Title

Statistical phylogenetic methods with applications to virus evolution

Permalink

https://escholarship.org/uc/item/362312kp

Author

Westesson, Oscar

Publication Date

2012

Statistical phylogenetic methods with applications to virus evolution

by

Oscar Westesson

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Joint Doctor of Philosophy
with University of California, San Francisco

in

Bioengineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Ian Holmes, Chair
Professor Rasmus Nielsen
Professor Joe DeRisi

Fall 2012

Statistical phylogenetic methods with applications to virus evolution

Abstract

Statistical phylogenetic methods with applications to virus evolution

by

Oscar Westesson

Joint Doctor of Philosophy

with University of California, San Francisco

in Bioengineering

University of California, Berkeley

Ian Holmes, Chair

This thesis explores methods for computational comparative modeling of genetic sequences. The framework within which this modeling is undertaken is that of sequence alignments and associated phylogenetic trees. The first part explores methods for building ancestral sequence alignments making explicit use of phylogenetic likelihood functions. New capabilities of an existing MCMC alignment sampler are discussed in detail, and the sampler is used to analyze a set of HIV/SIV gp120 proteins. An approximate maximum-likelihood alignment method is presented, first in a tutorial-style format and later in precise mathematical terms. An implementation of this method is evaluated alongside leading alignment programs. The second part describes methods utilizing multiple sequence alignments. First, mutation rate is used to predict positional mutational sensitivities for a protein. Second, the flexible, automated model-specification capabilities of the XRate software are presented. The final chapter presents recHMM, a method to detect recombination among sequence by use of a phylogenetic hidden Markov model with a tree in each hidden state.

För min farmor och farfar, som alltid undrade varför jag aldrig blev doktor.

Acknowledgments

This dissertation contains much of the work I did while in the lab of Ian Holmes at UC Berkeley. I am deeply indebted to Ian for taking me on as a student and providing a quiet example of what it means to be a modern computational scientist. The folks in and around the lab during my time there (anb before) were absolutely essential, each in their own ways, to my growth as a scientist and person. Mitch Skinner, Robert Bradley, Lars Barquist, Andrew Uzilov, and Alex James Hughes: thanks for the help, lunches, and pranks. Members of the Andino and DeRisi labs helped deepen my interest in virology and keep my ideas grounded in biology: Raul Andino, Joe DeRisi, Charles Runckel, Michael Schulte, and Cecily Burrill. Without the following teachers along the way I would perhaps never have come to love pure and applied maths as much as I do: Rich "R" Kelly, Naomi Jochnowitz, and Bernd Sturmfels. The very tricky transfer from University of Rochester to UC Berkeley would likely not have happened were it not for Maritza Aguilar, an extremely helpful and thorough admissions counselor.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Overview

That all living organisms are related by common ancestors [1], and that the history relating them is encoded in their DNA sequences [2], has profound implications for the study of evolution. The ease and accuracy with which genetic sequences can be extracted has paved the way for rigorous, quantitative approaches to studying the molecular basis for evolution, developing and testing mathematical models on real data.

By examining the sequences of extant species in an evolutionary context, a wide range of questions can be investigated. We can reconstruct what *has happened*: what mutations along the human lineage separate us from our closest primate relatives? Which evolutionary events are common, and which are rare? Is a newly-sequenced virus a mosaic of previously-known viral types, and when did this mixing occur? We can predict what *will happen*: given the observed mutational behavior of HIV, how likely are immune/drug escape mutations? If we mutate protein X at position n, is X likely to be functionally impaired? We can also make *annotation predictions*: using the known evolutionary patterns of functional RNA secondary structures in viruses, which regions in virus Y are likely to contain additional structured elements? Where are the protein-coding regions in a newly-sequenced organism's genome?

This dissertation focuses on approaching the above questions by way of the construction and analysis of sequence alignments. A sequence alignment is a compact summary of *homology* among a group of sequences, vertically orienting "equivalent" residues into columns. An alignment allows for a great possibility of analyses, ranging from simple heuristics to rigorous, complex probabilistic models. The structure of parts and chapters is shown graphically in Figure 1.1.

Part I (Chapters 2-4) of this dissertation focuses on methods for building sequence alignments. Chapter 2 presents the mathematical details of an alignment algorithm which approximates a maximum-likelihood solution under a broad class of evolutionary models. Chapter 3 evaluates ProtPal, an implementation of this method, next to other leading alignment methods in the context of evolutionary analyses. Chapter 4 outlines the capabilities of the Bayesian MCMC alignment sampler `handalign`.

Part II (Chapters 5-7) of this dissertation outlines three methods which utilize a multiple sequence alignment. Chapter 5 discusses a method to predict mutational sensitivity of particular positions in a protein by estimating position-specific mutation rates. Chapter 6 describes the XRate software, and in particular its capabilities for automated specification of large, complex evolutionary models. Chapter 7 describes RecHMM, a method for detecting recombination by partitioning alignments into regions associated with differing phylogenetic trees.

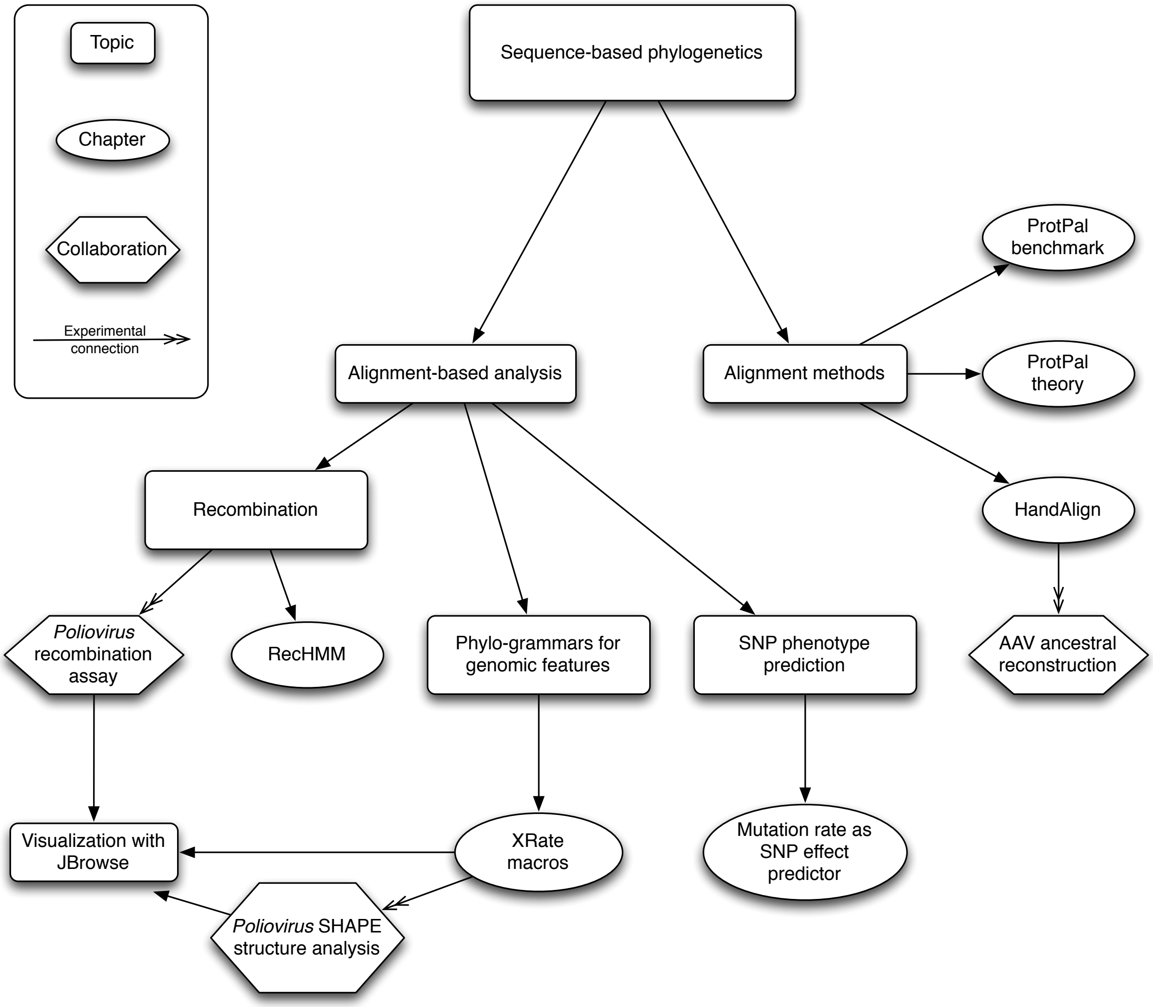

Topic
Chapter
Collaboration
Experimental connection
Sequence-based phylogenetics
Alignment-based analysis
Alignment methods
ProtPal benchmark
ProtPal theory
HandAlign
Recombination
Poliovirus recombination assay
RecHMM
Phylo-grammars for genomic features
SNP phenotype prediction
AAV ancestral reconstruction
Visualization with JBrowse
XRate macros
Mutation rate as SNP effect predictor
Poliovirus SHAPE structure analysis

Figure 1.1: The topics covered in this dissertation.

Methods for multiple sequence alignment

The first part of this dissertation explores methods for building sequence alignments. An alignment transforms the sequences of related organisms into a concise collection of evolutionary statements of homology: residues from related sequences are aligned together in columns if they are believed to originate from a common ancestor. Practically speaking, aligning sequences amounts to placing gaps where insertions and deletions have changed sequences' length so as to vertically homologous residues.

Three types of alignments are relevant to this work: *pairwise, multiple*, and *ancestral.* A *pairwise alignment* is an alignment of two sequences, one involving three or more is referred to as a *multiple sequence alignment* (MSA), and a multiple alignment in which the extant as

poliovirus	A	C	C	G	T
rhinovirus A	A	-	-	G	G

Figure 1.2: A **pairwise alignment** between the *poliovirus* sequence ACCGT and the *rhinovirus A* sequence AGG. Gaps are introduced such that aligned characters are believed to share a common ancestor. An insertion or deletion results in the *poliovirus* CC residues having no homologs in *rhinovirus*. A nucleotide substitution is evident in the last column, where G is aligned to T.

poliovirus	A	C	C	G	T
rhinovirus A	A	-	-	G	G
rhinovirus B	T	-	-	C	G

Figure 1.3: A **multiple alignment** between *poliovirus*, *rhinovirus A*, and *rhinovirus B*. Characters aligned in columns are hypothesized to share a common ancestor. Multiple alignment, as opposed to pairwise alignment, often cannot be globally optimized, and so alignment methods use heuristics to narrow the search space.

well as (predicted) ancestral sequences are included is called an *ancestral* alignment.

Figure 1 shows an example of a small alignment of sequences from the small RNA virus species *poliovirus* and *rhinovirus A*. Gaps (dashes) are used to account for the residues in *poliovirus* (CC) that do not occur in *rhinovirus A*. Note that homologous residues do not necessarily need to be the same: the G and T at the end of the sequences are predicted to be homologous, but a nucleotide substitution (mutation) some time since the *poliovirus* - *rhinovirus A* divergence has led to a difference between them.

Figure 1 shows a multiple alignment of related sequences from *poliovirus*, *rhinovirus A*, and *rhinovirus B*. Note that the history of CC in *poliovirus* is still unaccounted for - we cannot determine whether it was an insertion in the *poliovirus* lineage or a deletion in both the *rhinovirus A* and *rhinovirus B* lineages; we only know that this sequence is not shared among all the extant viruses.

Figure 1 shows an ancestral alignment of *poliovirus*, *rhinovirus A*, *rhinovirus B*, and their common ancestor. The characters of the ancestral sequence may be explicitly predicted or left as unknowns, indicated by asterisks. An ancestral alignment requires a phylogenetic tree relating the sequences to define the ancestors; for simplicity, a multifurcating "star" topology is assumed in Figure 1.

In Figure 1, since *poliovirus* has two characters CC that are not present in the ancestor, we can now confidently classify this as an insertion (rather than a deletion in the other two lineages). This was not possible using a pairwise or standard multiple alignment without making an additional prediction (e.g. based on parsimony). Note that while an ancestral alignment provides a finer-grained account of evolutionary history than does a multiple alignment (which is contained within an ancestral alignment), it still retains some ambiguity.

ancestor	*	-	-	*	*
poliovirus	A	C	C	G	T
rhinovirus A	A	-	-	G	G
rhinovirus B	T	-	-	C	G

Figure 1.4: An **ancestral alignment** between *poliovirus*, *rhinovirus A*, and *rhinovirus B*, and their common ancestor. Characters aligned in columns are hypothesized to share a common ancestor which is now explicitly predicted as an additional sequence in the alignment. The CC unique to *poliovirus* can now be classified as an insertion, since it is not present in the ancestral sequence. Ancestral characters are left as unknown, indicated by *.

For instance, in the example above, we do not know whether the individual Cs of *poliovirus* were deleted "one at a time" or as part of a multi-residue deletion, nor do we know the timing of this event - was it recent, or just after the three species diverged? This notion of reconstruction granularity will be explored further in Chapters 2 and 3.

While the optimal pairwise alignment can be found in polynomial time and memory for simple models [3], the space of possible multiple and ancestral alignments for of a group of sequences grows extremely quickly, and exact optimal solutions can rarely be obtained. To circumvent this, multiple alignment algorithms use a variety of heuristics in order to trim the search space of possible alignments. These can be decomposed into two parts: the objective function and the optimization strategy. The objective function is the criteria by which the algorithm ranks candidate alignments, and the optimization strategy is the way in which the objective function is maximized.

Curiously, though the MSA is often interpreted in an evolutionary context, few alignment programs are based on objective functions which are explicitly evolutionary likelihoods. That is, the goal of most alignment programs is not to discover alignments which correspond to reasonable evolutionary histories. The aims of Part I are to develop alignment methods based on phylogenetic objective functions, and to investigate how this shift in focus affects alignment quality.

In Chapter 2, we develop a method for building ancestral alignments whose objective function is based on a flexible class of evolutionary likelihood models (sequence transducers). While direct, exact inference under such an objective function is typically intractable, we derive an efficient hierarchical approximation method capable of typical alignment tasks. In Chapter 3, we evaluate an implementation of this method next to other leading alignment methods, using simulated evolutionary histories to determine the relative accuracy of the various programs. In Chapter 4, we describe the capabilities of `handalign`, a Bayesian MCMC alignment sampler capable of co-sampling alignments, trees, and model parameters.

Phylogenetic objective functions

We consider two methods using the objective function presented in [4], based on *string transducers*. String transducers are conditional pairwise models for sequence evolution: they describe the evolution of an ancestral sequence into its direct descendant. We can think of a transducer as an evolutionary machine, "absorbing" an ancestral sequence and "emitting" a descendant (possibly altered by substitutions, insertions, and deletions) conditional on the absorbed ancestor. Mathematically, a transducer is a probability distribution over descendants X conditional on an ancestor Y, taking the form $P(X|Y)$. Placing a transducer on each branch of a phylogenetic tree forms the basis of our objective function: we seek to find ancestral alignments which have a high likelihood under this evolutionary model. Direct, naive inference under such a model requires $\mathcal{O}(L^N)$ time and memory for N sequences of length L - intractable for all but the smallest alignment tasks. In Chapters 4 and 2 we explore two alternative inference strategies for this class of models.

Exact inference of alignment, tree, and parameters via MCMC A powerful approach to attacking large statistical inference problems is Markov chain Monte Carlo (MCMC) - stochastic sampling via local perturbations. If run sufficiently long, an MCMC chain is guaranteed to recover the full posterior distribution over hidden variables (which includes the maximum *a posteriori* (MAP) solution as a special case). Perhaps more importantly, all unobserved variables can be co-sampled, eliminating the need to arbitrarily set them beforehand. This is ideal for phylogenetic analysis, when only the sequences are truly observed: a pre-estimated phylogeny and/or evolutionary parameters can affect alignment inference in non-trivial ways, and the uncertainty associated with these inferences is not propagated forward.

Rigorous accounting for uncertainty and thoroughly exploring the solution space come at a steep computational cost; MCMC alignment samplers typically require orders of magnitude more time than heuristic tools. To combat this, in Chapter 4 we describe the capabilities of `handalign`, a Bayesian ancestral alignment sampler, including the range of models it is able to use, transition kernel options, accelerating/limiting heuristics, and trace post-processing tools. We run `handalign` on HIV/SIV gp120 proteins, finding that alignment of the "hypervariable" regions is so uncertain (most columns having posterior probability near zero) that alignment-based analysis of these regions should be avoided at all costs. We estimate the posterior distribution of indel rate of this protein to be approximately normally-distributed with mean 0.05 indels per substitution.

Approximate ancestral alignment using sequence profiles While `handalign` and other MCMC methods represent a statistically unbiased approach to ancestral alignment construction, there are many cases where its computational demands render it impractical or even impossible to use. Further, many downstream analyses require a single "best guess" alignment, and creating (let alone analyzing) an entire MCMC trace of alignments (and parameters and trees) may create a flood of data unnecessary to these analyses. For these reasons we sought to develop an alignment algorithm that uses the same phylogenetic objective function but is efficient enough to allow for typical alignment tasks. We endeavored to expose the mathematical underpinnings of such a method in sufficient detail so that an interested reader could implement or extend the method, but still readable enough so that non-experts could appreciate it.

We develop an algorithm for building ancestral alignments using an approximation to the phylogenetic likelihood function found in [4]. The algorithm is *progressive* - it traverses the phylogeny from leaves to root, with the final alignment constructed when the root is reached. At each progressive step (each corresponding to an internal node of the phylogeny), a subset of the possible subtree alignments is selected using stochastic traceback sampling. The size of this subset allows a strict bound on the number of reconstructions that are considered. In the limit of sampling infinitely many alignments at each progressive step, all solutions are retained, leading to exact inference over the set of $\mathcal{O}(L^N)$ possible alignments. If only one alignment is retained at each internal node, a progressive alignment algorithm similar to that of PRANK [5] is recovered. The former choice (generating infinitely many alignments) is intractable for reasonably-sized alignment tasks, and while the latter (using a single alignment) is computationally efficient, it may produce inaccurate alignments in situations when the optimal alignment of a subtree is only clear upon incorporating sequences outside that subtree.

In Chapter 2 we present this new method in two parts. The first is a tutorial-style overview of the relevant transducer algebra used by the underlying algorithms. The second part is a rigorous, complete description of the algebra and algorithms used for the method. While the two sections may be read independently, connections between them (e.g. visual representations of structures described in the second part) are clearly indicated.

In Chapter 3 we evaluate ProtPal, an implementation of this method next to other leading alignment methods. Our principal aim was to investigate the relative accuracy of the programs in a phylogenetic sense - how accurately do they reconstruct evolutionary history? This is related, though not equivalent, to the common practice of ranking programs by pairwise residue metrics (such as the Sum-of-Pairs (SPS), or Alignment Metric Accuracy (AMA)), which instead ask “how many pairwise residue homologies were correctly found?” Using insertion and deletion rates as measures of evolutionary reconstruction accuracy, we find that ProtPal is more accurate than all other tested programs. Though it is still susceptible to a bias towards deletions (as noted in [6] and Figure 3.3), the effect is significantly less than typical aligners, and even slightly better than PRANK, the previous state-of-the-art phylogenetic aligner. To demonstrate the practical capability and utility of ProtPal, we reconstruct the evolutionary history of $\sim$ 7500 human gene families from the Orthologous and Paralagous Transcripts in Clades (OPTIC) database. From these reconstructions, we find that indel rates are approximately gamma-distributed, with 95% of genes experiencing less than 0.1 indels per synonymous substitution, insertion and deletion lengths follow a comparable distribution, and an enrichment for regulatory and metabolic function among the 200 highest indel rate genes.

Alignment-based phylogenetic analysis

Part II of this dissertation explores analysis methods which utilize a multiple sequence alignment. The chapters of this part (Chapters 5-7) all involve modeling the substitution history of an alignment, as opposed to the insertion/deletion history as in Chapter 2, 3 and 4. Rather than treating gaps as manifestations of phylogenetic indel events, they are essentially not included in the likelihood. While this is somewhat unsatisfactory from a theoretical modeling perspective, the decrease in computational cost enables an extremely wide range of modeling possibilities.

In particular, it is possible to exactly summarize the spectrum of possible mutational histories generating an alignment, without resorting to heuristics or approximations, in $\mathcal{O}(LN)$ time and memory for N sequences of length L. This is quite remarkable, given that there are infinitely many mutational histories separating just two sequences separated by nonzero time if mutations are assumed to occur in continuous time. Modeling mutation histories over continuous time can be accomplished in a straightforward way (for finite state spaces) by using the *matrix exponential*, transforming a mutation rate matrix describing instantaneous change (e.g. at what rate does A mutate to T in an infinitesimally small time period) to a time-parametrized probability matrix (e.g. given that the sequence has an A at time zero, what is the probability that the sequence will have T after time t, accounting for possibly infinitely many substitutions under that time?).

Using such a probability matrix on each branch of a tree, it is possible to compute the likelihood of an entire alignment column by way of Felsenstein's "pruning" algorithm. The pruning recursion begins at the leaves of the tree and "prunes" leaves and subtrees one by one, tabulating the likelihood of each subtree conditional on a particular character present at its root. Upon reaching the root after $\mathcal{O}(N)$ operations, the likelihood of the column can be computed by summing over possible values of the root character. Under a simple "independent and identically-distributed" (IID) model, wherein each alignment column is assumed to be an independent realization of the same evolutionary process, computing the likelihood of an alignment simply involves multiplying the L individual column likelihoods [7].

Each chapter in Part II presents a venture beyond such a basic IID model. By quantifying these departures from IID, we can extract subtle evolutionary signals of interest, such as those of mutational sensitivity, RNA structures, or recombination. Chapter 5 describes a method to predict the mutational tolerance of individual positions of a protein based on their measured mutation rate. Chapter 6 describes the automated model-specification capabilities of the XRate software. Chapter 7 describes a method to detect recombination among sequences using a phylogenetic hidden Markov model with trees in each of the hidden states.

Predicting the functional effect of single nucleotide polymorphisms

One of the great hopes of personal genomic sequencing is to more accurately characterize the genetic components of disease. Genome-wide association studies (GWAS) aim to do this by correlating known disease phenotypes with genotypes, and have been successful with certain diseases [8–11]. An alternative, orthogonal approach is to use the protein's sequence itself, and predict the functional consequences of particular single nucleotide polymorphisms (SNPs). While this type of approach cannot, in itself, predict genotype-disease connections, it can allow investigators to zero in on the positions which may be more informative for phenotypes of interest.

Popular methods like SIFT [12], PolyPhen [13], PMut [14], and ASP [15] use structural, comparative, or evolutionary features to separate SNPs into neutral and deleterious classes. In Chapter 5, we develop a model in which each alignment column evolves at its own mutation rate, independent from neighboring columns. The estimated mutation rate for each column can then be used as a measure of "mutability" - the ability of a protein to tolerate a mutation at a given position without losing its function. Intuitively, if many mutations have been tolerated over the evolutionary history of the protein at a particular position (leading to a high measured mutation rate), it may be more likely that a mutation there will be tolerated in the extant state of the protein. We use large-scale SNP-phenytope datasets from HIV, *E. coli*, phage T4, and human proteins to estimate cutoffs allowing us to make binary (functional or nonfunctional) predictions from continuously-varying mutation rate measurements. Evaluating our predictions next to leading phenotype prediction programs, we find that using mutation rate as a mutability measure is the most accurate on all datasets tested.

XRate macros: detecting genomic features and structures

The model presented in Chapter 5 is a simple example of a broad class of phylogenetic models known as *phylogrammars*, a generalization of stochastic context-free grammars (SCFGs) originating from computational linguistics [16]. A phylogrammar is composed of two parts: a phylogenetic model (phylo-) and hidden transformations (grammar). Hidden transformations are repeatedly applied until only "emitting" symbols remain, which then give rise to the observed alignment columns according to the phylogenetic model(s). The high-level idea is that the grammar's transformations partition the alignment into distinct regions, each of which exhibits its own evolutionary patterns.

In practice the alignment is taken as observed and the sequence of hidden transformations (or "states") represents a structure of interest which is to be reconstructed probabilistically. For instance, a gene-finding phylogrammar could partition the alignment into "gene" and "intergenic" groups of columns, and an RNA structure grammar into "paired" and "unpaired" columns folding into stems, loops, and bulges. While this can be accomplished within reasonable time and memory constraints, implementation is non-trivial for all but the simplest of phylogrammars. However, the necessary algorithms all have the same general form, which can be summarized in terms of the structure of the grammar. For *regular* grammars (e.g. HMMs), these are known as the Forward and Backward recursion, named as such since they summarize subsequences starting at one end and move towards the other. For *context-free* grammars (e.g. models for RNA structure), which may include nested (as opposed to simpler right-left) correlations, the inference algorithms are the Inside and Outside recursions [17]. The XRate software [18] implements the relevant parameter estimation and inference algorithms for a wide range of phylogrammars. Models need only be specified in its Scheme-like grammar format.

However, even this specification can become limiting when prototyping large, repetitive, or data-dependent (such as the grammar in Chapter 5, whose structure depends on the length of the alignment) grammars. To combat this, XRate now includes a model-specification macro language allowing grammars to be defined using loops, conditionals, and data-dependent variables (such as the tree and alignment). This allows trivial implementation of simple grammars (such as an IID Jukes-Cantor grammar), and medium-size grammars (such as the column-specific rate grammar in Chapter 5) can be implemented and prototyped in an intuitive way with only a basic knowledge of the macro language. Large grammars which would be nearly impossible to implement coherently by hand are now possible with extensive use of the macro language. For instance, XDecoder, a model describing RNA structure which may (partially or fully) overlap protein-coding sequences, can be represented with a few hundred lines of macro code, whereas the “expanded” version is over 3500 lines long. A set of increasingly complex grammars, concluding with XDecoder, is presented in Chapter 6 to highlight the capabilities of XRate’s macro language for phylogrammar specification. Further, experimental evidence (from SHAPE chemistry performed on *poliovirus*) is presented alongside XDecoder’s pairing predictions, demonstrating its utility in identifying functional RNA structures in viral genomes.

RecHMM: detecting recombination

In Chapter 7 we consider a different kind of departure from a null IID phylogenetic model. The previous two chapters involve methods which assume two crucial pieces of input data: the alignment and the phylogenetic tree relating the aligned sequences. We now consider the situation where not only is the tree unknown (which, in practice, is quite common), but *multiple* trees may be needed to explain the diversity present in the alignment.

Biologically, a shift in tree topology between columns can be explained by *recombination* - an evolutionary event related to those discussed elsewhere in this work (e.g. substitutions, insertions, and deletions). A recombinant sequence is one which has more than one parent. Nearly all organisms participate in some form of genetic mixing, though its precise evolutionary roles may differ. Recombination in humans is extremely common-we are each a mixture of our two parents. In viruses, recombinant forms of HIV-1 [19–21] and *poliovirus* [22] frequently exhibit increased virulence, yet some hypothesize that population-level recombination may actually be deleterious in most cases [23].

Whatever it’s evolutionary role, accurate detection of recombination breakpoints remains a difficult problem. Previous methods such as SimPlot [24] and BootScan use sliding windows and summary statistics (similarity to reference strains and percent of trees showing reference-query clustering, respectively) to detect changes in ancestry along a query sequence. While this may suffice for coarse-grained analysis, predictions depend heavily on the window size used. Further, more detailed analysis aiming to estimate genomic breakpoint distributions or investigate the sequence features responsible for modulating local recombination rates [25,26] depend on fine-scale predictions. The HMM-based approach by Husmeier and Wright [27], embedding a phylogenetic tree in each hidden state, allows for dispensing with the sliding window: using standard HMM inference algorithms, all possible breakpoint locations can efficiently be considered. Their approach was limited to 4 taxa since the trees in each hidden state were fixed beforehand, severely hampering its practical utility.

In Chapter 7, we present an extension to the Husmeier and Wright [27] approach which avoids the need to fix trees beforehand. Instead, the method uses Structural EM [28] to simultaneously estimate trees in hidden states and determine recombination breakpoints. The result is a method free of sliding windows that can handle many taxa. We evaluate the method's accuracy via simulation studies in which both simulation and inference parameters are varied over a wide range. We investigate published recombinant Neisseria and HIV-1 data, finding that our method recovers all previously-predicted breakpoints and, in many cases, find additional possible breakpoints.

Recently, experimental systems have been designed which can measure recombination rates with high resolution and throughput. Such an assay was developed in *poliovirus*, finding that most of the recombination occurring in a single replication cycle is concentrated in a few “hotspots”, where GC tracts and RNA structure have elevated the recombination rate. The most pronounced of these hotspots is near the large structured RNAse-L element at the 3’ end of the genome. A recent recHMM-based analysis of circulating vaccine-derived *poliovirus* strains shows several recombination events in that region, suggesting that mechanistic factors may play a significant role in determining breakpoint distributions.

Part I

Methods for multiple sequence alignment

Chapter 2

Approximate alignment with transducers: theory

The following chapter contains work conducted together with Gerton Lunter, Benedict Paten, and Ian Holmes, submitted to the arXiv [29].

Overview

We present an extension of Felsenstein's algorithm to indel models defined on entire sequences, without the need to condition on one multiple alignment. The algorithm makes use of a generalization from probabilistic substitution matrices to weighted finite-state transducers. Our approach may equivalently be viewed as a probabilistic formulation of progressive multiple sequence alignment, using partial-order graphs to represent ensemble profiles of ancestral sequences. We present a hierarchical stochastic approximation technique which makes this algorithm tractable for alignment analyses of reasonable size.

2.1 Background

Felsenstein's pruning algorithm is routinely used throughout bioinformatics and molecular evolution [7]. A few common applications include estimation of substitution rates [30]; reconstruction of phylogenetic trees [31]; identification of conserved (slow-evolving) or recently-adapted (fast-evolving) elements in proteins and DNA [32]; detection of different substitution matrix "signatures" (e.g. purifying vs diversifying selection at synonymous codon positions [33], hydrophobic vs hydrophilic amino acid signatures [34], CpG methylation in genomes [35], or basepair covariation in RNA structures [36]); annotation of structures in genomes [37, 38]; and placement of metagenomic reads on phylogenetic trees [39].

The pruning algorithm computes the likelihood of observing a single column of a multiple sequence alignment, given knowledge of an underlying phylogenetic tree (including a map from leaf-nodes of the tree to rows in the alignment) and a substitution probability matrix associated with each branch of the tree. Crucially, the algorithm sums over all unobserved substitution histories on internal branches of the tree. For a tree containing N taxa, the algorithm achieves $\mathcal{O}(N)$ time and memory complexity by computing and tabulating intermediate probability functions of the form $G_n(x) = P(Y_n|x_n = x)$, where x_n represents the individual residue state of ancestral node n, and $Y_n = \{y_m\}$ represents all the data at leaves $\{m\}$ descended from node n in the tree (i.e. the observed residues at all leaf nodes m whose ancestors include node n).

The pruning recursion visits all nodes in postorder. Each G_n function is computed in terms of the functions G_l and G_r of its immediate left and right children (assuming a binary tree):

$$\begin{aligned} G_n(x) &= P(Y_n|x_n = x) \\ &= \begin{cases} \left(\sum_{x_l} B^{(l)}_{x,\,x_l} G_l(x_l)\right) \left(\sum_{x_r} B^{(r)}_{x,\,x_r} G_r(x_r)\right) & \text{if } n \text{ is not a leaf} \\ \delta(x = y_n) & \text{if } n \text{ is a leaf} \end{cases} \end{aligned}$$

where $B_{ab}^{(n)} = P(x_n = b|x_m = a)$ is the probability that node n has state b, given that its parent node m has state a; and $\delta(x = y_n)$ is a Kronecker delta function terminating the recursion at the leaf nodes of the tree.

The "states" in the above description may represent individual residues (nucleotides, amino acids), base-pairs (in RNA secondary structures) or base-triples (codons). Sometimes, the state space is augmented to include gap characters, or latent variables. In the machine learning literature, the G_n functions are often described as "messages" propagated from the leaves to the root of the tree [40], and corresponding to a summary of the information in the subtree rooted at n.

The usual method for extending this approach from individual residues to full-length sequences assumes both that one knows the alignment of the sequences, and that the columns of this alignment are each independent realizations of single-residue evolution. One uses pruning to compute the above likelihood for a single alignment column, then multiplies together the probabilities across every column in the alignment. For an alignment of length L, the time complexity is $\mathcal{O}(LN)$ and the memory complexity $\mathcal{O}(N)$. This approach works well for marginalizing substitution histories consistent with a single alignment, but does not readily generalize to summation over indel histories or alignments.

The purpose of this manuscript is to introduce another way of extending Felsenstein's recursion from single residues (or small groups of residues) to entire, full-length sequences, without needing to condition on a single alignment. With no constraints on the algorithm, using a branch transducer with c states, the time and memory complexities are $\mathcal{O}((cL)^N)$, with close similarity to the algorithms of Sankoff [41] and Hein [42]. For a user-specified maximum internal profile size $p \geq L$, the worst-case complexity drops to $\mathcal{O}(c^2p^3N)$ (typical case is $\mathcal{O}((cp)^2N)$ when a stochastic lower-bound approximation is used; in this form, the algorithm is similar to the partial order graph for multiple sequence alignment [43]. Empirical tests indicate that the typical-case complexity drops further to $\mathcal{O}(cpN)$ if an"alignment envelope" is provided as a clue to the algorithm. The alignment envelope is not a hard constraint, and may be controllably relaxed, or dispensed with altogether.

The new algorithm is, essentially, algebraically equivalent to Felsenstein's algorithm, if the concept of a "substitution matrix" over a particular alphabet is extended to the countably-infinite set of all sequences over that alphabet. Our chosen class of "infinite substitution matrix" is one that has a finite representation: namely, the *finite-state transducer*, a probabilistic automaton that transforms an input sequence to an output sequence, a familiar tool of statistical linguistics [44].

In vector form, Felsenstein's pruning recursion is

$$G_n = \begin{cases} \left(B^{(l)}G_l\right) \circ \left(B^{(r)}G_r\right) & \text{if } n \text{ is not a leaf} \\ \nabla(y_n) & \text{if } n \text{ is a leaf} \end{cases}$$

where $A \circ B$ is the pointwise (Hadamard) product and $\nabla(x)$ is the unit column vector in dimension x. By generalizing a few key algebraic ideas from matrices to transducers (matrix multiplication, the pointwise product, row vectors and column vectors), we are able to interpret this vector-space form of Felsenstein's algorithm as the specification of a composite phylogenetic transducer that spans all possible alignments (see Section 2.3).

The transducer approach offers a natural generalization of Felsenstein's pruning recursion to indels, since it can be used to calculate

$$P(S|T,\theta) = \sum_A P(S,A|T,\theta)$$

i.e. the likelihood of sequences S given tree T and parameters θ, summed over all alignments A. Previous attempts to address indels phylogenetically have mostly returned $P(S|\hat{A},T,\theta)$ where $\hat{A}$ represents a single alignment (typically estimated by a separate alignment program, which may introduce undetermined biases). The exceptions to this rule are the "statistical alignment" methods [42,45–48] which also marginalize alignments in an unbiased way—albeit more slowly, since they use Markov Chain Monte Carlo methods (MCMC). In this sense, the new algorithm may be thought of as a fast, non-MCMC approximation to statistical alignment.

The purpose of this manuscript is a clean theoretical presentation of the algorithm. In separate work [49] we find that the algorithm appears to recover more accurate reconstructions of simulated phylogenetic indel histories, as indicated by proxy statistics such as the estimated indel rate.

The use of transducers in bioinformatics has been reported before [4, 50–53] including an application to genome reconstruction that is conceptually similar to what we do here for proteins [53]. In particular, to maximize accessibility, we have chosen to use a formulation of finite-state transducers that closely mirrors the formulation available on Wikipedia at the time of writing
(`http://en.wikipedia.org/w/index.php?title=Finite_state_transducer&oldid=486381386`). This presentation is consistent with others described in the computer science literature [44].

Document structure

We will begin with a narrative, "tutorial" overview that introduces the main theoretical concepts using a small worked example. Following this we will present general, precise, technical definitions. The informal overview makes extensive use of illustrative figures, and is intended to be easily read, even by someone not familiar with transducer theory. The technical description is intended primarily for those wishing to integrate the internals of this method into their own algorithms. Either section may be read in isolation.

Each example presented in this informal section (e.g. single transducers, composite transducers, sampled paths) correspond to rigorously-defined mathematical constructs defined in the technical section. Whenever possible, we provide references between the examples and their technical definitions.

The final section of the tutorial, Section 2.2, gives a detailed account of the connections between the tutorial and formal sections.

2.2 Informal tutorial on transducer composition

In this section we introduce (via verbal descriptions and graphical representations) the various machines and manners of combining them necessary for the task of modeling evolution on the tree shown in Figure 2.1. (The arrangement of machines required for this tree is shown in Figure 2.2.) While the conceptual underpinnings of our algorithm are not unusually complex, a complete mathematical description demands a significant amount of technical notation (which we provide in Section 2.3). For this reason, we aim to minimize notation in this section, instead focusing on a selection of illustrative example machines ranging from simple to complex.

We first describe the sorts of state machines used, beginning with simple linear machines (which appear at the leaves of the tree in Figure 2.2) and moving on to the various possibilities of the branch model. Then we describe (and provide examples for) the techniques which allow us to co-ordinate these machines on a phylogeny: composition and intersection.

Finally, we outline how combinations of these machines allow a straightforward definition of Felsenstein's pruning algorithm for models allowing insertion/deletion events, and a stochastic approximation technique which will allow inference on datasets of common practical size.

Transducers as input-output machines

We begin with a brief definition of transducers from Wikipedia. These ideas are defined with greater mathematical precision in Section 2.3.

A finite state transducer is a finite state machine with two tapes: an input tape and an output tape. ... An automaton can be said to recognize *a string if we view the content of its tape as input. In other words, the automaton computes a function that maps strings into the set* $\{0, 1\}$. († † †) *Alternatively, we can say that an automaton* generates *strings, which means viewing its tape as an output tape. On this view, the automaton generates a formal language, which is a set of strings. The two views of automata are equivalent: the function that the automaton computes is precisely the indicator function of the set of strings it generates... Finite State Transducers can be weighted, where each transition is labeled with a weight in addition to the input and output labels.*
`http://en.wikipedia.org/w/index.php?title=Finite_state_transducer&oldid=486381386`

(† † †) For a weighted transducer this mapping is, more generally, to the nonnegative real axis $[0, \infty)$ rather than just the binary set $\{0, 1\}$.

In this tutorial section we are going to work through a small examples of using transducers on a tree for three tiny protein sequences (MF, CS, LIV). Specifically, we will compute the likelihood of the tree shown in Figure 2.1, explaining the common descent of these three sequences under the so-called TKF91 model (Figure 2.16), as well as a simpler model that only allows point substitutions. To do this we will construct (progressively, from the bottom up) the ensemble of transducer machines shown in Figure 2.2. We will

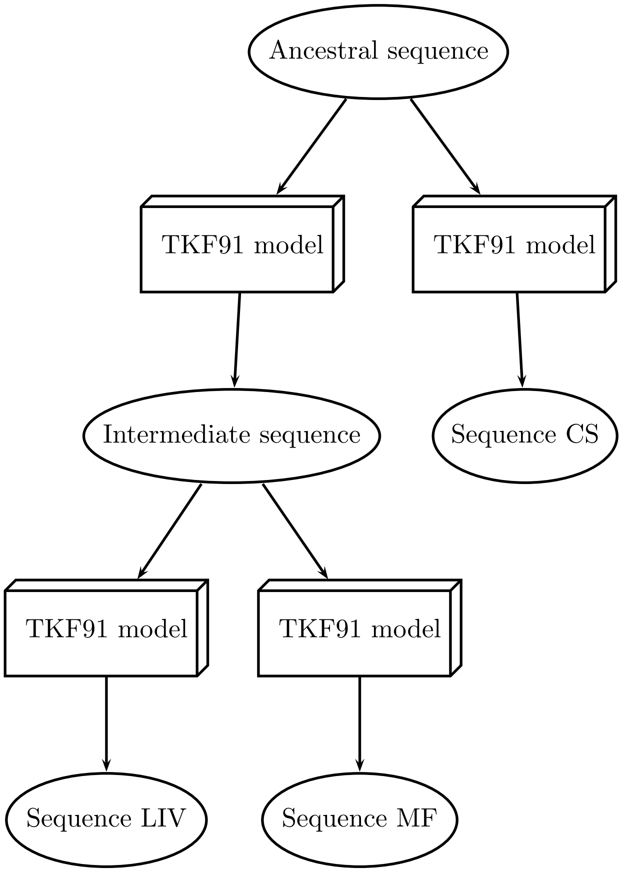


Figure 2.1: Example tree used in this tutorial. The TKF91 model is used as the branch transducer model, but our approach is applicable to a wider range of string transducer models.

see that the full state space of Figure 2.2 is equivalent to Hein's $\mathcal{O}(L^N)$ alignment algorithm for the TKF91 model [42]; $\mathcal{O}(NL^2)$ progressive alignment corresponds to a greedy Viterbi/maximum likelihood-traceback approximation, and partial-order graph alignment corresponds to a Forward/stochastic-traceback approximation.

Generators and recognizers

As noted in the Wikipedia quote, transducers can be thought of as generalizations of the related concepts of *generators* (state machines that emit output sequences, such as HMMs) and parsers or *recognizers* (state machines that match/parse input sequences, such as the UNIX 'lex' program). Both generators and recognizers are separate special cases of transducers. Of particular use in our treatment are generators/recognizers that generate/recognize a single unique sequence. Generators and recognizers are defined with greater precision in Section 2.3 and Section 2.3.

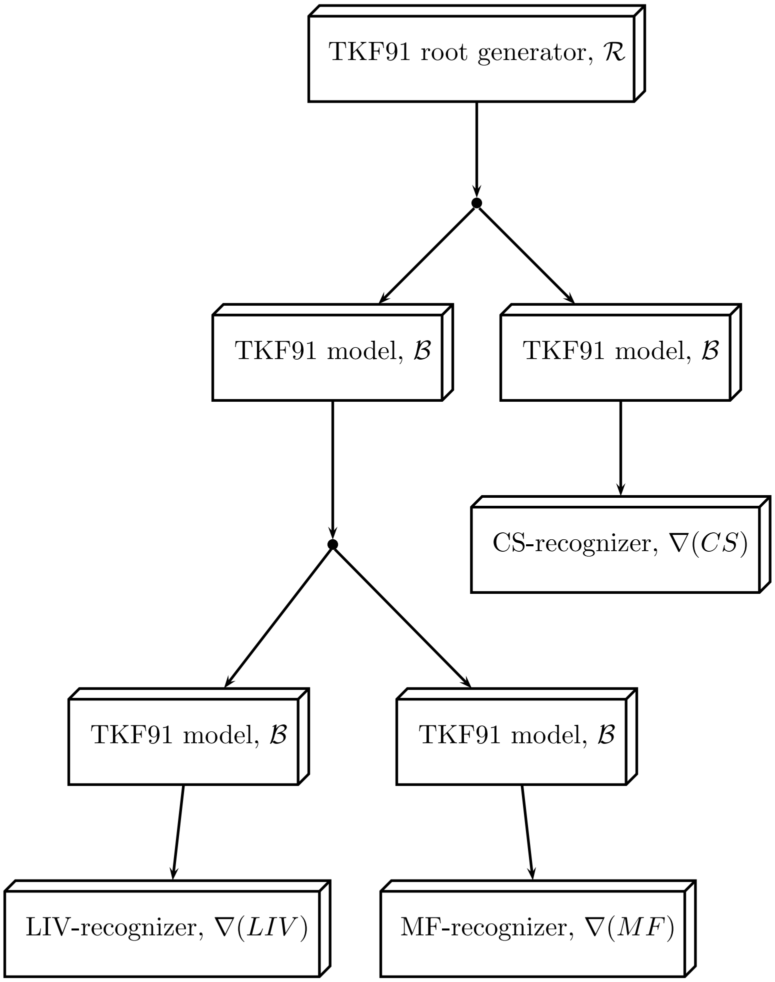

TKF91 root generator, $\mathcal{R}$
TKF91 model, $\mathcal{B}$
TKF91 model, $\mathcal{B}$
CS-recognizer, $\nabla(CS)$
TKF91 model, $\mathcal{B}$
TKF91 model, $\mathcal{B}$
LIV-recognizer, $\nabla(LIV)$
MF-recognizer, $\nabla(MF)$

Figure 2.2: An ensemble of transducers modeling the likelihood of the tree shown in Figure 2.1. We write this as $\mathcal{R} \cdot (\mathcal{B} \cdot (\mathcal{B} \cdot \nabla(LIV)) \circ (\mathcal{B} \cdot \nabla(MF))) \circ (\mathcal{B} \cdot \nabla(CS))$. The terms in this expression represent individual component transducers: $\mathcal{R}$ is shown in Figure 2.18, $\mathcal{B}$ is shown in Figure 2.17, $\nabla(LIV)$ is in Figure 2.10, $\nabla(MF)$ in Figure 2.11, and $\nabla(CS)$ in Figure 2.12. (The general notation $\nabla(\ldots)$ is introduced in Section 2.2 and formalized in Section 2.3.) The operations for combining these transducers, denoted "$\cdot$" and "$\circ$", are—respectively—*transducer composition* (introduced in Section 2.2, formalized in Section 2.3) and *transducer intersection* (introduced in Section 2.2, formalized in Section 2.3). The full state graph of this transducer is not shown in this manuscript: even for such a small tree and short sequences, it is too complex to visualize easily (the closest thing is Figure 2.42, which represents this transducer configuration minus the root generator, $\mathcal{R}$).

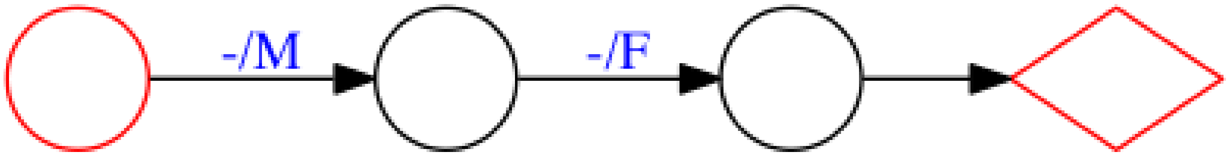


Figure 2.3: Generator for protein sequence MF. This is a trivial state machine which emits (generates) the sequence MF with weight (probability) 1. The red circle indicates the Start state, and the red diamond the End state.

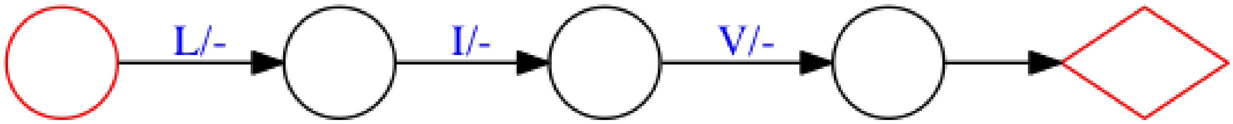


Figure 2.4: Recognizer for protein sequence LIV. This is a trivial state machine which absorbs (recognizes) the sequence LIV with weight (probability) 1, and all other sequences with weight 0. The red circle indicates the Start state, and the red diamond the End state.

Figure 2.3 is an example of a generator that uniquely generates (inserts) the protein sequence MF. Figure 2.4 is an example of a recognizer that uniquely recognizes (and deletes) the protein sequence LIV.

These Figures illustrate the visual notation we use throughout the illustrative Figures of this tutorial. States and transitions are shown as a graph. Transitions can be labeled with absorption/emission pairs, written x/y where x is the absorbed character and y the emitted character. Either x or y is allowed to be the empty string (shown in these diagrams as the gap character, a hyphen). In a Figure that shows absorption/emission pairs, if there is no absorption/emission labeled on a transition, then it can be assumed to be $-/-$ (i.e. no character is absorbed or emitted) and the transition is said to be a “null” transition.

Some transitions are also labeled with weights. If no transition label is present, the weight is usually 1 (some more complicated diagrams omit all the weights, to avoid clutter). The weight of a path is defined to be the product of transition weights occuring on the path.

The weight of an input-output sequence-pair is the sum over all path weights that generate the specified input and output sequences. The weight of this sequence-pair can be interpreted as the joint probability of both (a) successfully parsing the input sequence and (b) emitting the specified output sequence.

Note sometimes this weight is zero—e.g. in Figure 2.4 the weight is zero except in the unique case that the input tape is LIV, when the weight is one—this in fact makes Figure 2.4 a special kind of recognizer: one that only recognizes a single string (and recognizes that string with weight one). We call this an *exact-match* recognizer.

More generally, suppose that G, R and T are all probabilistically weighted finite-state transducers: G is a generator (output only), R is a recognizer (input only) and T is a general transducer (input *and* output). Then, conventionally, G defines a probability distribution $P(Y|G)$ over the emitted output sequence Y; R defines a probability $P(\text{accept}|X, R)$ of accepting a given input sequence X; and T defines a joint probability $P(\text{accept}, Y|X, T)$ that input X will be accepted and output Y emitted. According to this convention, it is reasonable to expect these weights to obey the following (here Ω^* denotes the set of all sequences):

$$\begin{aligned}
\sum_{Y \in \Omega^*} P(Y|G) &= 1 \\
P(\text{accept}|X, R) &\leq 1 \quad \forall X \in \Omega^* \\
\sum_{Y \in \Omega^*} P(\text{accept}, Y|X, T) &\leq 1 \quad \forall X \in \Omega^*
\end{aligned}$$

It is important to state that these are just conventional interpretations of the computed weights: in principle the weights can mean anything we want, but it is common to interpret them as probabilities in this way.

Thus, as noted in the Wikipedia quote, generators and recognizers are in some sense equivalent, although the probabilistic interpretations of the weights are slightly different. In particular, just as we can have a *generative profile* that generates some sequences with higher probability than others (e.g. a profile HMM) we can also have a *recognition profile*: a transducer that recognizes some sequences with higher probability than others. The exact-match transducer of Figure 2.4 is a (trivial and deterministic) example of such a recognizer; later we will see that the stored probability vectors in the Felsenstein pruning recursion can also be thought of as recognition profiles.

Moore machines

In our mathematical descriptions, we will treat transducers as *Mealy machines*, meaning that absorptions and emissions are associated with transitions between states. In the *Moore machine* view, absorptions and emissions are associated with states. In our case, the distinction between these two is primarily a semantic one, since the structure of the machines and the I/O functions of the states is intimately tied.

The latter view (Moore) can be more useful in bioinformatics, where rapid point substitution means that all combinations of input and output characters are possible. In such situations, the Mealy type of machine can suffer from an excess of transitions, complicating the presentation. For example, consider the Mealy-machine-like view of Figure 2.5, and compare it with the more compact Moore-machine-like view of Figure 2.6.

Figure 2.7 shows the visual notation we use in this tutorial for Moore-form transducer state types. There are seven *state types*: Start, Match, Insert, Delete, End, Wait, and Null, frequently abbreviated to S, M, I, D, E, W, N. State types are defined precisely in Section 2.3. Note that in the TKF91 model (Figure 2.16, the example we use for most of this tutorial) there are exactly one of each of these types, but this is not a requirement. For

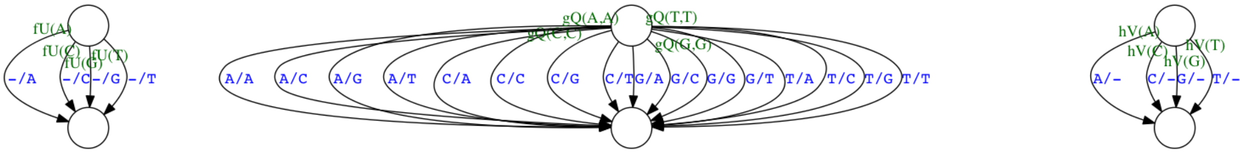


Figure 2.5: All combinations of input and output characters are frequently observed. The transitions in this diagram include all possible deletion, insertion, and substitution transitions between a pair of transducer states. Each transition is labeled with I/O characters (blue) and selected transitions are labeled with the transition weights (green). Insertions (I/O label "`-/y`") have weight $fU(y)$, deletions (I/O label "`x/-`") have weight $hV(x)$, and substitutions (I/O label "`x/y`") have weight $gQ(x, y)$. The large number of transitions complicates the visualization of such "Mealy-machine" transducers. We therefore use a "Moore-machine" representation, where all transitions of each type between a pair of states are collapsed into a single transition, and I/O weights are associated with states (Figure 2.6).

instance, the transducer in Figure 2.20 has two Insert, two Delete and three Wait states, while Figure 2.14 has no Insert or Delete states at all; Figure 2.9 has no Match states; and so on.

Some other features of this view include the following:

- The shape and color of states indicates their type. The six Moore normal form states are all red. Insert states point upwards; Delete states point downwards; Match states are rectangles; Wait states are octagons (like U.S. Stop signs); Start is a circle and End is a diamond. There is a seventh state type, Null, which is written as a black circle to distinguish it from the six Moore-form state types. (Null states have no associated inputs or outputs; they arise as a side-effect of algorithmically-constructed transducers in Section 2.2 and Section 2.2. In practice, they are nuisance states that must be eliminated by marginalization, or otherwise dealt with somehow.) This visual shorthand will be used throughout.

- We impose certain constraints on states that involve I/O: they must be typed as Insert, Delete, or Match, and their type determines what kinds of I/O happens on transitions into those states (e.g. a Match state always involves an absorption and an emission).

- We impose certain constraints on transitions into I/O states: their weights must be factorizable into transition and I/O components. Suppose j is a Match state and i is a state that precedes j; then all transitions $i \to j$ must both absorb a non-gap input character x and emit a non-gap output character y, so the transition can be written $i \xrightarrow{x/y} j$ and the transition weight must take the form $t_{ij} \times e_j(x, y)$ where t_{ij} is a component that can depend on the source and destination state (but not the I/O characters) and $e_j(x, y)$ is a component that can depend on the I/O characters and the destination state (but not the source state).

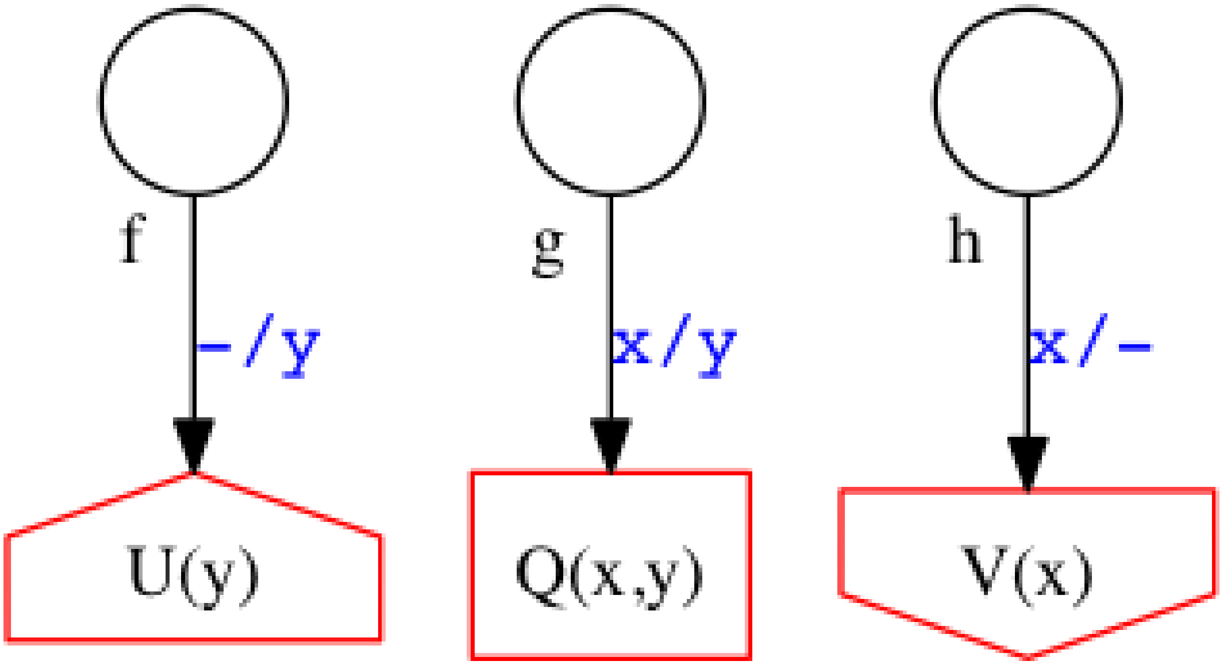


Figure 2.6: In a condensed Moore-machine-like representation, possible combinations of input and output characters are encoded in the distributions contained within each state, simplifying the display. In this diagram, all four insertion transitions from Figure 2.5 (`-/A`, `-/C`, etc.) are collapsed into a single `-/y` transition; similarly, all four deletion transitions from Figure 2.5 (`A/-`, etc.) are collapsed into one `x/-`, and all sixteen substitution transitions (`A/A`, `A/C`, . . . `G/G`) are collapsed to one `x/y`. To allow this, the transition weights for all the collapsed transitions must factorize into independent I/O- and transition-associated components. In this example (corresponding to Figure 2.5), the I/O weights are $U(y)$ for insertions, $Q(x, y)$ for substitutions and $V(x)$ for deletions; while the transition weights are f for insertions, g for substitutions, and h for deletions. **Visual conventions.** The destination state node is bordered in red, to indicate that transitions into it have been collapsed. Instead of just a plain circle, the node shape is an upward house (insertions), a downward house (deletions), or a rectangle (substitutions). Instead of specific I/O character labels (`A/G` for a substitution of A to G, `-/C` for an insertion of C, etc.) we now have generic labels like `x/y` representing the set of all substitutions; the actual characters (and their weights) are encoded by the I/O functions. For most Figures in the remainder of this manuscript, we will omit these blue generic I/O labels, as they are implied by the node shape of the destination state.

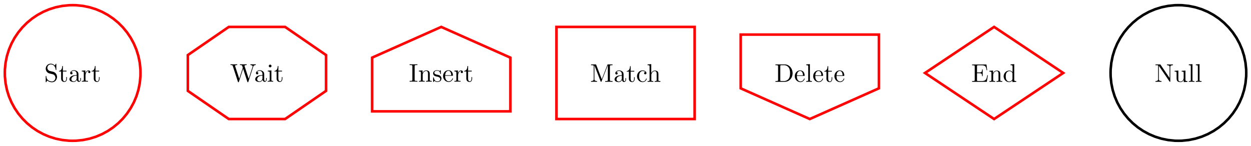


Figure 2.7: In a Moore machine, each state falls into one of several *types* (a transducer may contain more than one state of each type). A state's type determines its I/O capabilities: an Insert state emits (writes) an output character, a Delete state absorbs (reads) an input character, and a Match state both absorbs an input character and emits an output character. Mnemonics: Insert states point upwards, Delete states point Downwards, Wait states are octagonal like U.S. Stop signs.

- We can then associate the "*I/O weight function*" e_j with Match state j and the "*transition weight*" t_{ij} with a single conceptual transition $i \to j$ that summarizes all the transitions $i \overset{x/y}{\to} j$ (compare Figure 2.5 and Figure 2.6).
- The function e_j can be thought of as a conditional-probability substitution matrix (for Match states, c.f. Q in Figure 2.6), a row vector representing a probability distribution (for Insert states, c.f. U in Figure 2.6), or a column vector of conditional probabilities (for Delete states, c.f. V in Figure 2.6).
- Note that we call e_j an "I/O function" rather than an "emit function". The latter term is more common in bioinformatics HMM theory; however, e_j also describes probabilistic weights of *absorptions* as well as *emissions*, and we seek to avoid ambiguity.

Figure 2.8 shows the allowed types of transition in Moore-normal form transducers. In our "Moore-normal form" for transducers, we require that all input states (Match, Delete) are immediately preceded in the transition graph by a Wait state. This is useful for co-ordinating multiple transducers connected together as described in later sections, since it requires the transducer to "wait" for a parent transducer to emit a character before entering an absorbing state. The downside is that the state graph sometimes contains a few Wait states which appear redundant (for example, compare Figure 2.9 with Figure 2.3, or Figure 2.10 with Figure 2.4). For most Figures in the remainder of this manuscript, we will leave out the blue "x/y" labels on transitions, as they are implied by the state type of the destination state.

Note also that this graph depicts the transitions between *types* of states allowed under our formalism, rather than a *particular* state machine. It happens that the TKF91 model (Figure 2.17) contains exactly one state of each type, so its state graph appears similar to Figure 2.8, but this is not true of all transducers.

Generators and recognizers in Moore normal form

We provide here several examples of small transducers in Moore normal form, including versions of the transducers in Figure 2.3 and Figure 2.4.

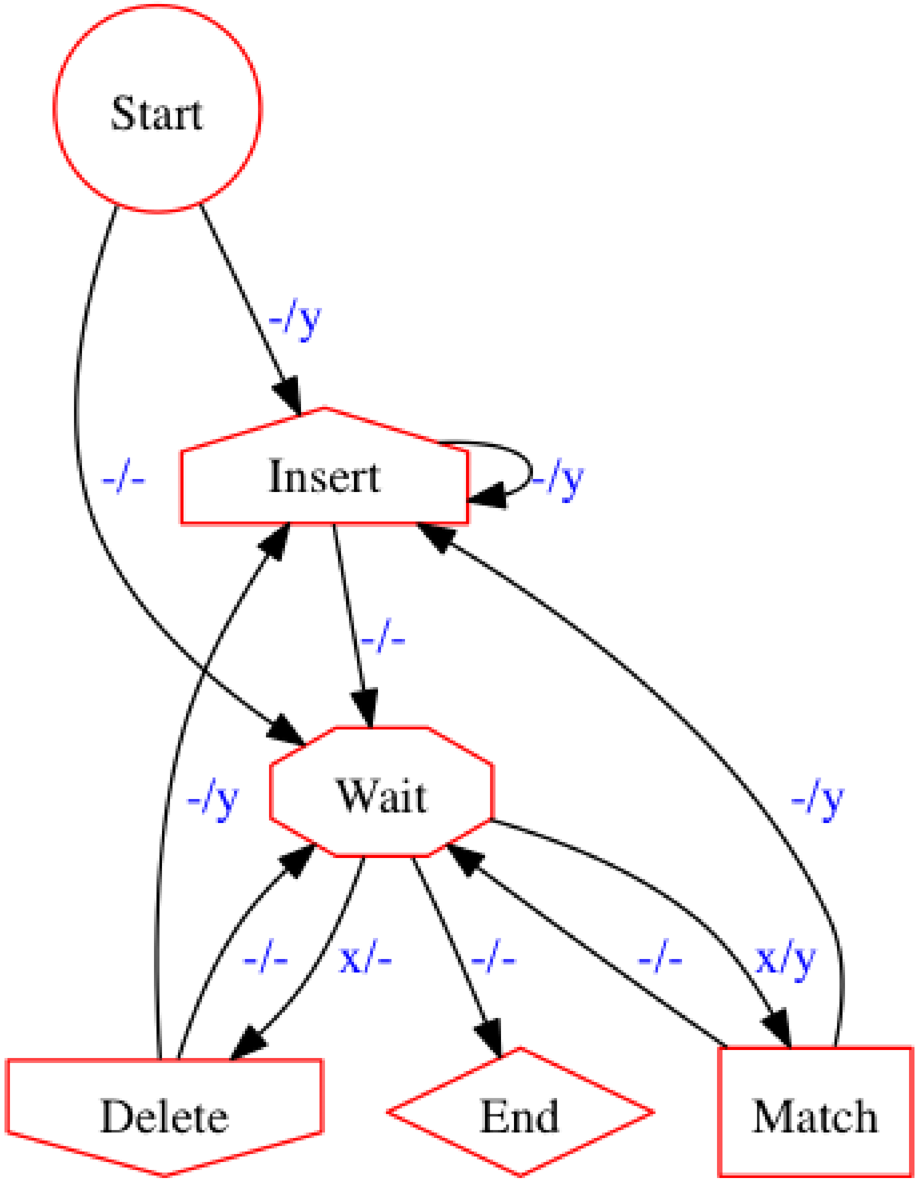


Figure 2.8: Allowed transitions between types of transducer states, along with their I/O requirements. In particular, note the Wait state(s) which must precede all absorbing (Match and Delete) states—the primary departure from the familiar pair HMM structure. Wait states are useful in co-ordinating multiple connected trandsucers, since they indicate that the transducer is "waiting" for an upstream transducer to emit a character before entering an absorbing state. Also note that this graph is not intended to depict a *particular* state machine, but rather it shows the transitions which are permitted between the *types* of states of arbitrary machines under our formalism. Since the TKF91 model (Figure 2.17) contains exactly one state of each type, its structure is similar to this graph (but other transducers may have more or less than one state of each type).

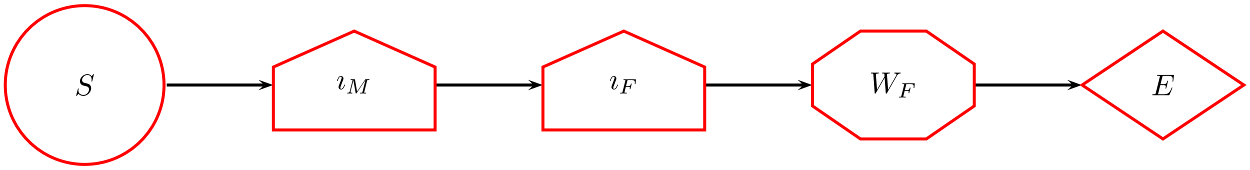


Figure 2.9: Transducer $\Delta(MF)$, the Moore-normal form generator for protein sequence MF. The states are labeled S (Start), E (End), $\imath_M$ and $\imath_F$ (Insert states that emit the respective amino acid symbols), and W_F (a Wait state that pauses after emitting the final amino acid; this is a requirement imposed by our Moore normal form). The state labeled $\imath_Z$ (for $Z \in \{M, F\}$) has I/O function $\delta(y = Z)$.

We introduce a notation for generators (Δ) and recognizers (∇); a useful mnemonic for this notation (and for the state types in Figure 2.7) is "insertions point up, deletions point down".

- Figure 2.9 uses our Moore-machine visual representation to depict the generator in Figure 2.3. We write this transducer as $\Delta(MF)$.
- Figure 2.10 is a Moore-form recognizer for sequence LIV. We write this transducer as $\nabla(LIV)$. The state labeled δ_Z (for $Z \in \{L, I, V\}$) has I/O function $\delta(x = Z)$, defined to be 1 if $x = Z$, 0 otherwise. The machine recognizes sequence LIV with weight 1, and all other sequences with weight 0.
- Figure 2.11 is the Moore-machine recognizer for MF, the same sequence whose generator is shown in Figure 2.9. We write this transducer as $\nabla(MF)$.
- Figure 2.12 is the Moore-machine recognizer for sequence CS. We write this transducer as $\nabla(CS)$.
- Figure 2.13 is a "null model" generator that emits a single IID sequence (with residue frequency distribution π) of geometrically-distributed length (with geometric parameter p).

Substitution and identity

Figure 2.14 shows how the Moore-normal notation can be used to represent a substitution matrix. The machine pauses in the Wait state before absorbing each residue x and emitting a residue y according to the distribution Q_{xy}. Since there are no states of type Insert or Delete, the output sequence will necessarily be the same length as the input.

This is something of a trivial example, since it is certainly not necessary to use transducer machinery to model point substitution processes. Our aim is to show explicitly how a familiar

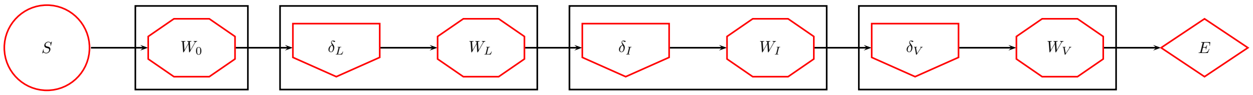


Figure 2.10: Transducer $\nabla(LIV)$, the Moore-normal form recognizer for protein sequence LIV. The states are labeled S (Start), E (End), δ_L, δ_I and δ_V (Delete states that recognize the respective amino acid symbols), W_L, W_I and W_V (Wait states that pause after recognizing each amino acid; these are requirements imposed by our Moore normal form). The states have been grouped (enclosed by a rectangle) to show four clusters: states that are visited before any of the sequence has been recognized, states that are visited after "L" has been recognized, states that are visited after "I" has been recognized, and states that are visited after "V" has been recognized. The I/O function associated with each Delete state δ_Z is $\delta(x = Z)$.

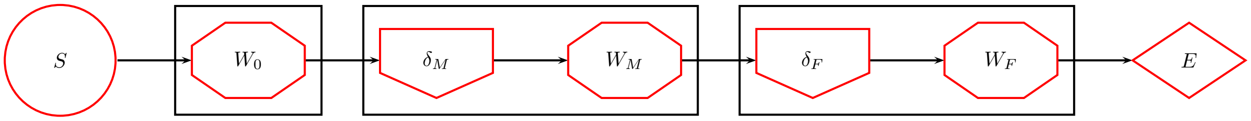


Figure 2.11: Transducer $\nabla(MF)$, the Moore-normal form recognizer for protein sequence MF. The states are labeled S (Start), E (End), δ_M and δ_F (Delete states that recognize the respective amino acid symbols), W_M and W_F (Wait states that pause after recognizing each amino acid; these are requirements imposed by our Moore normal form). The states have been grouped (enclosed by a rectangle) to show four clusters: states that are visited before any of the sequence has been recognized, states that are visited after "M" has been recognized, and states that are visited after "F" has been recognized. The I/O function associated with each Delete state δ_Z is $\delta(x = Z)$.

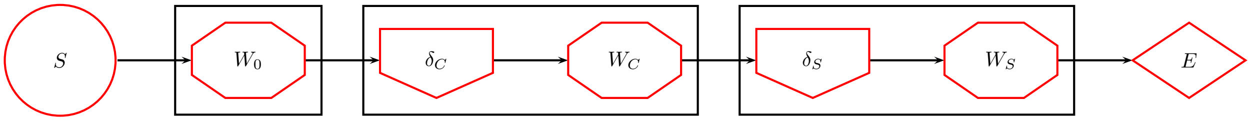


Figure 2.12: Transducer $\nabla(CS)$, the Moore-normal form recognizer for protein sequence CS. The states are labeled S (Start), E (End), δ_C and δ_S (Delete states that recognize the respective amino acid symbols), W_C and W_S (Wait states that pause after recognizing each amino acid; these are requirements imposed by our Moore normal form). The states have been grouped (enclosed by a rectangle) to show four clusters: states that are visited before any of the sequence has been recognized, states that are visited after "C" has been recognized, and states that are visited after "S" has been recognized. The I/O function associated with each Delete state δ_Z is $\delta(x = Z)$.

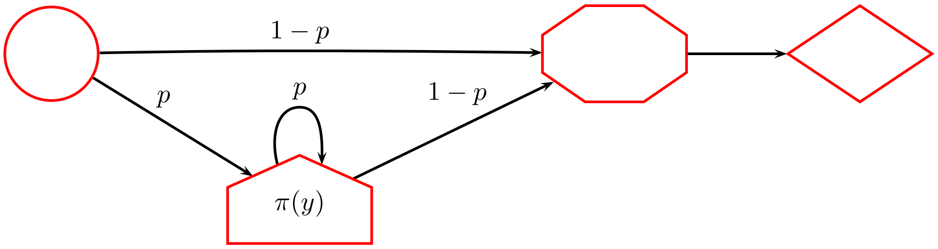


Figure 2.13: Transducer $\mathcal{N}$, a simple null-model generator with geometric length parameter p and residue frequency distribution π.

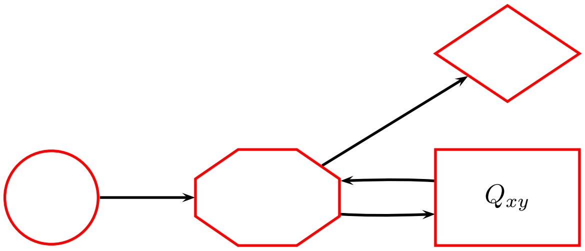


Figure 2.14: Transducer $\mathcal{S}(Q)$ ("the substituter") introduces substitutions (according to substitution matrix Q) but no indels. Whenever the machine makes a transition into the rectangular Match state, a character x is read from the input and a character y emitted to the output, with the output character sampled from the conditional distribution $P(y|x) = Q_{xy}$.

simple case (the Felsenstein algorithm for point substitution) is represented using the more elaborate transducer notation.

Figure 2.15 shows the identity transducer, $\mathcal{I}$, a special case of the substituter: $\mathcal{I} = \mathcal{S}(\delta)$, where

$$\delta_{xy} = \begin{cases} 1 & x = y \\ 0 & x \neq y \end{cases}$$

The identity essentially copies its input to its output directly. It is defined formally in Section 2.3.

The TKF91 model as a transducer

We use the TKF91 model of indels [54] as an example, not because it is the best model of indels (it has deficiencies, most notably the linear gap penalty); rather, because it is canonical, widely-known, and illustrative of the general properties of transducers.

The TKF91 transducer with I/O functions is shown in Figure 2.16. The underlying continuous-time indel model has insertion rate λ and deletion rate μ. The transition weights of the transducer, modeling the stochastic transformation of a sequence over a finite time interval t, are $a = \exp(-\mu t)$ is the "match probability" (equivalently, $1 - a$ is the deletion

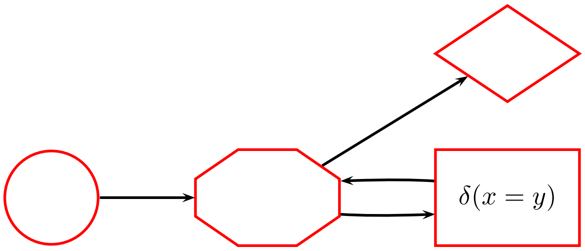


Figure 2.15: Transducer $\mathcal{I}$, the identity transducer, simply copies the input tape to the output tape. This can be thought of as a special case of the substituter transducer in Figure 2.14 where the substitution matrix is the identity matrix. **Formal definitions:** Transducer $\mathcal{I}$ is defined in Section 2.3.

probability), $b = \lambda \frac{1 - a \exp(\lambda t)}{\mu - a\lambda \exp(\lambda t)}$ is the probability of an insertion at the start of the sequence or following a match, and $c = \frac{\mu b}{\lambda(1-a)}$ is the probability of an insertion following a deletion. These may be derived from analysis of the underlying birth-death process [54].

The three I/O functions for Match, Delete and Insert states are defined as follows: conditional on absorbing character x, the Match state emits y with probability $exp(Rt)_{xy}$, where R is the rate matrix governing character substitutions and t is the time separating the input and output sequences. Characters are all deleted with the same weight: once a Delete state is entered the absorbed character is deleted with weight 1 regardless of its value. Inserted characters y are emitted according to the equilibrium distribution π_y.

In the language of [54], every state path contains a Start→Wait segment that corresponds to the "immortal link", and every Wait→Wait tour corresponds to an "mortal link".

Figure 2.17 is a version of Figure 2.16 where, rather than writing the I/O weight functions directly on each state (as in Figure 2.16), we have instead written a state label (as in Figure 2.10, Figure 2.11 and Figure 2.12). The state labels are S, M, I, D, E, W (interpretation: Start, Match, Insert, Delete, End, Wait).

It has been shown that affine-gap versions of the TKF91 model can also be represented using state machines [55]. An approximate affine-gap transducer, based on the work of Knudsen [56], Rivas [57] *et al*, is shown in Figure 2.19; a version that approximates a two-component mixture-of-geometric indel length distributions is shown in Figure 2.20, while a transducer that only inserts/deletes in multiples of 3 is in Figure 2.21.

A generator for the equilibrium distribution of the TKF91 model

TKF91 is an input/output transducer that operates on individual branches. In order to arrange this model on a phylogenetic tree, we need a generating transducer at the root. Shown in Figure 2.18 is the equilibrium TKF91 generator, conceptually the same as Figure 2.13, with insertion weight λ/μ (which is the parameter of the geometric length distribution of sequences at equilibrium under the TKF91 model).

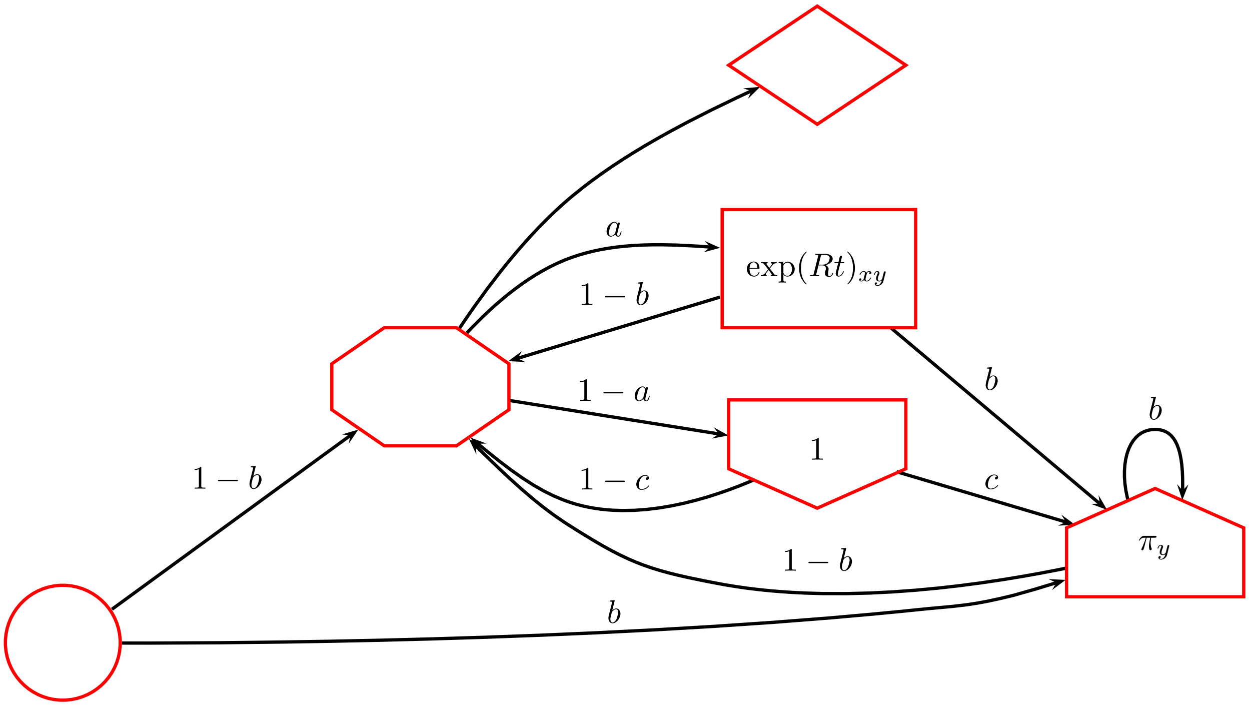


Figure 2.16: The TKF91 transducer, labeled with transition weights and I/O functions. This transducer models the sequence transformations of the TKF91 model [54]. The transition weights a, b, c are defined in Section 2.2. Figure 2.17 shows the same transducer with state type labels rather than I/O functions.

We have now defined all the component transducers we need to model evolution along a phylogeny. Conceptually, we can imagine the equilibrium machine generating a sequence at the root of the tree which is then repeatedly passed down along the branches, each of which mutates the sequence via a TKF91 machine (potentially introducing substitutions and indels). The result is a set of sequences (one for each node), whose evolutionary history is recorded via the actions of each TKF91 branch machine. What remains is to detail the various ways of connecting transducers by constraining some of their tapes to be the same—namely *composition* and *intersection*.

Composition of transducers

The first way of connecting transducers that we will consider is "composition": feeding the output of one transducer, T, into the input of another, U. The two transducers are now connected, in a sense, since the output of T must be synchronized with inputs from U: when T emits a character, U must be ready to absorb a character.

We can equivalently consider the two connected transducers as a single, *composite* machine that represents the connected ensemble of T followed by U. From two transducers T

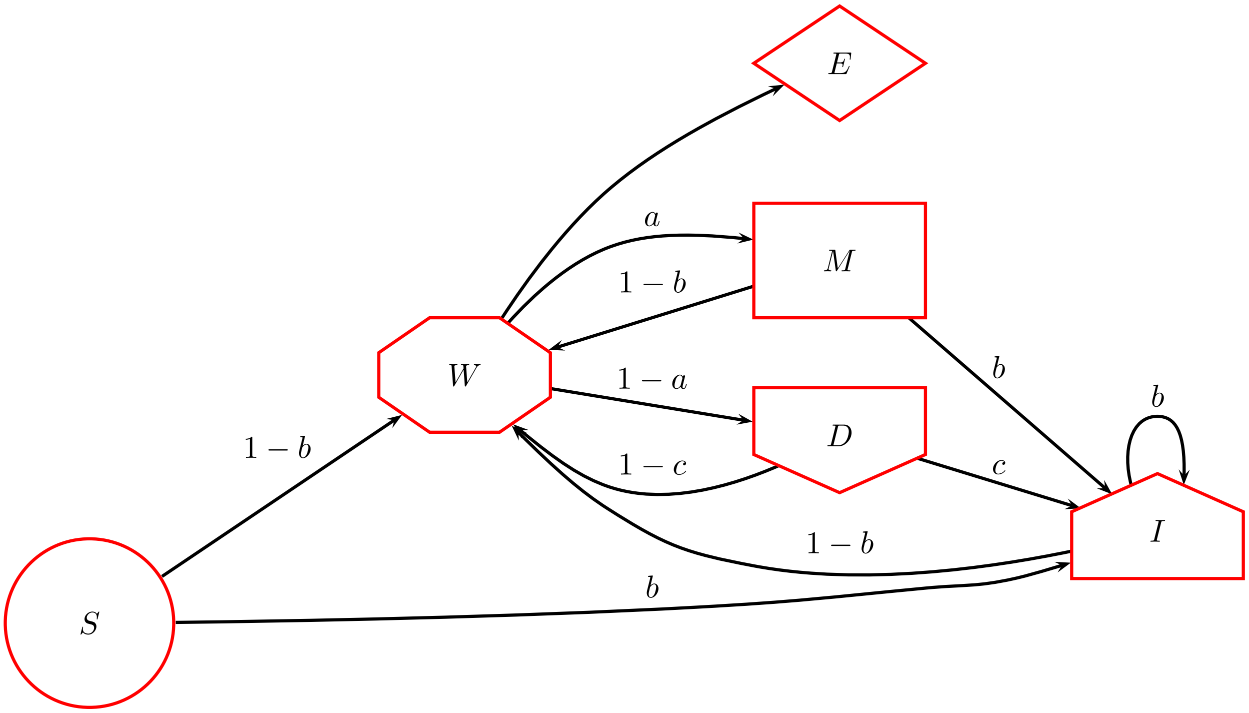


Figure 2.17: Transducer $\mathcal{B}$, the TKF91 model on a branch of length t. This transducer models the sequence transformations of the TKF91 model [54]. The machine shown here is identical to that of Figure 2.16 in nearly all respects. The only difference is this: in order that we can later refer to the states by name, rather than writing the I/O weight functions directly on each state we have instead written a state type label S, M, I, D, E, W (Start, Match, Insert, Delete, End, Wait). It so happens that the TKF91 transducer has one of each of these kinds of state. Identically to Figure 2.16, the transition weights a, b, c are defined in Section 2.2. For each of the I/O states (I, D and M) we must, of course, still specify an I/O weight function. These are also identical to Figure 2.16, and are reproduced here for reference: $\exp(Rt)$ is the substitution matrix for the M (Match) state, π is the vector of weights corresponding to the probability distribution of inserted characters for the I (Insert) state, and $(1, 1, \ldots, 1)$ is the vector of weights corresponding to the conditional probabilities that any given character will be deleted by the D (Delete) state (in the TKF91 model, deletion rate is independent of the actual character being deleted, which is why these delete weights are all 1).

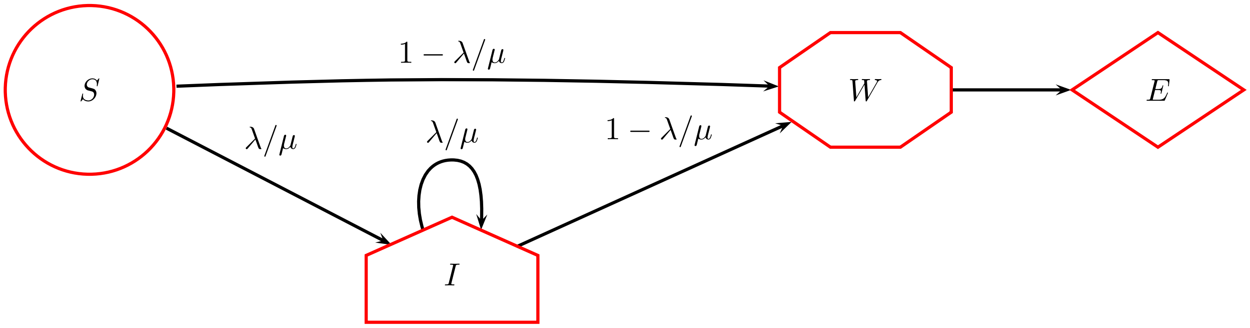


Figure 2.18: Transducer $\mathcal{R}$, the equilibrium TKF91 generator. The equilibrium distribution for the TKF91 model is essentially the same generator as Figure 2.13 with $p = \lambda/\mu$. The I/O function for the I (Insert) state is π_y. Note that this is the limit of Figure 2.16 on a long branch with empty ancestor: $\mathcal{R} = \lim_{t\to\infty}(\Delta(\epsilon)\cdot\mathcal{B}(t))$.

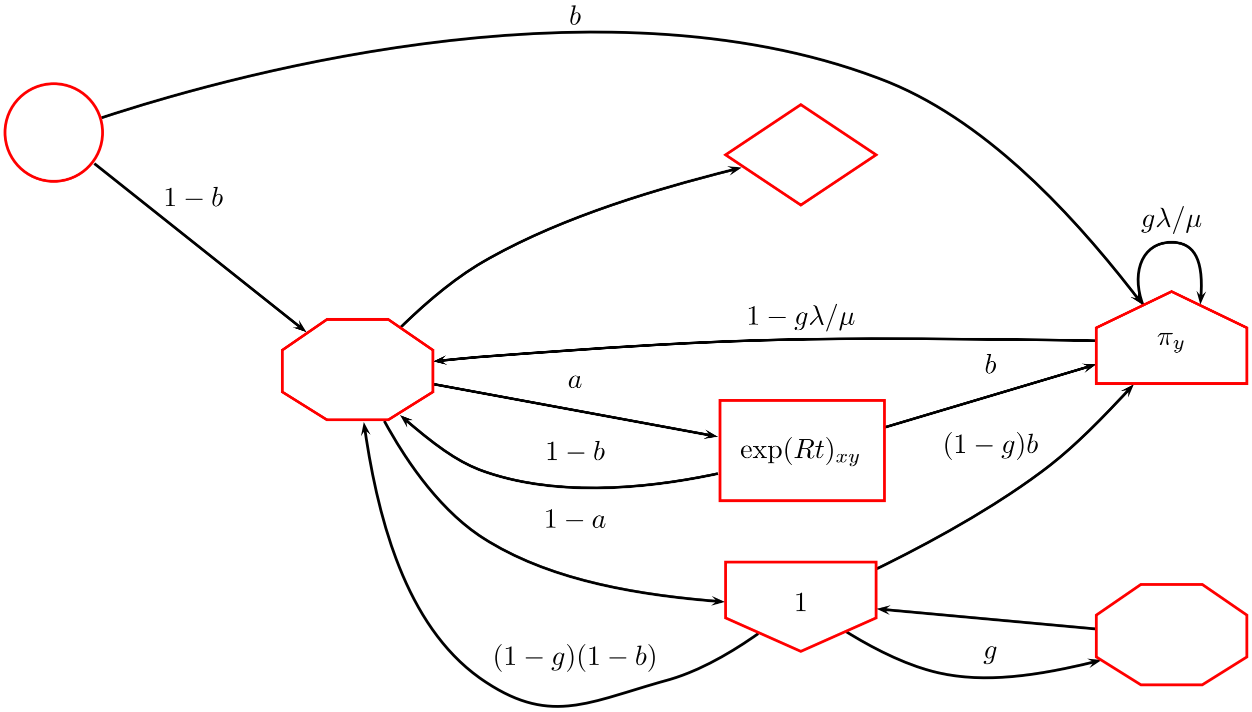


Figure 2.19: An approximate affine-gap transducer. Note that in contrast to Figure 2.16, this machine requires *two* Wait states, with the extra Wait state serving to preserve the "memory" of being inside a deletion. The parameters of this model are insertion rate λ, deletion rate μ, gap extension probability g, substitution rate matrix R, inserted residue frequencies π (typically the equilibrium distribution, so $\pi R = 0$), and branch length t. The transition weights use the quantities $a = \exp(-\mu t)$ and $b = 1 - \exp(-\lambda t)$. The transducer in Figure 2.18 serves as a suitable root generator.

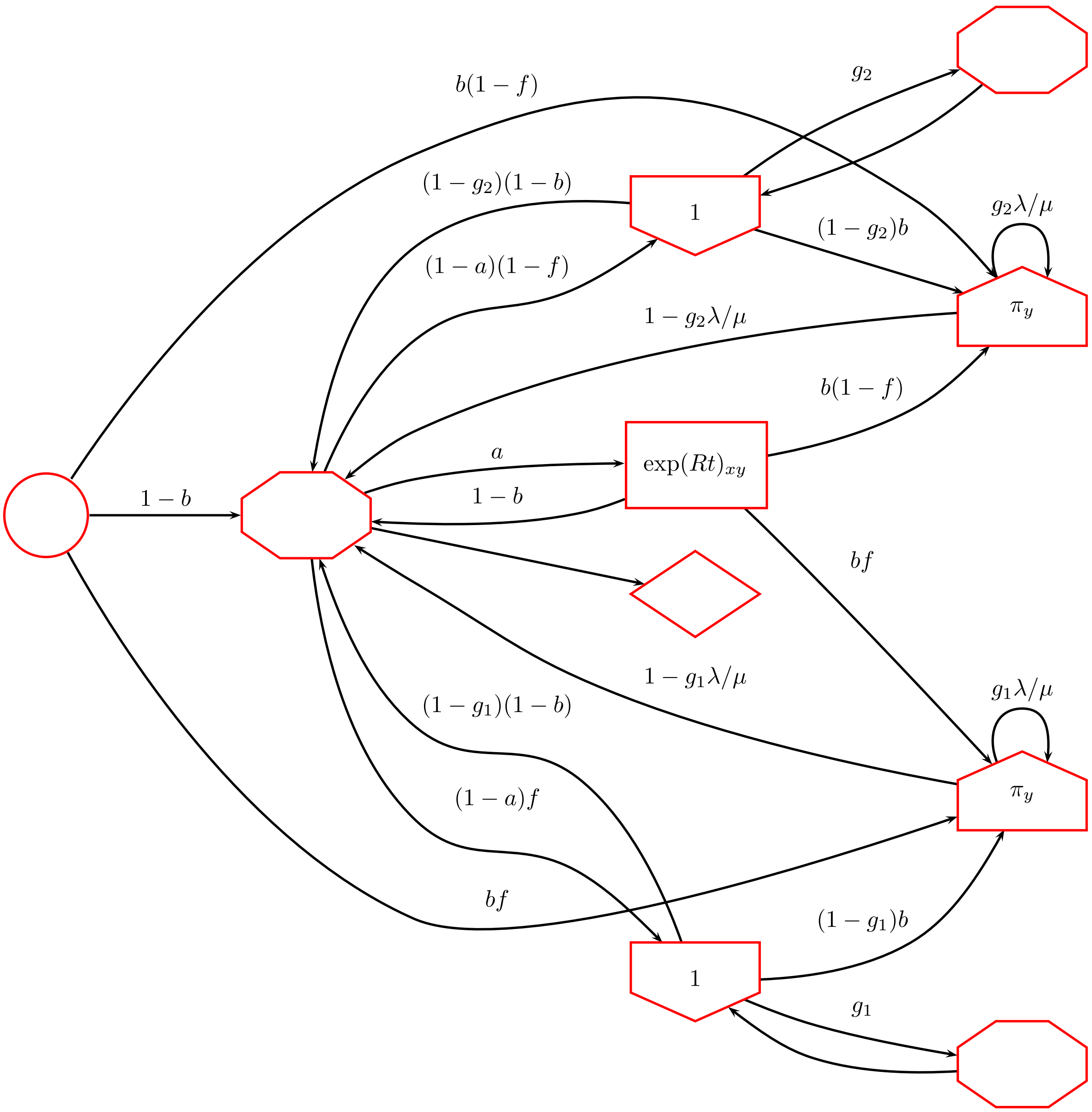


Figure 2.20: A transducer that models indel length distributions as a 2-component mixture of geometric distributions. Note that this machine requires two separate copies of the Insert and Delete states, along with *three* Wait states, with two of the Wait states serving to preserve the "memory" of being inside a deletion from the respective Delete states. The parameters of this model are insertion rate λ, deletion rate μ, gap extension probabilities g_1 and g_2 for the two mixture components, first component mixture strength f, substitution rate matrix R, inserted residue frequencies π (typically the equilibrium distribution, so $\pi R = 0$), and branch length t. The transition weights use the quantities $a = \exp(-\mu t)$ and $b = 1 - \exp(-\lambda t)$. The transducer in Figure 2.18 serves as a suitable root generator.

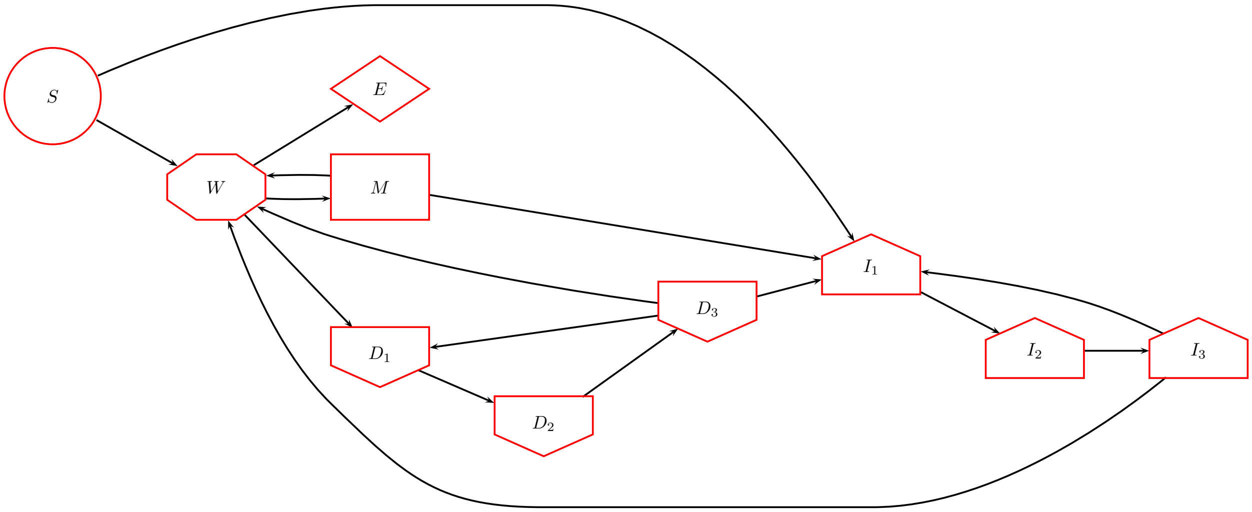


Figure 2.21: A transducer whose indels are a multiple of 3 in length. (This is not in strict normal form, but could be made so by adding a Wait state after every Delete state.)

and U, we make a new transducer, written TU (or $T \cdot U$), wherein every state corresponds to a pair (t, u) of T- and U-states.

In computing the weight of an input/output sequence pair (X, Y), where the input sequence X is fed to T and output Y is read from U, we must sum over all state paths through the composite machine TU that are consistent with (X, Y). In doing so, we are effectively summing over the intermediate sequence (the output of T, which is the input of U), just as we sum over the intermediate index when doing matrix multiplication. In fact, matrix multiplication of I/O functions is an explicit part of the algorithm that we use to automatically construct composite transducers in Section 2.3. The notation (TU or $T \cdot U$) reflects the fact that transducer composition is directly analogous to matrix multiplication.

Properties of composition include that if T is a generator then TU is also a generator, while if U is a recognizer then TU is also a recognizer. A formal definition of transducer composition, together with an algorithm for constructing the composite transducer TU, is presented in Section 2.3. This algorithmic construction of TU (which serves both as a proof of the existence of TU, and an upper bound on its complexity) is essential as a formal means of verifying our later results.

Multiplying two substitution models

As a simple example of transducer composition, we turn to the simple substituting transducers in Figure 2.14 and Figure 2.22. Composing these two in series models two consecutive branches $x \xrightarrow{Q} y \xrightarrow{R} z$, with the action of each branch modeled by a different substitution matrix, Q and R.

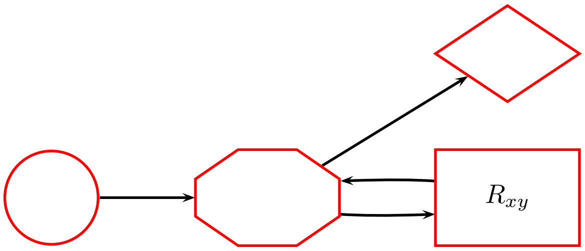


Figure 2.22: Transducer $\mathcal{S}(R)$ introduces substitutions (rate matrix R) but no indels. Compare to Figure 2.14, which is the same model but with substitution matrix Q instead of R.

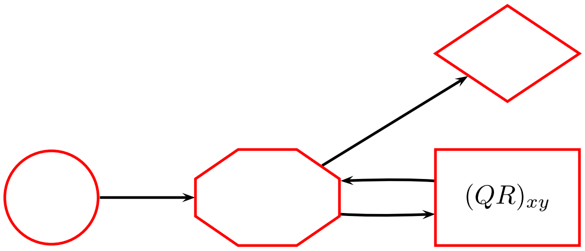


Figure 2.23: Transducer $\mathcal{S}(Q) \cdot \mathcal{S}(R)$, the composition of Figure 2.14 and Figure 2.22. Note that this is just $\mathcal{S}(QR)$: in the simple case of substituters, transducer composition is *exactly* the same as multiplication of substitution matrices. In the more general case, transducer composition is always analogous to matrix multiplication, though (unlike matrices) transducers tend in general to require more representational storage space when multiplied together (not the case here, but see e.g. Figure 2.24).

Constructing the $T \cdot U$ composite machine (shown in Figure 2.23) simply involves constructing a machine whose Match state I/O function is the matrix product of the component substitution matrices, QR. If x denotes the input symbol to the two-transducer ensemble (input into the Q-substituter), y denotes the output symbol from the two-transducer ensemble (output from the R-substituter), and z denotes the unobserved intermediate symbol (output from Q and input to R), then the I/O weight function for the composite state QR in the two-transducer ensemble is

$$(QR)_{xy} = \sum_{z} Q_{xz} R_{zy}$$

Multiplying two TKF91 models

For a more complex example, consider the composition of a TKF91 transducer with itself. This is again like two consecutive branches $x \to y \to z$, but now TKF91 is acting along each branch, rather than a simple substitution process. An input sequence is fed into first TKF91; the output of this first TKF91 transducer is an intermediate sequence that is fed

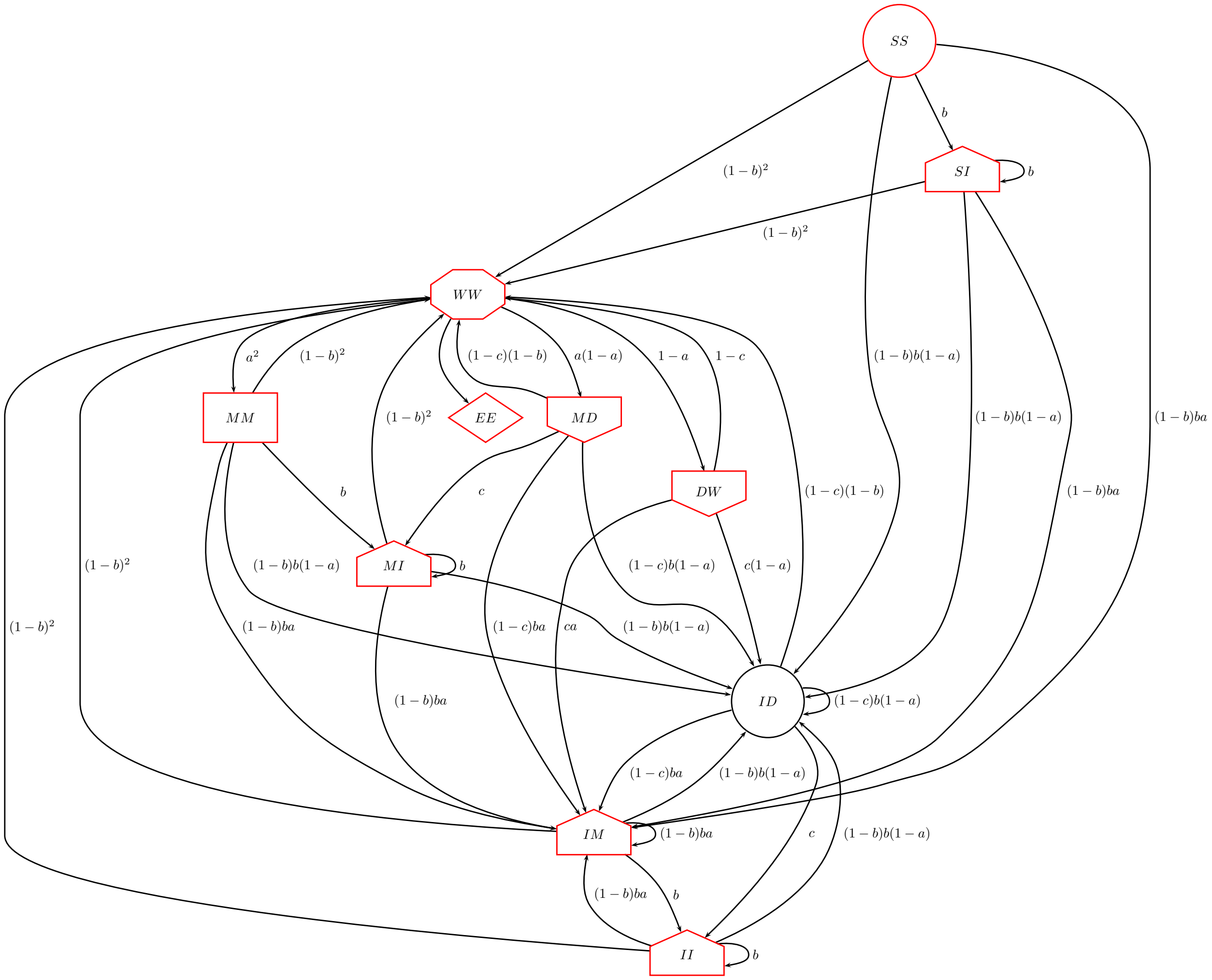


Figure 2.24: Transducer $\mathcal{B} \cdot \mathcal{B}$, the composition of the TKF91 model (Figure 2.17) with itself, representing the evolutionary drift over two consecutive branches of the phylogenetic tree. Each composite state is labeled with the states of the two component transducers. I/O weight functions, state labels, and meanings of transition parameters are explained in Table 2.1 and Section 2.2. **Formal definitions:** The algorithmic construction of the composite transducer is described in Section 2.3.

into the input of the second TKF91; output of this second TKF91 is the output of the entire ensemble.

The composite machine inputs a sequence and outputs a sequence, just like the machine in Figure 2.23, but these sequences may differ by insertions and deletions as well as substitutions. The intermediate sequence (emitted by the first TKF91 and absorbed by the second) is unobserved, and whenever we sum over state paths through the composite machine, we are essentially summing over values for this intermediate sequence.

Figure 2.24 shows the state space of this transducer. The meaning of the various states in this model are shown in Table 2.1.

The last two states in Table 2.1, II and MI, appear to contradict the co-ordination of the machines. Consider, specifically, the state MI: the first transducer is in a state that produces output (M) while the second transducer is in a state which does not receive input (I). The solution to this paradox is that the first transducer has not emitted a symbol, despite being in an M state, because symbols are only emitted on transitions *into* M states; and inspection of the composite machine (Figure 2.24) reveals that transitions into state MI only occur from states MD or MM. Only the second transducer changes state during these transitions; the M state of the first transducer is a holdover from a previous emission. The interpretation of such transitions is well-specified by our formal construction (Section 2.3).

In a specific sense (summing over paths) this composite transducer is equivalent to the single transducer in Figure 2.16, but with the time parameter doubled ($t \to 2t$). We can write this statement as $\mathcal{B}(t) \cdot \mathcal{B}(t) \equiv \mathcal{B}(2t)$. This statement is, in fact, equivalent to a form of the Chapman-Kolmogorov equation, $B(t)B(t') = B(t+t')$, for transition probability matrices $B(t)$ of a stationary continuous-time Markov chain. In fact TKF91 is currently the only nontrivial transducer known to have this property (by "nontrivial" we mean including all types of state, and so excluding substitution-only models such as Figure 2.14, which are essentially special limits of TKF91 where the indel rates are zero). An open question is whether there are any transducers for affine-gap versions of TKF91 which have this property (excluding TKF92 from this since it does not technically operate on strings, but rather sequences of strings (fragments) with immovable boundaries). The Chapman-Kolmogorov equation for transducers is stated in Section 2.3.

Constraining the input to the substitution model

Figure 2.25 is the composition of the MF-generator (Figure 2.9) with the Q-substituter (Figure 2.14). This is quite similar to a probabilistic weight matrix trained on the single sequence MF, as each of the Insert states has its own I/O probability weights.

Constraining the input to the TKF91 model

Figure 2.26 is the composition of the MF-generator (Figure 2.9) with TKF91 (Figure 2.17). In contrast to Figure 2.25, this machine generates samples from the distribution of descendants of a known ancestor (MF) via indels and substitutions. It is conceptually similar to a profile

$b_1 b_2$	Meaning	I/O fn.
SS	Both transducers are in the Start state. No characters are absorbed or emitted.	
EE	Both transducers are in the End state. No characters are absorbed or emitted.	
WW	Both transducers are in the Wait state. No characters are absorbed or emitted, but both are poised to absorb sequences from their input tapes.	
SI	The second transducer inserts a character y while the first remains in the Start state.	π_y
IM	The first transducer emits a character (via the Insert state) which the second transducer absorbs, mutates and re-emits as character y (via a Match state).	$(\pi \exp(Rt))_y$
MM	The first transducer absorbs character x (from the input tape), mutates and emits it (via the Match state); the second transducer absorbs the output of the first, mutates it again, and emits it as y (via a Match state).	$\exp(2Rt)_{xy}$
ID	The first transducer emits a character (via the Insert state) which the second transducer absorbs and deletes (via its Delete state).	
MD	The first transducer absorbs a character x, mutates and emits it (via the Match state) to the second transducer, which absorbs and deletes it (via its Delete state).	1
DW	The first transducer absorbs and deletes a character x, whereas the second transducer idles in a Wait state (since it has recieved no input from the first tranducer).	1
II	The second transducer inserts a character y while the first remains in the Insert state from a previous insertion. Only the second transducer is changing state (and thus emitting a character) here—the first is resting in an Insert state from a previous transition.	π_y
MI	The second transducer inserts a character y while the first remains in the Match state from a previous match. Only the second transducer is changing state (and thus emitting a character) here—the first is resting in a Match state from a previous transition.	π_y

Table 2.1: Meanings (and, where applicable, I/O functions) of all states in transducer $\mathcal{B} \cdot \mathcal{B}$ (Figure 2.24), the composition of two TKF91 transducers (an individual TKF91 transducer is shown in Figure 2.17). Every state corresponds to a tuple (b_1, b_2) where b_1 is the state of the first TKF91 transducer and b_2 is the state of the second TKF91 transducer.

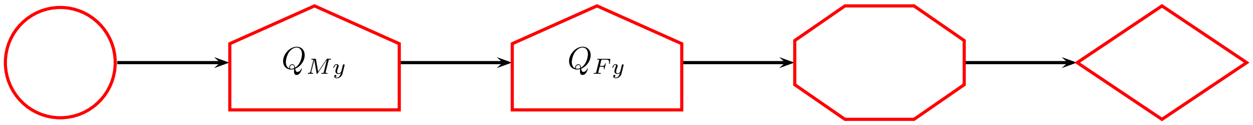


Figure 2.25: Transducer $\Delta(MF) \cdot \mathcal{S}(Q)$ corresponds to the generation of sequence MF (by the transducer in Figure 2.9) and its subsequent mutation via substitutions (by the transducer in Figure 2.14). Since no gaps are involved, and the initial sequence length is specified (by Figure 2.9), there is only one state path through this transducer. State types are indicated by the shape (as per Figure 2.7), and the I/O functions of the two Insert states are indicated. For more information see Section 2.2.

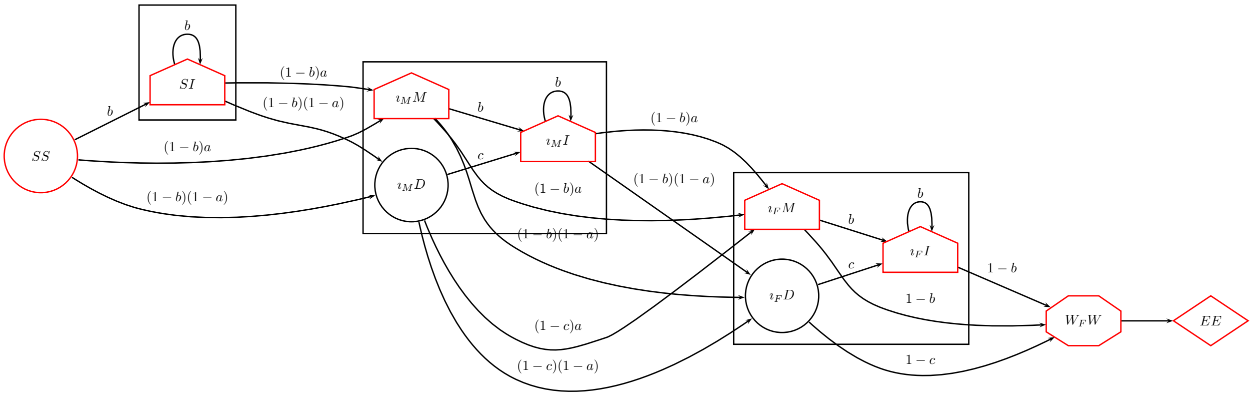


Figure 2.26: Transducer $\Delta(MF) \cdot \mathcal{B}$, the composition of the MF-generator (Figure 2.9) with the TKF91 transducer (Figure 2.17). This can be viewed as an HMM that generates sequences sampled from the distribution of TKF91-mutated descendants of ancestor MF. See Section 2.2 and Table 2.2 for the meaning of each state.

HMM trained on the single sequence MF (though without the Dirichlet prior distributions that are typically used when training a profile HMM).

The meaning of the various states in this machine are shown in Table 2.2.

As noted in Section 2.2, a generator composed with any transducer is still a generator. That is, the *composite* machine contains no states of type Match or Delete: it accepts null input (since its immediately-upstream transducer, the MF-generator, accepts null input). Even when the TKF91 is in a state that accepts input symbols (Match or Delete), the input symbol was inserted by the MF-generator; the MF-generator does not itself accept any input, so the entire ensemble accepts no input. Therefore the *entire composite machine is a generator*, since it accepts null input (and produces non-null output).

ib	Meaning	I/O fn.
SS	Both transducers are in the Start state. No characters are absorbed or emitted.	
EE	Both transducers are in the End state. No characters are absorbed or emitted.	
$W_F W$	Both transducers are in their respective Wait states. No characters are absorbed or emitted, and both are poised to enter the End state.	
SI	The TKF91 transducer inserts a character y while the generator remains in the start state.	π_y
$\imath_M M$	The generator emits the M character, which TKF91 absorbs via its Match state and emits character y.	$\exp(Rt)_{My}$
$\imath_M D$	The generator emits the M character, which TKF91 absorbs via its Delete state.	
$\imath_M I$	TKF91 inserts a character y while the generator remains in the Insert state from which it emitted its last character (M).	π_y
$\imath_F M$	The generator emits the F character, which TKF91 absorbs via its Match state and emits character y.	$\exp(Rt)_{Fy}$
$\imath_F D$	The generator emits the F character, which TKF91 absorbs via its Delete state.	
$\imath_F I$	TKF91 inserts a character y while the generator remains in the Insert state from which it emitted its last character (F).	π_y

Table 2.2: Meanings (and, where applicable, I/O functions) of all states in transducer $\Delta(MF) \cdot \mathcal{B}$ (Figure 2.26), the composition of the MF-generator (Figure 2.9) with the TKF91 transducer (Figure 2.17). Since this is an ensemble of two transducers, every state corresponds to a pair (i, b) where i is the state of the generator transducer and b is the state of the TKF91 transducer.

Constraining the output of the substitution model

Figure 2.27 and Figure 2.28 show the composition of the substituter with (respectively) the MF-recognizer and the CS-recognizer. These machines are similar to the one in Figure 2.25, but instead of the substituter taking its input from a generator, the order is reversed: we are now feeding the substituter's output to a recognizer. The composite machines accept sequence on input, and emit null output.

Constraining the output of the TKF91 model

Figure 2.29 shows the composition of a TKF91 transducer with the MF-recognizer. It is worth comparing the recognizer in Figure 2.29 with the analogous generator in Figure 2.26. While similar, the generator and the recognizer are not the same; for a sequence S, Fig-

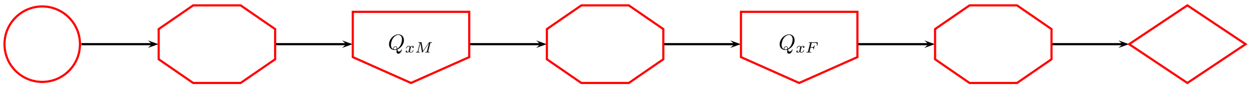


Figure 2.27: Transducer $\mathcal{S}(Q) \cdot \nabla(MF)$ ‘recognizes’ sequences ancestral to MF: it computes the probability that a given input sequence will mutate into MF, assuming a point substitution model. In contrast to the machine in Figure 2.25, this machine accepts an ancestral sequence as input and emits null output. Note that a sequence of any length besides 2 will have recognition weight zero.

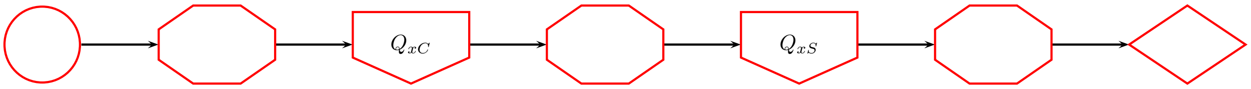


Figure 2.28: Transducer $\mathcal{S}(Q) \cdot \nabla(CS)$ ‘recognizes’ sequences ancestral to CS: it computes the probability that a given input sequence will mutate into CS, assuming a point substitution model. In contrast to the machine in Figure 2.25, this machine accepts an ancestral sequence as input and emits null output. Note that a sequence of any length besides 2 will have recognition weight zero.

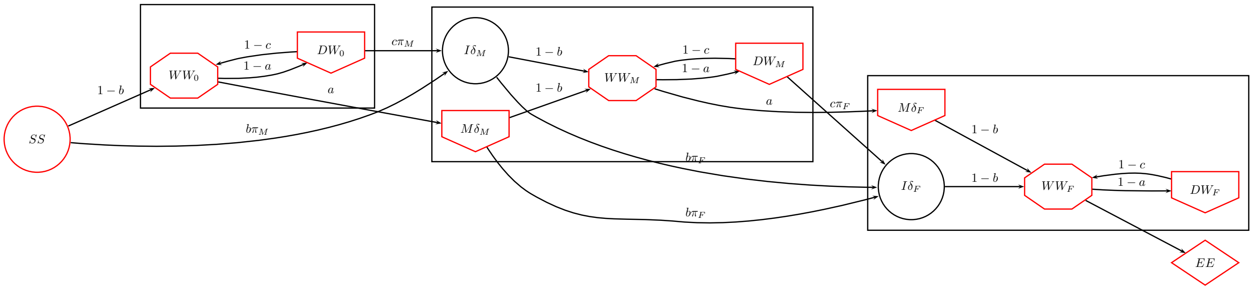


Figure 2.29: Transducer $\mathcal{B} \cdot \nabla(MF)$ computes the probability that a given ancestral input sequence will evolve (via the TKF91 model) into the specific descendant MF. It can therefore be thought of as a *recognizer* for the ancestor of MF. This machine absorbs sequence on its input and has null output due to the recognizer’s null output. See Section 2.2 and Table 2.3 for more information.

bd	Meaning	I/O fn.
SS	Both transducers are in the Start state. No characters are absorbed or emitted.	
EE	Both transducers are in the End state. No characters are absorbed or emitted.	
WW_0	Both transducers are in Wait states. No characters are absorbed or emitted; both are poised to accept input.	
DW_0	TKF91 absorbs a character x and deletes it; recognizer remains in the initial Wait state (W_0).	1
$I\delta_M$	TKF91 emits a character (M) via an Insert state and it is recognized by the δ_M state of the recognizer.	
$M\delta_M$	TKF91 absorbs a character x via a Match state, then emits an M, which is recognized by the δ_M state of the recognizer.	$\exp(Rt)_{xM}$
WW_M	Both transducers are in Wait states: TKF91 in its only Wait state and the recognizer in the Wait state following the M character.	
DW_M	TKF91 absorbs a character x and deletes it; recognizer remains in the Wait state (W_M).	1
$I\delta_F$	TKF91 emits a character (F) via an Insert state and it is recognized by the δ_F state of the recognizer.	
$M\delta_F$	TKF91 absorbs a character x via a Match state, then emits an F, which is recognized by the δ_F state of the recognizer.	$\exp(Rt)_{xF}$
WW_F	Both transducers are in Wait states: TKF91 in its only Wait state and the recognizer in the Wait state following the F character.	
DW_F	TKF91 absorbs a character x and deletes it; recognizer remains in the Wait state (W_F).	1

Table 2.3: Meanings (and, where applicable, I/O functions) of all states in transducer $\mathcal{B} \cdot \nabla(MF)$ (Figure 2.29), the composition of the TKF91 model (Figure 2.17) with the MF-recognizer (Figure 2.11). Since this is an ensemble of two transducers, every state corresponds to a pair (b, d) where b is the state of the TKF91 transducer and d is the state of the recognizer transducer.

ure 2.29 with S as input computes $P(MF|S)$ (probability that descendant of S is MF), whereas Figure 2.26 with S as output computes $P(S|MF)$ (probability that descendant of MF is S).

The meaning of the various states in this model are shown in Table 2.3.

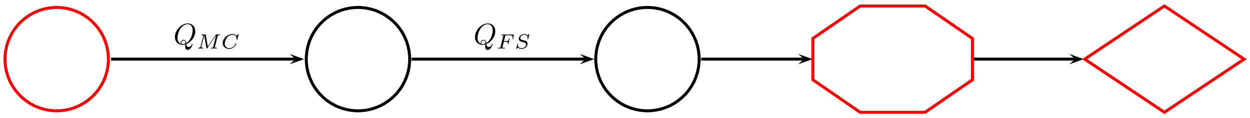


Figure 2.30: Transducer $\Delta(MF) \cdot \mathcal{S}(Q) \cdot \nabla(CS)$ is a pure Markov chain (with no I/O) whose single Start→End state path describes the transformation of MF to CS via substitutions only.

Specifying input and output to the substitution model

Figure 2.30 shows the composition of the MF-generator, substituter, and CS-recognizer. This machine is a pure Markov chain (with no inputs or outputs) which can be used to compute the probability of transforming MF to CS. Specifically, this probability is equal to the weight of the single path through Figure 2.30.

Note that composing $\Delta(MF) \cdot \mathcal{S}(Q) \cdot \nabla(LIV)$ would result in a state graph with no valid paths from Start to End, since $\mathcal{S}(Q)$ cannot change sequence length. This is correct: the total path weight of zero, corresponding to the probability of transforming MF to LIV by point substitutions alone.

Specifying input and output to the TKF91 model

Figure 2.31 shows the composition of three transducers, the MF-generator, TKF91 and LIV-recognizer (transitions are omitted for clarity). This is one step beyond the machine in Figure 2.30, as it allows insertions and deletions to separate the generated (MF) and recognized (CS) sequences.

The meaning of the various states in this model are shown in Table 2.4.

Figure 2.31, like Figure 2.30, contains only null states (no match, insert, or delete states), making it a pure Markov chain with no I/O. Such Markov models can be viewed as a special case of an input/output machine where the input and output are both null. (As noted previously, a *hidden* Markov model corresponds to the special case of a transducer with null input, i.e. a generator.)

Probability $P(Y = \text{LIV}|X = \text{MF})$ for the TKF91 model is equal to sum of all path weights from start to end in the Markov model of Figure 2.31. The set of paths corresponds to the set of valid evolutionary transformations relating sequences MF and LIV (valid meaning those allowed under the model, namely non-overlapping single-residue indels and substitutions).

This is directly analogous to computing a pairwise alignment (e.g. using the Needlman-Wunsch algorithm), and the structure of the Markov model shown in Figure 2.31 suggests the familiar structure of a pairwise dynamic programming matrix.

Removal of null states

In Figure 2.24, the state ID is of type Null: it corresponds to an insertion by the first TKF91 transducer that is then immediately deleted by the second TKF91 transducer, so there is no

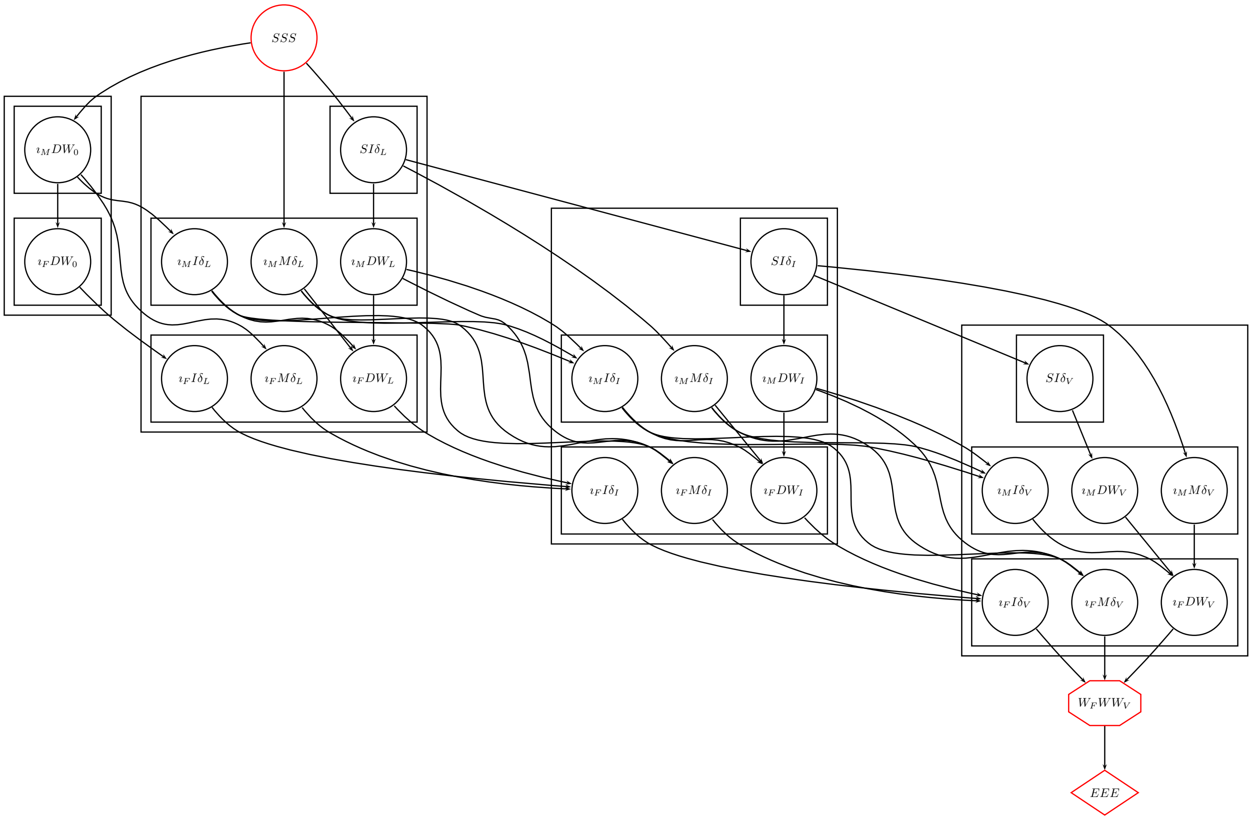


Figure 2.31: Transducer $\Delta(MF) \cdot \mathcal{B} \cdot \nabla(LIV)$ is a Markov chain (that is to say, a state machine, but one with no input or output characters) wherein every Start→End state path describes an indel history via which sequence MF mutates to sequence LIV. Note that the structure of this Markov chain is very similar to a dynamic programming matrix for pairwise sequence alignment. The "rows" and "cells" of this matrix are shown as boxes. Computation of the total Start→End path weight, using a recursion that visits all the states in topological sort order, is similar to the Forward algorithm of HMMs. See Section 2.2 and Table 2.4 for more information on the states of this model.

ibd	Meaning
SSS	All transducers are in the Start state. No characters are emitted or absorbed.
$W_F W W_V$	All transducers are in their Wait states which precede the End states for each transducer. No characters are emitted or absorbed.
EEE	All transducers are in their End states. No characters are emitted or absorbed.
$SI\delta_y$	The component TKF91 machine emits a character y via the Insert state (I) which is read by the δ_y state of the recognizer; the generator remains in the Start state (S), not yet having emitted any characters.
$\imath_x I\delta_y$	The component TKF91 machine emits a character y via the Insert state (I) which is read by the δ_y state of the recognizer; the generator remains in an Insert state ($\imath_x$) corresponding to the last character x which it generated.
$\imath_x M\delta_y$	The generator emits character x via an Insert state ($\imath_x$); TKF91 absorbs the x and emits a y via a Match state (M); the recognizer absorbs the y character via state δ_y.
$\imath_x DW_y$	The generator emits character x via an Insert state ($\imath_x$), and TKF91 absorbs and deletes it via a Delete state (D). The recognizer remains in W_y, the Wait state following character y (or W_0 if the recognizer has not yet read any characters).

Table 2.4: Meanings of states in transducer $\Delta(MF) \cdot \mathcal{B} \cdot \nabla(LIV)$ (Figure 2.31), the composition of the MF-generator (Figure 2.9), TKF91 (Figure 2.17) and the LIV-recognizer (Figure 2.10). Since this is an ensemble of three transducers, every state corresponds to a triple (i, b, d) where i is the state of the generator transducer, b is the state of the TKF91 transducer and d is the state of the recognizer transducer. Where states are labeled with x or y suffices (e.g. $SI\delta_y$), then the definition is valid $\forall\ x \in \{M, F\}$ and $\forall y \in \{L, I, V\}$.

net insertion or deletion.

It is often the case that Null states are irrelevant to the final analysis. (For example, we may not care about all the potential, but unlikely, insertions that were immediately deleted and never observed.) It turns out that is often useful to eliminate these states; i.e. transform the transducer into an equivalent transducer that does not contain null states. (Here "equivalent" means a transducer that defines the same weight for every pair of I/O sequences; this is made precise in Section 2.3.) Fortunately we can often find such an equivalent transducer using a straightforward matrix inversion [58]. Null state removal is described in more detail in Section 2.3 and Section 2.3.

Intersection of transducers

Our second operation for connecting two transducers involves feeding the same input tape into both transducers in parallel, and is called "intersection".

As with a transducer composition, an intersection is constructed by taking the Cartesian product of two transducers' state spaces. From two transducers T and U, we make a new transducer $T \circ U$ wherein every state corresponds to a pair (t, u) of T- and U-states, and whose output tape constitutes a *pairwise alignment* of the outputs of T and U.

A property of intersection is that if T and U are both recognizers, then there *is* no output, and so $T \circ U$ is a recognizer also.

Conversely, if either T or U is *not* a recognizer, then $T \circ U$ will have an output; and if neither T or U is a recognizer, then the output of $T \circ U$ will be a pairwise alignment of T's output with U's output (equivalently, we may regard $T \circ U$ as having two output tapes). Transducer Q_n in Section 2.3 is an example of such a two-output transducer. Indeed, if we do further intersections (such as $(T \circ U) \circ V$ where V is another transducer) then the resultant transducer may have several output tapes (as in the general multi-sequence HMMs defined in [46]). The models denoted F_n in Section 2.3 are of this nature.

A formal definition of transducer intersection, together with an algorithm for constructing the intersected transducer $T \circ U$, is presented in Section 2.3. Like the construction of TU, this algorithmic construction of $T \circ U$ serves both as a proof of the existence of $T \circ U$, and an upper bound on its complexity. It is also essential as a formal means of verifying our later results.

Composition, intersection and Felsenstein's pruning algorithm

If a composition is like matrix multiplication (i.e. the operation of evolution along a contiguous branch of the phylogenetic tree), then an intersection is like a *bifurcation* at a node in the phylogenetic tree, where a branch splits into two child branches.

Intersection corresponds to the pointwise multiplication step in the Felsenstein pruning algorithm — i.e. the calculation

$$P(\text{descendants}|\text{parent}) = P(\text{left child and its descendants}|\text{parent})P(\text{right child and its descendants}|\text{parent}$$

Specifically, in the Felsenstein algorithm, we define $G^{(n)}(x)$ to be the probability of all observed descendants of node n, conditional on node n having been in state x. Let us further suppose that $M^{(n)}$ is the conditional substitution matrix for the branch above node n (coming from n's parent), so

$$M^{(n)}_{ij} = P(\text{node } n \text{ is in state } j|\text{parent of node } n \text{ is in state } i)$$

Then we can write the core recursion of Felsenstein's pruning algorithm in matrix form; $G^{(n)}$ is a column vector, $M^{(n)}$ is a matrix, and the core recursion is

$$G^{(n)} = \left(M^{(l)} \cdot G^{(l)}\right) \circ \left(M^{(r)} \cdot G^{(r)}\right)$$

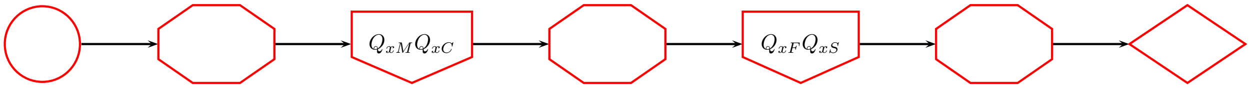


Figure 2.32: The transducer $(\mathcal{S}(Q) \cdot \nabla(CS)) \circ (\mathcal{S}(Q) \cdot \nabla(MF))$ recognizes the common ancestor of MF and CS.

where (l, r) are the left- and right-children of node n, "$\cdot$" denotes matrix multiplication, and "$\circ$" denotes the *pointwise product* (also called the *Hadamard product*), defined as follows for two vectors A and B:

$$(A \circ B)_i = A_i B_i, \qquad A \circ B = \begin{pmatrix} A_1 B_1 \\ A_2 B_2 \\ A_3 B_3 \\ \dots \\ A_K B_K \end{pmatrix}$$

Thus the two core steps of Felsenstein's algorithm (in matrix notation) are (a) matrix multiplication and (b) the pointwise product. Composition provides the transducer equivalent of matrix multiplication; intersection provides the transducer equivalent of the pointwise product.

Note also that $G^{(n)}(x)$ is the probability of node n's observed descendants *conditional on* x, the state of node n. Thus $G^{(n)}$ is similar to a recognition profile, where the computed weight for a sequence S represents the probability of some event (recognition) conditional on having S as input, i.e. a probability of the form $P(\dots|S)$ (as opposed to a probability distribution of the form $P(S|\dots)$ where the sequence S is the output, as is computed by generative profiles).

Finally consider the initialization step of the Felsenstein algorithm. Let n be a leaf node and y the observed character at that node. The initialization step is

$$G^{(n)}(x) = \delta(x = y)$$

i.e. we initialize $G^{(n)}$ with a unit column vector, when n is a leaf. This unit vector is, in some sense, equivalent to our exact-match recognizer. In fact, generators and recognizers are analogous to (respectively) row-vectors and column-vectors in our infinite-dimensional vector space (where every dimension represents a different sequence). The exact-match recognizer $\nabla(S)$ resembles a unit column-vector in the S-dimension, while the exact-match generator $\Delta(S)$ resembles a unit row-vector in the S-dimension.

Recognizer for a common ancestor under the substitution model

Figure 2.32 shows the intersection of Figure 2.27 and Figure 2.28. This is an ensemble of four transducers. Conceptually, what happens when a sequence is input is as follows. First, the input sequence is duplicated; one copy is then fed into a substituter (Figure 2.14), whose

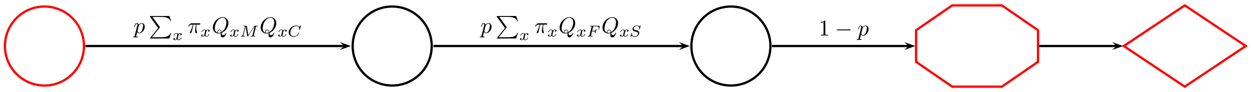


Figure 2.33: Transducer $\mathcal{N} \cdot ((\mathcal{S}(Q) \cdot \nabla(CS)) \circ (\mathcal{S}(Q) \cdot \nabla(MF)))$ allows computing the final Felsenstein probability for the two sequence MF and CS descended from an unobserved ancestor by character substitutions. Since characters emitted by the $\mathcal{N}$ transducer are all absorbed by the two recognizers, the composite machine has null input and output. The probabilities (involving the geometric parameter p, the prior π and the substitution matrix Q) are all encoded on the transitions. The use of summation operators in the weights of these transitions reflects the fact that multiple transitions have been collapsed into when by our composition algorithm, which marginalizes I/O characters that are not directly observed. Note that this machine is equivalent to the composition of a null model (Figure 2.13) with the machine in Figure 2.32.

output is fed into the exact-matcher for CS (Figure 2.12); the other copy of the input sequence is fed into a separate substituter (Figure 2.22), whose output is fed into exact-matcher for MF (Figure 2.11).

The composite transducer is a recognizer (the only I/O states are Deletes; there are no Matches or Inserts), since it absorbs input but the recognizers (exact-matchers) have null output. Note also that the I/O weight functions in the two Delete states in Figure 2.32 are equivalent to the $G^{(n)}$ probability vectors in Felsenstein's algorithm (Section 2.2).

Since this is an ensemble of four transducers, every state corresponds to a tuple (b_1, d_1, b_2, d_2) where b_1 is the state of the substituter transducer on the CS-branch (Figure 2.14), d_1 is the state of the exact-matcher for sequence CS (Figure 2.12), b_2 is the state of the substituter transducer on the MF-branch (Figure 2.22), d_1 is the state of the exact-matcher for sequence MF (Figure 2.11).

The I/O label for the general case where $(b_1, d_1, b_2, d_2) = (Q, \delta_A, R, \delta_B)$ denotes the vector whose elements are given by $Q_{xA} R_{xB}$. So, for example, the I/O label for state $(Q, \delta_C, R, \delta_M)$ is the vector whose x'th entry is the probability that an input symbol x would mutate to C on the Q-branch and M on the R-branch.

The Felsenstein likelihood for a two-branch tree using the substitution model

As noted, the previous machine (Figure 2.32) can be considered to compute the Felsenstein probability vector $G^{(n)}$ at an internal node n of a tree. When n is the root node of the tree, this must be multiplied by a prior over sequences if we are to compute the final Felsenstein probability,

$$P(\text{sequences}|\text{tree}) = \pi G^{(1)} = \sum_x \pi_x G_x^{(1)}$$

In transducer terms, this "multiplication by a prior for the root" is a composition: we connect a root generator to the input of the machine. Composing Figure 2.13 (a simple prior over root sequences: geometric length distribution and IID with frequencies π) with the transducer of Figure 2.32 (which represents $G^{(1)}$) yields a pure Markov chain whose total path weight from Start to End is the final Felsenstein probability (Figure 2.33). Furthermore, sampling traceback paths through this machine provides an easy way to sample from the posterior distribution over ancestral sequences relating MF and CS.

Recognizer for a common ancestor under the TKF91 model

The transducer shown in Figure 2.34, is the recognition profile for the TKF91-derived common ancestor of LIV and MF. LIV and MF may each differ from the ancestor by insertions, deletions, and substitutions; a particular path through this machine represents one such explanation of differences (or, equivalently, an alignment of LIV, MF, and their ancestor). This type of machine is denoted G_n in Section 2.3.

Figure 2.34 is an ensemble of four transducers. Conceptually, what happens to an input sequence is as follows. First, the input sequence is duplicated; one copy of the input is fed into TKF91 (Figure 2.17), whose output is fed into the exact-matcher for LIV (Figure 2.10); the other copy of the input is fed into a separate TKF91 machine (Figure 2.17), whose output is fed into the exact-matcher for MF (Figure 2.11).

Since this is an ensemble of four transducers, every state corresponds to a tuple (b_1, d_1, b_2, d_2) where b_1 is the state of the TKF91 transducer on the LIV-branch (Figure 2.17), d_1 is the state of the exact-matcher for sequence LIV (Figure 2.10), b_2 is the state of the TKF91 transducer on the MF-branch (Figure 2.17), and d_1 is the state of the exact-matcher for sequence MF (Figure 2.11).

Note that, as with Figure 2.31, the underlying structure of this state graph is somewhat like a DP matrix, with rows, columns and cells. In fact, (modulo some quirks of the automatic graph layout performed by graphviz's 'dot' program) Figure 2.34 and Figure 2.31 are structurally quite similar. However, compared to Figure 2.31, Figure 2.34 has more states in each "cell", because this transducer tracks events on two separate branches (whereas Figure 2.31 only tracks one branch).

Since this machine is a recognizer, it has Delete states but no Match or Insert states. The delete states present do not necessarily correspond to deletions by the TKF91 transducers: all characters are ultimately deleted by the exact-match recognizers for LIV and MF, so even if the two TKF91 transducers allow symbols to pass through undeleted, they will still be deleted by the exact-matchers.

In fact, this transducer's states distinguish between deletion events on one branch *vs* insertion events on the other: this is significant because a "deleted" residue is homologous to a residue in the ancestral sequence, while an "inserted" residue is not. There are, in fact, four delete states in each "cell" of the matrix, corresponding to four fates of the input symbol after it is duplicated:

1. Both copies of the input symbol pass successfully through respective TKF91 transducers and are then deleted by respective downstream exact-matchers for sequences LIV

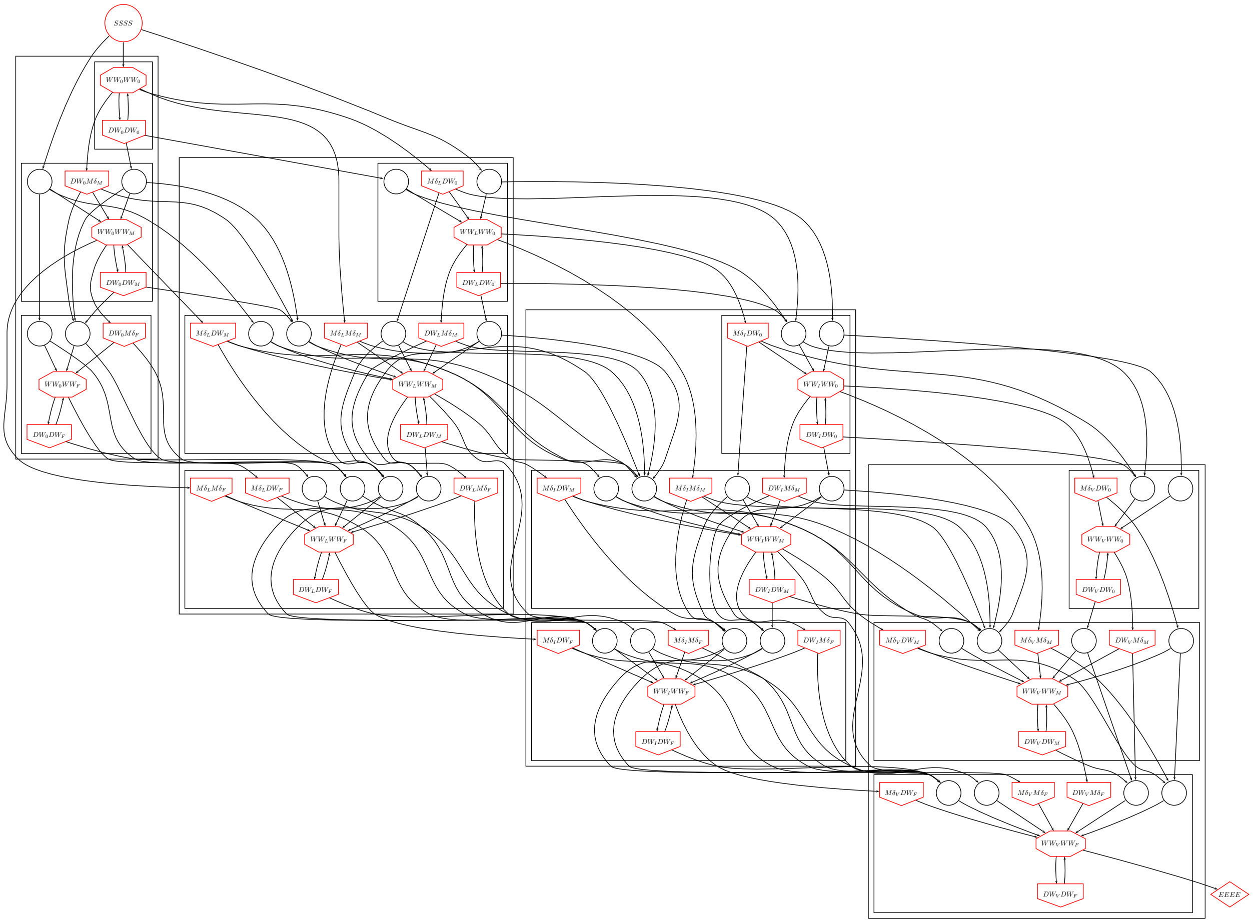


Figure 2.34: Transducer $(\mathcal{B} \cdot \nabla(LIV)) \circ (\mathcal{B} \cdot \nabla(MF))$ recognizes the ancestor of LIV and MF, assuming common descent under the TKF91 model. As with Figure 2.31, the structure of this machine is very similar to a dynamic programming matrix for pairwise sequence alignment; again, the "rows" and "cells" of this matrix have been shown as boxes. In contrast to Figure 2.31, this machine is not a Markov model (ancestral sequence already encoded in the state space), but a recognizer (ancestral sequence read from input tape). This Figure, along with several others in this manuscript, was autogenerated using phylocomposer [51] and graphviz [59]. **Formal definitions:** This type of machine is denoted G_n in Section 2.3. The algorithms for constructing composite and intersected transducer state graphs, are reviewed in Section 2.3 and Section 2.3.

and MF (e.g. MD_LMD_F);

2. One copy of the input symbol is deleted by the TKF91 transducer on the LIV-branch, leaving the downstream LIV-matcher idling in a Wait state; the other copy of the input symbol passes through the TKF91 transducer on the MF-branch and is then deleted by the downstream MF-matcher; (e.g. DW_0MD_F);

3. One copy of the input symbol passes through the TKF91 transducer on the LIV-branch and is then deleted by the downstream LIV-matcher; the other copy is deleted by TKF91 transducer on MF-branch, leaving the downstream MF-matcher idling in a Wait state; (e.g. MD_LDW_0);

4. Both copies of the input symbol are deleted by the respective TKF91 transducers, while the downstream exact-matchers idle in Wait states without seeing any input (e.g. DW_0DW_0).

The other states in each cell of the matrix are Null states (where the symbols recognized by the LIV- and MF-matchers originate from insertions by the TKF91 transducers, rather than as input symbols) and Wait states (where the ensemble waits for the next input symbol).

The Felsenstein likelihood for a two-branch tree using the TKF91 model

If we want to compute the joint marginal probability of LIV and MF as siblings under the TKF91 model, marginalizing the unobserved common ancestor, we have to use the same trick that we used to convert the recognizer in Figure 2.32 into the Markov model of Figure 2.33. That is, we must connect a generator to the input of Figure 2.34, where the generator emits sequences from the root prior (equilibrium) distribution of the TKF91 model. Using the basic generator shown in Figure 2.18, we construct the machine shown in Figure 2.35. (This type of machine is denoted M_n in Section 2.3.)

Summing over all Start→End paths in Figure 2.35, the total weight is the final Felsenstein probability, i.e. the joint likelihood $P(LIV, MF|\text{tree}, \text{TKF91})$. Sampling traceback paths through Figure 2.35 yields samples from the posterior distribution over common ancestors of LIV and MF. The sum-over-paths is computed via a form of the standard Forward dynamic programming algorithm, described in Section 2.3.

This ability to sample paths will allow us to constrain the size of the state space when we move from pairs of sequences to entire phylogenetic trees. Tracing paths back through the state graph according to their posterior probability is straightforward once the Forward matrix is filled; the algorithmic internals are detailed in Section 2.3.

Maximum likelihood ancestral reconstruction under the TKF91 model

In Figure 2.36, the highest-weight (Viterbi) traceback path is highlighted. This path, via the significances of each of the visited states, corresponds to an ancestral alignment relating LIV and MF: an alignment of the sequences and their ancestor.

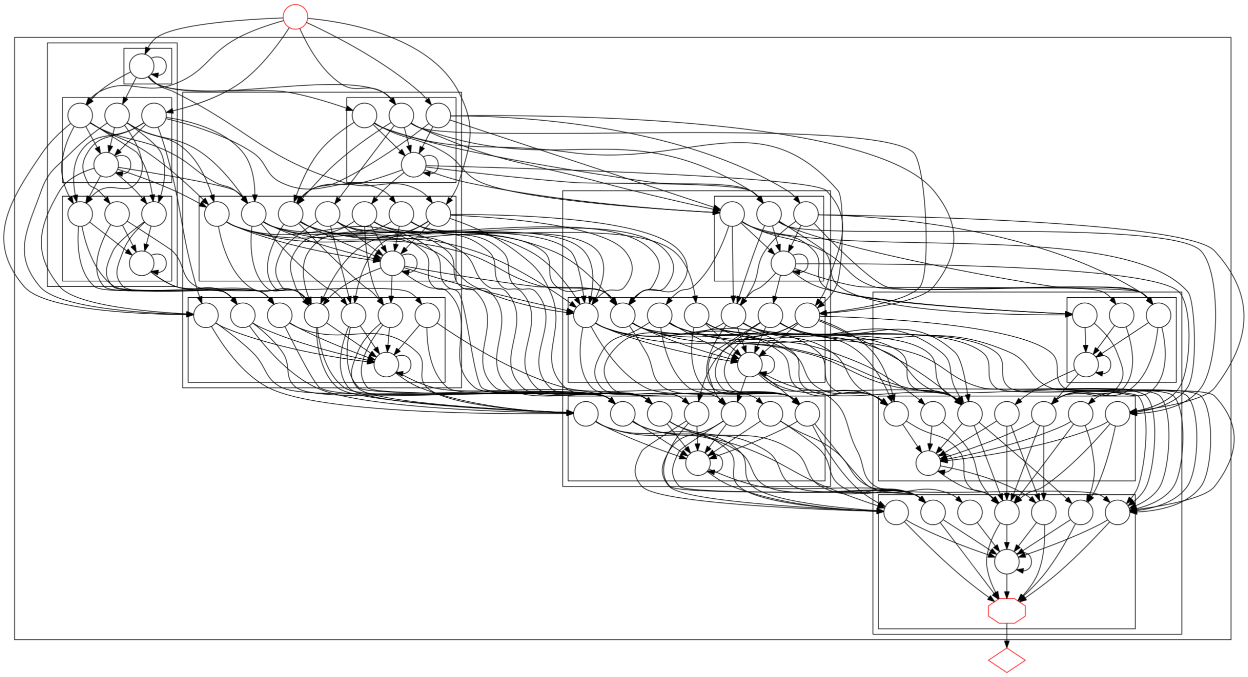


Figure 2.35: Transducer $\mathcal{R} \cdot ((\mathcal{B} \cdot \nabla(LIV)) \circ (\mathcal{B} \cdot \nabla(MF)))$ models the generation of an ancestral sequence which is then duplicated; the two copies are mutated (in parallel) by two TKF91 transducers into (respectively) LIV and MF. This machine is equivalent to the generator in Figure 2.18 coupled to the input of the recognizer in Figure 2.34. Because it is a generator coupled to a recognizer, there is no net input or output, and so we can think of this as a straightforward Markov model, albeit with some probability "missing": the total sum-over-paths weight from Start→End is less than one. Indeed, the total sum-over-paths weight through this Markov model corresponds to the joint likelihood of the two sibling sequences, LIV and MF. **Formal definitions:** This type of machine is denoted M_n in Section 2.3.

	Ancestral sequence	*	*	*
e.g.	Sequence 1	L	V	I
	Sequence 2	M	F	-

Computing this alignment/path is straightforward, and essentially amounts to choosing the highest-probability possibility (as opposed to sampling) in each step of the traceback detailed in section Section 2.3.

If we remove the root generator, we obtain a linear recognizer profile for the sequence at the ancestral node (Figure 2.38), as might be computed by progressive alignment. This can be thought of as a profile HMM trained on an alignment of the two sequences MF and LIV. It is a machine of the same form as E_n in Section 2.3.

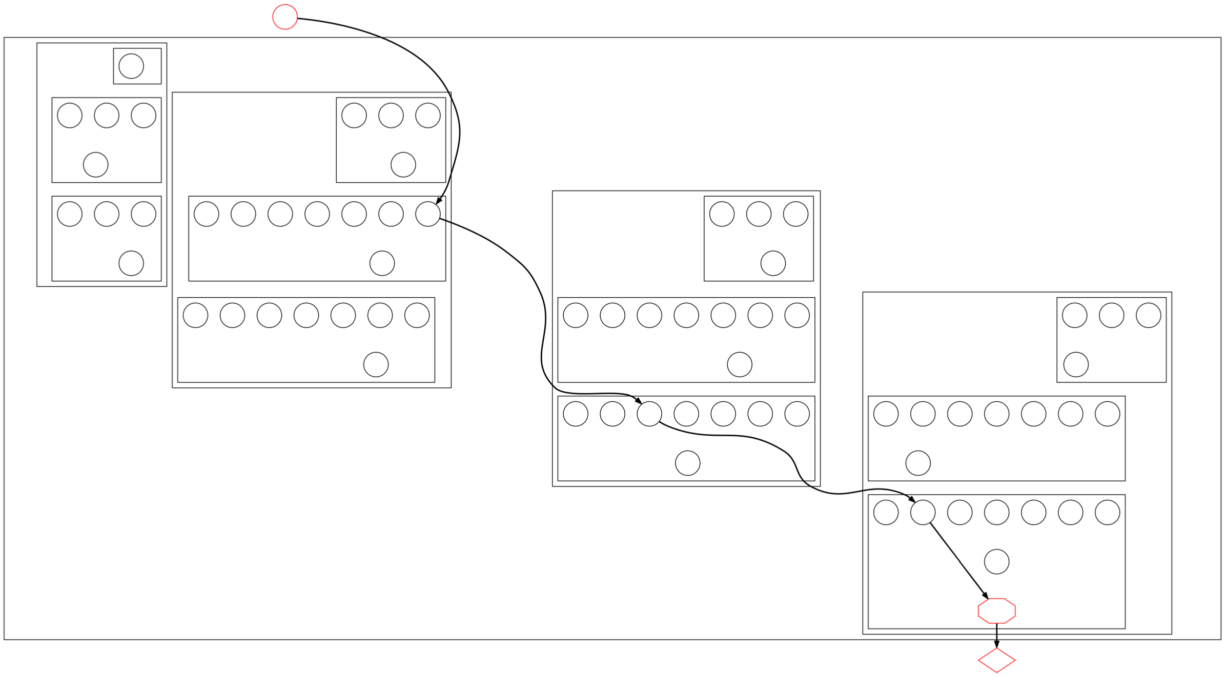


Figure 2.36: The highest-weight path through the machine of Figure 2.35 corresponds to the most likely evolutionary history relating the two sequences. Equivalently, this path corresponds to an alignment of the two sequences and their ancestors. **Formal definitions:** The subset of the state graph corresponding to this path is denoted M_n' in Section 2.3.

Sampling ancestral reconstructions under the TKF91 model

Instead of only saving the single highest-weight path through the machine of Figure 2.35 (as in Figure 2.36), we can sample several paths from the posterior probability distribution over paths. In Figure 2.39, two sampled paths through the state graph are shown. Tracing paths back through the state graph according to their posterior probability is straightforward once the Forward matrix is filled; the algorithmic internals are detailed in Section 2.3.

It is possible that many paths through the state graph have weights only slightly less than the Viterbi path. By sampling suboptimal paths according to their weights, it is possible to retain and propogate this uncertainty when applying this method to progressive alignment. Intuitively, this corresponds to storing multiple evolutionary histories of a subtree as progressive alignment climbs the phylogenetic tree.

For instance, in addition to sampling this path

Ancestral sequence	*	*	*
Sequence 1	L	V	I
Sequence 2	M	F	-

we also have this path

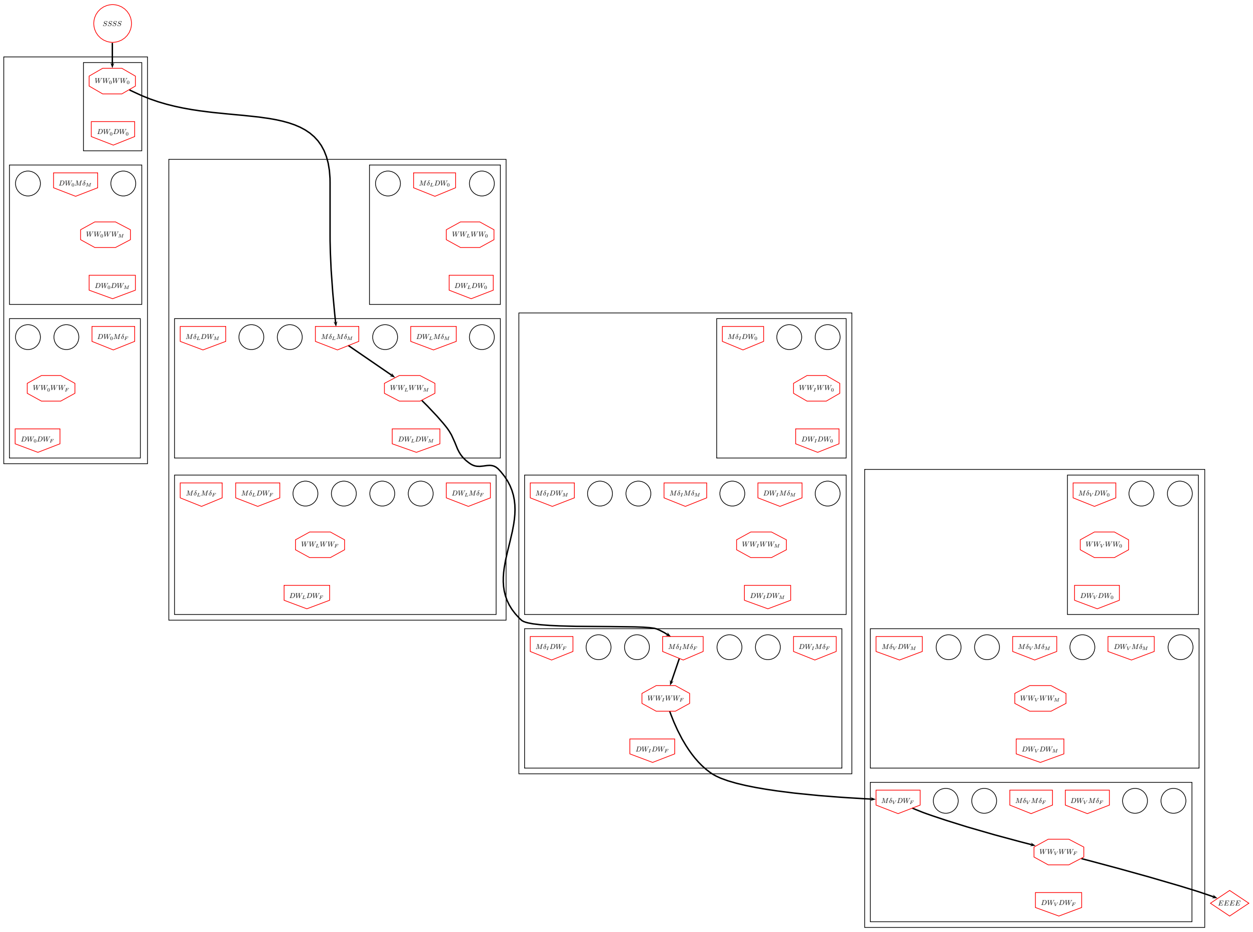


Figure 2.37: Removing the generator from Figure 2.36 leaves a recognition profile for the ancestral sequence relating MF and LIV. **Formal definitions:** This and related transformations to sampled state paths are described in Section 2.3.

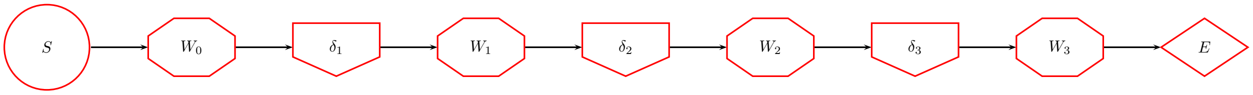


Figure 2.38: Transducer $\mathcal{P}_1$ is a linear recognition profile for the ancestor relating MF and LIV, created by taking the states visited by the Viterbi path shown in Figure 2.37. **Formal definitions:** This machine, and the one in Figure 2.41, are examples of the recognition profiles E_n described in Section 2.3.

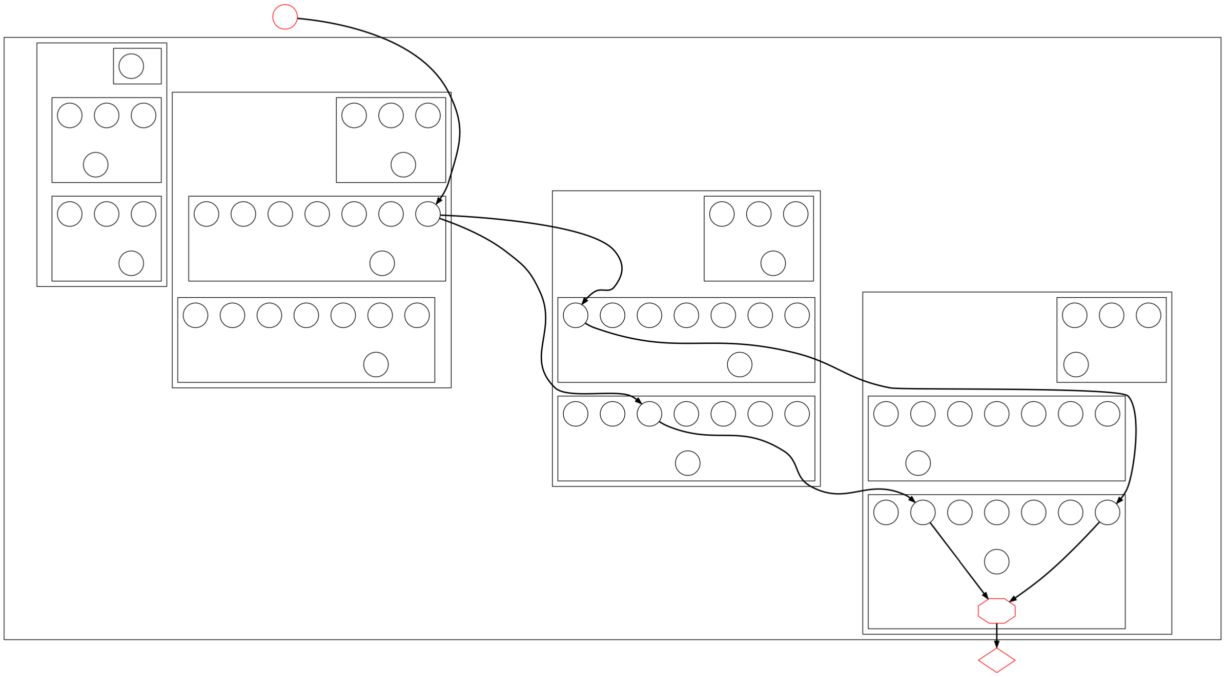


Figure 2.39: As well as finding just the highest-weight path through the machine of Figure 2.35 (as in Figure 2.36), it is possible to sample suboptimal paths proportional to their posterior probability. Here, two sampled paths through the state graph are shown. **Formal definitions:** The subset of the state graph covered by the set of sampled paths is denoted M'_n in Section 2.3. The mathematical detail of sampling paths is described in Section 2.3.

Ancestral sequence	*	*	*
Sequence 1	L	V	I
Sequence 2	M	-	F

Combining these paths into a single graph and relabeling again, we still have a recognition profile, but it is now branched, reflecting the possible uncertainty in our alignment/ancestral prediction, shown in Figure 2.41 (which is a transducer of the form E_n in Section 2.3). The exact series of transformations required to convert Figure 2.39 into Figure 2.41 (removing the root generator, eliminating null states, and adding wait states to restore the machine to Moore normal form) is detailed in Section 2.3.

Note that these are all still approximations to the full recognition profile for the ancestor, which is Figure 2.34. While we could retain this entire profile, progressively climbing up a tree would add so many states to the graph that inference would quickly become an intractable problem. Storing a subset allows a flexible way in which to retain many high-probability solutions while still allowing for a strict bound on the size of the state space.

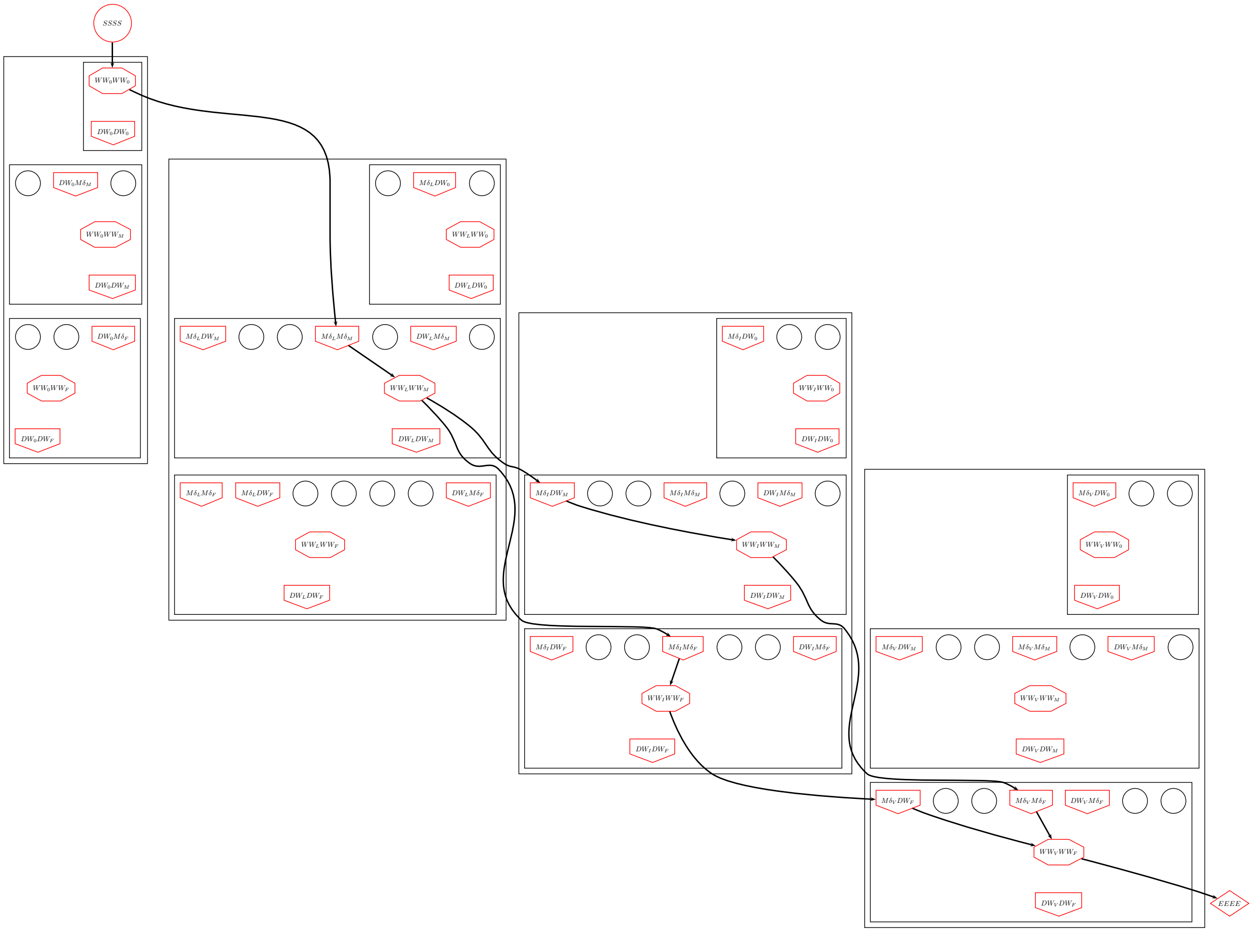


Figure 2.40: Two sample paths through the machine in Figure 2.35, representing possible evolutionary histories relating MF and LIV. These are the same two paths as in Figure 2.39, but we have removed the root generator (as we did to transform Figure 2.36 into Figure 2.37). Paths are sampled according to their posterior probability, allowing us to select a high-probability subset of the state graph. The mathematical details of sampling paths is described in Section 2.3.

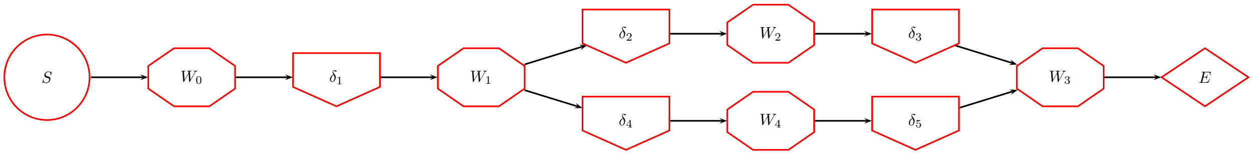


Figure 2.41: Transducer $\mathcal{P}_2$ is a branched recognizer for the ancestor common to MF and LIV. The branched structure results from sampling multiple paths through the state graph in Figure 2.35, mapping the paths back to Figure 2.34, and retaining only the subset of the graph visited by a sampled path. **Formal definitions:** This machine, and the one in Figure 2.38, are examples of the recognition profiles E_n described in Section 2.3.

Ancestral sequence recognizer on a larger TKF91 tree

Figure 2.42 shows the full recognition profile for the root-ancestral sequence in Figure 2.1, representing all the possible evolutionary histories relating the three sequences. For clarity, many transitions in this diagram have been removed or collapsed. (The recognition transducer for the root profile is denoted G_1 in Section 2.3.)

Felsenstein probability on a larger TKF91 tree

We have now seen the individual steps of the transducer version of the Felsenstein recursion. Essentially, composition replaces matrix multiplication, intersection replaces the pointwise product, and the initiation/termination steps involve (respectively) recognizers and generators.

The full recursion (with the same $\mathcal{O}(L^N)$ complexity as the Sankoff algorithm [41] for simultaneously aligning N sequences of length L, ignoring secondary structure) involves starting with exact-match recognition profiles at the leaves (Figure 2.11, Figure 2.10), using those to construct recognition profiles for the parents (Figure 2.34), and progressively climbing the tree toward the root, constructing ancestral recognition profiles for each internal node. At the root, compose the root generator with the root recognition profile, and the Forward probability can be computed.

Progressive alignment version of Felsenstein recursion

The "progressive alignment" version, equivalent to doing Felsenstein's pruning recursion on a single alignment found using the progressive alignment algorithm, involves sampling the single best linear recognition profile of the parent at each internal node, as in Figure 2.37.

We then repeat the process at the next level up in the tree, aligning the parent profile to its sibling, as shown in Figure 2.43. The machine in Figure 2.43 is an example of the transducer H_n in Section 2.3. (Technically, Figure 2.34 is also an example of H_n, as well as being an example of G_n. The difference is that G_n describes all possible histories below node n, since it is made by combining the two transducers G_l and G_r for the two children (l, r) of node n. By contrast, H_n only describes a *subset* of such histories, since it is made by combining the two transducers E_l and E_r, which are subsets of the corresponding H_l and H_r; just as Figure 2.38 and Figure 2.41 are subsets of Figure 2.34.)

This method can be recognized as a form of "sequence-profile" alignment as familiar from progressive alignment, except that we don't really make a distinction between a sequence and a profile (in that observed sequences are converted into exact-match recognition profiles in the very first steps of the procedure).

The "progressive" algorithm proceeds in the exact same way as the "full" version, except that at each internal node a linear profile is created from the Viterbi path through the state graph. This "best guess" of the alignment of each subtree is likely to work well in cases where the alignment is unambiguous, but under certain evolutionary parameters the alignment of a subtree may not be clear until more sequences are observed.

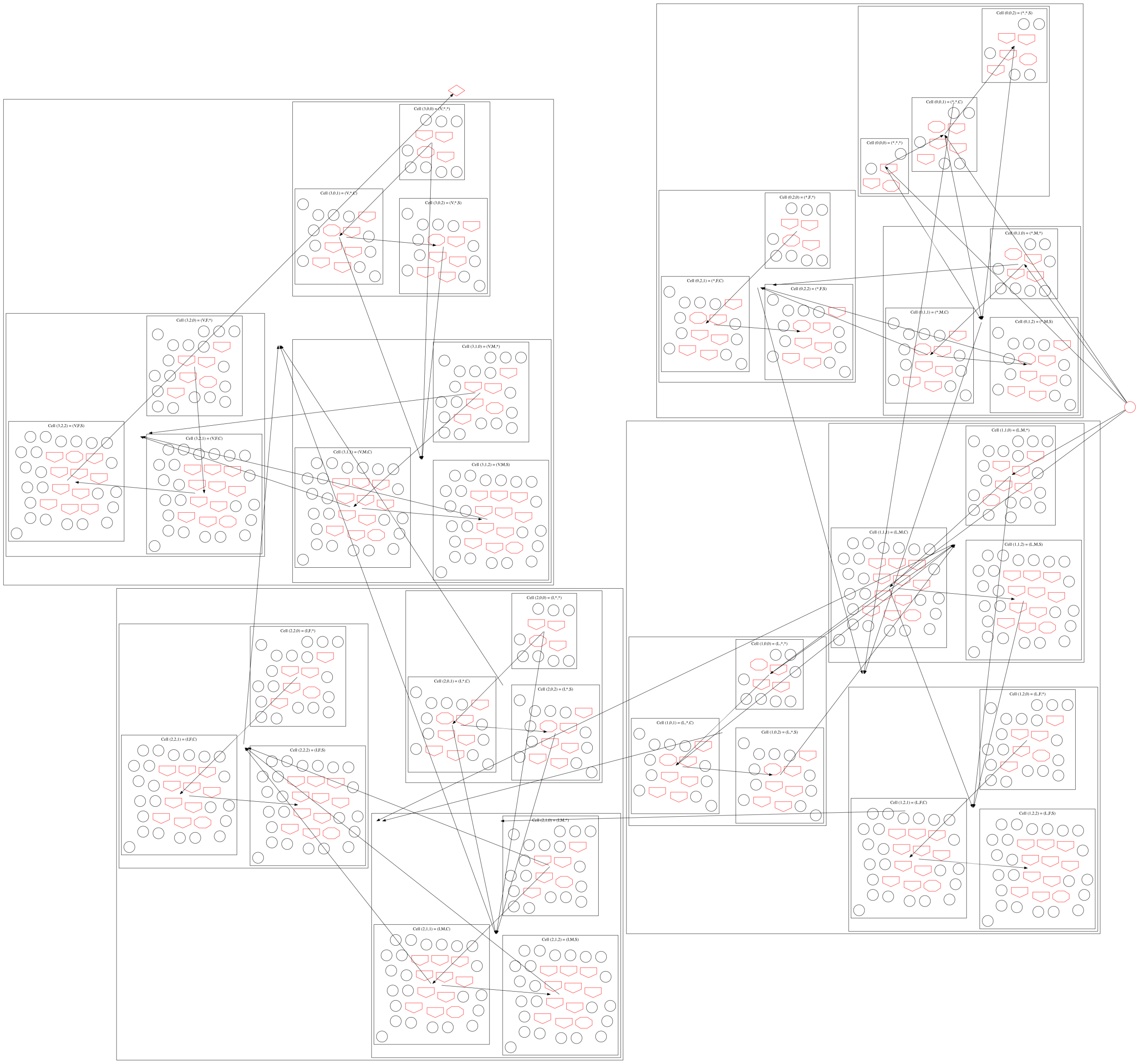


Figure 2.42: Transducer $(\mathcal{B}\cdot(\mathcal{B}\cdot\nabla(LIV))\circ(\mathcal{B}\cdot\nabla(MF)))\circ(\mathcal{B}\cdot\nabla(CS))$ is the full recognition profile for the root-ancestral sequence in Figure 2.1, representing all the possible evolutionary histories relating the three sequences. For clarity, many transitions in this diagram have been removed or collapsed. **Formal definitions:** This transducer, which recognizes the root sequence in Figure 2.1, is denoted G_1 in Section 2.3.

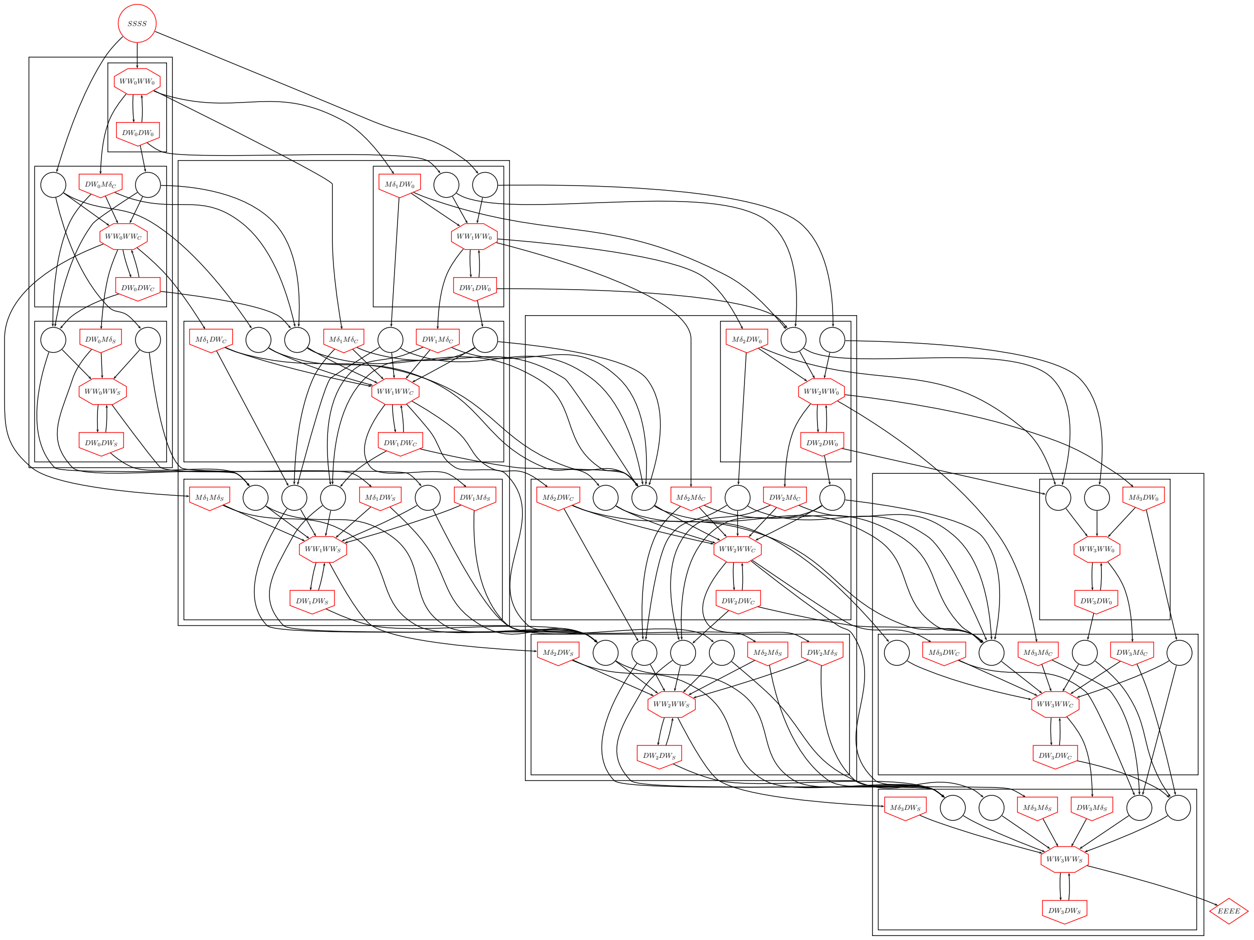


Figure 2.43: Transducer $(\mathcal{B} \cdot \mathcal{P}_1) \circ (\mathcal{B} \cdot \nabla(CS))$ recognizes the common ancestor of CS and $\mathcal{P}_1$. Transducer $\mathcal{P}_1$, shown in Figure 2.38, itself models the common ancestor of MF and LIV. Using profile $\mathcal{P}_1$, which is essentially a best-guess reconstructed ancestral profile, represents the most resource-conservative form of progressive alignment: only the maximum-likelihood indel reconstruction is kept during each step of the Felsenstein pruning recursion. **Formal definitions:** This type of transducer is denoted H_n in Section 2.3.

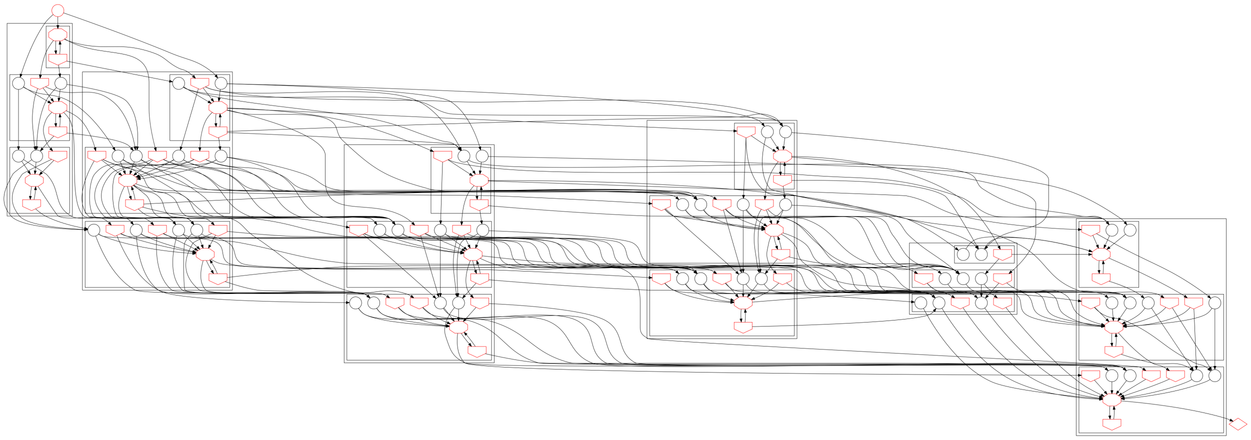


Figure 2.44: Transducer $(\mathcal{B} \cdot \mathcal{P}_2) \circ (\mathcal{B} \cdot \nabla(CS))$ shows the alignment of the sequence CS with the sampled profile $\mathcal{P}_2$. Transducer $\mathcal{P}_2$, shown in Figure 2.41, is a branched profile whose different paths represent alternate ancestries of sibling sequences MF and LIV. **Formal definitions:** The type of transducer shown in this Figure is denoted H_n in Section 2.3.

Stochastic lower bound version of Felsenstein recursion

Our stochastic lower bound version is intermediate to the progressive alignment (Viterbi-like) algorithm and the full Sankoff algorithm. Rather than just sampling the best linear profile for each parent, as in progressive alignment (Figure 2.37), we sample some fixed number of such paths (Figure 2.40). This allows us to account for some amount of alignment uncertainty, while avoiding the full complexity of the complete ancestral profile (Figure 2.42).

By sampling a fixed number of traceback paths, we can construct a recognition profile for the ancestral sequence that is linearly bounded in size and offers a stochastic "lower bound" on the probability computed by the full Felsenstein transducer in Figure 2.35 that (if we include the Viterbi path along with the sampled paths) is guaranteed to improve on the Viterbi lower-bound for the full Felsenstein probability.

Figure 2.44 shows the intersection of $\mathcal{P}_2$ (the ancestor of MF and LIV) with sequence CS, with TKF91 models accounting for the differences. Again this is a "sequence-profile" alignment, though it can also be called "profile-profile" alignment, since CS can be considered to be a (trivial) linear profile. Unlike in traditional progressive alignment (but quite like e.g. partial order alignment [43]), one of the profiles is now branched (because we sampled more than one path to construct it in Figure 2.40), allowing a tunable (by modulating how many paths are sampled in the traceback step) way to account for alignment uncertainty.

Just like Figure 2.43, the machine in Figure 2.44 is an example of the transducer H_n in Section 2.3.

Finally we compose the root generator (Figure 2.18) with Figure 2.44. (If there were more than two internal nodes in this tree, we would simply continue the process of aligning siblings and sampling a recognition profile for the parent, iterating until the root node was reached.) The sum of all path weights through the state graph of the final machine represents the stochastic lower-bound on the final Felsenstein probability for this tree. Since we have omitted some states at each progressive step, the computed probability does not sum over all possible histories relating the sequences, hence it is a lower bound on the true Felsenstein probability. (Each time we create a branched profile from a subset of the complete state graph, we discard some low-probability states and therefore leak a little bit of probability, but hopefully not much.) The hope is that, in practice, by sampling sufficiently many paths we are able to recover the maximum likelihood reconstruction at the root level, and so the lower bound is a good one. Furthermore, as long as we include the Viterbi path in with the retained sampled paths at every progressive step, then the final state machine will be a superset of the machine that Viterbi progressive alignment will have constructed, so we are guaranteed that the final likelihood will be greater than the Viterbi likelihood, while still representing a lower bound.

In terms of accuracy at estimating indel rates, we find that our method performs significantly better than Viterbi progressive alignment and approaches the accuracy of the true alignment. We simulated indel histories under 5 indel rates (0.005, 0.01, 0.02, 0.04, and 0.08 indels per unit time), and analyzed the unaligned leaf sequences using PRANK [6] and an implementation of our algorithm, ProtPal. Insertion and deletion rates were computed using basic event counts normalized by branch lengths (more details are provided in Methods). PRANK provides an excellent comparison since it is phylogenetically-oriented, but uses progressive Viterbi reconstruction rather than ensembles/profiles at internal nodes. Comparing indel rates computed using the true alignment gives an idea how well PRANK and ProtPal compare to a "perfect" method. Note that this does not necessarily correspond to sampling infinitely many paths in our method, since the maximum likelihood reconstruction may not always be the true alignment.

Figure 2.45 shows the error distributions of estimated insertion and deletion rates using the PRANK, ProtPal, and true reconstructions. The root mean squared error (RMSE), a measure of the overall distance from the 'true' value ($\frac{inferred}{true} = 1$, indicated with a vertical dashed line), is shown to the right of each distribution. ProtPal's RMSE values lie between PRANK's and the true alignment, indicating using alignment profiles at internal nodes allows for inferring more accurate alignments, or at least alignments which allow for more accurately computing summary statistics such as the indel rate. ProtPal's distributions are also more symmetric than PRANK's between insertions and deletions, indicating that it is more able to avoid the bias towards deletions described in [6].

A Pair HMM for aligning siblings

While we have constructed our hierarchy of phylogenetic transducers by using recognizers, it is also sometimes useful (and perhaps more conventional in bioinformatics) to think in terms of generators. For example, we can describe the multi-sequence HMM that simultaneously

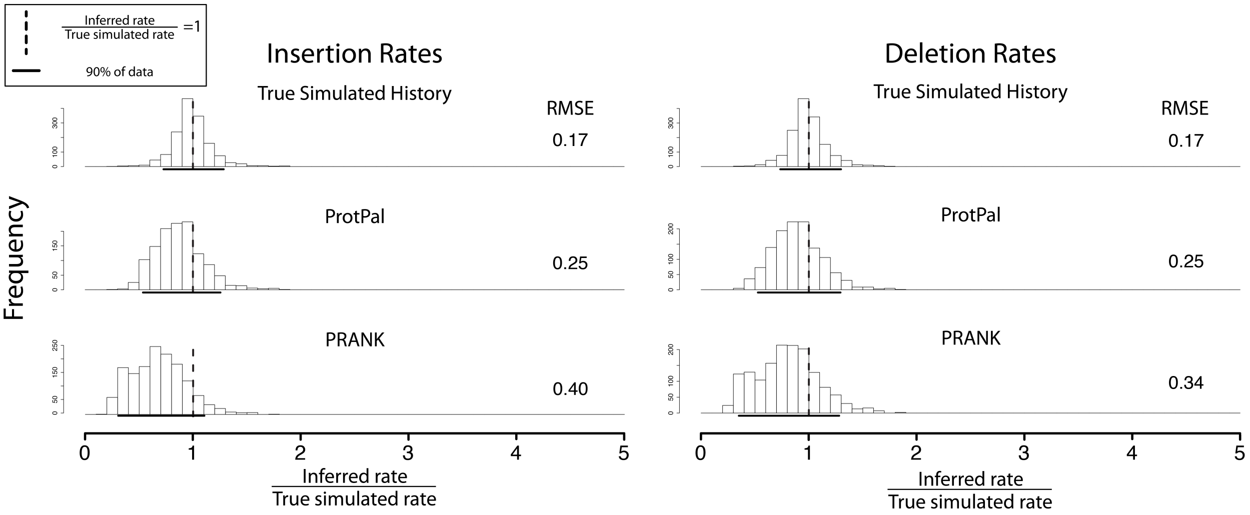


Figure 2.45: In a simulation experiment, an implementation of our algorithm outperforms PRANK [6] and approaches the accuracy of the true indel history. Alignments were simulated over range of evolutionary parameters, and unaligned leaf sequences along with a phylogeny were fed to PRANK and ProtPal, which each produced a predicted indel reconstruction. From each reconstruction, insertion and deletion rates were calculated by event counts normalized by branch lengths. The plot shows the ratio of $\frac{inferred}{true}$ rates pooled over all evolutionary parameters, with root mean squared error (RMSE) and middle 90% quantiles appearing to the right and below each histogram. Full results from these simulations are provided in Chapter 3.

emits an alignment of all of the sequences; this is a generator, and in fact is the model F_n described in Section 2.3. (An animation of such a multi-sequence generator emitting a small alignment can be viewed at `http://youtu.be/EcLj5MSDPyM` with more at `http://biowiki.org/PhyloFilm`.)

If we take the intersection of two TKF91 transducers, we obtain a transducer that has one input sequence and two output sequences (or, equivalently, one output tape that encodes a pairwise alignment of two sequences). This transducer is shown in Figure 2.46. If we then connect a TKF91 equilibrium generator (Figure 2.18) to the input of Figure 2.46, we get Figure 2.47: a generator with two output tapes, i.e. a Pair HMM. (To avoid introducing yet more tables and cross-references, we have confined the descriptions of the states in Figure 2.47 and Figure 2.46 to the respective Figure captions.)

The Pair HMM in Figure 2.47 is particularly useful, as it crops up in our algorithm whenever we have to align two recognizers. Specifically, Figure 2.35—which looks a bit like a pairwise dynamic programming matrix for aligning sequence LIV to sequence MF—is essentially Figure 2.47 with one output tape connected to the LIV-recognizer (Figure 2.10) and the other output tape connected to the MF-recognizer (Figure 2.11). In computing the state space of machines like Figure 2.35 (which are called M_n in Section 2.3), it is

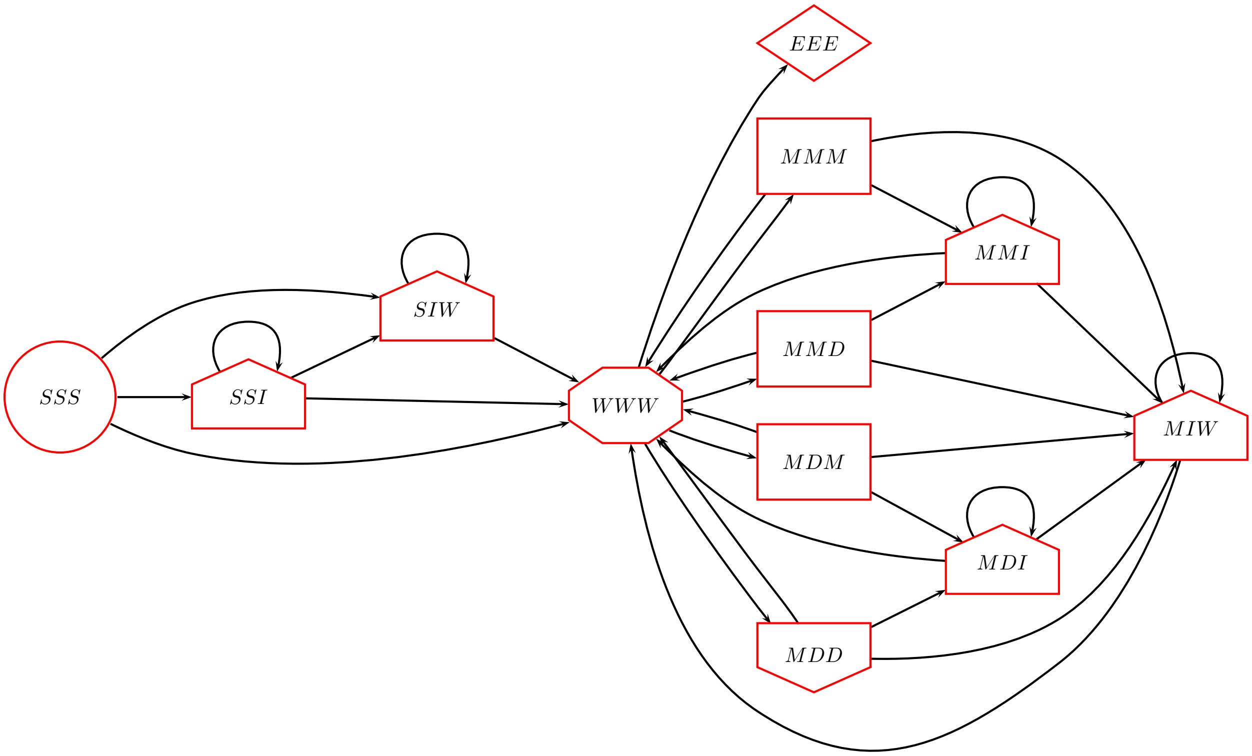


Figure 2.46: Transducer $\Upsilon \circ (\mathcal{B} \circ \mathcal{B})$ is a bifurcation of two TKF91 transducers. It can be viewed as a transducer with one input tape and two output tapes. Each state has the form $\upsilon b_1 b_2$ where υ is the state of the bifurcation transducer (Section 2.3), b_1 is the state of the first TKF91 machine (Figure 2.17) and b_2 is the state of the second TKF91 machine. The meaning of I/O states (Match, Insert, Delete) is subtle in this model, because there are two output tapes. Dealing first with the Inserts: in states SIW and MIW, the first TKF91 transducer is inserting symbols to the first output tape, while in states SSI, MMI and MDI, the second TKF91 transducer is emitting symbols to the second output tape. Dealing now with the Matches and Deletes: the four states that can receive an input symbol are MMM, MMD, MDM and MDD. Of these, MMM emits a symbol to both output tapes (and so is a Match); MMD only emits a symbol to the first output tape (and so qualifies as a Match because it has input and output); MDM only emits a symbol to the second output tape (and so qualifies as a Match); and MDD produces no output at all (and is therefore the only true Delete state).

useful to precompute the state space of the component machine in Figure 2.47 (called Q_n in Section 2.3). This amounts to a run-time optimization, though it also helps us verify correctness of the model.

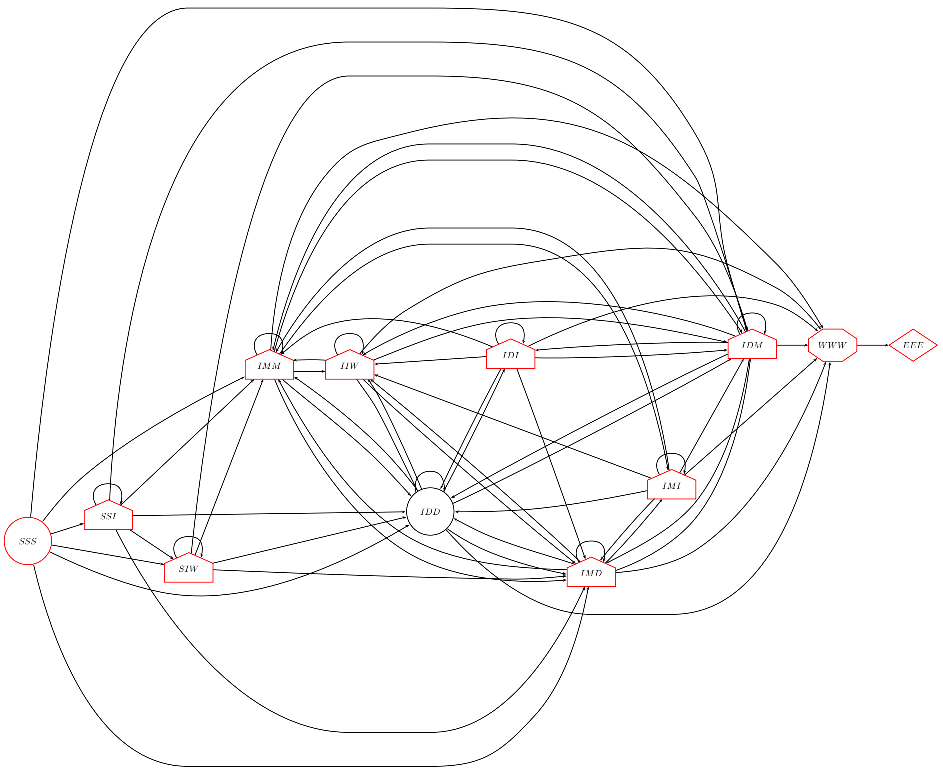


Figure 2.47: Transducer $\mathcal{R} \circ (\mathcal{B} \circ \mathcal{B})$ represents the operation of sampling a sequence from the TKF91 equilibrium distribution and then feeding that sequence independently into two TKF91 transducers. Equivalently, it is the composition of the TKF91 equilibrium generator (Figure 2.18) with a bifurcation of two TKF91 transducers (Figure 2.46). It can be viewed as a generator with two output tapes; i.e. a Pair HMM. Each state has the form $\rho b_1 b_2$ where ρ is the state of the generator, b_1 is the state of the first TKF91 machine and b_2 is the state of the second. As in Figure 2.46, the meaning of I/O states is subtle in this model, because there are two output tapes. We first deal with Insert states where one of the TKF91 transducers is responsible for the insertion. In states SIW and IIW, the first TKF91 transducer is emitting symbols to the first output tape; in states SSI, IDI and IMI, the second TKF91 transducer is emitting symbols to the second output tape. The remaining states (excluding SSS and EEE) all involve symbol emissions by the generator, that are then processed by the two TKF91 models in various ways. These four states that involve emissions by the generator are IMM, IMD, IDM and IDD. Of these, IMM emits a symbol to both output tapes (and so qualifies as an Insert state); IMD only emits a symbol to the first output tape (and is an Insert state); IDM only emits a symbol to the second output tape (and is an Insert state); and IDD produces no output at all (and is therefore a Null state). Note that Figure 2.35, which looks a bit like a pairwise dynamic programming matrix, is essentially this Pair HMM with one output tape connected to the LIV-recognizer (Figure 2.10) and the other output tape connected to the MF-recognizer (Figure 2.11), **Formal definitions:** This type of transducer is called Q_n in Section 2.3. When both its outputs are connected to recognizers (H_l and H_r), then one obtains a transducer of the form M_n.

A note on the relationship between this tutorial and the formal definitions section

As noted throughout the tutorial, the example phylogeny and transducers can be directly related to the "Hierarchy of phylogenetic transducers" described in Section 2.3 onwards.

Consider the tree of Figure 2.1. Let the nodes of this tree be numbered as follows: (1) Ancestral sequence, (2) Intermediate sequence, (3) Sequence LIV, (4) Sequence MF, (5) Sequence CS.

Some of the transducers defined for this tree in Section 2.3 include

$$
\begin{array}{rcll}
R & = & \mathcal{R} & \text{Figure 2.18} \\
\forall n \in \{2,3,4,5\} : \quad B_n & = & \mathcal{B} & \text{Figure 2.17} \\
E_5 = H_5 = G_5 & = & \nabla(CS) & \text{Figure 2.12} \\
E_4 = H_4 = G_4 & = & \nabla(MF) & \text{Figure 2.11} \\
E_3 = H_3 = G_3 & = & \nabla(LIV) & \text{Figure 2.10} \\
G_2 & = & (B_3 \cdot G_3) \circ (B_4 \cdot G_4) & \\
 & = & (\mathcal{B} \cdot \nabla(LIV)) \circ (\mathcal{B} \cdot \nabla(MF)) & \text{Figure 2.34} \\
H_2 & = & (B_3 \cdot E_3) \circ (B_4 \cdot E_4) & \\
 & = & (\mathcal{B} \cdot \nabla(LIV)) \circ (\mathcal{B} \cdot \nabla(MF)) & \text{Figure 2.34 again} \\
M_2 & = & R \cdot H_2 & \\
 & = & \mathcal{R} \cdot (\mathcal{B} \cdot \nabla(LIV)) \circ (\mathcal{B} \cdot \nabla(MF)) & \text{Figure 2.35} \\
E_2 & \subseteq & H_2 & \text{Figure 2.38, Figure 2.4} \\
G_1 & = & (B_2 \cdot G_2) \circ (B_5 \cdot G_5) & \\
 & = & (\mathcal{B} \cdot (\mathcal{B} \cdot \nabla(LIV)) \circ (\mathcal{B} \cdot \nabla(MF))) \circ (\mathcal{B} \cdot \nabla(CS)) & \text{Figure 2.42} \\
H_1 & = & (B_2 \cdot E_2) \circ (B_5 \cdot E_5) & \text{Figure 2.43, Figure 2.4} \\
G_0 & = & R \cdot G_1 & \\
 & = & \mathcal{R} \cdot (\mathcal{B} \cdot (\mathcal{B} \cdot \nabla(LIV)) \circ (\mathcal{B} \cdot \nabla(MF))) \circ (\mathcal{B} \cdot \nabla(CS)) & \text{Figure 2.2} \\
\forall n \in \{1,2\} : \quad Q_n & = & \mathcal{R} \circ (\mathcal{B} \cdot \mathcal{B}) & \text{Figure 2.47}
\end{array}
$$

2.3 Formal definitions

This report makes our transducer-related definitions precise, including notation for state types, weights (i.e. probabilities), transducer composition, etc.

Notation relating to mundane manipulations of sequences (sequence length, sequence concatenation, etc.) is deferred to the end of the document, so as not to interrupt the flow.

We first review the letter transducer T, transduction weight $\mathcal{W}(x : [T] : y)$ and equivalence $T \equiv T'$.

We then define two operations for combining transducers: composition (TU) which unifies T's output with U's input, and intersection ($T \circ U$) which unifies T's and U's input.

We define our "normal" form for letter transducers, partitioning the states and transitions into types $\{S, M, D, I, N, W, E\}$ based on their input/output labeling. (These types stand for Start, Match, Delete, Insert, Null, Wait, End.) This normal form is common in the bioinformatics literature [17] and forms the basis for our previous constructions of phylogenetic transducers [4, 58].

We define exact-match and identity transducers, and give constructions of these.

We define our hierarchy of phylogenetic transducers, and give constructions and inference algorithms, including the concept of "alignment envelopes" for size-limiting of transducers.

Input-output automata

The letter transducer is a tuple $T = (\Omega_I, \Omega_O, \Phi, \phi_S, \phi_E, \tau, \mathcal{W})$ where Ω_I is an input alphabet, Ω_O is an output alphabet, Φ is a set of states, $\phi_S \in \Phi$ is the start state, $\phi_E \in \Phi$ is the end state, $\tau \subseteq \Phi \times (I \cup \{\epsilon\}) \times (O \cup \{\epsilon\}) \times \Phi$ is the transition relation, and $\mathcal{W} : \tau \to [0, \infty)$ is the transition weight function.

Transition paths: The transitions in τ correspond to the edges of a labeled multidigraph over states in Φ. Let $\Pi \subset \tau^*$ be the set of all labeled transition paths from ϕ_S to ϕ_E.

I/O sequences: Let $S_I : \Pi \to \Omega_I^*$ and $S_O : \Pi \to \Omega_O^*$ be functions returning the input and output sequences of a transition path, obtained by concatenating the respective transition labels.

Transduction weight: For a transition path $\pi \in \Pi$, define the path weight $\mathcal{W}(\pi)$ and (for sequences $x \in \Omega_I^*, y \in \Omega_O^*$) the transduction weight $\mathcal{W}(x : [T] : y)$

$$\begin{aligned} \mathcal{W}(\pi) &= \prod_{\tau \in \pi} \mathcal{W}(\tau) \\ \mathcal{W}(x : [T] : y) &= \sum_{\pi \in \Pi, S_I(\pi) = x, S_O(\pi) = y} \mathcal{W}(\pi) \end{aligned}$$

Equivalence: If for transducers T, T' it is true that $\mathcal{W}(x : [T] : y) = \mathcal{W}'(x : [T'] : y)\ \forall x, y$ then the transducers are equivalent in weight, $T \equiv T'$.

State types and normal forms

Types of state and transition: If there exists a state type function, type : $\Phi \to \mathcal{T}$, mapping states to types in $\mathcal{T} = \{S, M, D, I, N, W, E\}$, and functions $\mathcal{W}^{\text{trans}} : \Phi^2 \to [0, \infty)$ and $\mathcal{W}^{\text{emit}} : (I \cup \{\epsilon\}) \times (O \cup \{\epsilon\}) \times \Phi \to [0, \infty)$, such that

$$\begin{aligned}
\Phi_U &= \{\phi : \phi \in \Phi, \text{type}(\phi) \in U \subseteq \mathcal{T}\} \\
\Phi_S &= \{\phi_S\} \\
\Phi_E &= \{\phi_E\} \\
\Phi &\equiv \Phi_{SMDINWE} \\
\tau_M &\subseteq \Phi_W \times \Omega_I \times \Omega_O \times \Phi_M \\
\tau_D &\subseteq \Phi_W \times \Omega_I \times \{\epsilon\} \times \Phi_D \\
\tau_I &\subseteq \Phi_{SMDIN} \times \{\epsilon\} \times \Omega_O \times \Phi_I \\
\tau_N &\subseteq \Phi_{SMDIN} \times \{\epsilon\} \times \{\epsilon\} \times \Phi_N \\
\tau_W &\subseteq \Phi_{SMDIN} \times \{\epsilon\} \times \{\epsilon\} \times \Phi_W \\
\tau_E &\subseteq \Phi_W \times \{\epsilon\} \times \{\epsilon\} \times \Phi_E \\
\tau &= \tau_M \cup \tau_D \cup \tau_I \cup \tau_N \cup \tau_W \cup \tau_E \\
\mathcal{W}(\phi_{\text{src}}, \omega_{\text{in}}, \omega_{\text{out}}, \phi_{\text{dest}}) &\equiv \mathcal{W}^{\text{trans}}(\phi_{\text{src}}, \phi_{\text{dest}}) \mathcal{W}^{\text{emit}}(\omega_{\text{in}}, \omega_{\text{out}}, \phi_{\text{dest}})
\end{aligned}$$

then the transducer is in *(weak) normal form.* If, additionally, $\Phi_N = \emptyset$, then the transducer is in *strict normal form.* The above transition and I/O constraints are summarized graphically in Figure 2.8.

Interpretation: A normal-form transducer can be thought of as associating inputs and outputs with states, rather than transitions. (Thus, it is like a Moore machine.) The state types are start (S) and end (E); wait (W), in which the transducer waits for input; match (M) and delete (D), which process input symbols; insert (I), which writes additional output symbols; and null (N), which has no associated input or output. All transitions also fall into one of these types, via the destination states; thus, τ_M is the set of transitions ending in a match state, etc. The transition weight ($\mathcal{W}$) factors into a term that is independent of the input/output label ($\mathcal{W}^{\text{trans}}$) and a term that is independent of the source state ($\mathcal{W}^{\text{emit}}$).

Universality: For any weak-normal form transducer T there exists an equivalent in strict-normal form which can be found by applying the state-marginalization algorithm to eliminate null states. For any transducer, there is an equivalent letter transducer in weak normal form, and therefore, in strict normal form.

Moore and Mealy machines

The following terms in common usage relate approximately to our definitions:

Mealy machines are transducers with I/O occurring on transitions, as with our general definition of the letter transducer.

Moore machines are transducers whose I/O is associated with states, as with our normal form. The difference between these two views is illustrated via a small example in Figure 2.6 and Figure 2.5.

Composition (TU) unifies output of T with input of U

Transducer composition: Given letter transducers $T = (\Omega_X, \Omega_Y, \Phi, \phi_S, \phi_E, \tau, \mathcal{W})$ and $U = (\Omega_Y, \Omega_Z, \Phi', \phi'_S, \phi'_E, \tau', \mathcal{W}')$, there exists a letter transducer $TU = (\Omega_X, \Omega_Z, \Phi'' \ldots \mathcal{W}'')$ such that $\forall x \in \Omega_X^*, z \in \Omega_Z^*$:

$$\mathcal{W}''(x : [TU] : z) = \sum_{y \in \Omega_Y^*} \mathcal{W}(x : [T] : y)\mathcal{W}'(y : [U] : z)$$

Example construction: Assume without loss of generality that T and U are in strict normal form. Then $\Phi'' \subset \Phi \times \Phi'$, $\phi''_S = (\phi_S, \phi'_S)$, $\phi''_E = (\phi_E, \phi'_E)$ and

$$\mathcal{W}''((t,u), \omega_x, \omega_z, (t',u')) = \begin{cases} \delta(t = t')\delta(\omega_x = \epsilon)\mathcal{W}'(u, \epsilon, \omega_z, u') & \text{if type}(u) \neq W \\ \mathcal{W}(t, \omega_x, \epsilon, t')\delta(\omega_z = \epsilon)\delta(u = u') & \text{if type}(u) = W, \text{type}(t') \notin \{M, I\} \\ \displaystyle\sum_{\omega_y \in \Omega_Y} \mathcal{W}(t, \omega_x, \omega_y, t')\mathcal{W}'(u, \omega_y, \omega_z, u') & \text{if type}(u) = W, \text{type}(t') \in \{M, I\} \\ 0 & \text{otherwise} \end{cases}$$

The resulting transducer is in weak-normal form (it can be converted to a strict-normal form transducer by eliminating null states).

In the tutorial section, many examples of simple and complex compositions are shown in Section 2.2, for instance Figure 2.24, Figure 2.26 and Figure 2.31.

Intersection ($T \circ U$) unifies input of T with input of U

Transducer intersection: Given letter transducers $T = (\Omega_X, \Omega_T, \Phi, \phi_S, \phi_E, \tau, \mathcal{W})$ and $U = (\Omega_X, \Omega_U, \Phi', \phi'_S, \phi'_E, \tau', \mathcal{W}')$, there exists a letter transducer $T \circ U = (\Omega_X, \Omega_V, \Phi'' \ldots \mathcal{W}'')$ where $\Omega_V \subseteq (T \cup \{\epsilon\}) \times (U \cup \{\epsilon\})$ such that $\forall x \in \Omega_X^*, t \in \Omega_T^*, u \in \Omega_U^*$:

$$\mathcal{W}(x : [T] : t)\mathcal{W}'(x : [U] : u) = \mathcal{W}''(x : [T \circ U] : (t, u))$$

where the term on the right is defined as follows

$$\mathcal{W}''(x : [T \circ U] : (t, u)) = \sum_{v \in \Omega_V^*, S_1(v) = t, S_2(v) = u} \mathcal{W}''(x : [T \circ U] : v)$$

Here Ω_V is the set of all possible pairwise alignment columns, $v \in \Omega_V^*$ is a pairwise alignment and $S_1(v)$ and $S_2(v)$ are the sequences in (respectively) the first and second rows of v.

Example construction: Assume without loss of generality that T and U are in strict normal form. Then $\Phi'' \subset \Phi \times \Phi'$, $\phi''_S = (\phi_S, \phi'_S)$, $\phi''_E = (\phi_E, \phi'_E)$ and

$$\mathcal{W}''((t,u),\omega_x,(\omega_y,\omega_z),(t',u')) = \\ \begin{cases} \delta(t=t')\delta(\omega_x=\omega_y=\epsilon)\mathcal{W}'(u,\epsilon,\omega_z,u') & \text{if type}(u) \neq W \\ \mathcal{W}(t,\epsilon,\omega_x,t')\delta(\omega_x=\omega_z=\epsilon)\delta(u=u') & \text{if type}(u) = W, \text{type}(t) \neq W \\ \mathcal{W}(t,\omega_x,\omega_y,t')\mathcal{W}'(u,\omega_x,\omega_z,u') & \text{if type}(t) = \text{type}(u) = W \\ 0 & \text{otherwise} \end{cases}$$

The resulting transducer is in weak-normal form (it can be converted to a strict-normal form transducer by eliminating null states). In the tutorial section, many examples of simple and complex intersections are shown in Section 2.2, for instance Figure 2.32 and Figure 2.35.

Identity and bifurcation transducers ($\mathcal{I}$, Υ)

Identity: There exists a transducer $\mathcal{I} = (\Omega, \Omega \ldots)$ that copies its input identically to its output. An example construction (not in normal form) is

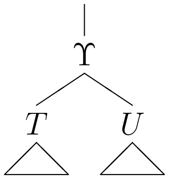
$$\begin{aligned} \mathcal{I} &= (\Omega, \Omega, \{\phi\}, \phi, \phi, \tau_{\mathcal{I}}, 1) \\ \tau_{\mathcal{I}} &= \{(\phi, \omega, \omega, \phi) : \omega \in \Omega\} \end{aligned}$$

Bifurcation: There exists a transducer $\Upsilon = (\Omega, \Omega^2 \ldots)$ that duplicates its input in parallel. That is, for input $x_1 x_2 x_3 \ldots$ it gives output $\left(\begin{array}{c} x_1 \\ x_1 \end{array}\right)\left(\begin{array}{c} x_2 \\ x_2 \end{array}\right)\left(\begin{array}{c} x_3 \\ x_3 \end{array}\right)\ldots$. An example construction (not in normal form) is

$$\begin{aligned} \Upsilon &= (\Omega, \Omega^2, \{\phi\}, \phi, \phi, \tau_{\Upsilon}, 1) \\ \tau_{\Upsilon} &= \left\{\left(\phi, \omega, \left(\begin{array}{c} \omega \\ \omega \end{array}\right), \phi\right) : \omega \in \Omega\right\} \end{aligned}$$

It can be seen that $\Upsilon \equiv \mathcal{I} \circ \mathcal{I}$.

An intersection $T \circ U$ may be considered a parallel composition of Υ with T and U. We write this as $\Upsilon(T, U)$ or, diagrammatically,



We use the notation $\Upsilon(T, U)$ in several places, when it is convenient to have a placeholder transducer Υ at a bifurcating node in a tree.

Exact-match recognizers ($\nabla(S)$)

Recognition profiles: A transducer T is a *recognizer* if it has a null output alphabet, and so generates no output except the empty string.

Exact-match recognizer: For $S \in \Omega^*$, there exists a transducer $\nabla(S) = (\Omega, \emptyset \ldots \mathcal{W})$ that accepts the specific sequence S with weight one, but rejects all other input sequences

$$\mathcal{W}(x : [\nabla(S)] : \epsilon) = \delta(x = S)$$

Note that $\nabla(S)$ has a null output alphabet, so its only possible output is the empty string, and it is a recognizer.

In general, if $T = (\Omega_X, \Omega_Y \ldots \mathcal{W}')$ is any transducer then $\forall x \in \Omega_X^*, y \in \Omega_Y^*$

$$\mathcal{W}'(x : [T] : y) \equiv \mathcal{W}(x : [T\nabla(y)] : \epsilon)$$

An example construction (not in normal form) is

$$\begin{aligned} \nabla(S) &= (\Omega, \emptyset, \mathbb{Z}_{\text{length}(S)+1}, 0, \text{length}(S), \tau_\nabla, 1) \\ \tau_\nabla &= \Big\{ (n, \text{symbol}(S, n+1), \epsilon, n+1) : n \in \mathbb{Z}_{\text{length}(S)} \Big) \} \end{aligned}$$

where $\mathbb{Z}_N$ is the set of integers modulo N, and symbol(S, k) is the k'th position of S (for $1 \leq k \leq \text{length}(S)$). Note that this construction has length$(S) + 1$ states.

For later convenience it is useful to define the function

$$\begin{aligned} t_{\nabla(S)}(i, j) &= \mathcal{W}^{\text{trans}}_{\nabla(S)}(i, j) \\ &= \delta(i + 1 = j) \end{aligned}$$

Figure 2.4 shows a small example of an exact-match transducer for sequence LIV, while Figure 2.10 shows an equivalent exact-match transducer in normal form.

Generators

Generative transducers: A transducer T is generative (or "a *generator*") if it has a null input alphabet, and so rejects any input except the empty string. Then T may be regarded as a state machine that generates an output, equivalent to a Hidden Markov Model. Define the probability (weight) distribution over the output sequence

$$P(x|T) \equiv \mathcal{W}(\epsilon : [T] : x)$$

Figure 2.9 and Figure 2.17 are both examples of generative transducers. Figure 2.9 is a specific generator that only emits one sequence (with probability 1), while Figure 2.17 can potentially emit (and defines a probability for) any output sequence.

Algorithmic complexities

$$\begin{aligned} |\Phi_{TU}| &= \mathcal{O}(|\Phi_T||\Phi_U|) \\ |\Phi_{T \circ U}| &= \mathcal{O}(|\Phi_T||\Phi_U|) \\ |\Phi_{\nabla(S)}| &= \mathcal{O}(\text{length}(S)) \end{aligned}$$

The complexity of computing $\mathcal{W}(x : [T] : y)$ is similar to the Forward algorithm: the time complexity is $\mathcal{O}(|\tau_T|\text{length}(x)\text{length}(y))$ and the memory complexity is $\mathcal{O}(|\Phi_T| \min(\text{length}(x), \text{length}(y)))$. Memory complexity rises to $\mathcal{O}(|\Phi_T|\text{length}(x)\text{length}(y))$ if a traceback is required. Analogously to the Forward algorithm, there are checkpointing versions which trade memory complexity for time complexity.

Chapman-Kolmogorov equation

If T_t is a transducer parameterized by a continuous time parameter t, modeling the evolution of a sequence for time t under a continuous-time Markov process, then the Chapman-Kolmogorov equation [60] can be expressed as a transducer equivalence

$$T_t T_u \equiv T_{t+u}$$

The TKF91 transducers, for example, have this property. Furthermore, for TKF91, T_{t+u} has the same number of states and transitions as T_t, so this is a kind of self-similarity. TKF91 composed with itself is shown in Figure 2.24.

In this paper, we have deferred the difficult problem of finding time-parameterized transducers that solve this equation (and so may be appropriate for Felsenstein recursions). For studies of this problem the reader is referred to previous work [54–57, 61].

Hierarchy of phylogenetic transducers

Phylogenetic tree (n, $\mathcal{L}$)

Suppose we have an evolutionary model defined on a rooted binary phylogenetic tree, and a set of κ observed sequences associated with the leaf nodes of the tree.

The nodes are numbered in preorder, with internal nodes $(1 \ldots \kappa - 1)$ preceding leaf nodes $\mathcal{L} = \{\kappa \ldots 2\kappa - 1\}$. Node 1 is the root.

Hidden and observed sequences ($\mathcal{S}_n$)

Let $\mathcal{S}_n \in \Omega^*$ denote the sequence at node n and let $\mathcal{S} = \{\mathcal{S}_n, n \in \mathcal{L}\}$ denote the observed leaf-node sequences.

Model components (B_n, R)

Let $B_n = (\Omega, \Omega, \ldots)$ be a transducer modeling the evolution on the branch to node $n > 1$, from n's parent. Let $R = (\emptyset, \Omega, \ldots)$ be a generator modeling the distribution of ancestral sequences at the root node.

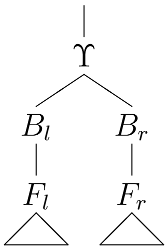
Figure 2.17 and Figure 2.19 are examples of B_n transducers. Figure 2.18 is an example of an R transducer.

The forward model (F_n)

If $n \geq 1$ is a leaf node, define $F_n = \mathcal{I}$. Otherwise, let (l, r) denote the left and right child nodes, and define

$$F_n = (B_l F_l) \circ (B_r F_r)$$

which we can represent as (recall that Υ is the bifurcation transducer).



The complete, generative transducer is $F_0 = RF_1$

The output alphabet of F_0 is $(\cup \{\epsilon\})^\kappa$ where κ is the number of leaf sequences. Letting $S_n : \tau^* \to \Omega^*$ denote the map from a transition path π to the n'th output leaf sequence (with gaps removed), we define the output distribution

$$P(\mathcal{S}|F_0) = \mathcal{W}(\epsilon : [F_0] : \mathcal{S}) = \sum_{\pi : S_n(\pi) = \mathcal{S}_n \forall n \in \mathcal{L}_n} \mathcal{W}(\pi)$$

where $\mathcal{L}_n$ denotes the set of leaf nodes that have n as a common ancestor.

Note that $|\Phi_{F_0}| \simeq \prod_n^{2\kappa-1} |\Phi_{B_n}|$ where $2\kappa - 1$ is the number of nodes in the tree. So the state space grows exponentially with the size of the tree—and this is before we have even introduced any sequences. We seek to avoid this with our hierarchy of approximate models, which will have state spaces that are bounded in size.

First, however, we expand the state space even more, by introducing the observed sequences explicitly into the model.

The evidence-expanded model (G_n)

Inference with stochastic grammars often uses a dynamic programming matrix (e.g. the Inside matrix) to track the ways that a given evidential sequence can be produced by a given grammar.

For our purposes it is useful to introduce the evidence in a different way, by transforming the model to incorporate the evidence directly. We augment the state space so that the model is no longer capable of generating any sequences *except* the observed $\{\mathcal{S}_n\}$, by composing F_0 with exact-match transducers that will only accept the observed sequences. This yields a model whose state space is very large and, in fact, is directly analogous to the Inside dynamic programming matrix.

If $n \geq 1$ is a leaf node, define $G_n = \nabla(\mathcal{S}_n)$. The number of states is $|\Phi_{G_n}| = \mathcal{O}(\text{length}(\mathcal{S}_n))$.

Otherwise, let (l, r) denote the left and right child nodes, and define

$$G_n = (B_l G_l) \circ (B_r G_r)$$

which we can represent as

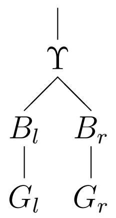


Figure 2.34 and Figure 2.42 are examples of G_n-transducers for the tree of Figure 2.1.

The complete evidence-expanded model is $G_0 = RG_1$. (In our tutorial example, the state graph of this transducer has too many transitions to show, but it is the configuration shown in Figure 2.2.)

The probability that the forward model F_0 generates the evidential sequences $\mathcal{S}$ is identical to the probability that the evidence-expanded model G_0 generates the empty string

$$P(\mathcal{S}|F_0) = \mathcal{W}(\epsilon : [F_0] : \mathcal{S}) = \mathcal{W}(\epsilon : [G_0] : \epsilon)$$

Note the astronomical number of states in G_0

$$|\Phi_{G_0}| \simeq \left(\prod_{n=1}^{\kappa} \text{length}(\mathcal{S}_n) \right) \left(\prod_{n=1}^{2\kappa-1} |\Phi_{B_n}| \right)$$

This is even worse than F_0; in fact, it is the same as the number of cells in the Inside matrix for computing $P(\mathcal{S}|F_0)$. The good news is we are about to start constraining it.

The constrained-expanded model (H_n, E_n, M_n, Q_n)

We now introduce a progressive series of approximating constraints to make inference under the model more tractable.

If $n \geq 1$ is a leaf node, define $H_n = \nabla(\mathcal{S}_n) \equiv G_n$. The number of states is $|\Phi_{H_n}| \simeq \text{length}(\mathcal{S}_n)$, just as with G_n.

Otherwise, let (l, r) denote the left and right child nodes, and define

$$H_n = (B_l E_l) \circ (B_r E_r)$$

where $\Phi_{E_n} \subseteq \Phi_{H_n}$.

We can represent H_n diagramatically as

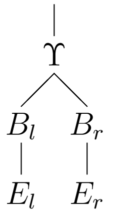


Figure 2.34, Figure 2.43 and Figure 2.44 are examples of H_n transducers.

Transducer E_n, which is what we mean by the "constrained-expanded model", is effectively a profile of sequences that might plausibly appear at node n, given the observed descendants of that node. Figure 2.38 and Figure 2.41 are examples of such transducers for the "intermediate sequence" in the tree of Figure 2.1. (Figure 2.37 and Figure 2.40 show the relationship to the corresponding H_n transducers).

The profile is constructed as follows.

The general idea is to generate a set of candidate sequences at node n, by sampling from the posterior distribution of such sequences **given only the descendants of node n,** ignoring (for the moment) the nodes outside the n-rooted subtree. To do this, we need to introduce a prior distribution over the sequence at node n. This prior is an approximation to replace the true (but as yet unknown) posterior distribution due to nodes outside the n-rooted subtree (including n's parent, and ancestors all the way back to the root, as well as siblings, cousins etc.)

A plausible choice for this prior, equivalent to assuming *stationarity* of the underlying evolutionary process, is the same prior that we use for the root node; that is, the generator model R. We therefore define

$$\begin{aligned} M_n &= RH_n \\ &= R((B_l E_l) \circ (B_r E_r)) \end{aligned}$$

We can represent M_n diagramatically as

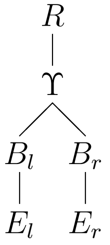
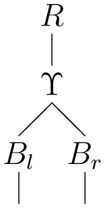


Figure 2.35 is an example of the M_n type of model.

The transducer $Q_n = R(B_l \circ B_r)$, which forms the comparison kernel of M_n, is also useful. It can be represented as



Conceptually, Q_n is a generator with two output tapes (i.e. a Pair HMM). These tapes are generated by sampling a sequence from the root generator R, making two copies of it, and feeding the two copies into B_l and B_r respectively. The outputs of B_l and B_r are the two outputs of the Pair HMM. The different states of Q_n encode information about how each output symbol originated (e.g. by root insertions that were then matched on the branches, *vs* insertions on the branches). Figure 2.47 shows an example of a Q_n-like transducer.

Transducer M_n can be thought of as the dynamic programming matrix that we get if we use the Pair HMM Q_n to align the recognition profiles E_l and E_r.

Component state tuples $(\rho, \upsilon, b_l, e_l, b_r, e_r)$

Suppose that $a \in \Phi_A, b \in \Phi_B, \upsilon \in \Phi_\Upsilon$. Our construction of composite transducers allows us to represent any state in $A \circ B = \Upsilon(A, B)$ as a tuple (υ, a, b). Similarly, any state in AB can be represented as (a, b). Each state in M_n can thus be written as a tuple $(\rho, \upsilon, b_l, e_l, b_r, e_r)$ of component states, where

- ρ is the state of the generator transducer R
- υ is the state of the bifurcation transducer Υ
- b_l is the state of the left-branch transducer B_l
- e_l is the state of the left child profile transducer E_l
- b_r is the state of the right-branch transducer B_l
- e_r is the state of the right child profile transducer E_r

Similarly, each state in H_n (and E_n) can be written as a tuple $(\upsilon, b_l, e_l, b_r, e_r)$.

Constructing E_n from H_n

The construction of E_n as a sub-model of H_n proceeds as follows:

1. sample a set of K paths from $P(\pi|M_n) = \mathcal{W}(\pi)/\mathcal{W}(\epsilon : [M_n] : \epsilon)$;
2. identify the set of M_n-states $\{(\rho, \upsilon, b_l, e_l, b_r, e_r)\}$ used by the sampled paths;
3. strip off the leading ρ's from these M_n-states to find the associated set of H_n-states $\{(\upsilon, b_l, e_l, b_r, e_r)\}$;
4. the set of H_n-states so constructed is the subset of E_n's states that have type D (wait states must be added to place it in strict normal form).

Here K plays the role of a bounding parameter. For the constrained-expanded transducer, $|\Phi_{E_n}| \simeq KL$, where $L = \max_n \text{length}(\mathcal{S}_n)$. Models H_n and M_n, however, contain $\mathcal{O}(b^2K^2L^2)$ states, where $b = \max_n |\Phi_{B_n}|$, as they are constructed by intersection of two $\mathcal{O}(bKL)$-state transducers (B_lE_l and B_rE_r).

Explicit construction of Q_n

$$
\begin{aligned}
Q_n &= R(B_l \circ B_r) \\
&= (\emptyset, (\cup \{\epsilon\})^2, \Phi_{Q_n}, \phi_{S;Q_n}, \phi_{E;Q_n}, \tau_{Q_n}, \mathcal{W}_{Q_n}) \\
\phi_{S;Q_n} &= (\phi_{S;R}, \phi_{S;\Upsilon}, \phi_{S;B_l}, \phi_{S;B_r}) \\
\phi_{E;Q_n} &= (\phi_{E;R}, \phi_{E;\Upsilon}, \phi_{E;B_l}, \phi_{E;B_r})
\end{aligned}
$$

States of Q_n

Define $\text{type}(\phi_1, \phi_2, \phi_3 \ldots) = (\text{type}(\phi_1), \text{type}(\phi_2), \text{type}(\phi_3) \ldots)$.

Let $q = (\rho, \upsilon, b_l, b_r) \in \Phi_{Q_n}$. We construct Φ_{Q_n} from classes, adopting the convention that each class of states is defined by its associated types:

$$\Phi_{\text{class}} = \{q : \text{type}(\rho, \upsilon, b_l, b_r) \in \mathcal{T}_{\text{class}}\}$$

The state typings are

$$\begin{aligned}
\mathcal{T}_{\text{match}} &= \{(I, M, M, M)\} \\
\mathcal{T}_{\text{right-del}} &= \{(I, M, M, D)\} \\
\mathcal{T}_{\text{left-del}} &= \{(I, M, D, M)\} \\
\mathcal{T}_{\text{null}} &= \{(I, M, D, D)\} \\
\mathcal{T}_{\text{right-ins}} &= \{(S, S, S, I),\ (I, M, M, I),\ (I, M, D, I)\} \\
\mathcal{T}_{\text{left-ins}} &= \{(S, S, I, W),\ (I, M, I, W)\} \\
\mathcal{T}_{\text{wait}} &= \{(W, W, W, W)\} \\
\mathcal{T}_{\text{right-emit}} &= \mathcal{T}_{\text{left-del}} \cup \mathcal{T}_{\text{right-ins}} \\
\mathcal{T}_{\text{left-emit}} &= \mathcal{T}_{\text{left-ins}} \cup \mathcal{T}_{\text{right-del}}
\end{aligned}$$

The state space of Q_n is

$$\Phi_{Q_n} = \{\phi_{S;Q_n},\ \phi_{E;Q_n}\} \cup \Phi_{\text{match}} \cup \Phi_{\text{left-emit}} \cup \Phi_{\text{right-emit}} \cup \Phi_{\text{null}} \cup \Phi_{\text{wait}}$$

It is possible to calculate transition and I/O weights of Q_n by starting with the example constructions given for TU and $T \circ U$, then eliminating states that are not in the above set. This gives the results described in the following sections.

I/O weights of Q_n

Let $(\omega_l, \omega_r) \in (\cup \{\epsilon\})^2$.

The I/O weight function for Q_n is

$$\mathcal{W}^{\text{emit}}_{Q_n}(\epsilon, (\omega_l, \omega_r), q) = \begin{cases}
\sum_{\omega \in \Omega} \mathcal{W}^{\text{emit}}_{R}(\epsilon, \omega, R) \mathcal{W}^{\text{emit}}_{B_l}(\omega, \omega_l, b_l) \mathcal{W}^{\text{emit}}_{B_r}(\omega, \omega_r, b_r) & \text{if } q \in \Phi_{\text{match}} \\
\sum_{\omega \in \Omega} \mathcal{W}^{\text{emit}}_{R}(\epsilon, \omega, R) \mathcal{W}^{\text{emit}}_{B_r}(\omega, \omega_r, b_r) & \text{if } q \in \Phi_{\text{left-del}} \\
\sum_{\omega \in \Omega} \mathcal{W}^{\text{emit}}_{R}(\epsilon, \omega, R) \mathcal{W}^{\text{emit}}_{B_l}(\omega, \omega_l, b_l) & \text{if } q \in \Phi_{\text{right-del}} \\
\mathcal{W}^{\text{emit}}_{B_l}(\epsilon, \omega_l, b_l) & \text{if } q \in \Phi_{\text{left-ins}} \\
\mathcal{W}^{\text{emit}}_{B_r}(\epsilon, \omega_r, b_r) & \text{if } q \in \Phi_{\text{right-ins}} \\
1 & \text{otherwise}
\end{cases}$$

Transition weights of Q_n

The transition weight between two states $q = (\rho, \upsilon, b_l, b_r)$ and $q' = (\rho', \upsilon', b_l', b_r')$ always takes the form

$$\mathcal{W}^{\text{trans}}_{Q_n}(q, q') \equiv \mathcal{W}^{\text{trans}}_{R}(\{\pi_R\}).\mathcal{W}^{\text{trans}}_{\Upsilon}(\{\pi_\Upsilon\}).\mathcal{W}^{\text{trans}}_{B_l}(\{\pi_{B_l}\}).\mathcal{W}^{\text{trans}}_{B_r}(\{\pi_{B_r}\})$$

where $\mathcal{W}^{\text{trans}}_{T}(\{\pi_T\})$ represents a sum over a set of paths through component transducer T. The allowed paths $\{\pi_T\}$ are constrained by the types of q, q' as shown in Table 2.5. Table 2.5 uses the following conventions:

- A 0 in any column means that the corresponding state must remain unchanged. For example, if $|\pi_{B_l}| = 0$ then

$$\mathcal{W}^{\text{trans}}_{B_l}(\{\pi_{B_l}\}) \equiv \delta(b_l = b_l')$$

- A 1 in any column means that the corresponding transducer makes a single transition. For example, if $|\pi_{B_l}| = 1$ then

$$\mathcal{W}^{\text{trans}}_{B_l}(\{\pi_{B_l}\}) \equiv \mathcal{W}^{\text{trans}}_{B_l}(b_l, b_l')$$

- A 2 in any column means that the corresponding transducer makes two transitions, via an intermediate state. For example, if $|\pi_{B_l}| = 2$ then

$$\mathcal{W}^{\text{trans}}_{B_l}(\{\pi_{B_l}\}) \equiv \sum_{b_l'' \in \Phi_{B_l}} \mathcal{W}^{\text{trans}}_{B_l}(b_l, b_l'')\mathcal{W}^{\text{trans}}_{B_l}(b_l'', b_l')$$

  (Since the transducers are in strict normal form, and given the context in which the 2's appear, it will always be the case that the intermediate state b_l'' has type W.)

- An asterisk ($*$) in a type-tuple is interpreted as a wildcard; for example, $(S, S, S, *)$ corresponds to $\{(S, S, S, S),\ (S, S, S, I)\}$.

- If a transition does not appear in the above table, or if any of the $\mathcal{W}^{\text{trans}}$'s are zero, then $\nexists(h, \omega, \omega', h') \in \tau_{H_n}$.

State types of Q_n

$$\text{type}(q) = \begin{cases} S & \text{if } q = \phi_{S;Q_n} \\ E & \text{if } q = \phi_{E;Q_n} \\ W & \text{if } q \in \Phi_{\text{wait}} \\ I & \text{if } q \in \Phi_{\text{match}} \cup \Phi_{\text{left-emit}} \cup \Phi_{\text{right-emit}} \\ N & \text{if } q \in \Phi_{\text{null}} \end{cases}$$

type$(\rho, \upsilon, b_l, b_r)$	type$(\rho', \upsilon', b_l', b_r')$	$\lvert\pi_R\rvert$	$\lvert\pi_\Upsilon\rvert$	$\lvert\pi_{B_l}\rvert$	$\lvert\pi_{B_r}\rvert$
$(S, S, S, *)$	(S, S, S, I)	0	0	0	1
	(S, S, I, W)	0	0	1	1
	(I, M, M, M)	1	2	2	2
	(I, M, M, D)	1	2	2	2
	(I, M, D, M)	1	2	2	2
	(I, M, D, D)	1	2	2	2
	(W, W, W, W)	1	1	1	1
(S, S, I, W)	(S, S, I, W)	0	0	1	0
	(I, M, M, M)	1	2	2	1
	(I, M, M, D)	1	2	2	1
	(I, M, D, M)	1	2	2	1
	(I, M, D, D)	1	2	2	1
	(W, W, W, W)	1	1	1	0
$(I, M, M, *)$	(I, M, M, I)	0	0	0	1
	(I, M, I, W)	0	0	1	1
	(I, M, M, M)	1	2	2	2
	(I, M, M, D)	1	2	2	2
	(I, M, D, M)	1	2	2	2
	(I, M, D, D)	1	2	2	2
	(W, W, W, W)	1	1	1	1
$(I, M, D, *)$	(I, M, D, I)	0	0	0	1
	(I, M, I, W)	0	0	1	1
	(I, M, M, M)	1	2	2	2
	(I, M, M, D)	1	2	2	2
	(I, M, D, M)	1	2	2	2
	(I, M, D, D)	1	2	2	2
	(W, W, W, W)	1	1	1	1
(I, M, I, W)	(I, M, I, W)	0	0	1	0
	(I, M, M, M)	1	2	2	1
	(I, M, M, D)	1	2	2	1
	(I, M, D, M)	1	2	2	1
	(I, M, D, D)	1	2	2	1
	(W, W, W, W)	1	1	1	0
(W, W, W, W)	(E, E, E, E)	1	1	1	1

Table 2.5: Transition types of Q_n, the transducer described in Section 2.3 This transducer requires its input to be empty: it is 'generative'. It jointly models a parent sequence (hidden) and a pair of sibling sequences (outputs), and is somewhat analogous to a Pair HMM. It is used during progressive reconstruction.

Note that Q_n contains null states (Φ_{null}) corresponding to coincident deletions on branches $n \to l$ and $n \to r$. These states have $\text{type}(\rho, \upsilon, b_l, b_r) = (I, M, D, D)$. There are transition paths that go through these states, including paths that cycle indefinitely among these states.

We need to eliminate these states before constructing M_n. Let $Q'_n \equiv Q_n$ denote the transducer obtained from Q_n by marginalizing Φ_{null}

$$\Phi_{Q'_n} = \{\phi_{S;Q_n},\ \phi_{E;Q_n}\} \cup \Phi_{\text{match}} \cup \Phi_{\text{left-emit}} \cup \Phi_{\text{right-emit}} \cup \Phi_{\text{wait}}$$

The question arises, how to restore these states when constructing E_n? Ortheus samples them randomly, but (empirically) a lot of samples are needed before there is any chance of guessing the right number, and in practice it makes little difference to the accuracy of the reconstruction. In principle it might be possible to leave them in as self-looping delete states in E_n, but this would make E_n cyclic.

Explicit construction of M_n using Q'_n, E_l and E_r

Refer to the previous section for definitions pertaining to Q'_n.

$$\begin{aligned} M_n &= R((B_l E_l) \circ (B_r E_r)) \\ Q'_n &\equiv R(B_l \circ B_r) \end{aligned}$$

States of M_n

The complete set of M_n-states is

$$\begin{aligned} \Phi_{M_n} = \ & \{\phi_{S;M_n},\ \phi_{E;M_n}\} \\ & \cup \{(\rho, \upsilon, b_l, e_l, b_r, e_r) : (\rho, \upsilon, b_l, b_r) \in \Phi_{\text{match}},\ \text{type}(e_l) = \text{type}(e_r) = D\} \\ & \cup \{(\rho, \upsilon, b_l, e_l, b_r, e_r) : (\rho, \upsilon, b_l, b_r) \in \Phi_{\text{left-emit}},\ \text{type}(e_l) = D, \text{type}(e_r) = W\} \\ & \cup \{(\rho, \upsilon, b_l, e_l, b_r, e_r) : (\rho, \upsilon, b_l, b_r) \in \Phi_{\text{right-emit}},\ \text{type}(e_l) = W, \text{type}(e_r) = D\} \\ & \cup \{(\rho, \upsilon, b_l, e_l, b_r, e_r) : (\rho, \upsilon, b_l, b_r) \in \Phi_{\text{wait}},\ \text{type}(e_l) = \text{type}(e_r) = W\} \end{aligned}$$

I/O weights of M_n

Let $m = (\rho, \upsilon, b_l, e_l, b_r, e_r)$ be an M_n-state and $q = (\rho, \upsilon, b_l, b_r)$ the subsumed Q'_n-state.

Similarly, let $m' = (\rho', \upsilon', b'_l, e'_l, b'_r, e'_r)$ and $q' = (\rho', \upsilon', b'_l, b'_r)$.

The I/O weight function for M_n is

$$\mathcal{W}^{\text{emit}}_{M_n}(\epsilon, \epsilon, m) = \begin{cases} \displaystyle\sum_{\omega_l \in \Omega} \sum_{\omega_r \in \Omega} \mathcal{W}^{\text{emit}}_{Q'_n}(\epsilon, (\omega_l, \omega_r), q).\mathcal{W}^{\text{emit}}_{E_l}(\omega_l, \epsilon, e_l).\mathcal{W}^{\text{emit}}_{E_r}(\omega_r, \epsilon, e_r) & \text{if } q \in \Phi_{\text{match}} \\ \displaystyle\sum_{\omega_l \in \Omega} \mathcal{W}^{\text{emit}}_{Q'_n}(\epsilon, (\omega_l, \epsilon), q).\mathcal{W}^{\text{emit}}_{E_l}(\omega_l, \epsilon, e_l) & \text{if } q \in \Phi_{\text{left-emit}} \\ \displaystyle\sum_{\omega_r \in \Omega} \mathcal{W}^{\text{emit}}_{Q'_n}(\epsilon, (\epsilon, \omega_r), q).\mathcal{W}^{\text{emit}}_{E_r}(\omega_r, \epsilon, e_r) & \text{if } q \in \Phi_{\text{right-emit}} \\ 1 & \text{otherwise} \end{cases}$$

Transitions of M_n

As before,

$$\begin{aligned} q &= (\rho, \upsilon, b_l, b_r) \\ m &= (\rho, \upsilon, b_l, e_l, b_r, e_r) \\ q' &= (\rho', \upsilon', b_l', b_r') \\ m' &= (\rho', \upsilon', b_l', e_l', b_r', e_r') \end{aligned}$$

An "upper bound" (i.e. superset) of the transition set of M_n is as follows

$$\begin{aligned} \tau_{M_n} \subseteq\ & \{(m, \epsilon, \epsilon, m') : q' \in \Phi_{\text{match}}, \text{type}(q) \in \{S, I\}, \text{type}(e_l', e_r') = (D, D)\} \\ & \cup \{(m, \epsilon, \epsilon, m') : q' \in \Phi_{\text{left-emit}}, \text{type}(q) \in \{S, I\}, \text{type}(e_l', e_r') = (D, W)\} \\ & \cup \{(m, \epsilon, \epsilon, m') : q' \in \Phi_{\text{right-emit}}, \text{type}(q) \in \{S, I\}, \text{type}(e_l', e_r') = (W, D)\} \\ & \cup \{(m, \epsilon, \epsilon, m') : q' \in \Phi_{\text{wait}}, \text{type}(q) \in \{S, I\}, \text{type}(e_l', e_r') = (W, W)\} \\ & \cup \{(m, \epsilon, \epsilon, m') : \text{type}(q, e_l, e_r) = (W, W, W), \text{type}(q', e_l', e_r') = (E, E, E)\} \end{aligned}$$

More precisely, τ_{M_n} contains the transitions in the above set for which the transition weight (defined in the next section) is nonzero. (This ensures that the individual transition paths $q \to q'$, $e_l \to e_l'$ and $e_r \to e_r'$ exist with nonzero weight.)

Transition weights of M_n

Let $\mathcal{W}^{\text{via-wait}}_{E_n}(e, e')$ be the weight of **either** the direct transition $e \to e'$, **or** a double transition $e \to e'' \to e'$ summed over all intermediate states e''

$$\mathcal{W}^{\text{via-wait}}_{E_n}(e, e') = \begin{cases} \sum_{e'' \in \Phi_{E_n}} \mathcal{W}^{\text{trans}}_{E_n}(e, e'') \mathcal{W}^{\text{trans}}_{E_n}(e'', e') & \text{if type}(e) \in \{S, D\} \\ \mathcal{W}^{\text{trans}}_{E_n}(e, e') & \text{if type}(e) = W \end{cases}$$

Let $\mathcal{W}^{\text{to-wait}}_{E_n}(e, e')$ be the weight of a transition (or non-transition) that leaves E_n in a wait state

$$\mathcal{W}^{\text{to-wait}}_{E_n}(e, e') = \begin{cases} \mathcal{W}^{\text{trans}}_{E_n}(e, e') & \text{if type}(e) \in \{S, D\},\ \text{type}(e') = W \\ 1 & \text{if } e = e',\ \text{type}(e') = W \\ 0 & \text{otherwise} \end{cases}$$

The transition weight function for M_n is

$$\mathcal{W}^{\text{trans}}_{M_n}(m, m') = \mathcal{W}^{\text{trans}}_{Q_n'}(q, q') \times \begin{cases} \mathcal{W}^{\text{via-wait}}_{E_l}(e_l, e_l') \mathcal{W}^{\text{via-wait}}_{E_r}(e_r, e_r') & \text{if } q' \in \Phi_{\text{match}} \\ \mathcal{W}^{\text{via-wait}}_{E_l}(e_l, e_l') \mathcal{W}^{\text{to-wait}}_{E_r}(e_r, e_r') & \text{if } q' \in \Phi_{\text{left-emit}} \\ \mathcal{W}^{\text{to-wait}}_{E_l}(e_l, e_l') \mathcal{W}^{\text{via-wait}}_{E_r}(e_r, e_r') & \text{if } q' \in \Phi_{\text{right-emit}} \\ \mathcal{W}^{\text{to-wait}}_{E_l}(e_l, e_l') \mathcal{W}^{\text{to-wait}}_{E_r}(e_r, e_r') & \text{if } q' \in \Phi_{\text{wait}} \\ \mathcal{W}^{\text{trans}}_{E_l}(e_l, e_l') \mathcal{W}^{\text{trans}}_{E_r}(e_r, e_r') & \text{if } q' = \phi_{E;Q_n'} \end{cases}$$

Explicit construction of H_n

This construction is somewhat redundant, since we construct M_n from Q_n, E_l and E_r, rather than from RH_n. It is retained for comparison.

$$\begin{aligned} H_n &= (B_l E_l) \circ (B_r E_r) \\ &= (\Omega, \emptyset, \Phi_{H_n}, \phi_{S;H_n}, \phi_{E;H_n}, \tau_{H_n}, \mathcal{W}_{H_n}) \end{aligned}$$

Assume B_l, B_r, E_l, E_r in strict-normal form.

States of H_n

Define $\text{type}(\phi_1, \phi_2, \phi_3 \ldots) = (\text{type}(\phi_1), \text{type}(\phi_2), \text{type}(\phi_3) \ldots)$.

Let $h = (\upsilon, b_l, e_l, b_r, e_r) \in \Phi_{H_n}$. We construct Φ_{H_n} from classes, adopting the convention that each class of states is defined by its associated types:

$$\Phi_{\text{class}} = \{h : \text{type}(\upsilon, b_l, e_l, b_r, e_r) \in \mathcal{T}_{\text{class}}\}$$

Define $\Phi_{\text{ext}} \subset \Phi_{H_n}$ to be the subset of H_n-states that follow *externally-driven cascades*

$$\begin{aligned} \mathcal{T}_{\text{ext}} &= \{(M, M, D, M, D),\ (M, M, D, D, W), \\ &\qquad (M, D, W, M, D),\ (M, D, W, D, W)\} \end{aligned}$$

Define $\Phi_{\text{int}} \subset \Phi_{H_n}$ to be the subset of H_n-states that follow *internal cascades*

$$\begin{aligned} \mathcal{T}_{\text{int}} &= \mathcal{T}_{\text{left-int}} \cup \mathcal{T}_{\text{right-int}} \\ \mathcal{T}_{\text{left-int}} &= \{(S, I, D, W, W),\ (M, I, D, W, W)\} \\ \mathcal{T}_{\text{right-int}} &= \{(S, S, S, I, D),\ (M, M, D, I, D),\ (M, D, W, I, D)\} \end{aligned}$$

Remaining states are the start, end, and wait states:

$$\begin{aligned} \phi_{S;H_n} &= (\phi_{S;\Upsilon}, \phi_{S;B_l}, \phi_{S;E_l}, \phi_{S;B_r}, \phi_{S;E_r}) \\ \phi_{E;H_n} &= (\phi_{E;\Upsilon}, \phi_{E;B_l}, \phi_{E;E_l}, \phi_{E;B_r}, \phi_{E;E_r}) \\ \mathcal{T}_{\text{wait}} &= \{(W, W, W, W, W)\} \end{aligned}$$

The complete set of H_n-states is

$$\Phi_{H_n} = \{\phi_{S;H_n},\ \phi_{E;H_n}\} \ \cup\ \Phi_{\text{ext}}\ \cup\ \Phi_{\text{int}}\ \cup\ \Phi_{\text{wait}}$$

It is possible to calculate transition and I/O weights of H_n by starting with the example constructions given for TU and $T \circ U$, then eliminating states that are not in the above set. This gives the results described in the following sections.

I/O weights of H_n

Let $\omega, \omega' \in (\cup \{\epsilon\})$.

Let $C_n(b_n, e_n)$ be the I/O weight function for $B_n E_n$ on a transition into composite state (b_n, e_n) where type$(b_n, e_n) = (I, D)$

$$C_n(b_n, e_n) = \sum_{\omega \in \Omega} \mathcal{W}^{\text{emit}}_{B_n}(\epsilon, \omega, b_n) \mathcal{W}^{\text{emit}}_{E_n}(\omega, \epsilon, e_n)$$

Let $D_n(\omega, b_n, e_n)$ be the I/O weight function for $B_n E_n$ on a transition into composite state (b_n, e_n) where type$(b_n, e_n) \in \{(M, D), (D, W)\}$ with input symbol ω

$$D_n(\omega, b_n, e_n) = \begin{cases} \sum_{\omega' \in \Omega} \mathcal{W}^{\text{emit}}_{B_n}(\omega, \omega', b_n) \mathcal{W}^{\text{emit}}_{E_n}(\omega', \epsilon, e_n) & \text{if type}(b_n, e_n) = (M, D) \\ \mathcal{W}^{\text{emit}}_{B_n}(\omega, \epsilon, b_n) & \text{if type}(b_n, e_n) = (D, W) \end{cases}$$

The I/O weight function for H_n is

$$\mathcal{W}^{\text{emit}}_{H_n}(\omega, \epsilon, h) = \begin{cases} D_l(\omega, b_l, e_l) D_r(\omega, b_r, e_r) & \text{if } h \in \Phi_{\text{ext}} \\ C_l(b_l, e_l) & \text{if } h \in \Phi_{\text{left-int}} \\ C_r(b_r, e_r) & \text{if } h \in \Phi_{\text{right-int}} \\ 1 & \text{otherwise} \end{cases}$$

Transition weights of H_n

The transition weight between two states $h = (\upsilon, b_l, e_l, b_r, e_r)$ and $h' = (\upsilon', b'_l, e'_l, b'_r, e'_r)$ always takes the form

$$\mathcal{W}^{\text{trans}}_{H_n}(h, h') \equiv \mathcal{W}^{\text{trans}}_{\Upsilon}(\{\pi_{\Upsilon}\}).\mathcal{W}^{\text{trans}}_{B_l}(\{\pi_{B_l}\}).\mathcal{W}^{\text{trans}}_{E_l}(\{\pi_{E_l}\}).\mathcal{W}^{\text{trans}}_{B_r}(\{\pi_{B_r}\}).\mathcal{W}^{\text{trans}}_{E_r}(\{\pi_{E_r}\})$$

where the RHS terms again represent sums over paths, with the allowed paths depending on the types of h, h' as shown in Table 2.6. Table 2.6 uses the same conventions as Table 2.5.

State types of H_n

$$\text{type}(h) = \begin{cases} S & \text{if } h = \phi_{S;H_n} \\ E & \text{if } h = \phi_{E;H_n} \\ W & \text{if } h \in \Phi_{\text{wait}} \\ D & \text{if } h \in \Phi_{\text{ext}} \\ N & \text{if } h \in \Phi_{\text{int}} \end{cases}$$

Since H_n contains states of type N (the internal cascades), it is necessary to eliminate these states from E_n (after sampling paths through M_n), so as to guarantee that E_n will be in strict normal form.

type$(\upsilon, b_l, e_l, b_r, e_r)$	type$(\upsilon', b_l', e_l', b_r', e_r')$	$\lvert\pi_\Upsilon\rvert$	$\lvert\pi_{B_l}\rvert$	$\lvert\pi_{E_l}\rvert$	$\lvert\pi_{B_r}\rvert$	$\lvert\pi_{E_r}\rvert$
(S,S,S,S,S)	(S,S,S,I,D)	0	0	0	1	2
	(S,I,D,W,W)	0	1	2	1	1
	(W,W,W,W,W)	1	1	1	1	1
(S,S,S,I,D)	(S,S,S,I,D)	0	0	0	1	2
	(S,I,D,W,W)	0	1	2	1	1
	(W,W,W,W,W)	1	1	1	1	1
(S,I,D,W,W)	(S,I,D,W,W)	0	1	2	0	0
	(W,W,W,W,W)	1	1	1	0	0
(W,W,W,W,W)	(M,M,D,M,D)	1	1	1	1	1
	(M,M,D,D,W)	1	1	1	1	0
	(M,D,W,M,D)	1	1	0	1	1
	(M,D,W,D,W)	1	1	0	1	0
	(E,E,E,E,E)	1	1	1	1	1
(M,M,D,M,D)	(M,M,D,I,D)	0	0	0	1	2
	(M,I,D,W,W)	0	1	2	1	1
	(W,W,W,W,W)	1	1	1	1	1
(M,M,D,D,W)	(M,M,D,I,D)	0	0	0	1	1
	(M,I,D,W,W)	0	1	2	1	0
	(W,W,W,W,W)	1	1	1	1	0
(M,D,W,M,D)	(M,D,W,I,D)	0	0	0	1	2
	(M,I,D,W,W)	0	1	1	1	1
	(W,W,W,W,W)	1	1	0	1	1
(M,D,W,D,W)	(M,D,W,I,D)	0	0	0	1	1
	(M,I,D,W,W)	0	1	1	1	0
	(W,W,W,W,W)	1	1	0	1	0
(M,M,D,I,D)	(M,M,D,I,D)	0	0	0	1	2
	(M,I,D,W,W)	0	1	2	1	1
	(W,W,W,W,W)	1	1	1	1	1
(M,D,W,I,D)	(M,D,W,I,D)	0	0	0	1	2
	(M,I,D,W,W)	0	1	1	1	1
	(W,W,W,W,W)	1	1	0	1	1
(M,I,D,W,W)	(M,I,D,W,W)	0	1	2	0	0
	(W,W,W,W,W)	1	1	1	0	0

Table 2.6: Transition types of H_n, the transducer described in Section 2.3 This transducer requires non-empty input: it is a 'recognizing profile' or 'recognizer'. It models a subtree of sequences conditional on an absorbed parental sequence. It is used during progressive reconstruction.

Explicit construction of M_n using R and H_n

This construction is somewhat redundant, since we construct M_n from Q_n, E_l and E_r, rather than from RH_n. It is retained for comparison.

The following construction uses the fact that $M_n = RH_n$ so that we can compactly define M_n by referring back to the previous construction of H_n. In practice, it will be more efficient to precompute $Q_n = R(B_l \circ B_r)$.

Refer to the previous section ("Explicit construction of H_n") for definitions of $\Phi_{\text{ext}}, \Phi_{\text{int}}, \Phi_{\text{wait}}, \mathcal{W}^{\text{trans}}_{H_n}($

Assume that R is in strict normal form.

$$\begin{aligned}
M_n &= RH_n \\
&= R((B_l E_l) \circ (B_r E_r)) \\
&= (\emptyset, \emptyset, \Phi_{M_n}, \phi_{S;M_n}, \phi_{E;M_n}, \tau_{M_n}, \mathcal{W}_{M_n}) \\
\phi_{S;M_n} &= (\phi_{S;R}, \phi_{S;\Upsilon}, \phi_{S;B_l}, \phi_{S;E_l}, \phi_{S;B_r}, \phi_{S;E_r}) \\
\phi_{E;M_n} &= (\phi_{E;R}, \phi_{E;\Upsilon}, \phi_{E;B_l}, \phi_{E;E_l}, \phi_{E;B_r}, \phi_{E;E_r})
\end{aligned}$$

States of M_n

The complete set of M_n-states is

$$\begin{aligned}
\Phi_{M_n} = \; & \{\phi_{S;M_n},\ \phi_{E;M_n}\} \\
& \cup \{(\rho, \upsilon, b_l, e_l, b_r, e_r) : (\upsilon, b_l, e_l, b_r, e_r) \in \Phi_{\text{ext}},\ \text{type}(\rho) = I\} \\
& \cup \{(\rho, \upsilon, b_l, e_l, b_r, e_r) : (\upsilon, b_l, e_l, b_r, e_r) \in \Phi_{\text{wait}},\ \text{type}(\rho) = W\} \\
& \cup \{(\rho, \upsilon, b_l, e_l, b_r, e_r) : (\upsilon, b_l, e_l, b_r, e_r) \in \Phi_{\text{int}},\ \text{type}(\rho) = \text{type}(\upsilon) = S\} \\
& \cup \{(\rho, \upsilon, b_l, e_l, b_r, e_r) : (\upsilon, b_l, e_l, b_r, e_r) \in \Phi_{\text{int}},\ \text{type}(\rho) = I,\ \text{type}(\upsilon) = M\}
\end{aligned}$$

I/O weights of M_n

Let $m = (\rho, \upsilon, b_l, e_l, b_r, e_r)$ be an M_n-state and $h = (\upsilon, b_l, e_l, b_r, e_r)$ the subsumed H_n-state.

Similarly, let $m' = (\rho', \upsilon', b'_l, e'_l, b'_r, e'_r)$ and $h' = (\upsilon', b'_l, e'_l, b'_r, e'_r)$.

The I/O weight function for M_n is

$$\mathcal{W}^{\text{emit}}_{M_n}(\epsilon, \epsilon, m) = \begin{cases} \displaystyle\sum_{\omega \in \Omega} \mathcal{W}^{\text{emit}}_{R}(\epsilon, \omega, \rho) \mathcal{W}^{\text{emit}}_{H_n}(\omega, \epsilon, h) & \text{if } h \in \Phi_{\text{ext}} \\ \mathcal{W}^{\text{emit}}_{H_n}(\epsilon, \epsilon, h) & \text{otherwise} \end{cases}$$

Transition weights of M_n

The transition weight function for M_n is

$$\mathcal{W}^{\mathrm{trans}}_{M_n}(m,m') = \begin{cases} \mathcal{W}^{\mathrm{trans}}_{H_n}(h,h') & \text{if } h' \in \Phi_{\mathrm{int}} \\ \mathcal{W}^{\mathrm{trans}}_{R}(\rho,\rho') \sum_{h'' \in \Phi_{\mathrm{wait}}} \mathcal{W}^{\mathrm{trans}}_{H_n}(h,h'')\mathcal{W}^{\mathrm{trans}}_{H_n}(h'',h') & \text{if } h' \in \Phi_{\mathrm{ext}} \\ \mathcal{W}^{\mathrm{trans}}_{R}(\rho,\rho')\mathcal{W}^{\mathrm{trans}}_{H_n}(h,h') & \text{otherwise} \end{cases}$$

If E_l and E_r are acyclic, then H_n and E_n will be acyclic too. However, M_n does contain cycles among states of type (I, M, D, W, D, W). These correspond to characters output by R that are then deleted by both B_l and B_r. It is necessary to eliminate these states from M_n by marginalization, and to then restore them probabilistically when sampling paths through M_n.

Dynamic programming algorithms

The recursion for $\mathcal{W}(\epsilon : [M_n] : \epsilon)$ is

$$\begin{aligned} \mathcal{W}(\epsilon : [M_n] : \epsilon) &= Z(\phi_E) \\ Z(m') &= \sum_{m:(m,\epsilon,\epsilon,m') \in \tau} Z(m)\mathcal{W}(m,\epsilon,\epsilon,m') \qquad \forall m' \neq \phi_S \\ Z(\phi_S) &= 1 \end{aligned}$$

The algorithm to fill $Z(m)$ has the general structure shown in Algorithm 1. (Some optimization of this algorithm is desirable, since not all tuples $(\rho, \upsilon, b_l, e_l, b_r, e_r)$ are states of M_n. If E_n is in strict-normal form its W- and D-states will occur in pairs (c.f. the strict-normal version of the exact-match transducer $\nabla(S)$). These (D, W) pairs are largely redundant: the choice between D and W is dictated by the parent B_n, as can be seen from Table 2.6 and the construction of Φ_{H_n}.)

Time complexity The skeleton structure of Algorithm 1 is three nested loops, over Φ_{E_l}, Φ_{E_r}, and Φ_{Q_n}. The state spaces Φ_{E_l}, Φ_{E_r}, and Φ_{Q_n} are independent of each other, and so Algorithm 1 has time complexity $\mathcal{O}(|\Phi_{E_l}||\Phi_{E_l}||\Phi_{Q_n}|t_{Z(m)})$, where $t_{Z(m)}$ is the time required to compute $Z(m)$ for a given $m \in M_n$.

The quantities $|\Phi_{E_*}|$ can be bounded by a user-specified constant p by terminating stochastic sampling such that $|\Phi_{E_*}| \leq p$ as described in Section 2.3. Φ_{Q_n} is comprised of pairs of states from transducers B_l and B_r, (detailed in Section 2.3), and so it has size $\mathcal{O}(|\Phi_{B_l}||\Phi_{B_r}|)$. Computing $Z(m)$ (outlined in Algorithm 4) requires summing over all incoming states, so $t_{Z(m)}$ has time complexity $\mathcal{O}(|m : (m, \epsilon, \epsilon, m') \in \tau|)$. In typical cases, this set will be small (e.g. a linear profile will have exactly one incoming transition per state), though the worst-case size is $\mathcal{O}(p)$. If we assume the same branch transducer B is used throughout, the full forward recursion has worst-case time complexity $\mathcal{O}(|\Phi_B|^2 p^3)$.

```
Initialize $Z(\phi_S) \leftarrow 1$;
foreach $e_l \in \Phi_{E_l}$ do                          /* topologically-sorted */
    foreach $e_r \in \Phi_{E_r}$ do                      /* topologically-sorted */
        foreach $(\rho, \upsilon, b_l, b_r) \in \Phi_{Q_n}$ do    /* topologically-sorted */
            Let $m = (\rho, \upsilon, b_l, e_l, b_r, e_r)$;
            if $m \in \Phi_{M_n}$ then
                Compute $Z(m)$;
Return $Z(\phi_E)$.
```

Algorithm 1: The analog of the Forward algorithm for transducer M_n, described in Section 2.3. This is used during progressive reconstruction to store the sum-over-paths likelihood up to each state in Φ_{M_n}. The value of $Z(\phi_{E;)})$ is the likelihood of sequences descended from node n.

For comparison, the Forward algorithm for computing the probability of two sequences (S_l, S_r) being generated by a Pair Hidden Markov Model (M) has the general structure shown in Algorithm 2.

```
Initialize cell $(0, 0, \mathrm{START})$;
foreach $0 \leq i_l \leq length(S_l)$ do                 /* ascending order */
    foreach $0 \leq i_r \leq length(S_r)$ do             /* ascending order */
        foreach $\sigma \in M$ do                        /* topologically-sorted */
            Compute the sum-over-paths up to cell $(i_l, i_r, \sigma)$;
Return cell $(\mathrm{length}(S_l), \mathrm{length}(S_r), \mathrm{END})$.
```

Algorithm 2: The general form of the Forward algorithm for computing the joint probability of two sequences generated by the model M, a Pair HMM.

The generative transducer $Q_n \equiv R(B_l \circ B_r)$ in Algorithm 1 is effectively identical to the Pair HMM in Algorithm 2.

Pseudocode for DP recursion

We outline a more precise version of the Forward-like DP recursion in Algorithm 3 and the associated Function *sum_paths_to*. Let get_state_type$(q, side)$ return the state type for the profile on *side* which is consistent with q.

Transition sets

Since all transitions in the state spaces Q'_n, E_l, and E_r are known, we can define the following sets :

$$\begin{aligned}
\text{incoming\_left\_profile\_indices}(j) &= \{i : t_l(i,j) \neq 0\} \\
\text{incoming\_right\_profile\_indices}(j) &= \{i : t_r(i,j) \neq 0\} \\
\text{incoming\_match\_states}(q') &= \{q : q \in \Phi_{\text{match}}, \mathcal{W}^{\text{trans}}_{Q'_n}(q,q') \neq 0\} \\
\text{incoming\_left\_emit\_states}(q') &= \{q : q \in \Phi_{\text{left-emit}}, \mathcal{W}^{\text{trans}}_{Q'_n}(q,q') \neq 0\} \\
\text{incoming\_right\_emit\_states}(q') &= \{q : q \in \Phi_{\text{right-emit}}, \mathcal{W}^{\text{trans}}_{Q'_n}(q,q') \neq 0\}
\end{aligned}$$

Traceback

Sampling a path from $P(\pi|M_n)$ is analogous to stochastic traceback through the Forward matrix. The basic traceback algorithm is presented in Algorithm 5, and a more precise version is presented in Algorithm 6.

Let the function sample(set, $weights$) input two equal-length vectors and return a randomly-chosen element of set, such that set_i is sampled with probability $\frac{weights_i}{sum(weights)}$. A state path with two sampled paths is shown in Figure 2.40.

Alternative sampling schemes The above stochastic sampling strategy was chosen for its ease of presentation and implementation, but our approach is sufficiently general to allow any algorithm which selects a subset of complete paths through M_n. This selection may be by random (as ours is) or deterministic means. Randomized algorithms are widespread in computer science [62], though deterministic algorithms may be easier to analyze mathematically.

For instance, if an analog to the backward algorithm for HMMs was developed for the state space of M_n (e.g. Algorithm 3 reversed), we could select a set of states according to their posterior probability (e.g. the n states with highest posterior probability), and determine the most likely paths (via Viterbi paths) from start to end which include these states. Alternatively, a decision theory-based "optimal accuracy" approach could be used to optimize the total posterior probability of the selected states. These approaches require an additional dynamic programming recursion (the backward algorithm) at each step, and we suspect the improvement in accuracy may be minimal in the limit of sampling many paths. Empirically, we have observed that sampling paths is very fast compared to filling a DP matrix, and so we have focused our attention on the outlined stochastic approach.

Alignment envelopes

Note that states $e \in \Phi_{E_n}$ of the constrained-expanded model, as with states $g \in \Phi_{G_n}$ of the expanded model, can be associated with a vector of subsequence co-ordinates (one

```
Initialize $Z(\phi_S) \leftarrow 1$;
foreach $1 \leq i'_r \leq N_r$ do
    foreach $q' \in \{q : type(q) = (S, S, S, I)\}$ do
        Let $(e'_l, e'_r) = (\phi_S, \phi_D^{(i'_r)})$;
        $sum\_paths\_to(q', e'_l, e'_r)$;
foreach $1 \leq i'_l \leq N_l$ do
    foreach  $1 \leq i'_r \leq N_r$ do
        if $is\_in\_envelope(i'_l, i'_r)$ then
            foreach $q' \in \Phi_{match}$ do
                Let $(e'_l, e'_r) = (\phi_D^{(i'_l)}, \phi_D^{(i'_r)})$;
                $sum\_paths\_to(q', e'_l, e'_r)$;
            foreach $q' \in \Phi_{left\text{-}emit}$ do
                Let $(e'_l, e'_r) = (\phi_D^{(i'_l)}, \phi_W^{(i'_r)})$;
                $sum\_paths\_to(q', e'_l, e'_r)$;
            foreach $q' \in \Phi_{right\text{-}emit}$ do
                if $type(q') == (S, S, S, I)$ then
                    continue
                Let $\tau =$ get_state_type$(q', left)$;
                $(e'_l, e'_r) = (\phi_\tau^{(i'_l)}, \phi_D^{(i'_r)})$;
                $sum\_paths\_to(q', e'_l, e'_r)$;
            foreach $q' \in \Phi_{wait}$ do
                Let $(e'_l, e'_r) = (\phi_W^{(i'_l)}, \phi_W^{(i'_r)})$;
                $sum\_paths\_to(q', e'_l, e'_r)$;
foreach $1 \leq i'_l \leq N_l$ do
    foreach $q' \in \Phi_{left\text{-}emit}$ do
        Let $(e'_l, e'_r) = (\phi_D^{(i'_l)}, \phi_W^{(\mathrm{end})})$;
        $sum\_paths\_to(q', e'_l, e'_r)$;
foreach  $1 \leq i'_r \leq N_r$ do
    foreach $q' \in \Phi_{right\text{-}emit}$ do
        Let $(e'_l, e'_r) = (\phi_W^{(\mathrm{end})}, \phi_D^{(i'_r)})$;
        $sum\_paths\_to(q', e'_l, e'_r)$;
foreach $q' \in \Phi_{wait}$ do
    Let $(e'_l, e'_r) = (\phi_W^{(\mathrm{end})}, \phi_W^{(\mathrm{end})})$;
    $sum\_paths\_to(q', e'_l, e'_r)$;
Initialize $Z(\phi_E) \leftarrow 0$;
foreach $q \in \Phi_{wait}$ do
    Let $m = (q, \phi_W^{(\mathrm{end})}, \phi_W^{(\mathrm{end})})$;
    $Z(\phi_E) \leftarrow Z(\phi_E) + Z(m)\mathcal{W}_{Q'_n}^{\mathrm{trans}}(q, \phi_{E;Q'_n})$;
```

Algorithm 3: The full version of the analog of the Forward algorithm for transducer M_n, described in Section 2.3. This to visit each state in Φ_{M_n} in the proper order, storing the sum-over-paths likelihood up to that state using sum_paths_to(...) (defined separately). The value of $Z(\phi_{E;)}$ is the likelihood of sequences descended from node n.

```
Input: $(q', e'_l, e'_r)$.
Result: The cell in $Z$ for $m = (q', e'_l, e'_r)$ is filled.

Let $m' = (q', e'_l, e'_r)$;
Let $\mathcal{E} = \mathcal{W}^{\mathrm{emit}}_{M_n}(\epsilon, \epsilon, m')$;
Initialize $Z(m') \leftarrow \mathcal{W}^{\mathrm{trans}}_{Q'_n}(\phi_S, q')t_l(0, i'_l)t_r(0, i'_r)\mathcal{E}$;
foreach $i_l \in$ incoming_left_profile_indices($i'_l$) do
    foreach $i_r \in$ incoming_right_profile_indices($i'_r$) do
        Let $(e_l, e_r) = (\phi_D^{(i_l)}, \phi_D^{(i_r)})$;
        foreach $q \in$ incoming_match_states($q'$) do
            Let $m = (q, e_l, e_r)$;
            $Z(m') \leftarrow Z(m') + Z(m)\mathcal{W}^{\mathrm{trans}}_{Q'_n}(q, q')t_l(i_l, i'_l)t_r(i_r, i'_r)\mathcal{E}$
foreach $i_l \in$ incoming_left_profile_indices($i'_l$) do
    foreach $q \in$ incoming_left_emit_states($q'$) do
        Let $(e_l, e_r) = (\phi_D^{(i_l)}, \phi_W^{(i'_r)})$;
        Let $m = (q, e_l, e_r)$;
        $Z(m') \leftarrow Z(m') + Z(m)\mathcal{W}^{\mathrm{trans}}_{Q'_n}(q, q')t_l(i_l, i'_l)\mathcal{E}$;
foreach $i_r \in$ incoming_right_profile_indices($i'_r$) do
    foreach $q \in$ incoming_right_emit_states($q'$) do
        Let $\tau$ = get_state_type($q, left$);
        Let $(e_l, e_r) = (\phi_\tau^{(i'_l)}, \phi_D^{(i_r)})$;
        Let $m = (q, e_l, e_r)$;
        $Z(m') \leftarrow Z(m') + Z(m)\mathcal{W}^{\mathrm{trans}}_{Q'_n}(q, q')t_r(i_l, i'_l)\mathcal{E}$;
```

Algorithm 4: sum_paths_to() used by Algorithm 3. This is used during the Forward algorithm to compute the sum-over-paths likelihood ending at a given state. This quantity is later used to guide stochastic sampling (Algorithms 5 and 6).

subsequence co-ordinate for each leaf-sequence in the clade descended from node n). For example, in our non-normal construction of $\nabla(S)$, the state $\phi \in \mathbb{Z}_{|S|+1}$ is itself the co-ordinate. The co-ordinate information associated with e_l and e_r can, therefore, be used to define some sort of *alignment envelope*, as in [63]. For example, we could exclude (e_l, e_r) pairs if they result in alignment cutpoints that are too far from the main diagonal of the sequence co-ordinate hypercube.

Let the function *is_in_envelope*(i_l, i_r) return true if the state-index pair (i_l, i_r) is allowed by the alignment envelope, and false otherwise. Further, assume that Z is a sparse container data structure, such that $Z(m)$ always evaluates to zero if $m = (\rho, \upsilon, b_l, e_l, b_r, e_r)$ is not in the envelope.

```
Input: Z
Output: A path $\pi$ through $M_n$ sampled proportional to $\mathcal{W}^{\mathrm{trans}}_{M_n}(\pi)$.
Initialize $\pi \leftarrow (\phi_{E;M_n})$
Initialize $m' \leftarrow \phi_{E;M_n}$
while $m' \neq \phi_{S;M_n}$ do
    Let $K = \{k \in M_n : \mathcal{W}^{\mathrm{trans}}_{M_n}(k, m') \neq 0\}$
    With probability $\frac{Z(m)\mathcal{W}^{\mathrm{trans}}_{M_n}(m,m')}{\sum_{k \in K} Z(k)\mathcal{W}^{\mathrm{trans}}_{M_n}(k,m')}$, prepend $m$ to $\pi$.
    Set $m' \leftarrow m$
return $\pi$
```

Algorithm 5: Pseudocode for Stochastic traceback for sampling paths through the transducer M_n, described in Section 2.3. Stochastic sampling is done such that a path π through M_n is visited proportional to its likelihood weight. By tracing a series of paths through M_n and storing the union of these paths as a sequence profile, we are able to limit the number of solutions considered during progressive reconstruction, reducing time and memory complexity.

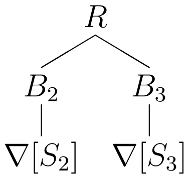
Construction of alignment envelopes

Let $\nabla(S)$ be defined such that it has only one nonzero-weighted path

$$X_0 \rightarrow W_0 \overset{\mathrm{symbol}(S,1)}{\rightarrow} M_1 \rightarrow W_1 \overset{\mathrm{symbol}(S,2)}{\rightarrow} M_2 \rightarrow \ldots \rightarrow W_{L-1} \overset{\mathrm{symbol}(S,\mathrm{length}(S))}{\rightarrow} M_{\mathrm{length}(S)} \rightarrow W_{\mathrm{lengt}}$$

so a $\nabla(S)$-state is either the start state (X_0), the end state ($X_{\mathrm{length}(S)}$), a wait state (W_i) or a match state (M_i). All these states have the form ϕ_i where i represents the number of symbols of S that have to be read in order to reach that state, i.e. a "co-ordinate" into S. All $\nabla(S)$-states are labeled with such co-ordinates, as are the states of any transducer that is a composition involving $\nabla(S)$, such as G_n or H_n.

For example, in a simple case involving a root node (1) with two children (2,3) whose sequences are constrained to be S_2, S_3, the evidence transducer is $G = RG_{\mathrm{root}} = R(G_2 \circ G_3) = R(\Upsilon(B_2\nabla(S_2), B_3\nabla(S_3))) =$



All states of G have the form $g = (r, b_2, \phi_2 i_2, b_3, \phi_3 i_3)$ where $\phi_2, \phi_3 \in \{X, W, M\}$, so $\phi_2 i_2 \in \{X_{i_2}, W_{i_2}, M_{i_2}\}$ and similarly for $\phi_3 i_3$. Thus, each state in G is associated with a co-ordinate pair (i_2, i_3) into (S_2, S_3), as well as a state-type pair (ϕ_2, ϕ_3).

Let n be a node in the tree, let $\mathcal{L}_n$ be the set of indices of leaf nodes descended from n, and let G_n be the phylogenetic transducer for the subtree rooted at n, defined in Section 2.3. Let Φ_n be the state space of G_n.

```
Input: Z
Output: A path $\pi$ through $M_n$ sampled proportional to $\mathcal{W}^{\text{trans}}_{M_n}(\pi)$.
Initialize $\pi \leftarrow (\phi_{E;M_n})$
Initialize $m' \leftarrow \phi_{E;M_n}$
while $m' \neq \phi_{S;M_n}$ do
    Initialize $states\_in \leftarrow ()$
    Initialize $weights\_in \leftarrow ()$
    Set $(q', e'_l, e'_r) \leftarrow m'$
    if $m' = \phi_{E;M_n}$ then
        foreach $q \in \Phi_{wait}$ do
            Add $(q', \phi_W^{(\text{end})}, \phi_W^{(\text{end})})$ to $states\_in$
            Add $Z((q, \phi_W^{(\text{end})}, \phi_W^{(\text{end})}))\mathcal{W}^{\text{trans}}_{Q'_n}(q, \phi_{E;Q'_n})$ to $weights\_in$
    else
        if $\mathcal{W}^{trans}_{Q'_n}(\phi_{S;Q'_n}, qTo)t_l(0, i'_l)t_r(0, i'_r) \neq 0$ then
            Add $(\phi_{S;Q'_n}, \phi_S, \phi_S)$ to $states\_in$
            Add $\mathcal{W}^{\text{trans}}_{Q'_n}(\phi_{S;Q'_n}, qTo)t_l(0, i'_l)t_r(0, i'_r)$ to $weights\_in$
        foreach $i_l \in incoming\_left\_profile\_indices(i'_l)$ do
            foreach $i_r \in incoming\_right\_profile\_indices(i'_r)$ do
                Let $(e_l, e_r) = (\phi_D^{(i_l)}, \phi_D^{(i_r)})$;
                foreach $q \in incoming\_match\_states(q')$ do
                    Add $(q, e_l, e_r)$ to $states\_in$;
                    Add $Z((q, e_l, e_r))\mathcal{W}^{\text{trans}}_{Q'_n}(q, q')t_l(i_l, i'_l)t_r(i_r, i'_r)$ to $weights\_in$;
        foreach $i_l \in incoming\_left\_profile\_indices(i'_l)$ do
            Let $(e_l, e_r) = (\phi_D^{(i_l)}, \phi_W^{(i'_r)})$;
            foreach $q \in incoming\_left\_emit\_states(q')$ do
                Add $(q, e_l, e_r)$ to $states\_in$;
                Add $Z((q, e_l, e_r))\mathcal{W}^{\text{trans}}_{Q'_n}(q, q')t_l(i_l, i'_l)$ to $weights\_in$;
        foreach $i_r \in incoming\_right\_profile\_indices(i'_r)$ do
            Let $\tau = \text{get\_state\_type}(q', left)$;
            Let $(e_l, e_r) = (\phi_\tau^{(i'_l)}, \phi_D^{(i_r)})$;
            foreach $q \in incoming\_right\_emit\_states(q')$ do
                Add $(q, e_l, e_r)$ to $states\_in$;
                Add $Z((q, e_l, e_r))\mathcal{W}^{\text{trans}}_{Q'_n}(q, q')t_r(i_r, i'_r)$ to $weights\_in$;
    Set $m \leftarrow \text{sample}(states\_in, weights\_in)$
    Prepend $m$ to $\pi$
    Set $m' \leftarrow m$
return $\pi$
```

Algorithm 6: Pseudocode for stochastic traceback for sampling paths through the transducer M_n, described in Section 2.3. Stochastic sampling is done such that a path π through M_n is visited proportional to its likelihood weight. By tracing a series of paths through M_n and storing the union of these paths as a sequence profile, we are able to limit the number of solutions considered during progressive reconstruction, reducing time and memory complexity.

If $m \in \mathcal{L}_n$ is a leaf node descended from n, then G_n includes, as a component, the transducer $\nabla(S_m)$. Any G_n-state, $g \in \Phi_n$, is a tuple, one element of which is a $\nabla(S_m)$-state, ϕ_i, where i is a co-ordinate (into sequence S_m) and ϕ is a state-type. Define $i_m(g)$ to be the co-ordinate and $\phi_m(g)$ to be the corresponding state-type.

Let $A_n : \Phi_n \to 2^{\mathcal{L}_n}$ be the function returning the set of *absorbing leaf indices* for a state, such that the existence of a finite-weight transition $g' \to g$ implies that $i_m(g) = i_m(g') + 1$ for all $m \in A_n(g)$.

Let (l, r) be two sibling nodes. The *alignment envelope* is the set of sibling state-pairs from G_l and G_r that can be aligned. The function $E\colon \Phi_l \times \Phi_r \to \{0, 1\}$ indicates membership of the envelope. For example, this basic envelope allows only sibling co-ordinates separated by a distance s or less

$$E_{\text{basic}}(f, g) = \max_{m \in A_l(f), n \in A_r(g)} |i_m(f) - i_n(g)| \leq s$$

An alignment envelope can be based on a *guide alignment.* For leaf nodes x, y and $1 \leq i \leq \text{length}(S_x)$, let $\mathcal{G}(x, i, y)$ be the number of residues of sequence S_y in the section of the guide alignment from the first column, up to and including the column containing residue i of sequence S_x.

This envelope excludes a pair of sibling states if they include a homology between residues which is more than s from the homology of those characters contained in the guide alignment:

$$E_{\text{guide}}(f, g) = \max_{m \in A_l(f), n \in A_r(g)} \max(\ |\mathcal{G}(m, i_m(f), n) - i_n(g)|\ , |\mathcal{G}(n, i_n(g), m) - i_m(f)|\ ) \leq s$$

Let $K(x, i, y, j)$ be the number of match columns (those alignment columns in which both S_x and S_y have a non-gap character) between the column containing residue i of sequence S_x and the column containing residue j of sequence S_y. This envelope excludes a pair of sibling states if they include a homology between residues which is more than s *matches* from the homology of those characters contained in the guide alignment:

$$\begin{aligned} E_{\text{guide}}(f, g) = \quad & \max_{m \in A_l(f), n \in A_r(g)} \max(\ |\mathcal{G}(m, i_m(f), n) - K(m, i_m(f), n, i_n(g))|, \\ & |\mathcal{G}(n, i_n(g), m) - K(n, i_n(g), m, i_m(f))|\ ) \leq s \end{aligned}$$

Explicit construction of profile E_n from M_n, following DP

Having sampled a set of paths through M_n, profile E_n is constructed by applying a series of transformations. Refer to the previous sections for definitions pertaining to M_n and E_n.

Transformation $M_n \to M_n'$: sampled paths

Let $\Pi' \subseteq \Pi_{M_n'}$ be a set of complete paths through M_n, corresponding to K random samples from $P(\pi | M_n)$.

The state space of M'_n is the union of all states used by the paths in $\Pi_{M'_n}$

$$\Phi_{M'_n} = \bigcup_{\pi \in \Pi'} \bigcup_{i=1}^{|\pi|} \pi_i$$

where π_i is the i^{th} state in a path $\pi \in (\tau_{M_n})^*$.

The I/O and transition weight functions for M'_n are the same as those of M_n:

$$\begin{aligned} \mathcal{W}^{\text{trans}}_{M'_n}(m, m') &= \mathcal{W}^{\text{trans}}_{M_n}(m, m') \\ \mathcal{W}^{\text{emit}}_{M'_n}(\epsilon, \epsilon, m') &= \mathcal{W}^{\text{emit}}_{M_n}(\epsilon, \epsilon, m') \end{aligned}$$

M'_n can be updated after each path π is sampled:

$$\Phi_{M'_n} = \Phi_{M'_n} \cup \bigcup_{i=1}^{|\pi|} \pi_i$$

For some arbitrary limiting factor $p \geq L$, if $|\Phi_{M'_n}| \geq p$ upon addition of states of π, sampling may be terminated. This allows bounding $|\Phi_{E_n}|$ as the algorithm traverses up the phylogeny.

Transformation $M'_n \to E''_n$: stripping out the prior

Let E''_n be a the transducer constructed via removing the R states from the states of M'_n.

$$\Phi_{E''_n} = \{(\upsilon, b_l, e_l, b_r, e_r) : \exists(\rho, \upsilon, b_l, e_l, b_r, e_r) \in M'_n\}$$

The I/O and transition weight functions for E''_n are the same as those of H_n:

$$\begin{aligned} \mathcal{W}^{\text{trans}}_{E''_n}(e, e') &= \mathcal{W}^{\text{trans}}_{H_n}(e, e') \\ \mathcal{W}^{\text{emit}}_{E''_n}(\epsilon, \epsilon, e') &= \mathcal{W}^{\text{emit}}_{H_n}(\epsilon, \epsilon, e') \end{aligned}$$

Transformation $E''_n \to E'_n$: eliminating null states

Let E'_n be the transducer derived from E''_n by marginalizing its null states.

The state space of E'_n is the set of non-null states in E''_n:

$$\Phi_{E'_n} = \{e : e \in E''_n, type(e) \in \{S, E, D, W\}\}$$

The transition weight function is the same as that of H_n (and also E''_n) with paths through null states marginalized. Let

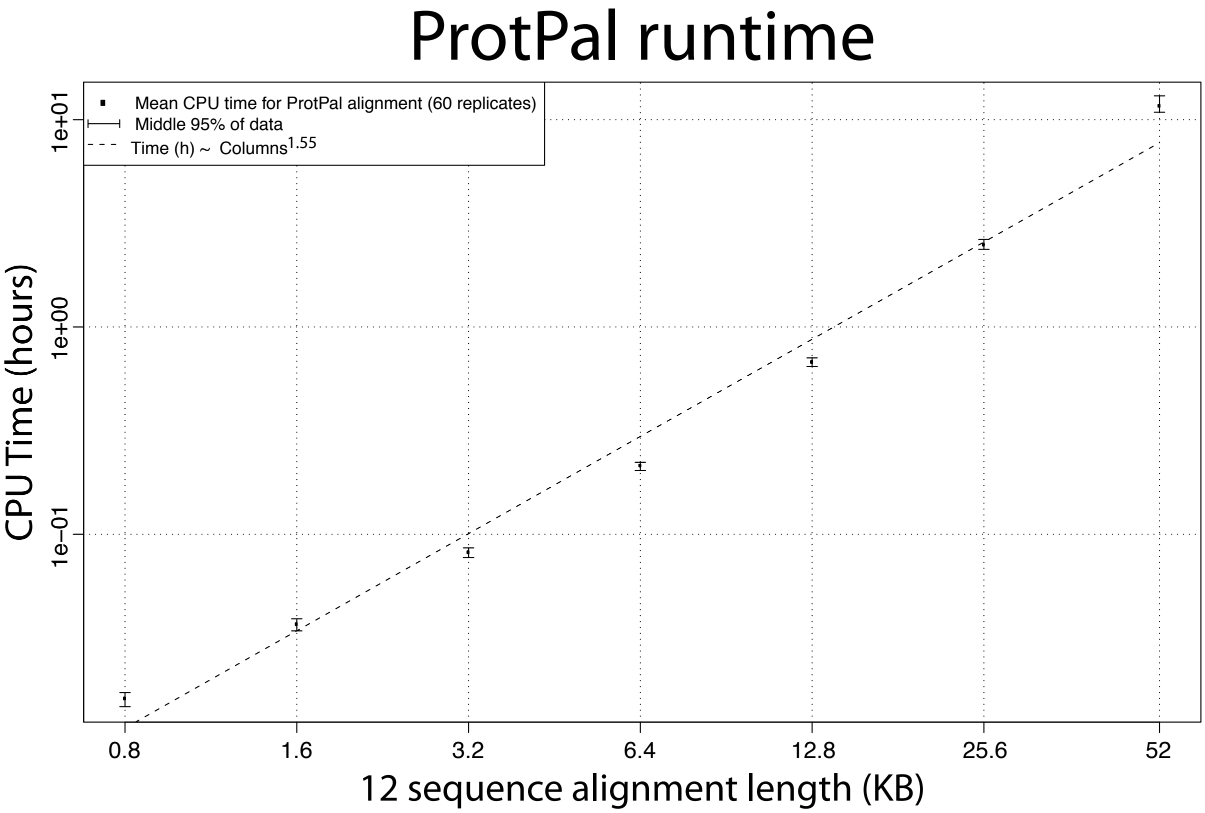


Figure 2.48: When using an alignment envelope, our implementation of the alignment algorithm scales with polynomial complexity with respect to sequence length. Alignments consisting of 12 sequences were simulated for lengths ranging from 800bp to 52kb and aligned using our software. Mean and middle 90% quantiles (60 replicates) for CPU time required on a 2GHz, 2GB RAM Linux machine are shown next to a linear fit in log-log space (dashed line).

$$\begin{aligned}
\pi_{E''_n(e,e')} &= \{\pi : \pi_1 = e, \pi_{|\pi|} = e', type(\pi_i) = N \ \forall \ i : 2 \leq i < |\pi|\} \\
\mathcal{W}^{\text{trans}}_{E'_n}(e, e') &= \sum_{\pi \in \pi_{E''_n(e,e')}} \mathcal{W}^{\text{trans}}_{H_n}(\pi)
\end{aligned}$$

The transition weight function resulting from summing over null states in E''_n can be done with a preorder traversal of the state graph, outlined in Algorithm 7. The *stack* data structure has operations *stack.push*(e), which adds e to the top of the stack, and *stack.pop*(), which removes and returns the top element of the stack. The *weights* container maps states in $\Phi_{E''_n}$ to real-valued weights.

```
**Input**: $\Phi_{E''_n}, \mathcal{W}^{\text{trans}}_{E''_n}$
**Output**: $\mathcal{W}^{\text{trans}}_{E'_n}$
Let $\Phi_{E'_n} = \{e : e \in E''_n, \textit{type}(e) \in \{S, E, D, W\}\}$
Initialize $t_{\text{new}}(e, e') = 0 \ \forall (e, e') \in \Phi_{E'_n}$
**foreach** $\textit{source\_state} \in \Phi_{E''_n}$ **do**
    Initialize $\textit{stack} = [\textit{source\_state}]$
    Initialize $\textit{weights}[\textit{source\_state}] = 0$

    **while** $\textit{stack} \neq []$ **do**
        Set $e = \textit{stack.pop}()$
        **foreach** $e' \in \{e' \in \Phi_{E''_n} : \mathcal{W}^{\textit{trans}}_{E''_n}(e, e') \neq 0\}$ **do**
            **if** $\textit{type}(e') \neq N$ **then**
                $t_{\text{new}}(\textit{source}, e') + = \textit{weights}[\mathrm{e}] \cdot \mathcal{W}^{\text{trans}}_{E''_n}(e, e')$
            **else**
                $\textit{stack.push}(e')$
                $\textit{weights}[\mathrm{e}'] = \textit{weight}[\mathrm{e}] \cdot \mathcal{W}^{\text{trans}}_{E''_n}(e, e')$
**return** $\mathcal{W}^{\textit{trans}}_{E'_n}(e, e') \equiv t_{\textit{new}}(e, e') \ \forall (e, e') \in \Phi_{E'_n}$
```

Algorithm 7: Pseudocode for transforming the transition weight function of E''_n into that of E'_n via summing over null state paths (insertions). This is done after stochastic sampling as the first of two steps transforming a sampled M_n transducer into a recognizer E_n, described in Section 2.3. Summing over null states ensures that these states cannot align to sibling states in the parent round of profile-profile alignment. An insertion at branch n is, by definition, not homologous to any characters outside the n-rooted subtree, and null state elimination is how this is explicitly enforced in our algorithm.

Besides the states $\{\phi_S, \phi_E, \phi_W^{(\text{end})}\}$, the remaining states in E'_n are of type D, which we now index in ascending topological order:

$$\Phi_{E'_n} = \{\phi_S, \phi_E, \phi_W^{(\text{end})}\} \cup \{\phi_D^{(n)} : 1 \leq n \leq N_n\}$$

where $1 \leq i, j \leq N_n$,

$$\begin{aligned}
t_n(i, j) &\equiv \mathcal{W}^{\text{trans}}_{E'_n}(\phi_D^{(i)}, \phi_D^{(j)}) \\
t_n(0, j) &\equiv \mathcal{W}^{\text{trans}}_{E'_n}(\phi_S, \phi_D^{(j)}) \\
t_n(i, N_n + 1) &\equiv \mathcal{W}^{\text{trans}}_{E'_n}(\phi_D^{(i)}, \phi_W^{(\text{end})})
\end{aligned}$$

Transformation $E'_n \to E_n$: adding wait states

Let E_n be the transducer derived from transforming E'_n into strict normal form. Since N states were removed in the transformation from $E''_n \to E'_n$, we need only add W states before each D state:

$$\Phi_{E_n} = \{\phi_S, \phi_E, \phi_W^{(\mathrm{end})}\} \cup \{\phi_D^{(n)} : 1 \leq n \leq N\} \cup \{\phi_W^{(n)} : 1 \leq n \leq N\}$$

Note the following correspondences between the previously-defined $\mathcal{W}^{\mathrm{trans}}_{E_n}(e, e')$ and new notation.

$$\begin{aligned}
\mathcal{W}^{\mathrm{trans}}_{E_n}(\phi_D^{(i)}, \phi_W^{(j)}) &= t_n(i,j) \\
\mathcal{W}^{\mathrm{trans}}_{E_n}(\phi_W^{(j)}, \phi_D^{(j)}) &= 1 \\
\mathcal{W}^{\mathrm{via\text{-}wait}}_{E_n}(\phi_D^{(i)}, \phi_D^{(j)}) &= t_n(i,j) \\
\mathcal{W}^{\mathrm{to\text{-}wait}}_{E_n}(\phi_D^{(i)}, \phi_W^{(j)}) &= t_n(i,j) \\
\mathcal{W}^{\mathrm{to\text{-}wait}}_{E_n}(\phi_W^{(i)}, \phi_W^{(j)}) &= \delta(i = j)
\end{aligned}$$

Message-passing interpretation

In the interpretation of Felsenstein's pruning algorithm [7] and Elston and Stewart's more general peeling algorithm [64] as message-passing on factor graphs [40], the tip-to-root messages are functions of the form $P(\mathcal{S}_n | S_n = x)$ where S_n (a random variable) denotes the sequence at node n, x denotes a particular value for this r.v., and $\mathcal{S}_n = \{S_m : m \in \mathcal{L}_n\}$ denotes the observation of sequences at nodes in $\mathcal{L}_n$, the set of leaf nodes that have n as a common ancestor.

These tip-to-root messages are equivalent to our evidence-expanded transducers (Section 2.3):

$$P(\mathcal{S}_n | S_n = x) = \mathcal{W}(x : [G_n] : \epsilon)$$

The corresponding root-to-tip messages take the form $P(\bar{\mathcal{S}}_n, S_n = x)$ where $\bar{\mathcal{S}}_n = \{S_m : m \in \mathcal{L}, m \notin \mathcal{L}_n\}$ denotes the observation of sequences at leaf nodes that do *not* have n as a common ancestor. These messages can be combined with the tip-to-root messages to yield posterior probabilities of ancestral sequences

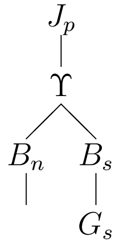
$$P(S_n = x | \mathcal{S}) = \frac{P(\bar{\mathcal{S}}_n, S_n = x) P(\mathcal{S}_n | S_n = x)}{P(\mathcal{S})}$$

We can define a recursion for transducers that model these root-to-tip messages, just as with the tip-to-root messages.

First, define $J_1 = R$.

Next, suppose that $n > 1$ is a node with parent p and sibling s. Define

$$J_n = J_p(B_n \circ (B_s G_s)) =$$



Note that J_n is a generator that outputs only the sequence at node n (because G_s has null output). Note also that $J_n G_n \equiv G_0$.

The root-to-tip message is

$$P(\bar{\mathcal{S}}_n, \mathcal{S}_n = x) = \mathcal{W}(\epsilon : [J_n] : x)$$

The equations for G_n and J_n are transducer formulations of the pruning and peeling recursions

$$\begin{aligned}
P(\mathcal{S}_n|\mathcal{S}_n = x) &= \left(\sum_y P(\mathcal{S}_l = y|\mathcal{S}_n = x) P(\mathcal{S}_l|\mathcal{S}_l = y) \right) \left(\sum_z P(\mathcal{S}_r = z|\mathcal{S}_n = x) P(\mathcal{S}_r|\mathcal{S}_r = z) \right) \\
P(\bar{\mathcal{S}}_n, \mathcal{S}_n = x) &= \sum_y P(\bar{\mathcal{S}}_p, \mathcal{S}_p = y) P(\mathcal{S}_n = x|\mathcal{S}_p = y) \sum_z P(\mathcal{S}_s = z|\mathcal{S}_p = y) P(\mathcal{S}_s|\mathcal{S}_s = z)
\end{aligned}$$

where (l, r) are the left and right children of node n. For comparison,

$$\begin{aligned}
G_n &= (B_l G_l) \circ (B_r G_r) \\
J_n &= J_p(B_n \circ (B_s G_s)) \\
\mathcal{W}(x : [B_n] : y) &= P(\mathcal{S}_n = y|\mathcal{S}_p = x) \\
\mathcal{W}(x : [TU] : z) &= \sum_y \mathcal{W}(x : [T] : y) \mathcal{W}(y : [U] : z) \\
\mathcal{W}(x : [T_l] : \mathcal{S}_l) \mathcal{W}(x : [T_r] : \mathcal{S}_r) &= \mathcal{W}(x : [T_l \circ T_r] : \mathcal{S}_n)
\end{aligned}$$

2.4 Conclusions

In this article we have presented an algorithm that may be viewed in two equivalent ways: a form of Felsenstein's pruning algorithm generalized from individual characters to entire sequences, or a phylogenetic generalization of progressive alignment. Our algorithm extends the concept of a character substitution matrix (e.g. [65, 66]) to finite-state transducers, replacing matrix multiplication with transducer composition.

We described a hierarchical approximation technique enabling inference in $\mathcal{O}(c^2 p^3 N)$ time and memory (typical-case $\mathcal{O}((cp)^2 N)$) , as opposed to $\mathcal{O}(L^N)$ for exact, exhaustive inference (N sequences of length L, limiting factor $m \geq L$ and branch transducer with c states). Empirical tests indicate that adding additional constraints (in the form of an "alignment envelope") brings typical-case time complexity down to $\mathcal{O}(cpN)$, making the algorithm practical for typical alignment problems.

Much of computational biology depends on a multiple sequence alignment as input, yet the uncertainty and bias engendered by an alignment is rarely accounted for. Further, most alignment programs account for the phylogeny relating sequences either in a heuristic sense, or not at all. Previous studies have indicated that alignment uncertainty and/or accuracy strongly affects downstream analyses [67], particularly if evolutionary inferences are to be made [6].

In this work we have described the mathematics and algorithms required for an alignment algorithm that is simultaneously explicitly phylogenetic and avoids conditioning on a single multiple alignment. In extensive simulations (in separate work, submitted), we find that our implementation of this algorithm recovers significantly more accurate reconstructions of simulated indel histories, indicating the need for mathematically rigorous alignment algorithms, particularly for evolutionary applications.

The source code to this paper, including the graphviz and phylocomposer files used to produce the diagrams, can be found at `https://github.com/ihh/transducer-tutorial`

2.5 Methods

Accuracy simulation study The simulation study was performed as described in Appendix B.

Time simulation study In order to empirically determine the time complexity of our method, alignments of varying lengths (800, 1600, 3200, 6400, 12800, 25600, 52000 positions) were simulated using indel-seq-gen (default parameters) on the 12 *Drosophila* species tree and aligned with ProtPal using a guide alignment with restriction parameter (`-s` option described in Section 2.3) set to 10. The analysis time on a 2GHz, 2GB machine was averaged over 60 replicates and plotted next to a linear fit computed using R (www.r-project.org).

Additional notation

This section defines commonplace notation used earlier in the document.

Sequences and alignments

Sequence notation: Let Ω^* be the set of sequences over Ω, including the empty sequence ϵ. Let $x + y$ denote the concatenation of sequences x and y, and $\sum_n x_n$ the concatenation of sequences $\{x_n\}$.

Gapped-pair alignment: A gapped pairwise alignment is a sequence of individual columns of the form (ω_1, ω_2) where $\omega_1 \in (1 \cup \{\epsilon\})$ and $\omega_2 \in (2 \cup \{\epsilon\})$.

Map from alignment to sequences: The function $S_k : ((1 \cup \{\epsilon\}) \times (2 \cup \{\epsilon\}))^* \to \Omega_k^*$ returns the k'th row of a pairwise alignment, with gaps removed

$$\begin{aligned} S_1(x) &= \sum_{(\omega_1,\omega_2)\in x} \omega_1 \\ S_2(x) &= \sum_{(\omega_1,\omega_2)\in x} \omega_2 \end{aligned}$$

Transition paths: A transition path $\pi \in \Pi$ is a sequence of transitions of the form $(\phi_1, \omega_I, \omega_O, \phi_2)$ where $\phi_1, \phi_2 \in \Phi$, $\omega_I \in (I \cup \{\epsilon\})$, and $\omega_O \in (O \cup \{\epsilon\})$.

Input and output sequences: Define the input and output sequences $S_I : \Pi \to \Omega_I^*$ and $S_O : \Pi \to \Omega_O^*$

$$\begin{aligned} S_I(\pi) &= \sum_{(\phi_1,\omega_I,\omega_O,\phi_2)\in\pi} \omega_I \\ S_O(\pi) &= \sum_{(\phi_1,\omega_I,\omega_O,\phi_2)\in\pi} \omega_O \end{aligned}$$

Chapter 3

Approximate alignment with transducers: simulation-based evaluation

The following chapter contains work published in PLoS ONE, together with Gerton Lunter, Benedict Paten, and Ian Holmes [49].

Overview

The Multiple Sequence Alignment (MSA) is a computational abstraction that represents a partial summary either of indel history, or of structural similarity. Taking the former view (indel history), it is possible to use formal automata theory to generalize the phylogenetic likelihood framework for finite substitution models (Dayhoff's probability matrices and Felsenstein's pruning algorithm) to arbitrary-length sequences. In this paper, we report results of a simulation-based benchmark of several methods for reconstruction of indel history. The methods tested include a relatively new algorithm for statistical marginalization of MSAs that sums over a stochastically-sampled ensemble of the most probable evolutionary histories. For mammalian evolutionary parameters on several different trees, the single most likely history sampled by our algorithm appears less biased than histories reconstructed by other MSA methods. The algorithm can also be used for alignment-free inference, where the MSA is explicitly summed out of the analysis. As an illustration of our method, we discuss reconstruction of the evolutionary histories of human protein-coding genes.

3.1 Background

The Multiple Sequence Alignment (MSA), indispensable to computational sequence analysis, represents a hypothetical claim about the homology beteen sequences. MSAs have many different uses, but the underlying hypothesis can often be classified as a claim either of *structural* homology (the 3D structures align in a particular way) or of *evolutionary* homology (the sequences are related by a particular history on a given phylogenetic tree). These types of hypothesis are similar, but with subtle (and important) distinctions: at the residue level, a claim of evolutionary homology (direct shared descent) is far stronger than a claim of structural homology (same approximate fold). Furthermore, both types of MSA—evolutionary and structural—typically only represent *summaries* of the respective homologies: some fine detail is often omitted. For example, an evolutionary MSA may—or may not—include the ancestral sequences at internal nodes of the underlying tree.

Structural and evolutionary MSAs are often conflated, but they have quite different applications. For example, a common use for a structural MSA is *template-based structure prediction*, where a query sequence is aligned to a target of known structure; the success of this prediction reflects the number of query-template residues correctly aligned [68]. By way of contrast, a common application for an evolutionary MSA is to identify regions or sites under selection, the success of which depends on accurate reconstruction of the evolutionary history [69, 70].

Evaluation of alignment methods is typically done with implicit regard for the structural interpretation. Many benchmarks have used metrics based on the Sum of Pairs Score (SPS) [71]. In the situation that a query-template pairwise alignment is randomly picked out of the MSA, the SPS effectively estimates the proportion of homologous residues that are correctly identified. Several alignment methods attempt to maximize the posterior expectation of SPS or similar metrics. This appears to improve accuracy, particularly when measured with reference to structural homology. However, it does not automatically confer *evolutionary* accuracy — a correct reconstruction of the evolutionary history of the sequences.

Several studies suggest that multiple alignment for evolutionary purposes is still a highly uncertain procedure [67], and that errors therein may significantly bias analyses of evolutionary effects [6, 72–76]. A useful component of these studies is simulation of genetic sequence evolution [6], which appears to better indicate evolutionary accuracy than benchmarks derived from protein structure alignments. Simulations can be made quite realistic given the abundance of comparative sequence data [77].

The current state-of-the-art in phylogenetic alignment software is a choice between (on the one hand) programs that lack explicit models of the underlying evolutionary process, and so are not framed as statistical inference problems [6], and (on the other hand) Bayesian Markov chain Monte Carlo (MCMC) methods, which are statistically exact but prohibitively slow [46, 48].

A telling observation is that while substitution rate is routinely measured from MSAs and used as an indicator of natural selection, there is relatively little analogous use of indel rate. As we report here, it seems highly likely that even if indel rate is a useful evolutionary signal (which is eminently plausible), the present alignment methods distort measurements of this rate so far as to make it meaningless (Figure 3.1 and Figure 3.2).

In this paper, we frame phylogenetic sequence alignment as an approximate maximum likelihood (ML) inference. Our inference algorithm assumes that the tree is known, requiring a separate tree estimation protocol. While this is a strong assumption, it is in principle shared among all progressive aligners (e.g. PRANK [5], Muscle [78], ClustalW [79], MAFFT [80]). The alignment-marginalized likelihoods reported by our algorithm allow for statistical tests between alternative trees, and the functionality to estimate an initial alignment and guide tree from unaligned sequences exists elsewhere in the DART package. Our framing uses automata-theoretic methods from computational linguistics to unify several previously-disjoint areas of bioinformatics: Felsenstein's pruning algorithm for the phylogenetic likelihood function [7], progressive multiple sequence alignment [81], and alignment ensemble representation using partial order graphs [43]. Our algorithm may be viewed as a stochastic generalization of pruning to infinite state spaces: it retains the linear time and memory complexity of pruning ($\mathcal{O}(NL)$ for N sequences of length L), while moderating the biasing effect of the MSA. The algorithmic details of our method are outlined briefly in the Methods, and in more complete, mathematically precise terms (with a tutorial introduction) in a separately submitted work.

Our software implementation of this algorithm is called ProtPal. We measured the accuracy of ProtPal relative to leading non-MCMC alignment/reconstruction protocols by simulating indels and substitutions on a known phylogeny, withholding the true history and attempting to reconstruct it from the sequences at the tips of the tree. The results show that all previous approaches to the reconstruction of ancestral sequences introduce significant biases, including systematic underestimation of insertions and overestimation of deletions. This contradicts previous claims that advances in the statistical foundations of alignment tools, supported by improvements in protein-structure benchmarks, necessarily improve the accuracy of evolutionary parameter estimates like the indel rate [6, 82, 83].

ProtPal introduces less bias than any other methods we tested, including PRANK, the state-of-the-art phylogenetic progressive aligner [6]. Both PRANK and ProtPal treat insertions and deletions as phylogenetic events (Figure 3.3). Based on our tests, ProtPal appears to be the best choice for small to moderately-sized analyses, such as a reconstruction of the history of proteins at the inter-species level in human evolutionary history. Using ProtPal to estimate indel rates for $\sim 7,500$ human protein-coding gene families, we find that per-gene indel rates are approximately gamma-distributed, with 95% of genes experiencing a mean rate of less than 0.1 indel events per synonymous substitution event. We find that lengths of inserted and deleted sequences are comparably distributed, having medians 5 and 7, respectively. The human lineage appears to have experienced unusually many insertions since the human-mouse split. By mapping genes to Gene Ontology (GO) terms, we find that the 200 fastest-indel genes are enriched for regulatory and metabolic functions. Possible applications and extensions of our algorithm include phylogenetic placement, homology detection, and reconstruction of structured RNA.

3.2 Results

Computational reconstruction of simulated histories

We undertook to determine the ability of leading bioinformatics programs, including ProtPal, to characterize mutation event histories. We simulated indel histories on a tree, then attempted to reconstruct the MAP history, $\hat{H}$, using only knowledge of the sequences S and the phylogeny T (but not the sequence alignment). The history $\hat{H}$ is the aligned set of observed extant and predicted ancestral sequences, such that insertion, deletion, and substitution events can be pinpointed to specific tree branches (though not to specific time points on those branches).

We then characterized the reconstruction quality both directly, by comparison of $\hat{H}$ to the true H, and indirectly, by using $\hat{H}$ to estimate θ, the evolutionary parameters:

$$\hat{\theta}_{\hat{H}} = \text{argmax}_{\theta'} P(\theta'|\hat{H}, S, T) = \text{argmax}_{\theta'} P(\hat{H}, S|T, \theta') \tag{3.1}$$

where the latter step assumes a flat prior, $P(\theta') = \text{const}$. We then compared the history-conditioned parameter estimate $\hat{\theta}_{\hat{H}}$ to the true θ.

This statistic is not without its problems. For one thing, we use an initial guess of θ to estimate $\hat{H}$. Furthermore, for an unbiased estimate, we should sum over all histories, rather than conditioning on the MAP reconstructed history. This summing over histories would, however, require multiple expensive calculations of $P(S|T,\theta)$, where conditioning on $\hat{H}$ requires only one such calculation. Furthermore, parameter estimation conditioned on a MAP-reconstructed history is the *de facto* method employed by large-scale genomics studies focusing on indels [84–87].

Simulation model parameters

The model parameters are $\theta = (\lambda^i, \lambda^d, \mathbf{p}^i, \mathbf{p}^d, \mathbf{R})$: the insertion and deletion rates (λ^i, λ^d), indel length distributions $(\mathbf{p}^i, \mathbf{p}^d)$ and substitution rate matrix $(\mathbf{R})$. Here we focus on the rates (λ^i, λ^d).

As described in Appendix B, we generated data using an external simulation tool, indel-seq-gen, varying insertion (λ^i), deletion (λ^d) and substitution rates (r) over a range representative of per-gene rates in *Amniota* evolution (Figure 3.4). We varied indel rates (with $\lambda^i = \lambda^d$) between 0.005 and 0.08 expected indels per unit time, estimating that this range accounts for 95% of human gene families. We left the substitution model $(\mathbf{R})$ and indel length distributions $(\mathbf{p}^i, \mathbf{p}^d)$ fixed, employing indel-seq-gen's empirically-estimated values.

We performed simulations on mammalian, amniote and fruitfly phylogenies, using the taxa in those clades for which genomic sequence is actually available. We found generally consistent results, with common trends that were most pronounced on the largest of the three trees that we used (the twelve sequenced *Drosophila* species [88]). In discussing the trends, we will refer specifically to the results on this largest of the trees.

Indel rate estimates

Overall most accurate We first set out to determine which program, when used to analyze a set of unaligned sequences, returns the indel rate estimate closest to the true rate. We report the ratio of inferred rate to true rate for insertions $\frac{\hat{\lambda}^i_{\hat{H}}}{\lambda^i}$ and deletions $\frac{\hat{\lambda}^d_{\hat{H}}}{\lambda^d}$ in Figure 3.1, with each $\hat{\lambda}^*_{\hat{H}} \in \{\hat{\lambda}^i_{\hat{H}}, \hat{\lambda}^d_{\hat{H}}\}$ defined as $\hat{\theta}_{\hat{H}}$ in Equation B.2. No parameter estimate derived from a computationally reconstructed history approaches the level of accuracy achieved using the true history (labeled “True simulated history” in Figure 3.1).

The results do not always concord with previous benchmarks that have measured accuracy using 3D structural alignments: for example, the FSA program, one of the most accurate aligners on structural benchmarks [83], performs poorly here. This discordance may be due to the fundamental differences between evolutionary and structural homology, and the metrics used to assess each. For instance, consider a region with many nearby and overlapping insertions and deletions. The spatial and temporal location of these insertion and deletion events (in particular, the pinpointing of events to branches on the tree) defines what the “perfect” evolutionary reconstruction is. In contrast, even given perfect knowledge of the insertion/deletion history, a “perfect” structural alignment depends only on the proteins at the tips of the tree, and this alignment could differ from the true evolutionary reconstruction.

Fundamentally, the difference between FSA and ProtPal is the underlying metric that is being optimized by each program: FSA attempts to maximize a metric (AMA=Alignment Metric Accuracy) which is essentially "structural" (in the sense that it predicts how many residues would be correctly aligned in a pairwise alignment of two leaf-node sequences, as might be used in structure prediction by target-template alignment), while ProtPal attempts to maximize a "phylogenetic" metric (the probability of a given evolutionary history). The metric we have used in our benchmark (which counts correct reconstruction of the number of indel events on branches of the tree) is also "phylogenetic". When ranking the programs using the AMA metric, FSA perfoms well, with accuracy exceeding that of ProtPal in the highest indel rate category (Figure B.2). This suggests that the differences between our evolutionary benchmark and previous benchmarks are not due to the data, but rather the types of metrics that are used to measure alignment accuracy; similarly, the differences between the leading programs are primarily due to what types of benchmark they are explicitly trying to perform well at.

All programs other than ProtPal display insertion-*versus*-deletion biases that are, to a varying degree, asymmetric. Typically, the asymmetry is that insertions are underrepresented and deletions overrepresented. ProtPal's bias, which is generally less than the other programs, is also the most symmetric: reconstructed insertions and deletions are roughly equally reliable, with both slightly underestimated. Over the distribution of human gene rates used by this benchmark, our phylogenetic likelihood approach, ProtPal, provides the most accurate reconstructions of both insertion and deletion counts. PRANK, which also uses a tree (but no likelihood), avoids insertion-deletion biases to a certain extent, although insertion rates are slightly underestimated relative to deletion rates. Since ProtPal's MAP history estimation appears similar to the optimization algorithm of PRANK, we suspect that ProtPal's marginally better performance is due primarily to its main difference in implementation: ProtPal tracks an *ensemble* of possible reconstructions during progressive tree traversal (Section 3.4), whereas PRANK uses a single "current best guess."

Effect of indel rate variation To investigate the effect of indel rate variation on estimation accuracy, we separate each program's error distributions by indel rate (Figure 3.2). We find that all programs' accuracy is strongly affected by the indel rate used in simulation. As the true indel rate increases, most programs' estimates drift towards $\frac{\hat{\lambda}^*_{\hat{H}}}{\lambda^*} \to 0$. This is consistent with the so-called "gap attraction" effect, where indels that are nearby in sequence can be misinterpreted as substitution events [90]. Depending on the phylogenetic orientation of the events, estimated rates can be elevated or lowered, with different biases for insertion and deletion rates (Figure 3.3).

Gap attraction and other biases operate simultaneously, and are sometimes opposed. MUSCLE over-estimates the deletion rate under most conditions, but (consistent with a trend where programs have lower $\frac{\hat{\lambda}^*_{\hat{H}}}{\lambda^*}$ at higher indel rates) gets the deletion rate roughly correct in the highest-indel-rate category of our benchmark. However, the alignments produced by MUSCLE at high indel rates are no more “accurate” by pairwise metrics (Figure B.2). We conjecture that multiple, contradictory types of gap attraction are at work, e.g. Figures 3.3B and 3.3C.

After ProtPal, the two most accurate reconstruction methods are PRANK and ProbCons (the latter combined with a parsimonious indel reconstruction). ProbCons produces more reliable insertion estimates than PRANK in a broad range of benchmark categories, is tied with PRANK for deletion estimates, and appears robust to indel rate variation. PRANK performs slightly better than ProbCons in the slowest indel rate category we considered. ProtPal produces the most reliable estimates overall, outperforming ProbCons in all but the fastest indel rate category, and PRANK in all but the slowest.

Sensitivity to substitution rate As compared to variation of simulated indel rate, variation of simulated substitution rate appears to have little effect on the accuracy of indel reconstruction (Figure B.3). One notable exception is FSA, which appears to be affected by the substitution rate more than the other programs. For example, when the simulated indel and substitution rates are both low, FSA is comparable to the most accurate of the other programs (ProtPal); but when the substitution rate is increased, FSA's error is greater than the least accurate program (CLUSTALW). Errors in estimating the substitution rate are comparable among the programs tested, and are similarly correlated with the simulation indel rate (Figure B.4).

Reconstructed indel histories of human genes

We present here a comprehensive set of reconstructions accounting for the evolutionary history of individual codons in human genes. We used genes in the **O**rthologous and **P**aralogous **T**ranscripts in **C**lades (OPTIC) database's *Amniota* set, comprised of the 5 mammals H. *sapiens*, M. *musculus*, C. *familiaris*, M. *domestica*, O. *anatinus* and G. *gallus* as an outgroup [91]. Considering only those families with one unique ortholog per species (approximately 7,500 families), we combined tree branch statistics across genes, using the species tree in Figure B.6 . Our reconstructions are available at `http://biowiki.org/~oscar/optic_reconstruction.tar`, and we provide here various graphical summaries of *Amniota* evolutionary history. Several negative results stand in contrast to earlier-reported trends.

Indel rates Insertion and deletion rates are approximately gamma-distributed (Figure 3.4). Roughly 95% of genes have indel rates < 0.1 indels per synonymous substitution.

Phylogenetic origins In our simulations, ProtPal pinpoints residues' "branch of origin" more reliably than other tools, with a 93% accuracy rate (Figure B.1). Many codons appeared to have been inserted following the human-mouse split (Figure B.1)

Branch-specific indel rates Using our reconstructions to estimate the rates of indel mutations along specific tree branches, we find evidence of an elevated insertion rate in the human (black) branch, as well as on the the *Amniota* - *Australophenids* (pink) branch (Figure B.5).

Amino acid distributions Distributions over amino acids differ significantly between inserted, deleted and non-indel sequences (Figure B.7). In general, small residues are over-represented in insertions, in agreement with previous studies [92].

Indel lengths We find, contrary to a previous study in *Nematode* [93], that length distributions in the Amniotes are nearly identical between insertions and deletions (Figure B.8). The previously-reported result may be attributable to the deletion-biased nature of the methods used, particularly CLUSTALW and MUSCLE [93].

Indel position The position of indels within genes is highly biased towards the ends of genes, presumably in large part reflecting annotation error (Figure B.9). The bias is strongest for deletions at the N-terminus of the gene, but both insertions and deletions are enriched in both C- and N- termini.

Evolutionary context of indel SNPs We find no general correlation between the indel rate for a gene and the number of indel polymorphisms recorded for that gene in dbSNP [94] (Figure B.10).

Gene ontology indel rates No Gene Ontology (GO) categories stand out as having significantly lowered or heightened indel rates in any of the three ontologies, contrasting with the reported results of a 2007 study using a smaller number of genes [92]. An enrichment analysis conducted with GOstat [95] showed that the 200 fastest evolving genes in our data are significantly enriched for regulatory and metabolic functions.

3.3 Discussion

We developed and analyzed a simulation benchmark that compares programs based on their reconstructions of evolutionary history, using instantaneous mutation rates representative of Amniote evolution. We tested several different tree topologies; results were similar on all trees, but most pronounced on the tree with the longest branch lengths. We find that most programs distort indel rate measurements, despite claims to the contrary. Moreover, the systematic bias varies significantly when the rates of substitutions and indels are varied within a biologically reasonable range. Many of the programs we rated have been ranked in the past, but using benchmarks that use protein structural alignments as a gold standard, rather than evolutionary simulations. Furthermore, these previous benchmarks have not directly assessed the reconstruction of evolutionary history (or summary statistics such as the indel rate), but have used other alignment accuracy metrics such as the *Sum of Pairs Score*. Alignment programs that perform weakly on our benchmark have apparently performed well on these previous benchmarks. We hypothesize that these benchmarks, compared to ours, are less directly predictive of a program's accuracy at historical reconstruction, although they may better reflect the program's suitability to assist in tasks relating more closely to folded structure, like prediction of a protein's 3D structure from a homologous template.

We have introduced a new notation that describes a general, hidden Markov model-structured likelihood function for indel histories on a tree, as well as the structure of the corresponding inference algorithm. We have implemented the new method in a freely-available program, ProtPal, that allows, for the first time, phylogenetic reconstruction with accuracy over a broad range of indel rates. ProtPal is written in C++ as a part of the DART package: `www.biowiki.org/ProtPal`. The evolutionary reconstructions ProtPal produces are, according to our simulated tests, the most accurate of any available tool, for a range of parameters typical of human genes.

We applied ProtPal to the reconstruction of human gene indel history, using families of human gene orthologs from the OPTIC database. We find some patterns that agree with previous studies, such as the non-uniform distributions over amino acids seen in [92]. Other results stand in contrast - a previous study found significantly different length distributions for insertions and deletions [93], whereas in our data they appear very similar. Another prediction of our reconstruction is an elevated rate of insertions on the human branch since the human-mouse split. This contrasts with a previous analysis [97], though the data therein was whole genomes, rather than individual protein-coding genes. In contrast to [92], we find no obvious predictive power of the Gene Ontology (GO) for indel rates; that is, the indel rate does not appear strongly correlated with the presence or absence of any particular GO term-gene association. However, enrichment analysis for GO terms using GOstat [95] showed that the 200 fastest-evolving genes are significantly enriched for regulatory and metabolic function. This apparent discrepancy might be explained by a group of regulatory and metabolic genes which have very high indel rates, but whose small number prevent them from skewing the average within their GO categories.

Many applications which use a fixed-alignment phylogenetic likelihood could potentially benefit from ProtPal's reconstruction profiles. For example, phylogenetic placement algorithms estimate taxonomic distributions by evaluating the relative likelihoods of placing sequence reads on tree branches [39]. By using sequence profiles exported from ProtPal, these reads could be placed with greater attention to indels and a more realistic accounting for alignment uncertainty. Homology detection could be done in a similar way, thereby making use of the phylogenetic relationship of the sequences within the reference family. It has been observed that the detection of positive selection is highly sensitive to the alignment used [72]. ProtPal could be modified to detect selection using entire profiles rather than single alignments, potentially eliminating the bias brought on by an inaccurate alignment.

In summary, multiple alignments are frequently constructed for use in downstream evolutionary analyses. However, except for our method and slow-performing MCMC methods, there are no software tools for reconstructing molecular evolutionary history that explicitly maximize a phylogenetic likelihood for indels. Our results strongly indicate that algorithms such as ProtPal (which use such a phylogenetic model) produce significantly more reliable estimates of evolutionary parameters, which we believe to be highly indicative of evolutionary accuracy. These results falsify previous assertions that existing, non-phylogenetic tools are well-suited to this purpose. Furthermore, we have demonstrated that it is possible to achieve such accuracy without sacrificing asymptotic guarantees on time/memory complexity, or resorting to expensive MCMC methods. ProtPal can reconstruct phylogenetic histories of entire databases on commodity hardware, enabling the large-scale study of evolutionary history in a consistent phylogenetic framework.

3.4 Methods

The details concerning generation and analysis of simulated data are contained in Appendix B .

Felsenstein's algorithm for indel models

Our algorithm may be viewed as a generalization of Felsenstein's pruning recursion [7], a widely-used algorithm in bioinformatics and molecular evolution. A few common applications of this algorithm include estimation of substitution rates [30]; reconstruction of phylogenetic trees [31]; identification of conserved (slow-evolving) or recently-adapted (fast-evolving) elements in proteins and DNA [32]; detection of different substitution matrix "signatures" (e.g. purifying vs diversifying selection at synonymous codon positions [33], hydrophobic vs hydrophilic amino acid signatures [34], CpG methylation in genomes [35], or basepair covariation in RNA structures [36]); annotation of structures in genomes [37,38]; and placement of metagenomic reads on phylogenetic trees [39]. We present a compact description of our method in the context of Felsenstein's algorithm here, while a complete mathematical treatment can be found in Chapter 2.

Felsenstein's algorithm computes $P(S|T, \theta)$ for a substitution model by tabulating intermediate probability functions of the form $G_n(x) = P(S_n|x_n = x, \theta)$, where x_n represents the individual residue state of ancestral node n, and S_n represents all the sequence data that is causally descended from node n in the tree (i.e. the observed residues at the set of leaf nodes whose most recent common ancestor is node n).

The pruning recursion visits all nodes in postorder. Each G_n function is computed in terms of the functions G_l and G_r of its immediate left and right children (assuming a binary tree):

$$\begin{aligned} G_n(x) &= P(S_n|x_n = x, \theta) \\ &= \begin{cases} \left(\sum_{x_l} M^{(l)}_{x,\ x_l} G_l(x_l)\right) \left(\sum_{x_r} M^{(r)}_{x,\ x_r} G_r(x_r)\right) & \text{if } n \text{ is not a leaf} \\ \delta(x = S_n) & \text{if } n \text{ is a leaf} \end{cases} \end{aligned}$$

where $M^{(n)}_{ab} = P(x_n = b|x_m = a)$ is the probability that node n has state b, given that its parent node m has state a; and $\delta(x = S_n)$ is a Kronecker delta function terminating the recursion at the leaf nodes of the tree. These G_n functions are often referred to as "messages" in the machine-learning literature [40].

Our new algorithm is algebraically equivalent to Felsenstein's algorithm, if the concept of a "substitution matrix" over a particular alphabet is extended to the countably-infinite set of all sequences over that alphabet. Our chosen class of "infinite substitution matrix" is one that has a finite representation: namely, the *finite-state transducer*, a probabilistic automaton that transforms an input sequence to an output sequence, and a familiar tool of statistical linguistics [44].

By generalizing the idea of matrix multiplication (AB) to two transducers (A and B), and introducing a notation for feeding the same input sequence to two transducers in parallel ($A \circ B$), we are able to write Felsenstein's algorithm in a new form:

$$G_n = \begin{cases} \left(M^{(l)} G_l\right) \circ \left(M^{(r)} G_r\right) & \text{if } n \text{ is not a leaf} \\ \nabla(S_n) & \text{if } n \text{ is a leaf} \end{cases}$$

where $\nabla(S_n)$ is the transducer equivalent of the Kronecker delta $\delta(x = S_n)$. The function G_n is now encapsulated by a transducer "profile" of node n.

This representation has complexity $\mathcal{O}(L^N)$ for N sequences of length L, which we reduce to $\mathcal{O}(LN)$ by stochastic approximation of the G_n. This approximation relies on the *alignment envelope* [53], a data structure introduced by prior work on efficient alignment methods. The alignment envelope is a subset of all the possible histories in which most of the probability mass is concentrated. A related data structure is the *partial order graph* [43]. Both these data structures can be viewed as ensembles of possible histories, in contrast to a single "best-guess" reconstruction of the history. Figure 3.5 shows a state graph, with paths through it corresponding to histories relating the two sequences GL and GIV. The paths highlighted in blue form a partial order graph, corresponding to a subset of these histories generated by a stochastic traceback. At each progressive traversal step, we sample a high-probability subset of alignments of two sibling profiles in order to maintain a bound on the state space size. Note that if we sample only the most likely path at every internal node, we essentially recover the progressive algorithm of PRANK, and if we sample and store all solutions, we recover the machine G_n with state space of size $\mathcal{O}(L^N)$.

OPTIC data analysis

Data Amniote gene families were downloaded from `http://genserv.anat.ox.ac.uk/downloads/clades/`. We restricted our analysis to the $\sim$ 7,500 families having simple 1:1 orthologies. The same species tree topology (downloaded from `http://genserv.anat.ox.ac.uk/clades/amniota/displayPhylogeny` was used for all reconstructions, though branch lengths were estimated separately for each family as part of OPTIC. When computing branch-specific indel rates, the branch lengths of the species tree were used.

Reconstruction and rate estimation Gene families were aligned and reconstructed using ProtPal with a 3-rate class Markov chain over amino acids, insertion and deletion rates set to 0.01, and 250 traceback samples. Averaged and per-branch indel rates were computed with ProtPal using the `-pi` and `-pb` options. The indel rates were then normalized by the synonymous substitution rate for each corresponding nucleotide alignment (taken directly from OPTIC), computed with PAML [96]. Residues' origins were determined by finding the tree node closest to the root containing a non-gap reconstructed character.

External data Genes were mapped to Gene Ontology terms via the mapping downloaded from `http://www.ebi.ac.uk/GOA/human_release.html` during 10/2010. Indel SNPs per gene were taken from a table downloaded from Supplemental Table 5 of [98].

Additional supporting information

Appendix B contains techinical details concerning generation of simulation data, analysis of OPTIC data, as well as additional figures pertaining to both simulated and OPTIC data.

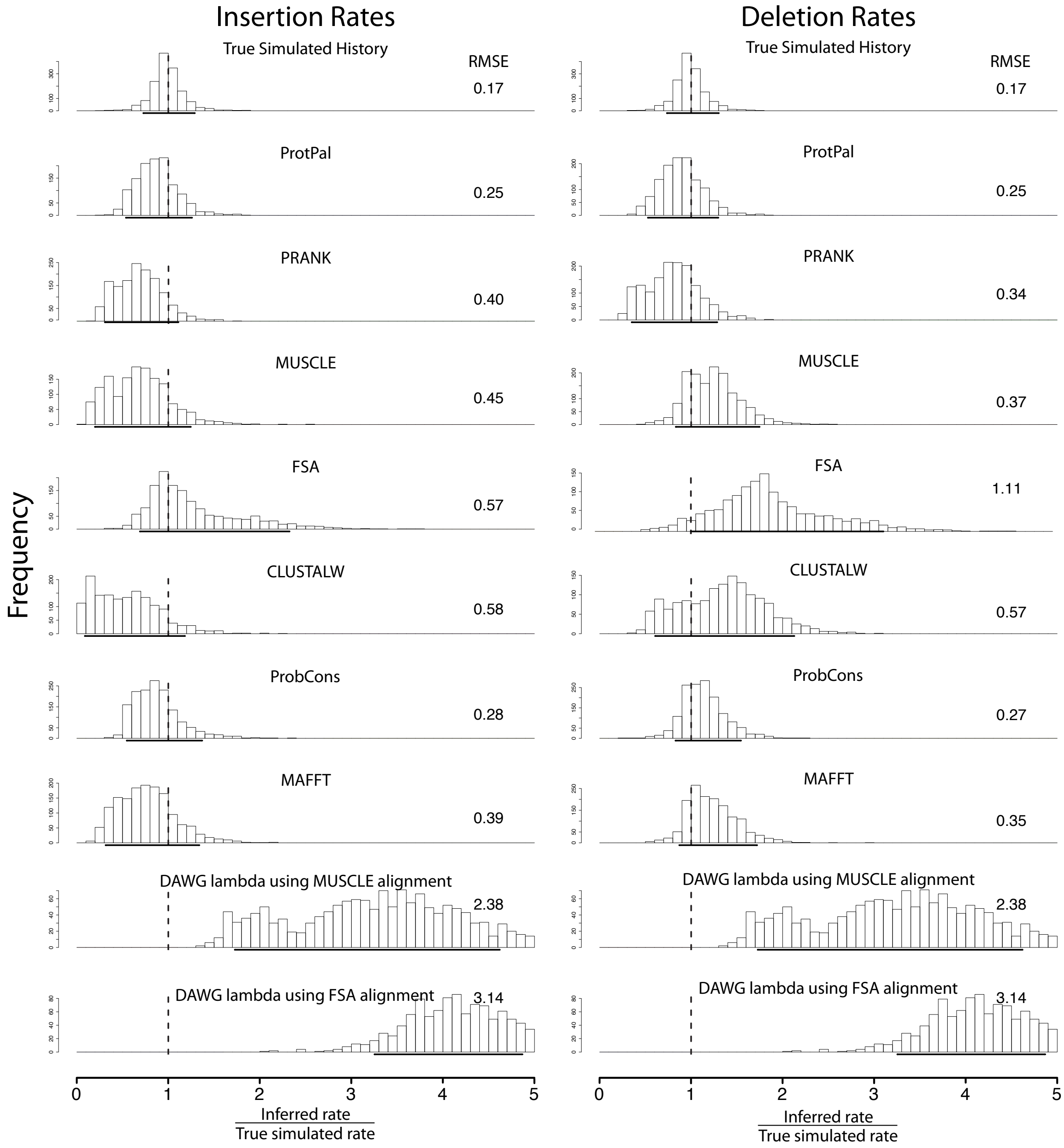


Figure 3.1: ProtPal's estimates of insertion and deletion rates are the most accurate of any program tested, as measured by the RMSE of $\frac{\hat{\lambda}^*_{\hat{H}}}{\lambda^*}$ values aggregated over all substitution/indel rate categories. Quantiles containing 90% of the data are shown as a bolded portion of the x-axis, and RMSE is shown to the right of each distribution, the latter computed as described in Appendix B Equation 1. No aligner approaches the accuracy of the rates estimated with the true alignment, though ProtPal, PRANK, and ProbCons are the top three, with ProtPal as the most accurate over all. Many aligners, particularly MUSCLE, CLUSTALW, and MAFFT, significantly underestimate insertion rates and overestimate deletion rates. ProtPal and PRANK perform their own ancestral reconstruction and other alignment programs were augmented with a most-recent-common-ancestor (MRCA) parsimony as described in [89].

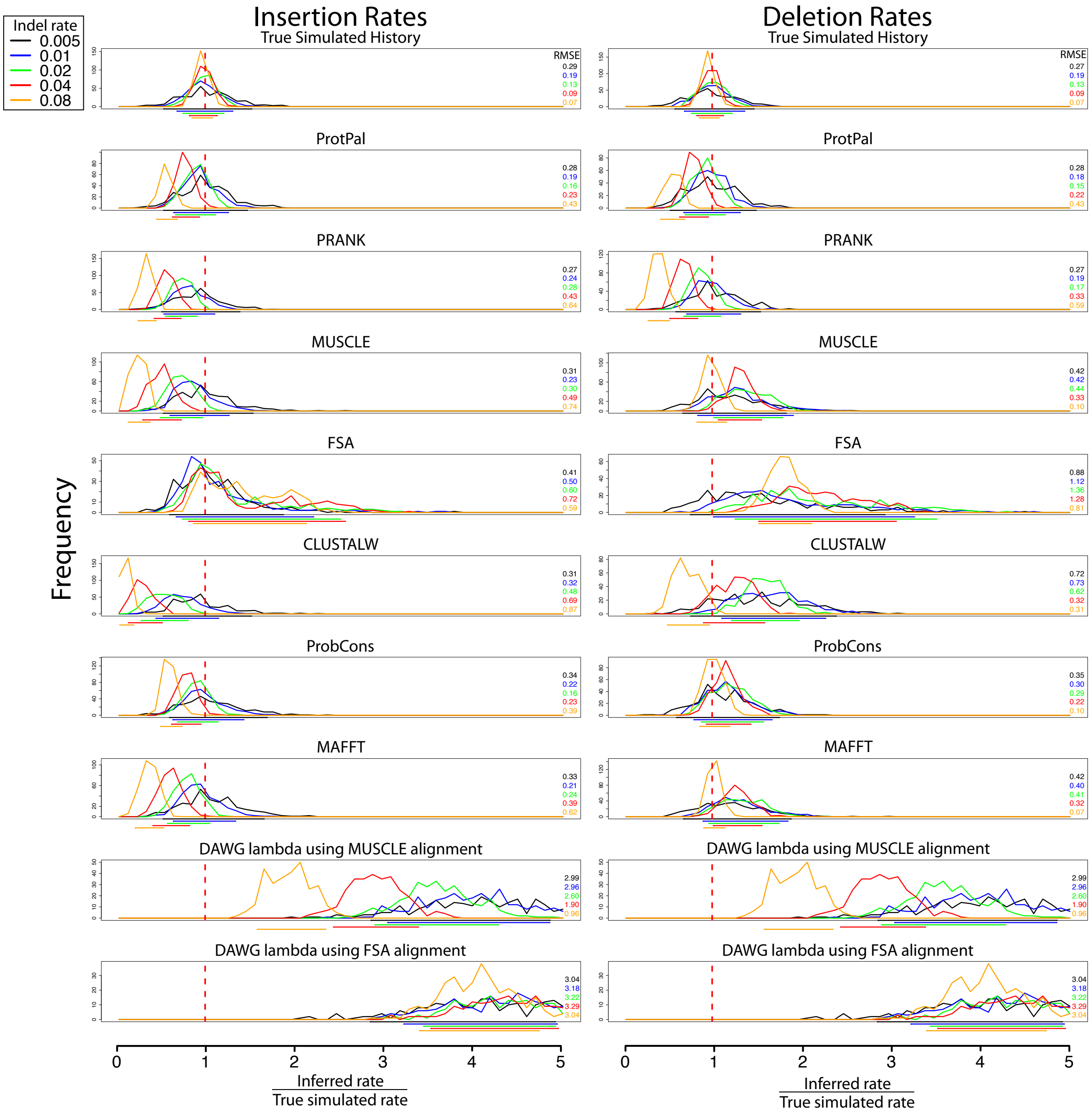
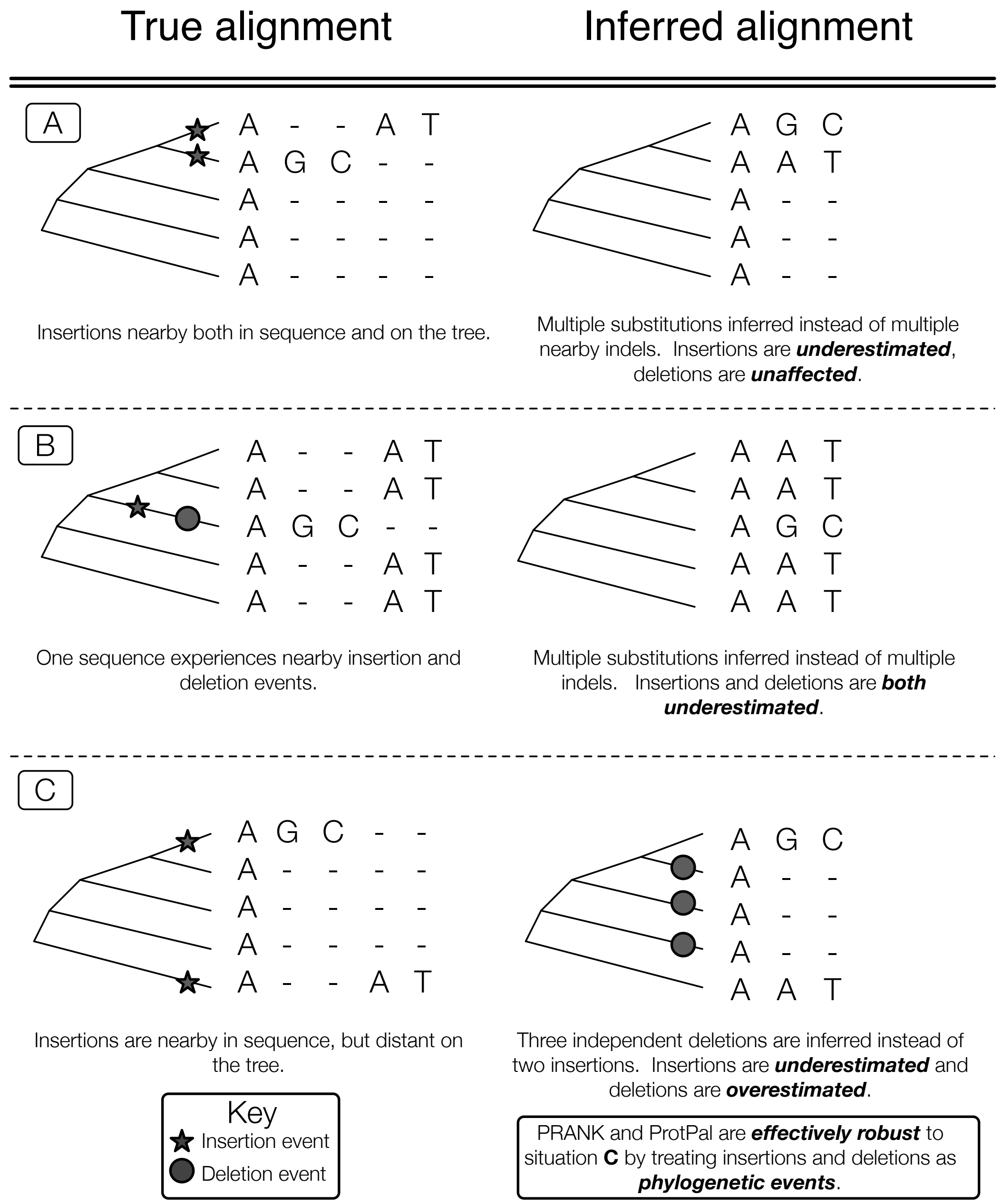


Figure 3.2: Rate estimation accuracy is highly dependent on the simulated indel rate. For instance, PRANK is more accurate at lower indel rates, ProbCons is more accurate at higher rates. ProtPal is more accurate than PRANK in all but one rate (0.005) and equal or more accurate than ProbCons in all but one rate (0.08). The drift towards $\frac{inferred}{true} = 0$ exhibited by most programs indicates that most programs infer proportionally fewer indels as rates are increased, likely due to various forms of gap attraction. Color-coded 90% quantiles and RMSEs are shown underneath and to the right of each group of distributions, respectively. RMSE is computed as described in Appendix B Equation 1



Insertions nearby both in sequence and on the tree.

Multiple substitutions inferred instead of multiple nearby indels. Insertions are ***underestimated***, deletions are ***unaffected***.

One sequence experiences nearby insertion and deletion events.

Multiple substitutions inferred instead of multiple indels. Insertions and deletions are ***both underestimated***.

Insertions are nearby in sequence, but distant on the tree.

Three independent deletions are inferred instead of two insertions. Insertions are ***underestimated*** and deletions are ***overestimated***.

PRANK and ProtPal are ***effectively robust*** to situation **C** by treating insertions and deletions as ***phylogenetic events***.

Figure 3.3: *Gap attraction*, the canceling of nearby complementary indels, can affect insertion and deletion rates in various ways depending on the phylogenetic relationship of the sequences involved. All programs are, to some extent, sensitive to situations **A** and **B** whereas phylogenetic aligners can avoid situation **C**. An insertion at a leaf requires gaps at all other leaves - an understandably costly alignment move when gaps are added without regard to the phylogeny, resulting in **multiple penalization** for each insertion. Such a penalization would cause most non-phylogenetic aligners to prefer the "Inferred alignment" in case **C** where there are fewer total gaps. Aligners treating indels as phylogenetic events would penalize each of the implied multiple deletions and only penalize each insertion once, thus preferring the "True alignment" in case **C**.

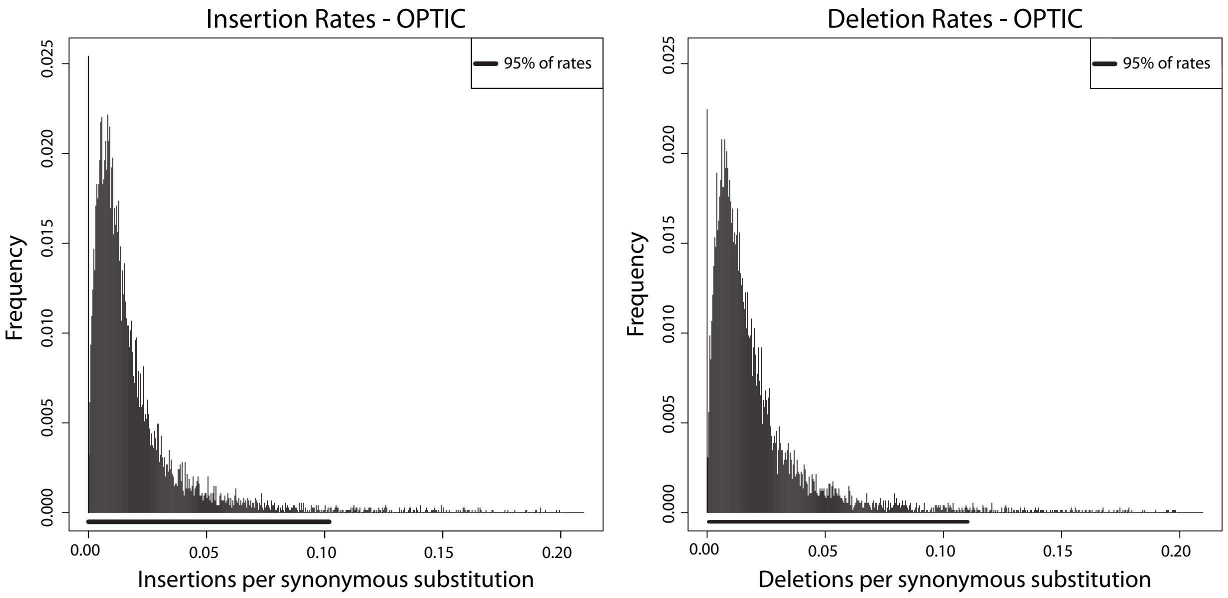


Figure 3.4: Insertion and deletion rates in *Amniota* show similar distributions, with 95% of genes having rates less than approximately 0.1 indels per synonymous substitution. Insertion and deletion rates were estimated using reconstructions done with ProtPal from a set of approximately 7,500 protein-coding genes from the OPTIC amniote database [91]. Indel rates were normalized by the synonymous substitution rate of each gene as computed with PAML [96] so that the plotted rate represents the number of expected indels per synonymous substitution. Since these rates are conditioned on the MAP reconstructed history, there are many alignments whose inferred indel rates are zero (197, 174, and 54 for insertions, deletions, and both, respectively).

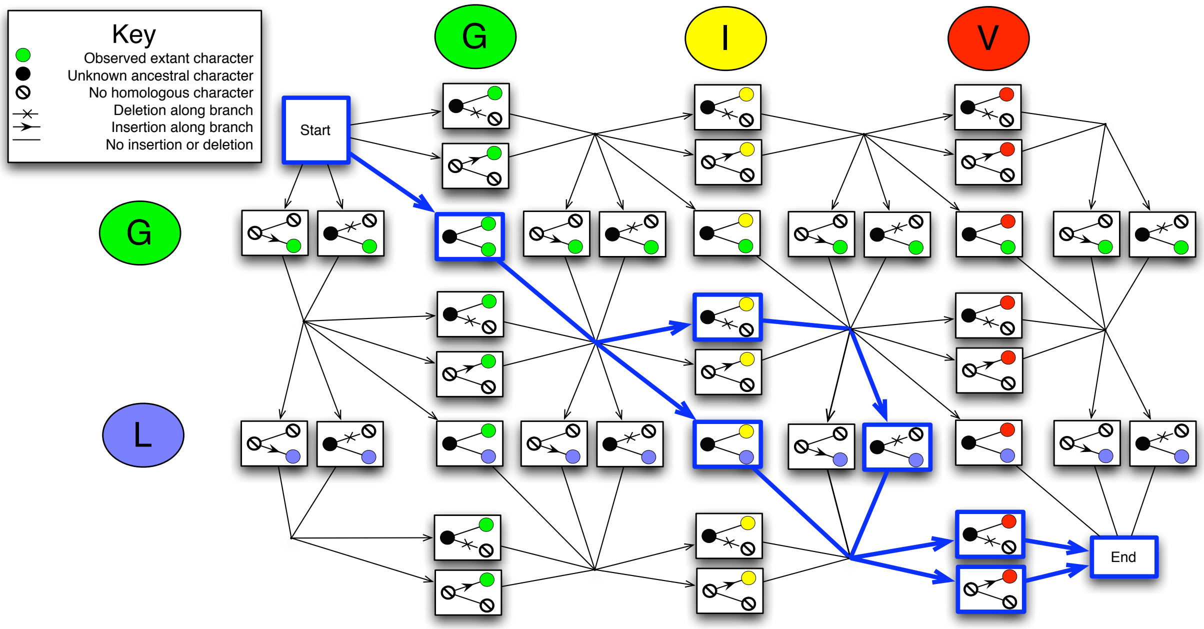


Figure 3.5: Each path through this state graph represents a possible evolutionary history relating sequences GL and GIV. By using stochastic traceback algorithms (sampling paths proportional to their posterior probability, blue highlighted states and transitions), it is possible to select a high-probability subset of the full state graph. By constructing such a subset at each internal node, it is possible to maintain a bound on the state space size during progressive tree traversal while still retaining an ensemble of possible histories.

Chapter 4

HandAlign: MCMC for exact Bayesian inference of alignment, tree, and parameters

The following section contains work conducted with Lars Barquist and Ian Holmes published in Bioinformatics [99].

Overview

We describe handalign, a software package for Bayesian reconstruction of phylogenetic history. The underlying model of sequence evolution describes indels and substitutions. Alignments, trees, and model parameters are all treated as jointly-dependent random variables and sampled via Metropolis-Hastings Markov chain Monte Carlo (MCMC), enabling systematic statistical parameter inference and hypothesis testing. handalign implements several different MCMC proposal kernels, allows sampling from arbitrary target distributions via Hastings ratios, and uses standard file formats for trees, alignments and models.

Availability and Implementation:

Installation and usage instructions are at http://biowiki.org/HandAlign

4.1 Background

Multiple sequence alignments constitute a central part of many bioinformatics workflows. Commonly, an alignment is built from primary sequences, a tree is reconstructed from this alignment, and various analyses are run using the alignment and/or tree.

This can be problematic for several reasons. First, inference of the tree and alignment is likely to be *uncertain*: many alternative trees or alignments may explain the data comparably well. Downstream analyses which assume the tree and alignment to be true ignore this uncertainty, and potentially inherit embedded bias and error [67, 73, 100–102].

Second, this flow of information is *circular*: alignment algorithms often (implicitly or explicitly) make use of a guide tree, while tree- and model-fitting algorithms typically use an alignment as input. This leads to a chicken-and-egg situation [73, 103, 104].

In attempted resolution of these problems, the field of **statistical alignment** methods unifies alignment and tree-building as related inference tasks under a phylogenetic likelihood function [42, 105–107]. The handalign software is one such tool, building on a range of prior work in this area [46, 108, 109].

4.2 Sampling alignments, trees, and parameters

handalign implements a Bayesian model of sequence phylogeny with separate substitution and indel components. To perform inference of unknown variables (i.e. trees, ancestral sequences, or parameters) under this model, handalign makes use of Markov chain Monte Carlo sampling (MCMC). We cannot directly observe the indel history (H), tree (T), or evolutionary model parameters (θ). However, we can estimate their *a posteriori* probability distribution, conditional on what we do observe: the extant sequences (S). That is, we aim to sample ($H, T, \theta|S$). Explicitly marginalizing H, T and θ is infeasibly expensive: there are combinatorially many trees T and histories H, and continuously-varying parameters θ. MCMC provides a powerful alternative way to sample from $(H, T, \theta|S)$ that is often not much more expensive than computing the joint likelihood of (H, T, θ, S).

Informally, MCMC randomly walks the space of (H, T, θ) tuples, the number of steps spent at a particular tuple converging to the posterior probability $P(H, T, \theta|S)$. The result is a series $\{(H_n, T_n, \theta_n)\}$ of samples from the posterior.

Depending on the investigator's goals, various analyses can then be performed using the collection of tuples $\{(H_n, T_n, \theta_n)\}$. The ensemble can be summarized with a single "consensus" alignment or tree,including confidence levels (e.g. the probability that a given subset of species form a monophyletic clade, or that a given column is correct) [108]. Alternatively, downstream computations can be averaged over the ensemble: the sampled parameters can be used to estimate modes and moments (e.g. the most likely indel rate), or detect signatures of interest (e.g. positive selection).

As well as MCMC, `handalign` can perform a stochastic search using the same underlying model, but returning a single best-guess (H, T, θ) rather than a collection of such tuples.

4.3 Capabilities

Given unaligned FASTA-format sequences, or (optionally) an "initial guess" in the form of a Stockholm-format alignment with an embedded Newick-format tree. `handalign` performs $N \times |S|$ sampling steps (where $|S|$ is the number of input sequences). Each step uses one of the MCMC kernel moves (described below) to update one of H, T, or θ. If requested (via a command-line option), the new tuple (H, T, θ) is logged to a file. If operated in "Stochastic search" mode, a greedy local search is performed every K samples to find the most likely nearby alignment. After $N \times |S|$ samples, the final (H, T, θ) tuple is output in Stockholm+Newick format.

Indel models: The insertion-deletion model is an affine-gap transducer approximation [110] to a Long Indel birth-death process [55] with insertion rate λ, deletion rate μ, deletion extension probability r. The approximation is that indel events never overlap on the same branch. Other indel length distributions, such as TKF91 [54] or mixture-geometric, can be used.

Substitution models: Any parametric continuous-time Markov chain can be used to model character evolution, via the file format of the companion program `xrate` [18]. For instance, $20N$-state amino acid models (with N-valued hidden states [111]) and 64-state codon models have been used. Parameters of these substitution models can be sampled, allowing alignment-free estimation of statistics such as K_a/K_s. Ancestral characters are summed out [46]; they can be imputed using `xrate`.

Tree prior: The prior over tree topologies is uniform, with a weak exponential prior over total branch length. Alternate priors can be implemented using the "Arbitrary target distribution" mechanism.

Arbitrary target distribution: `handalign` allows any tree/alignment probability model to be implemented over a Unix pipe and used in a Metropolis-Hastings accept/reject step.

MCMC kernel moves: The relative proportions of the various sampling moves can be set on the command line. All are variants of Gibbs-sampling moves. Some are full Gibbs (perfectly mixing the sampled variables at every step); others utilize Metropolis-within-Gibbs [112] or a variant of importance sampling that includes the current point in the list of accessible points. The individual moves vary in the dependence of their complexities on the input sequence length, L. The worst-case complexity with default settings is $\mathcal{O}(L^2)$, comparable to BaliPhy [108].

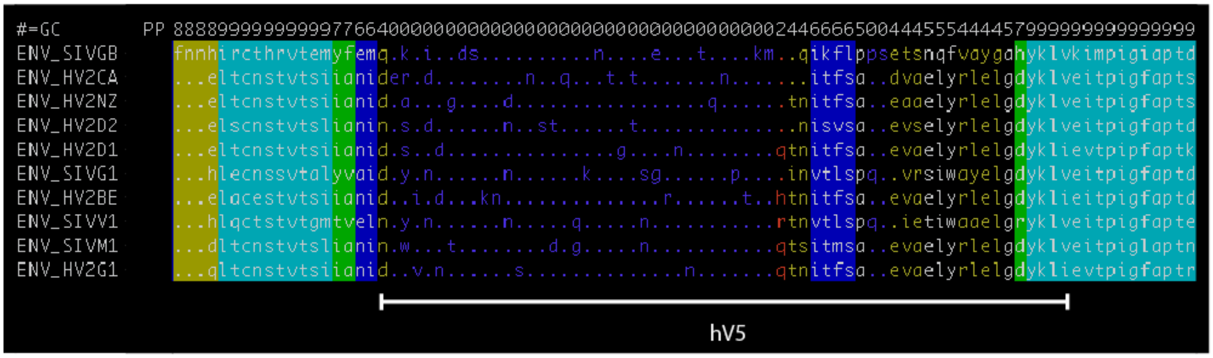
Stochastic search: `handalign` can be used to do a partially-randomized greedy search, yielding a relatively quick, approximate maximum likelihood estimate for the alignment and/or tree, in addition to a full MCMC trace. The *iterative refinement* command-line option interrupts the MCMC run periodically to perform a greedy (Viterbi) search for the locally-maximal alignment close to the current sample.

Alignment banding: As the DP matrix may be costly to fill, both in time and memory, users may specify an alignment "band" as a heuristic constraint. Command-line options can be used to prevent visiting cells more than M positions away from the current alignment path. This has the effect of causing indels longer than M to be excluded, but is otherwise ergodic, and generally converts an $\mathcal{O}(L^a)$ step into an $\mathcal{O}(LM^{a-1})$ step (for $a \geq 2$).

HMMoC adapter: `handalign` can optionally make use of the Hidden Markov Model Compiler `hmmoc` [113] to craft optimized C++ code for DP-based sampling steps. This typically speeds things up by a large constant factor.

MCMC trace analysis: The DART package includes several scripts for summarizing MCMC traces. `constock.pl` finds a consensus alignment and uses ANSI terminal color to render posterior probabilities of individual columns (Figure 4.1). Alternatively, trees can be extracted using `stocktree.pl` and a separate program, such as CONSENSE in the PHYLIP package, used to estimate consensus trees. `handalign` sampling traces use Stockholm alignment format to embed trees and parameters. A trace can be rendered as an ANSI terminal color animation using `stockfilm.pl`, converted into other common formats (see `http://biowiki.org/StockholmTools`) or the parameters extracted and their distribution analyzed (Figure 4.2).

Current limitations and performance: The $\mathcal{O}(L^2)$ complexity may be limiting for longer sequences (e.g. genomes); alignment banding should ameliorate this (but its effect on mixing performance is untested). Underflow/precision issues may potentially be an issue with larger trees. A ongoing compilation of comparison tests is here: `http://biowiki.org/HandAlignBenchmark`



```
#=GC       PP 8888999999999997766400000000000000000000000000000000000002446666500444555444457999999999999999
ENV_SIVGB     fnnhircthrvtemyfemq.k.i..ds.........n....e...t....km..qikflppsetsnqfvaygahyklvkimpigiaptd
ENV_HV2CA     ...eltcnstvtsiianider.d........n..q...t.t.......n.......itfsa..dvaelyrlelgdyklveitpigfapts
ENV_HV2NZ     ...eltcnstvtsiianid.a...g....d................q......tnitfsa..eaaelyrlelgdyklveitpigfapts
ENV_HV2D2     ...elscnstvtslianin.s.d......n..st......t.............nisvsa..evselyrlelgdyklveitpigfaptd
ENV_HV2D1     ...eltcnstvtsiianid.s..d...............g....n........qtnitfsa..evaelyrlelgdyklievtpipfaptk
ENV_SIVG1     ...hlecnssvtalyvaid.y.n......n......k....sg......p....invtlspq..vrsiwayelgdyklveitpigfaptd
ENV_HV2BE     ...elacestvtsiianid..i.d...kn..............r......t..htnitfsa..evaelyrlelgdyklieitpigfaptd
ENV_SIVV1     ...hlqctstvtgmtveln.y.n......n.....q.....n...........rtnvtlspq..ietiwaaelgryklveitpigfapte
ENV_SIVM1     ...dltcnstvtslianin.w...t........d.g.....n...........qtsitmsa..evaelyrlelgdyklveitpiglaptn
ENV_HV2G1     ...qltcnstvtsiianid..v.n......s..............n........qtnitfsa..evaelyrlelgdyklievtpigfaptr
```

Figure 4.1: A summary alignment of SIV/HIV gp120 proteins produced by `constock.pl`. Posterior probabilities of alignment columns are shown on the "PP" line (most significant decimal digit) and by ANSI terminal color (white-on-cyan is most reliable, blue-on-black is least). Hypervariable (hV) region 5 [114] corresponds with a low-confidence region.

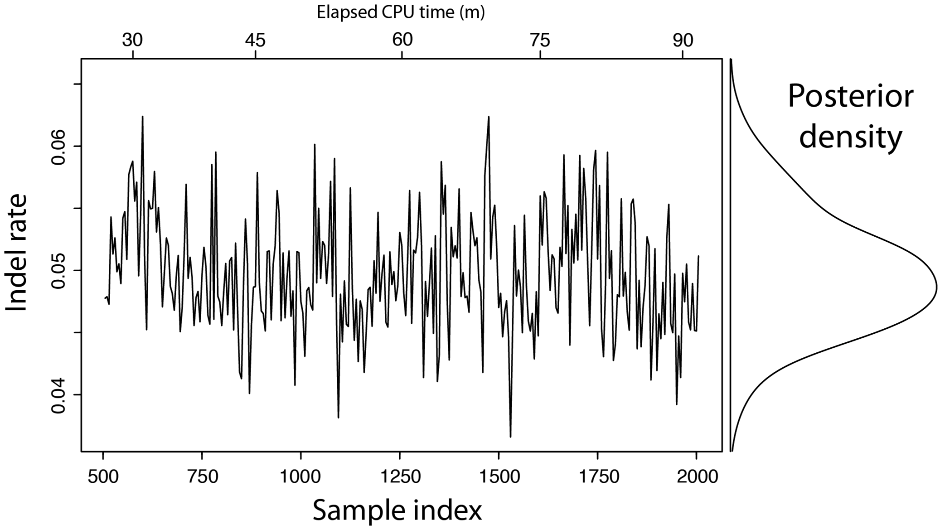


Figure 4.2: The indel rate of the SIV/HIV gp120 protein has most of its probability mass concentrated between 0.04 and 0.06 indels per substitution. `handalign` was run for 90 minutes on a 3.4 GHz CPU, generating 2000 samples (500 discarded as burn-in). Every fifth sample is plotted; the entire trace was used to estimate the density.

Part II

Methods utilizing a multiple sequence alignment

Chapter 5

Predicting the functional effects of polymorphisms using mutation rate

Overview

Predicting the phenotype resulting from non-synonymous mutations is an important goal of genomic sequence analysis. Computational approaches to this problem have made use of biochemical, structural, or comparative information in order to predict whether or not a mutation will impair the protein's function. In this work we employ phylogenetic models for sequence evolution to predict the mutability at each position of a protein, using the mutation rate as an informative statistic. Using four large-scale mutagenesis datasets, we evaluate our approach next to leading methods, finding that ours is the most accurate among the programs and datasets tested. More datasets are urgently needed to robustly evaluate the growing number of tools for this sort of analysis.

5.1 Background

Large-scale sequencing projects are steadily discovering single nucleotide polymorphisms (SNPs) that underlie human genetic diversity [115–117]. Beyond providing insights into human evolution and ancestry, these SNPs could provide clues to the genetic basis underlying diseases. Associating SNPs with disease is the goal of the so called genome-wide association study (GWAS): given a set of haplotypes annotated with disease conditions, determine which SNPs are statistically associated with diseases. For many diseases, relevant genetic mutations have been discovered using statistical GWAS studies [8–11]. However, SNPs may coincidentally lie on the same haplotype as a disease-causing SNP, or otherwise be unrelated to its predicted disease.

A complementary approach involves predicting the functional effect that a given SNP will have on its protein, without knowledge of associated disease. Popular methods like SIFT [12], PolyPhen [13], PMut [14], and ASP [15] use computations involving structural, comparative, or evolutionary features to separate SNPs into neutral (do not affect function) and deleterious (disrupt function) classes.

Structural predictors use biochemical or mechanistic models to predict whether a mutation is likely to impair function, either via misfolding or modification of key residues. Comparative analysis (e.g. SIFT [12]) uses related sequences, with the idea that if particular mutations are accepted in homologs (which are assumed functional), then they are likely to be tolerated in the query sequence. Evolutionary approaches take comparative analysis one step further, incorporating the phylogenetic relationship between the query's homologs in the mutability prediction. For instance, the ASP statistic measures how long ago (in evolutionary time) a given position experienced a mutation [15].

Our method extends the phylogenetic approach of ASP by using models for sequence evolution to estimate the mutation *rate* for each column of a query-homologs alignment. Modeling variations in mutation rate to detect regions of interest is a common technique in phylogenetics, and has been used to detect conserved regions, lineage-specific selection [118], and evolutionary differences in HIV subtypes [119]. In applying mutation rate measurements to SNP analysis, the high level idea is that a faster evolutionary rate correlates with tolerance to mutation: if the evolutionary process has “tried” and accepted many mutations at a given position, it is more likely that mutation of the current sequence will be tolerated.

Our approach is distinct from SIFT in that it leverages phylogenetics to account for the non-independence of the selected homologs. It may often be the case that homologous sequences do not lie at perfectly uniform evolutionary distance from the query, and by modeling the evolutionary relationships between them, this bias need not affect the resulting predictions. For instance, in Figure 5.1, related sequences display large divergence but slow mutation rates at the position in question. SIFT would predict greater mutability at this position than would a mutation rate-based score. Our method is distinct from the ASP score as our quantitative measure is based on the entire tree relating all the alignment’s sequences as opposed to focusing only on the query’s lineage. Further, we estimate the mutation rate as a parameter to a probabilistic model, rather than using a discrete tree-dependent count.

Despite the simplicity of the current model used, our approach performs best on the four datasets we investigated, leading us to believe that mutation rate is an informative quantity in predicting SNP phenotype. It is possible that other phylogenetic measures or models could be applied to this problem. For instance, features such as protein secondary structure and differences in amino acid sizes could be incorporated in a straightforward way.

It is difficult to draw robust conclusions from so few datasets, and so we hope that the improvement of SNP phenotype prediction methods spurs the creation of more datasets of this kind.

The model used to estimate mutation rates is available as an XRate grammar as part of the DART package [18], available via git: `https://github.com/ihh/dart`. The script used to convert a grammar to a list of SNP mutability scores is also included in DART.

5.2 Results

We tested four programs on four datasets (as described in Materials and Methods) in order to determine the relative accuracy of phylogenetic measurements to predict mutation tolerance. The methods tested were SIFT [12], PMut [14], ASP [15], and our XRate-based rate measurement ColumnRates. The four publicly-available SNP-phenotype datasets used were HIV protease [120], lysozyme [121], LacI [122], and CBS [123]. For each of the datasets, the alignment that SIFT generates was used for the XRate and ASP analyses, so that this variable would be unable to contribute to differences between them. Since ASP and ColumnRates return quantitative measures rather than dichotomous results, score cutoffs separating neutral and deleterious SNPs were estimated for each dataset using the remaining three datasets. PMut and SIFT were run using default settings. After each of the methods was run on the datasets, the predictions were compared against the measured phenotypes, with the results summarized via the BER statistic, shown in Figure 5.2. Additional detail on the testing setup is provided in Materials and Methods, and performance measured using other summary statistics is shown in Appendix C.

Overall accuracy

Figure 5.2 displays the accuracies of the four programs on each of the four datasets, as computed by the balanced error rate (BER) statistic. The BER measure (defined in Methods) was used since the datasets have different ratios of functional to nonfunctional mutations. Most methods lie between 0.6 and 0.8, with the exception of our implementation of the ASP metric, which categorizes only approximately 50% of SNPs correctly. It should be emphasized that since we were unable to find a general-purpose implementation of ASP, we implemented our own version following the description in [15] as closely as possible. Thus, it is possible that certain aspects of their method are missing in our implementation, which could contribute to its poor performance.

Despite this caveat, the ASP score does remarkably poorly on all the datasets tested. When optimizing the score cutoff separately for each dataset (shown in Appendix C), ASP does much better, outperforming SIFT on the Lysozyme and LacI datasets. This suggests that ASP is sensitive to the specified score cutoff, which may limit its applicability to proteins with no training data. In [15], the ASP score cutoff was set to ensure categorizing all the functional SNPs correctly, which essentially amounts to using 100% of the test data and maximizing the True Negative Rate during training. Further, their dataset on the MTHFR protein contained only 30 SNPs - too few to allow statistically reliable conclusions. These two factors may have contributed to the underestimation of ASP's robustness.

The accuracy of PMut [14] lies between ASP and SIFT on all datasets. ColumnRates is slightly more accurate than SIFT on all datasets. The quantities that SIFT and ColumnRates measure are different but related. SIFT estimates the prevalence of a given residue at a particular position given a collection of aligned related sequences and a prior distribution over residues. ColumnRates measures the evolutionary rate of a particular position given an alignment of related sequences and a tree relating them. Intuitively, a higher mutation rate will likely give rise to more diversity at the leaves of the tree (which is what SIFT measures). Depending on the phylogenetic relationship between the sequences, however, the opposite may not always be true: sequence diversity among observed sequences could be due to *divergence* rather than frequent mutations. Figure 5.1 shows examples of cases where SIFT, ASP, and ColumnRates may make differing predictions. ColumnRates's integration of phylogenetic relationships into its predictions may contribute to its increased accuracy relative to SIFT.

5.3 Discussion

We have presented a new tool for the prediction of SNP phenotypes which uses uses the phylogenetic mutation rate at each position as the mutability score. This method uses XRate,

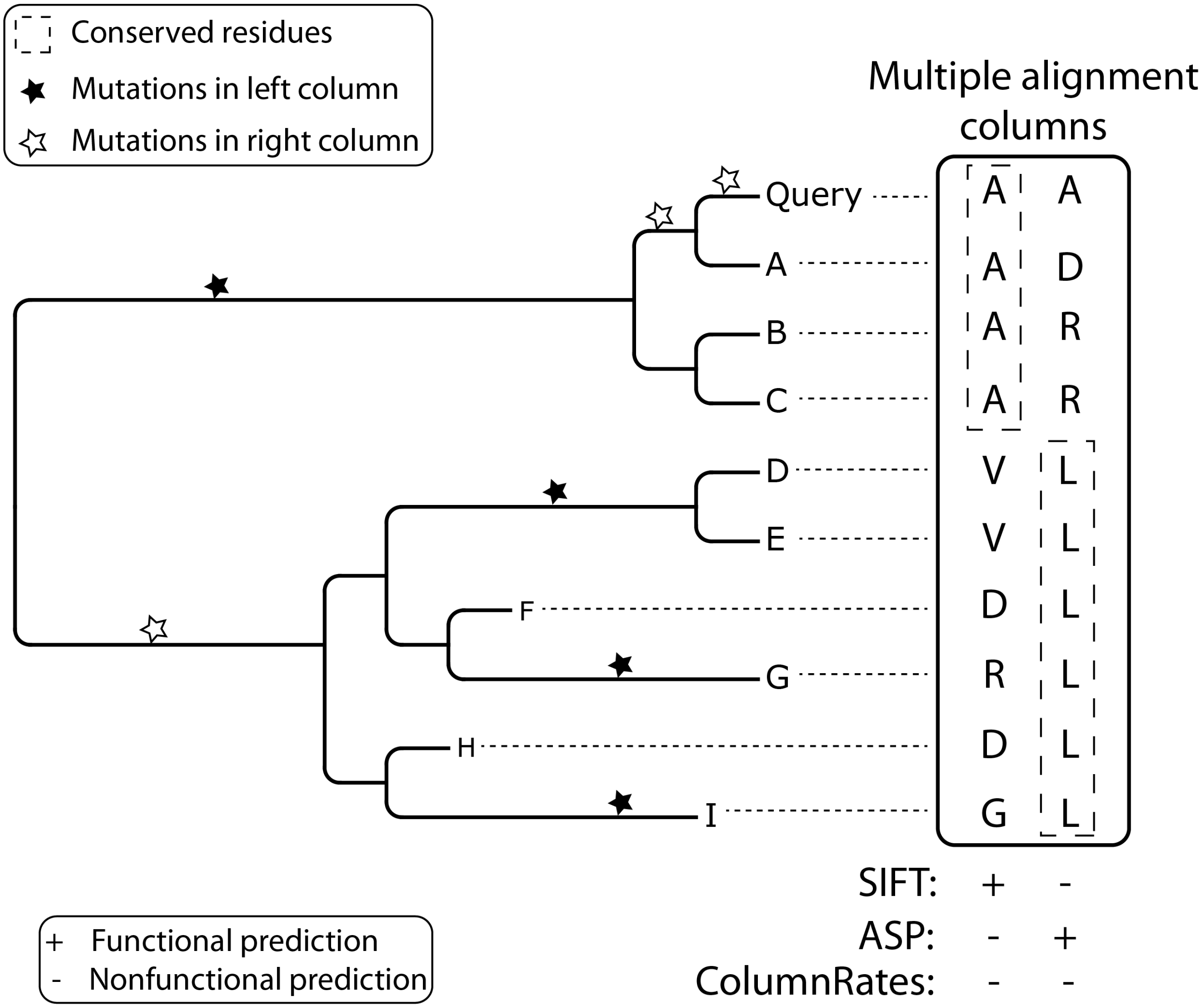


Figure 5.1: SIFT, ASP, and ColumnRates differ in the ways they use comparative sequence information to predict SNP phenotype, often resulting in different predictions, indicated with + and - symbols under each of the columns. The stars indicate possible locations of mutations required to explain the diversity in the multiple alignment at right. The left column displays high diversity of characters, causing SIFT to predict mutation tolerance. Viewed in a phylogenetic context, the mutations required for the observed diversity all lie on relatively long branches, causing ColumnRates to estimate a slow mutation rate, leading to a prediction of mutational sensitivity. In the right column, ASP and ColumnRates make different predictions: the multiple mutations in the root-Query path cause ASP to predict mutational tolerance, whereas ColumnRates's whole-tree measurement makes use of the strong conservation in the D-I clade, measuring a low mutation rate and hence predicting mutational sensitivity.

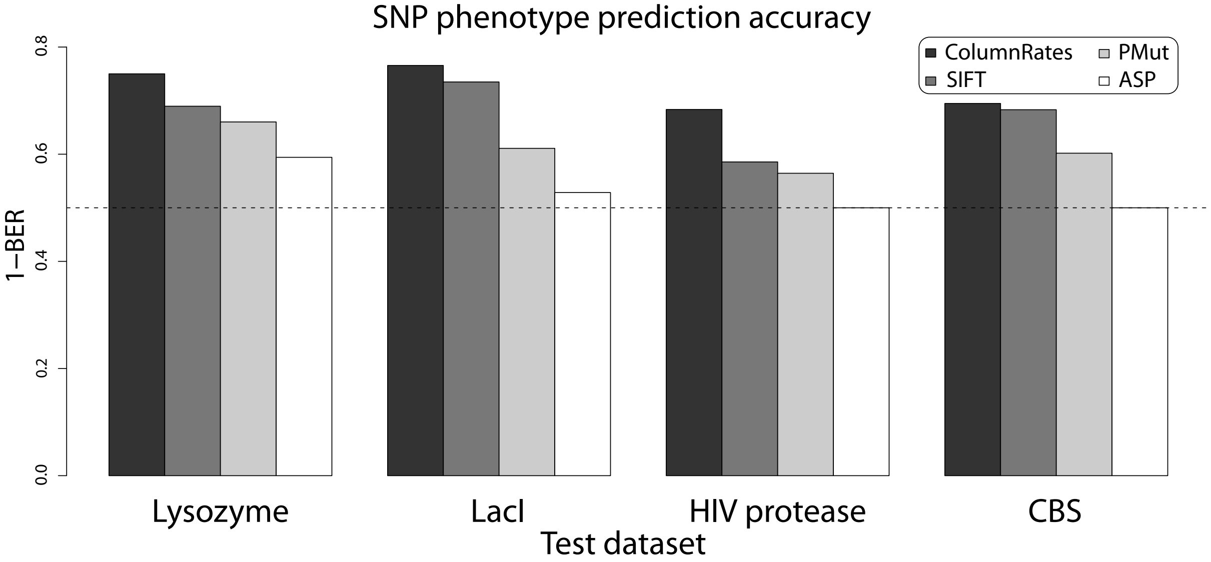


Figure 5.2: In a comparison using four publicly-available datasets, phylogenetic measurements outperform other tested methods on all datasets. The balanced error rate (BER) was calculated for each method and dataset as described in Materials and Methods. The horizontal dashed line indicates BER=0.5, the expected accuracy if SNPs are randomly classified as neutral or deleterious.

a general-purpose phylogenetic modeling package capable of implementing a wide class of user-specified models [18, 124]. We evaluate the performance of using one such phylogenetic model next to leading phenotype prediction methods. We find that despite its simplicity and lack of biochemical/structural awareness, it performs remarkably well, achieving the highest accuracy (BER statistic) on all of the four datasets.

Diversity of evolutionary patterns and rates across proteins and the extreme paucity of datasets leads to weak statistical power in evaluating prediction programs' relative accuracy. Still, the ranking of these programs is remarkably consistent across the four tested proteins. ColumnRates's performance on these data lead us to conclude that phylogenetic measurements are a promising avenue for SNP phenotype prediction. Especially attractive is the ease with which the models used could be extended to include other informative features. Phylogenetic models have been used to model protein secondary structure [125–127], amino acid size [128], and even experimental data [124], and it would be feasible to integrate such features into a phenotype prediction model within XRate's framework. We hope that these results spur further developments in evolutionary phenotype prediction methods as well as the creation of additional datasets of this kind.

5.4 Materials and methods

Datasets used

We evaluated four methods (SIFT [12], PMut [14], ASP [15], and ColumnRates), on the following four datasets: Lysozyme [121], LacI [122], HIV protease [120] and CBS [123]. Lysozyme, LacI, and HIV protease SNP data were downloaded from the SIFT website: `http://sift.bii.a-star.edu.sg/`. CBS data was downloaded from the supplementary info of [123]. The datasets had functional and nonfunctional mutations according to the distribution shown in Table 5.1.

Dataset	Functional	Nonfunctional
Lysozyme [121]	1377	175
LacI [122]	2267	1166
HIV protease [120]	111	159
CBS [123]	125	79

Table 5.1: Four SNP-phenotype datasets were used for comparision. Since the datasets had varying sizes and functional-nonfunctional ratios, we used the balanced error rate (BER) statistic in evaluating the prediction methods.

Optimizing score cutoffs

For a queried mutation, each program returns either a binary value or a continuously-valued score. For those returning a numerical score, a “cutoff” separating functional and nonfunctional predictions must be determined. This is done by partitioning the data into training and test sets - cutoffs are optimized on the training set and the method is evaluated using the test set. Given a training set (a collection of SNPs scores and phenotypes), the cutoff is chosen so as to maximize a given statistic - in this case the Balanced Error Rate (defined in Table 5.3). The cutoffs for ASP and ColumnRates were determined in this way; for SIFT we used the published cutoff (0.05), and PMut natively returns binary results.

For the data shown in Figure 5.2, a cutoff for each dataset was determined using the remaining datasets as a training set. This is intended to mimic the situation where we are analyzing a protein with no SNP phenotype data: we use all the data we do have to estimate a score cutoff that will (hopefully) be accurate on the protein of interest.

It can also be informative to see how each method performs when the cutoff is optimized for each individual dataset. To investigate this we trained cutoffs for ASP and ColumnRates using 50% of each data as training (and the remainder as test data). The BER scores for each dataset using these optimized cutoffs are shown in Appendix C.

Term	Definition
True positives (TP)	Deleterious mutations predicted as deleterious
True negatives (TN)	Neutral mutations predicted as neutral
False positives (FP)	Neutral mutations predicted as deleterious
False negatives (FN)	Deleterious mutations predicted as neutral
$P_{exp} = (TP + FN)$	Total number of experimentally-determined deleterious mutations
$N_{exp} = (TN + FP)$	Total number of experimentally-determined neutral mutations
$P_{pred} = (TP + FP)$	Total number of predicted deleterious mutations
$N_{pred} = (TN + FN)$	Total number of predicted neutral mutations

Table 5.2: Definitions of counts relevant to computing accuracy statistics, following [123]

Statistic	Abbreviation	Formula
Total accuracy	ACC	$\frac{TP+TN}{P_{exp}+N_{exp}}$
True positive rate	TPR	$\frac{TP}{P_{exp}}$
Positive predictive value	PPV	$\frac{TP}{P_{pred}}$
True negative rate	TNR	$\frac{TN}{N_{exp}}$
Negative predictive value	NPV	$\frac{TN}{N_{pred}}$
Balanced error rate	BER	$\frac{(1-TNR)+(1-TPR)}{2}$

Table 5.3: We define the following accuracy statistics following [123]. We believe the BER to be the most illustrative and reliable since the datasets used have differing ratios of functional/nonfunctional mutations (listed in Table 5.1); we display only BER in Figure 5.2. For specific applications, other statistics may be of greater interest, and so in Appendix C all of these metrics are shown for each dataset.

Testing and evaluation

Following [123], we define the counts in Table 5.2 and accuracy statistics in Table 5.3.

Performance measured using other summary statistics is shown in Appendix C.

XRate grammars

ColumnRates simply involves training an XRate grammar (`single-rate-rind.eg`, available as part of the DART package) on a multiple alignment of the query sequence along with homologous sequences (referred to as `alignment.stk` below) and converting the grammar into a common mutability score format. XRate uses the EM algorithm to train grammars' probability and rate parameters. Users need only specify the grammars to be trained; all parameter estimation and inference algorithms are internally implemented in XRate.

The grammar `single-rate-rind.eg` models sequence evolution along a tree with a continuous-time Markov chain with rate matrix with entries $Q_{ij} = r_c \pi_{cj}$ where i and j are residues, r_c is a column-specific mutation rate, and π_{cj} is the equilibrium frequency of character j in column c. That is, the model assumes that the substitution rate and equilibrium distribution differ at each column independently, and these are to be estimated when the grammar is trained. The following command creates the trained grammar `trained.eg`, which contains a mutation rate for each column:

```
xrate alignment.stk -g single-rate-rind.eg -t trained.eg -obl --noannotate
```

Using a simple script we convert the trained grammar to a SNP file where each row stores the WT residue, position, SNP, and score:

```
cat trained.eg | python parseRateGrammar.py alignment.stk
```

This outputs a file wherein each line contains the mutability score for a certain position in the reference sequence. For instance, the following line indicates the mutation from A to T at position 74 has score 0.01:

```
A74T 0.01
```

Chapter 6

Modeling genomic features using phylo-grammars

The following section contains work conducted with Ian Holmes published in PLoS ONE [124].

Overview

Modeling sequence evolution on phylogenetic trees is a useful technique in computational biology. Especially powerful are models which take account of the heterogeneous nature of sequence evolution according to the "grammar" of the encoded gene features. However, beyond a modest level of model complexity, manual coding of models becomes prohibitively labor-intensive.

We demonstrate, via a set of case studies, the new built-in model-prototyping capabilities of XRate (macros and Scheme extensions). These features allow rapid implementation of phylogenetic models which would have previously been far more labor-intensive. XRate's new capabilities for lineage-specific models, ancestral sequence reconstruction, and improved annotation output are also discussed.

XRate's flexible model-specification capabilities and computational efficiency make it well-suited to developing and prototyping phylogenetic grammar models. XRate is available as part of the DART software package: `http://biowiki.org/DART` .

6.1 Background

Phylogenetics, the modeling of evolution on trees, is an extremely powerful tool in computational biology. The better we can model a system, the more can learn from it, and vice-versa. Especially attractive, given the plethora of available sequence data, is modeling sequence evolution at the molecular level. Models describing the evolution of a single nucleotide began simply (e.g. JC69 [65]), later evolving to capture such biological features as transition/transversion bias (e.g. K80 [129]) and unequal base frequencies (e.g. HKY85 [66]). Felsenstein's "pruning" algorithm allows combining these models with phylogenetic trees to compute the likelihood of multiple sequences [7].

As powerful as phylogenetic models are for explaining the evolutionary depth of a sequence alignment, they are even more powerful when combined with a model for the feature structure: the partition of the alignment into regions, each evolving under a particular model. The phylogenetic grammar, or "phylo-grammar", is one such class of models. Combining hidden Markov models (and, more generally, stochastic grammars) and phylogenetic substitution models provides computational modelers with a rich set of comparative tools to analyze multiple sequence alignments (MSAs): gene prediction, homology detection, finding structured RNA, and detecting changes in selective pressure have all been approached with this general framework [36, 130–132]. Readers unfamiliar with phylo-grammars may benefit from relevant descriptions and links available here: `http://biowiki.org/PhyloGrammars` or the original paper describing XRate [18]. Also, a collection of animations depicting various evolutionary models at work (generating multiple alignments or evolving sequences) has been compiled here: `http://biowiki.org/PhyloFilm` .

While the mathematics of sequence modeling is straightforward, manual implementation can quickly become the limiting factor in iterative development of a computational pipeline. To streamline this step, general modeling platforms have been developed. For instance, Exonerate allows users to specify a wide variety of common substitution and gap models when aligning pairs of sequences [133]. Dynamite uses a specification file to generate code for dynamic programming routines [134]. HMMoC is a similar model compiler sufficiently general to work with arbitrary HMMs [113]. The BEAST program allows users to choose from a wide range of phylogenetic substitution models while also sampling over trees [135]. The first three of these are non-phylogenetic, only able to model related pairs of sequences. Dynamite and HMMoC are unique in that they allow definition of arbitrary models via specification files, whereas users of BEAST and Exonerate are limited to the range of models which have been hard-coded in the respective programs.

Defining models' structure manually can be limiting as models grow in size and/or complexity. For instance, a Nielsen-Yang model incorporating both selection and transition/transversion bias has nearly 4000 entries - far too many for a user to manually specify [33]. Such a large matrix requires specific model-generating code to be written and integrated with the program in use - not always possible or practical for the user depending on the program's implementation.

XRate is a phylogenetic modeling program that implements the key parameterization and inference algorithms given two ingredients: a user-specified phylo-grammar, and a multiple sequence alignment. (A phylogeny can optionally be specified by the user, or it can be inferred by the program.) XRate's models describe the parametric structure of substitution rate matrices, along with grammatical rules governing which rate matrices can account for which alignment columns. This essentially amounts to partitioning the alignment (e.g. marking up exon boundaries and reading frames) and factoring in the transitions between the different types of region.

Parameter estimation and decoding (alignment annotation) algorithms are built in, allowing fast model prototyping and fitting. Model training (estimating the rate and probability parameters of the grammar) is done via a form of the Expectation Maximization (EM) algorithm, described in more detail in the original XRate paper [18]. Most recently, XRate allows programmatic model construction via its macros and Scheme extensions. XRate's built-in macro language allows large, repetitive grammars to be compactly represented, and also enables the model structure to depend on aspects of the data, such as the tree or alignment. Scheme extensions take this even further, interfacing XRate to a full-featured functional scripting language, allowing complex XRate-oriented workflows to be written as Scheme programs.

In this paper we demonstrate XRate's new model-specification tools via a set of progressively more complex examples, concluding with XDecoder, a phylo-grammar modeling RNA secondary structure overlapping protein-coding regions. We also describe additional improvements to XRate since its initial publication, namely ancestral sequence reconstruction, GFF/WIG output, and hybrid substitution models. Finally, we show how XRate's features are exposed as function extensions in a dialect of the Scheme programming language, typifying a Functional Programming (FP) style of model development and inference for phylogenetic sequence analysis. Terminology relevant to modeling with XRate are defined in detail in Appendix E. We also provide an online tutorial for making nontrivial modifications to existing grammars, going step-by-step from a Jukes-Cantor model to an autocorrelated Gamma-distributed rates phylo-HMM: `http://biowiki.org/XrateTutorial`.

6.2 Methods

The XRate generative model

A phylo-grammar generates an alignment in two steps: nonterminal transformations and token evolution. The sequence of nonterminal transformations comprises the "grammar" portion of a phylo-grammar, and the "phylo" portion refers to the evolution of tokens along a phylogeny. First, transformation rules are repeatedly applied, beginning with the `START` nonterminal, until only a series of pseudoterminals remains. From each group of pseudoterminals (a group may be a single column, two "paired" columns in an RNA structure, or a codon triplet of columns), a tuple of tokens is sampled from the initial distribution of the chain corresponding to the pseudoterminal. These tokens then evolve down the phylogenetic tree according to the mutation rules of the chain, resulting in the observed alignment columns.

If the nonterminal transformations contain no bifurcations and all emissions occur on the same side of the nonterminal, the grammar is a phylogenetic hidden Markov model (phylo-HMM), a special subclass of phylo-grammars. Otherwise, it is a phylogenetic stochastic context-free grammar (phylo-SCFG), the most general class of models implemented by XRate. This distinction, along with other related technical terms, are described in greater detail in Appendix E, the Glossary of XRate model terminology.

The generality of XRate requires a slight tradeoff against speed. Since the low-level code implementing core operations is shared among the set of possible models, XRate will generally be slower than programs with source code optimized for a narrower range of models. Computing the Felsenstein likelihood under the HKY85 [66] model of a 5-taxon, 1Mb alignment, XRate required 1.25 minutes of CPU time and 116MB RAM, while PAML required 9 seconds of CPU time and 19MB RAM for the same operation. Running PFOLD [136] on a 5-taxon, 1KB alignment required 11 seconds and 164MB RAM, and running XRate on the same alignment with a comparable grammar required 25 seconds and 62MB RAM. All programs were run with default settings on a 3.4 GHz Intel i7 processor. Model-fitting also takes longer with XRate: a previous work found that XRate's parameter estimation routines were approximately 130 times slower than those in PAML [137].

In an attempt to improve XRate's performance, we tried using Beagle, a library that provides CPU and accelerated parallel GPU implementations of Felsenstein's algorithm along with related matrix operations [138]. We have, however, been so far unable to generate significant performance gains by this method.

Despite these caveats, XRate has proved to be fast enough for genome-scale applications, such as a screen of *Drosophila* whole-genome alignments [76]. Furthermore, it implements a significantly broader range of models than the above-cited tools.

XRate inputs, outputs and operations

The formulation of the XRate model presented in the previous section is generative: that is, it describes the generation of data on a tree. In practice, the main reason for doing this is to generate simulation data for benchmarking purposes. This is possible using the tool `simgram` [103], which is provided with XRate as part of the DART package.

Most common use cases for generative models involve not simulation, but inference: that is, reconstructing aspects of the generative process (sequence of nonterminal transformations, token mutations, or grammar parameters) given observed sequence data (in the form of a multiple sequence alignment). Using a phylo-grammar, a set of aligned sequences, and a phylogeny relating these sequences (optionally inferred by XRate), XRate implements the relevant parameterization and inference algorithms, allowing researchers to analyze sequence data without having to implement their own models.

Sequences are read and written in Stockholm format [139] (converters to and from common formats are included with DART). This format allows for the option of embedding a tree in Newick format [140] (via the `#=GF NH` tag) and annotations in GFF format [141]. By construction, Newick format necessarily specifies a rooted tree, rather than an unrooted one. However, the root placement is only relevant for time-irreversible models; when using time-reversible models, the placement of the root is arbitrary and can safely be ignored. Given these input ingredients, a call to XRate proceeds in the following order (more detail is provided at `http://biowiki.org/XRATE` and `http://biowiki.org/XrateFormat`):

1. The Stockholm file and grammar alphabet are parsed (as macros may depend on these).

2. Any grammar macros are expanded, followed by Scheme functions.

3. If requested, or a tree was not provided in the input data, one is estimated using neighbor-joining [142]. As noted above, this is a rooted tree, but the root placement is arbitrary if a time-reversible model is used.

4. Grammar parameters are estimated (if requested).

5. Alignment is annotated (if requested).

6. Ancestral sequences are reconstructed (if requested).

After the analysis is complete, the alignment (along with an embedded tree) is printed to the output stream along with ancestral sequences (if requested) as well as any `#=GC` and `#=GR` column annotations. GFF and WIG annotations are sent to standard output by default, but these can be directed to separate files by way of the `-gff` and `-wig` options, respectively.

6.3 Results and discussion

The XRate format macro language for phylo-grammar specification: case studies

The following sections describe case studies of repetitively-structured models which motivate the need for grammar-generating code. Historically, we have attempted several solutions to the case studies described. We first briefly review the factors that influenced our eventual choice of Scheme as a macro language.

XRate was preceded by Searls' Prolog-based automata [50] and Birney's Dynamite parser-generator [134], and roughly contemporaneous with Slater's Exonerate [133] and Lunter's HMMoC [113]. In early versions of XRate (circa 2004), and in Exonerate, the only way for the user to specify their own phylo-grammar models was to write C/C++ code that would compile directly against the program's internal libraries. This kind of compilation step significantly slows model prototyping, and impedes re-use of model parameters.

Current versions of XRate, along with Dynamite and HMMoC, understand a machine-readable grammar format. In the case of XRate, this format is based on Lisp S-expressions. In such formats (as the case studies illustrate) the need arises for code that generates repetitively-structured grammar files. It is often convenient, and sometimes sufficient, to write such grammar-generating code in an external language: for example, we have written Perl, Python and C++ libraries to generate XRate grammar files [18, 137]. However, this approach still has the disadvantage (from a programmer's or model developer's perspective) that (a) code to generate real grammars tends to require an ungainly mix of grammar-related S-expression constants embedded in Perl/Python/C++ code, and (b) the requirement for an explicit model-generation step can delay prototyping and evaluation of new phylo-grammar models.

XRate's macro language provides an alternate way to generate repetitive models within XRate, without having to resort to external code-generating scripts. This allows the model-specifying code to remain compact, readable, and easy to edit. As we report in this manuscript, the XRate grammar format now also natively includes a Scheme-based scripting language that can be embedded directly within grammar files, whose syntax blends seamlessly with the S-expression format used by XRate and whose functional nature fits XRate's problem domain. We provide here examples of common phylogenetic models which make use of various macro features, and refer the reader to the online documentation for a complete introduction to XRate's macro features: `http://biowiki.org/XrateMacros`. All of the code snippets presented here are available as minimal complete grammars in Text S1. The full, trained grammars corresponding to those presented here are available as part of DART. This correspondence is described here: `http://biowiki.org/XratePaper2011`

A repetitively-structured HMM specified using simple macros

Probabilistic models for the evolution of biological sequences tend to contain repetitive structure. Sometimes, this structure arises as a reflection of symmetries in the phylo-grammar; other times, it arises due to structure in the data, such as the tree or the alignment. While small repetitive models can be written manually, developing richer evolutionary models and grammars often demands writing code to model the underlying structure.

Markov chain symmetry The most familiar source of repetition derives from the substitution model's structure: different substitutions share parameters based on prior knowledge or biological intuition. Perhaps most repetitive is the Jukes-Cantor model for DNA. The matrix entries Q_{ij} denote the rate of substitution from i to j:

$$Q^{JC} = \left\{ \begin{array}{ccccc} & \mathbf{A} & \mathbf{C} & \mathbf{G} & \mathbf{T} \\ \mathbf{A} & * & u & u & u \\ \mathbf{C} & u & * & u & u \\ \mathbf{G} & u & u & * & u \\ \mathbf{T} & u & u & u & * \end{array} \right\}$$

Here u is an arbitrary positive rate parameter. The $*$ character denotes the negative sum of the remaining row entries (here equal to $-3u$ in every case). The parameter u is typically set to 1/3 in order that the stochastic process performs, on average, one substitution event per unit of time.

This matrix can be specified in XRate with two nested loops over alphabet tokens. Each loop over alphabet tokens has the form `(&foreach-token X expression...)` where `expression...` is a construct to be expanded for each alphabet token `X`. Here, `expression` sets the substitution rate between each pair of source and destination tokens (except for the case when the source and destination tokens are identical, for which case we simply generate an empty list, `()`, which will be ignored by the XRate grammar parser). We do not explicitly need to write the negative values of the on-diagonal matrix elements (labeled $*$ in the above description of the matrix); XRate will figure these out for itself. To check whether source and destination tokens are equal in the loop, we use a conditional `&if` statement, which has the form `(&if (condition) (expansion-if-true) (expansion-if-false))`. The `condition` is implemented using the `&eq` macro, which tests if its two arguments are equal. Putting all these together, the nested loops look like this:

```
(&foreach-token tok1
 (&foreach-token tok2
  (&if (&eq tok1 tok2)
   ()  ;; If tok1==tok2, expand to an empty list (ignored by parser)
   (mutate (from (tok1)) (to (tok2)) (rate u)))))
```

While this illustrates XRate's looping and conditional capabilities, such a simple model would almost be easier to code by hand. For a slightly more complex application, we turn to the model of Pupko *et al* in their 2008 work. In their RASER program the authors used a chain augmented with a latent variable indicating "slow" or "fast" substitution. Reconstructing ancestral sequences on an HIV phylogeny allowed them to infer locations of transitions between slow and fast modes - indicating a possible gain or loss of selective pressure [143]. The chain shown below, Q^{RASER}, shows a simplified version of their model: substitutions *within* rate classes occur according to a JC69 model scaled by rate parameters s and f (slow and fast, respectively), and transitions *between* rate classes occur with rates r_{sf} and r_{fs} (slow → fast and fast → slow, respectively).

$$Q^{RASER} = \left\{ \begin{array}{ccccc|cccc} & \mathbf{A}_s & \mathbf{C}_s & \mathbf{G}_s & \mathbf{T}_s & \mathbf{A}_f & \mathbf{C}_f & \mathbf{G}_f & \mathbf{T}_f \\ \mathbf{A}_s & * & us & us & us & r_{sf} & 0 & 0 & 0 \\ \mathbf{C}_s & us & * & us & us & 0 & r_{sf} & 0 & 0 \\ \mathbf{G}_s & us & us & * & us & 0 & 0 & r_{sf} & 0 \\ \mathbf{T}_s & us & us & us & * & 0 & 0 & 0 & r_{sf} \\ \hline \mathbf{A}_f & r_{fs} & 0 & 0 & 0 & * & uf & uf & uf \\ \mathbf{C}_f & 0 & r_{fs} & 0 & 0 & uf & * & uf & uf \\ \mathbf{G}_f & 0 & 0 & r_{fs} & 0 & uf & uf & * & uf \\ \mathbf{T}_f & 0 & 0 & 0 & r_{fs} & uf & uf & uf & * \end{array} \right\}$$

While this chain contains four times as many rates as the basic JC69 model, there are only five parameters: u, s, f, r_{sf}, r_{fs} since the model contains repetition via its symmetry. While manual implementation is possible, the model can be expressed in just a few lines of XRate macro code. Further, additional "modes" of substitution (corresponding to additional quadrants in the matrix above) can be added by editing the first two lines of the following code.

XRate represents latent variable chains as tuples of the form `(state class)`, where `state` is a particular state of the Markov chain and `class` is the value of a hidden variable. In this case, standard DNA characters are augmented with a latent variable indicating substitution rate class: $\mathbf{A}_f$ indicates an $\mathbf{A}$ which evolves "fast." The following syntax is used to declare a latent variable chain (in this case, this variable may take values `s` or `f`), with the `row` tag specifying `CLASS` as the Stockholm `#=GR` identifier for per-sequence, per-column annotations:

```
(hidden-class (row CLASS) (label (s f)))
```

Combining loops, conditionals, hidden classes, and the `(&cat LIST)` function (which concatenates the elements of `LIST`), we get the following XRate code for the RASER chain:

```
(rate (s 0.1) (f 2.0) (r_sf 0.01) (r_fs 0.01) (u 1.0))
(chain
 (hidden-class (row CLASS) (label (s f)))
 (terminal RASER)
 (&foreach class1 (s f)
  (&foreach class2 (s f)
   (&foreach-token tok1
    (&foreach-token tok2
     (&if (&eq class1 class2)
      (&if (&eq tok1 tok2)
       () ;; if class1==class2 && tok1==tok2, expand to empty list (will be ignored)
       ;; The following line handles the case (class1==class2 && tok1!=tok2)
       (mutate (from (tok1 class1)) (to (tok2 class2)) (rate u class1)))
      (&if (&eq tok1 tok2)
       ;; The following line handles the case (class1!=class2 && tok1==tok2)
       (mutate (from (tok1 class1)) (to (tok2 class2)) (rate (&cat r_ class1 class2)))
       ())))))))  ;; if class1!=class2 && tok1!=tok2, expand to empty list (ignored)
```

Phylo-HMM-induced repetition The previous examples both involved specifying the Markov chain component of a phylo-grammar. Coupled with a trivial top-level grammar (a `START` state and an `EMIT` state which emits the chain via the `EMIT*` pseudoterminal), these models describe an alignment where each column's characters evolve according to the same substitution model. A common extension to this is using sequences of hidden states which generate alignment columns according to different substitution models. These "phylo-grammars" (which can include phylo-SCFGs and the more restricted phylo-HMMs) allow modelers to describe and/or detect alignment regions exhibiting different evolutionary patterns. Phylo-HMMs model left-to-right correlations between alignment columns, and phylo-SCFGs are capable of modeling nested correlations (such as "paired" columns in an RNA secondary structure). Readers unfamiliar with phylo-grammars may benefit from relevant descriptions and links available here: `http://biowiki.org/PhyloGrammars`, animations available here: `http://biowiki.org/PhyloFilm`, and the original paper describing XRate [18].

We outline here a phylo-HMM that is simple to describe, but would take a substantial amount of code to implement without XRate's macro language. The model is based on PhastCons, a program by Siepel *et al* which uses an HMM whose three states (or, in XRate terminology, nonterminals) use substitution models differing only by rate multipliers [118]. This model, depicted schematically in Figure 6.1, can be used to detect alignment regions evolving at different rates. If the rates of each hidden state correspond to quantiles of the Gamma distribution, then summing over hidden states of this model is equivalent to the commonly-used Gamma model of rate heterogeneity. We provide this grammar in Text S1, which is essentially identical to the PhastCons grammar with n states except for its invocation of a Scheme function returning the n Gamma-derived rates for a given shape parameter. We can define such a model in XRate easily due to the symmetric structure: all three nonterminals have similar underlying substitution models (varying only by a multiplier) and also similar probabilities of making transitions to other nonterminals via grammar transformation rules.

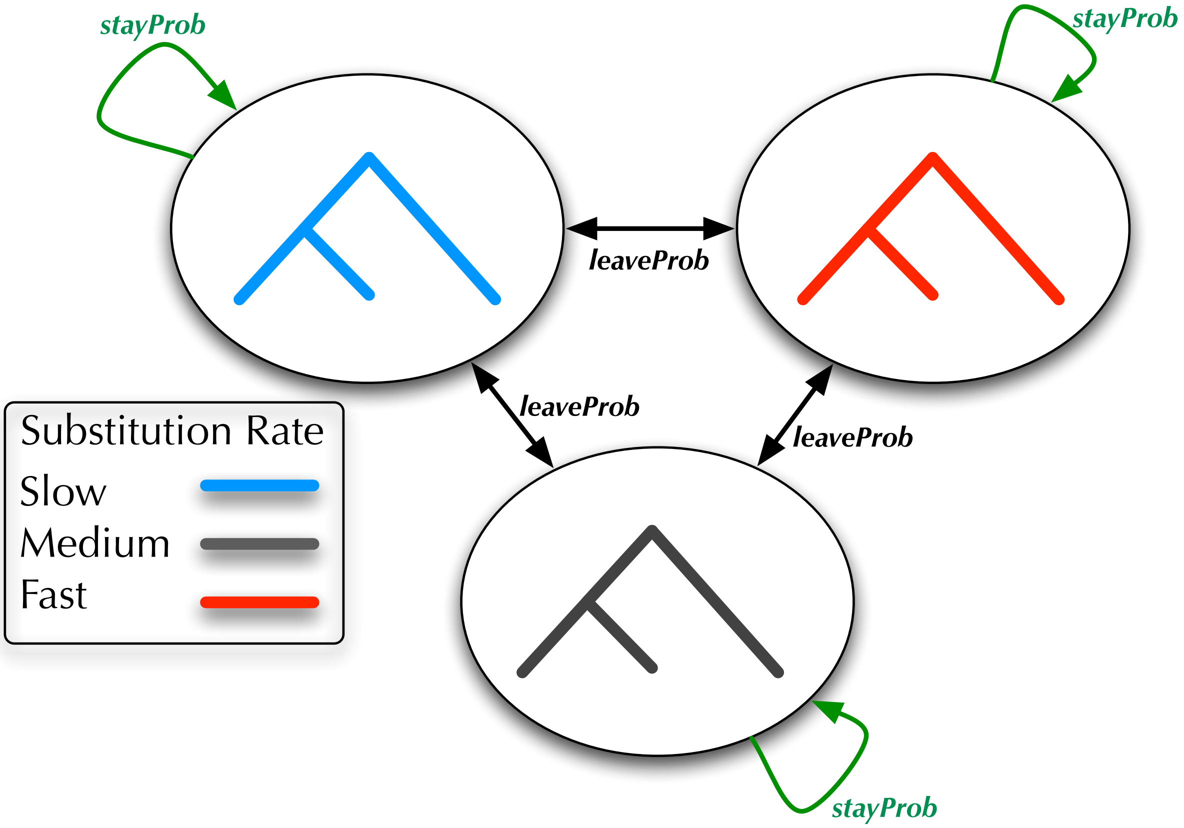


Figure 6.1: The model used by PhastCons, a 3-nonterminal HMM with rate multipliers, is compactly expressed by XRate's macro language. Different nonterminal have different evolutionary rates, but they all share the same underlying substitution model. Transition probabilities are shared: a transition between nonterminals happens with probability *leaveProb*, and self-transitions happen with probability *stayProb*. This model (with any number of nonterminals) can be expressed in XRate's macro language in approximately 20 lines of code.

The grammar will have nonterminals named "1", "2"...up to `numNonTerms`, each one associated with a rate parameter (`r_1, r_2...`) and substitution chain (`chain_1, chain_2...`). To express this grammar in XRate macro code, we'll need to declare each of these nonterminals, the production rules which govern transitions between them, rate parameters, and the nonterminal-associated substitution chains. (For a fully-functional grammar, an alphabet is also needed; these are omitted in code snippets included in the main text, but the corresponding grammars in Text S1 contain alphabets.)

First, define how many nonterminals the model will have: adding more nonterminals to the model later on can be done simply by adjusting this variable. We define a `SEED` value to initialize the rate parameters (this is not a random number seed, but rather an initial guess at the parameter value necessary for the EM algorithm to begin), which is done inside a foreach-integer loop using the numNonterms variable. The (foreach-integer X (1 K) expression) expands expression for all values of X from 1 to K. In this case, we define a rate parameter for each of our nonterminals 1..K.

```
(&define numNonterms 3)
(&define SEED 0.001)
(&foreach-integer nonterminal (1 numNonterms)
    (rate ((&cat r_ nonterminal) SEED)))
```

Next, define a Markov chain for each nonterminal: all make use of the same underlying substitution model (e.g. JC69 [65], HKY85 [66]) whose entries are stored as Q_a_b for the transition rate between characters a and b. This "underlying" chain must be defined elsewhere - either in an included file (using the (&include) directive), or directly in the grammar file. For instance, we could re-use the JC69 chain, declaring rate parameters for later use:

```
(&foreach-token tok1
 (&foreach-token tok2
  (&if (&eq tok1 tok2)
   ()  ;; If tok1==tok2, expand to an empty list (ignored by parser)
   (rate  (&cat Q_ tok1 _ tok2) u ))))
```

Each nonterminal has an associated substitution model which is Q_a_b scaled by a different rate multiplier r_nonterminal. Using an integer loop, we create a chain for each nonterminal using the rate parameters we defined in the two previous code snippets:

```
(&foreach-integer nonterminal (1 numNonterms)
 (chain
  (terminal (&cat chain_ nonterminal))
  (&foreach-token tok1
   (&foreach-token tok2
    (&if (&eq tok1 tok2)
     ()
     (mutate (from (tok1)) (to (tok2))
     (rate (&cat Q_ tok1 _ tok2) (&cat r_ nonterminal))))))))
```

Next, define the production rules which govern the nonterminal transitions. For simplicity of presentation (but not required), we assume here that transitions between nonterminals all occur with probability proportional to leaveProb, and all self-transitions have probability stayProb.

The pgroup declaration defines a probability distribution over a finite outcome space, with the parameters declared therein normalized to unity during parameter estimation. In this grammar we declare stayProb and leaveProb within a pgroup since they describe the two outcomes at each step of creating the alignment: staying at the current nonterminal or moving to a different one.

```
(pgroup (stayProb 0.9) (leaveProb 0.1))
(&foreach-integer nonterm1 (1 numNonterms)
 ;; Each nonterminal has a transition from start
 (transform (from (start)) (to (nonterm1)) (prob (&/ 1 numNonterms)))
 ;; Each nonterminal can transition to end - we assign this prob 1
 ;; since the alignment length directs when this transition occurs
 (transform (from (nonterm1)) (to ()) (prob 1))
 (&foreach-integer nonterm2 (1 numNonterms)
  (&if (&eq nonterm1 nonterm2))
   ;; If nonterm1==nonterm2, this is a self-transition
   (transform (from (nonterm1)) (to (nonterm2)) (prob stayProb))
   ;; Otherwise, this is an inter-nonterminal transition
   ;; with probability changeProb / (numNonterms - 1)
   (transform (from (nonterm1)) (to (nonterm2))
   (prob (&/ changeProb (&- numNonterms 1))))))
```

Lastly, associate each nonterminal with its specially-designed Markov chain for emitted alignment columns:

```
(&foreach-integer nonterminal (1 numNonterms)
 (transform (from (nonterminal))(to ((&cat chain_ nonterminal) (&cat nonterminal *))))
 (transform (from ((&cat nonterminal *))) (to (nonterminal))))
```

Data-induced repetition Models whose symmetric structure depends on the input data are less common in phylogenetic analysis, perhaps because normally their implementation requires creating a new model for each new dataset to be analyzed. XRate allows the user to create models based on different parts of the input data, namely the tree and the alignment, "on the fly" via its macro language. This is accomplished by making use of the tree iterators (e.g. &BRANCHES, &NODES, and &LEAVES) and alignment data (e.g. &COLUMNS) to create nonterminals and/or terminal chains associated with these parts of the input data.

In their program DLESS, Haussler and colleagues used such an approach in a tree-dependent model to detect lineage-specific selection. Their model used a phylo-HMM with different nonterminals for each tree node, with the substitution rate below this node scaled to reflect gain or loss of functional elements [118]. We show a simplified form of their model as a schematic in Figure 6.2, with blue colored branches representing a slowed evolutionary rate.

Using XRate's macros we can express this model in a compact way just as was done with the PhastCons model. Since both models use a set of nonterminals with their own scaled substitution models, we need simply to replace the integer-based loop `(&foreach-integer nonterminal` with the tree-based loop `(&foreach-node state expression)` to create a nonterminal for each node in the tree. Then, define each node-specific chain as a *hybrid chain*, such that the chain associated with tree node n has all the branches below node n scaled to reflect heightened selective pressure. Hybrid chains, substitution processes which vary across the tree, are discussed briefly in the section on "Recent enhancements to XRate", and the details of their specification is thoroughly covered in the XRate format documentation, available here: `http://biowiki.org/XrateFormat` . A minimal working form of the DLESS-style grammar included in Text S1.

A repetitively-structured codon model specified using Scheme functions

While XRate's macro language is very flexible, there are some relatively common models that are difficult to express within the language's constraints. For example, a Nielsen-Yang codon matrix incorporating transition bias and selection has nearly 4,000 entries whose rates are determined by the following criteria:

$$Q_{ij}^{NY} = \begin{cases} 0 & \text{if } i \text{ and } j \text{ differ at more than one position} \\ \pi_j & \text{if } i \text{ and } j \text{ differ by a synonymous transversion} \\ \kappa\pi_j & \text{if } i \text{ and } j \text{ differ by a synonymous transition} \\ \omega\pi_j & \text{if } i \text{ and } j \text{ differ by a nonsynonymous transversion} \\ \omega\kappa\pi_j & \text{if } i \text{ and } j \text{ differ by a nonsynonymous transition} \end{cases}$$

This sort of Markov chain is difficult to express in XRate's macro language since its entries are determined by aspects of the codons (synonymous changes and transitions/transversions) which in turn depend on knowledge of the properties of nucleotides and codons that would have to be hard-coded directly into the loops and conditionals afforded by XRate's macros. The conditions on the right side of the above equation are better framed as values returned from a function: given a pair of codons, the function returns the "type" of difference between them, which in turn determines the rate of substitution between the two codons.

Scheme extensions It is this sort of situation which motivates extensions to XRate that are more general-purpose than the simple macros described up to this point. There are several valid choices for the programming language that can be used to implement such extensions. For example, a chain such as Q^{NY} can be generated fairly easily by way of a Perl or Python script tailored to generate XRate grammar code. While this is a convenient scripting mechanism for many users (and is perfectly possible with XRate), it tends to lead to an awkward mix of code and embedded data (i.e. snippets of grammar-formatting text).

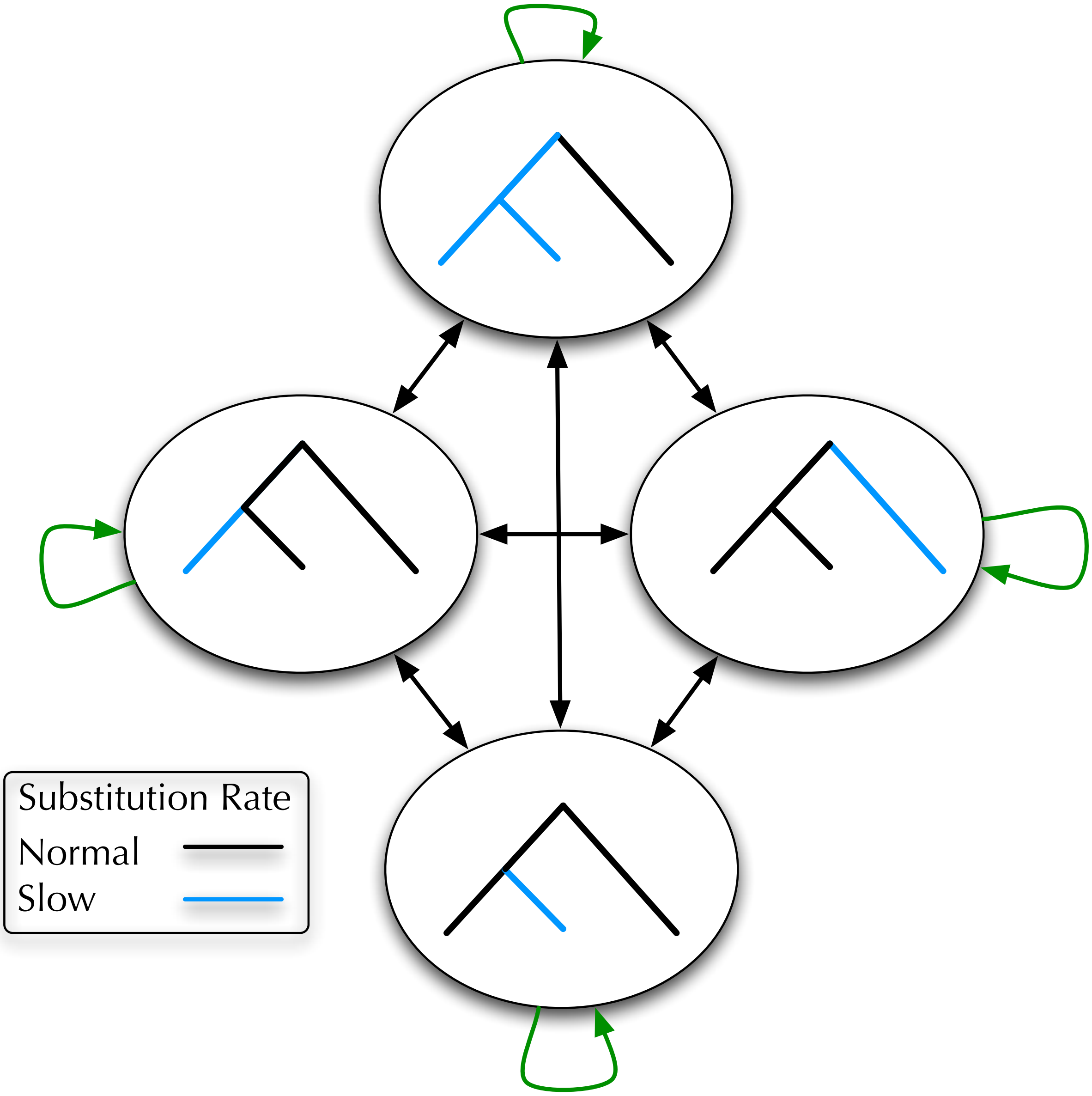


Figure 6.2: A schematic of a DLESS-style phylo-HMM: each node of the tree has its own nonterminal, such that the node-rooted subtree evolves at a slower rate than the rest of the tree. Inferring the pattern of hidden nonterminals generating an alignment allows for detecting regions of lineage-specific selection. Expressing this model compactly in XRate's macro language allows it to be used with any input tree without having to write data-specific code or use external model-generating scripts.

This obscures both the generating script and the final generated grammar file (the former due to the code/data mix, and the latter due to sheer size).

Another choice of programming language for implementing XRate extensions, which suffers slightly less from these limitations, is Scheme. As XRate's macro language is based on Lisp (the parent language to Scheme), the syntaxes are very similar, so the "extension" blends naturally with the surrounding XRate grammar file. Scheme is inherently functional and is also "safe" (in that it has garbage collection). Lastly, data and code have equivalent formats in Scheme, enabling the sort of code/data mingling outlined above.

To implement the Q^{NY} chain in XRate, we can use the XRate Scheme standard library (found in `dart/scheme/xrate-stdlib.scm`). This standard library implements all the necessary functions to define the Nielsen-Yang model, with the genetic code implemented as a Scheme association list (facilitating easy substitution of alternate genetic codes, such as the mitochondrial code) as well as a wrapper function to initialize the entire model.

Without stepping through every detail of the Scheme implementation of the Nielsen-Yang model in the XRate standard library, we will simply note that this implementation (the Nielsen-Yang model on a DNA alphabet) is available via the following XRate code (the include path to `dart/scheme` is searched by default by the Scheme function `load-from-path`):

```
(&scheme
 (load-from-path "xrate-stdlib.scm")
 xrate-dna-alphabet
 (xrate-NY-grammar))
```

Note that `xrate-dna-alphabet` is a simple variable, but `xrate-NY-grammar` is a function and is therefore wrapped in parentheses (as per the syntax of calling a function in Scheme). The reason that `xrate-NY-grammar` is a function is so that the user can optionally redefine the genetic code, which (as noted above) is stored as a Scheme association list, in the variable `codon-translation-table` (the standard library code can be examined for details).

A macro-heavy grammar for RNA structures in protein-coding exons

As a final example of the possibilities that XRate's new model-specification features enable, we present a new grammar for predicting RNA structures which overlap protein-coding regions. XDecoder is based closely on the RNADecoder grammar first developed by Pederson and colleagues [144]. This grammar is designed to detect phylogenetic evidence of conserved RNA structures, while also incorporating the evolutionary signals brought on by selection at the amino-acid level. In eukaryotes, RNA structure overlapping protein coding sequence is not yet well-known, but in viral genomes this is a common phenomenon due to constraints on genome size acting on many virus families. XDecoder is available as an XRate grammar, linked here: `http://biowiki.org/XratePaper2011`

Motivation for implementation Our endeavor to re-implement the RNADecoder grammar was based both on practical and methodological reasons. The original RNADecoder code is no longer maintained, but performs well on published viral datasets [145]. Running RNADecoder on an alignment of full viral genomes is quite involved: the alignment must first be split up into appropriately-sized chunks (˜300 columns), converted to COL format [146], and linked to a tree in a special XML file which directs the analysis. The grammar and its parameters, also stored in an XML format, are difficult to read and interpret. RNADecoder attains remarkably higher specificity in genome-wide scans as compared to protein-naive prediction programs like PFOLD [136] or MFOLD [147].

Using XDecoder We developed our own variant of the RNADecoder model as an XRate grammar, called XDecoder. This would have been a protracted task without XRate's macro capabilities: the expanded grammar is nearly 4,000 lines of code. Using XRate's macros, the main grammar (excluding the pre-estimated dinucleotide Markov chain) is only ˜100 lines of macro code. Starting with an alignment of full-length *poliovirus* genomes, annotated with reading frames, an analysis can be run with a single simple command:

```
xrate -g XDecoder.eg -l 300 -wig polio.wig polio.stk > polio_annotated.stk
```

This runs XRate with the XDecoder grammar on the Stockholm-format alignment `polio.stk`, allowing no more than 300 positions between paired columns, creating the wiggle file `polio.wig`, annotating the original alignment with maximum likelihood secondary structure and rate class indicators, and writing the annotated alignment to the the file `polio_annotated.stk`.

Each analysis with RNADecoder requires an XML file to coordinate the alignment and tree as well as direct parts of the analysis (training and annotation). XRate reads Stockholm format alignments which natively allows for alignment-tree association, enabling simple batch processing of many alignments. The grammar can be run on arbitrarily long alignments, provided a suitable maximum pair length is specified via the `-l N` argument. This prevents XRate from considering any pairing whose columns are more than `N` positions apart, effectively limiting both the memory usage and runtime.

Training the grammar's parameters, which may be necessary for running the grammar on significantly different datasets, is also accomplished with a single command:

```
xrate -g XDecoder.eg -l 300 -t XDecoder.trained.eg polio.stk
```

The results of an analysis using XDecoder are shown in Figure 6.3, together with gene and RNA structure annotations. Also shown are three related analyses (all done using XRate grammars): PhastCons conservation, coding potential, and pairing probabilities computed using PFOLD. These three separate analyses reflect the signals that XDecoder must tease apart in order to reliably predict RNA structures. DNA-level conservation could be due to protein-coding constraints, regional rate variation, pressure to maintain a particular RNA structure, or a combination of all three. Using codon-position rate multipliers, multiple rate classes, and a secondary structure model, XDecoder unifies all of these signals in a single phylogenetic model, resulting in the highly-specific predictions shown at the top of Figure 6.3. The full JBrowse instance is provided as a demo at `http://jbrowse.org/poliovirus-xrate-demo-by-oscar-westesson/`.

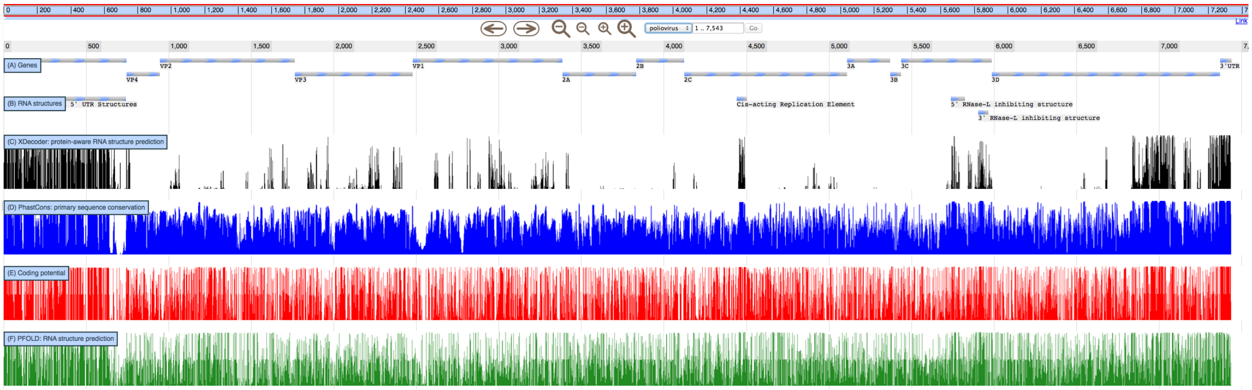


Figure 6.3: Data from several XRate analyses, shown alongside genes **(A)** and known RNA structures **(B)** in *poliovirus*. XDecoder **(C)** recovers all known structures with high posterior probability and predicts a promising target for experimental probing (region 6800-7100). XDecoder was run on an alignment of 27 *poliovirus* sequences with the results visualized as a track in JBrowse [148] via a wiggle file. Alongside XDecoder probabilities are the three signals which XDecoder aims to disentangle: **(D)** conservation, **(E)** coding potential, and **(F)** RNA structure. Paradoxically, the CRE and RNase-L inhibition elements show both conservation and coding sequence preservation, whereas PFOLD's predictions show only a slight increase in probability density around the known structures. XDecoder is the only grammar which returns predictions of reasonable specificity.

Recent enhancements to XRate

Lineage-specific models

All Markov chains in phylo-grammars describe the evolution of characters starting at the root and ending at the tips of the tree. In lineage-specific models, or *hybrid chains* in XRate terminology, the requirement that all branches share the same substitution process is relaxed. Phylogenetic analysis is often used to detect a departure from a "null model" representing some typical evolutionary pattern. Standard applications of HMMs and SCFGs focus on modeling this departure on the alignment level, enabling different columns of the alignment to show different patterns of evolution. Using hybrid chains, users can explicitly model differences in evolution across parts of the tree. By combining a hybrid chain with grammar nonterminals, this could be used to detect alignment regions (i.e. subsets of the set of all sites) which display unusually high (or low) mutation rates in a particular part of the tree, such as in the DLESS model described in the section on "Data-induced repetition". The details of specifying such models are contained within the XRate format documentation, at `http://biowiki.org/XrateFormat`

Ancestral sequence reconstruction

A phylo-grammar is a generative model: it generates a hidden parse tree, then further generates observed data conditional on that parse tree. The observed data here is an alignment of sequences; the hidden parse tree describes which alignment columns are to be generated by the evolutionary models associated with which grammar nonterminals. Inference involves reversing the generative process: reconstructing the hidden parse structure and evolutionary trajectories that explain the alignment.

The original version of XRate was focused on reconstructing the parse tree, for the purposes of annotating hidden structures such as gene boundaries or conserved regions. A newly-implemented feature in XRate allows an additional feature: reconstruction of ancestral sequences. This functionality is already implicit in the phylogenetic model: no additional modification to the grammar is necessary to enable reconstruction. The user can ask XRate to return the most probable ancestral sequence at each internal node, or the entire posterior distribution over such sequences, via the `-ar` and `-arpp` command-line options. Since XRate does marginal state reconstruction, the character with the highest posterior probability returned by the `-arpp` option will always correspond to the single character returned by the `-ar` option. Ancestral sequence reconstruction can be used to answer paleogenetic questions: what did the sequence of the ancestor to all of clade X look like? Similarly, evolutionary events such as particular substitutions or the gain or loss of function (also called trait evolution) can be pinpointed to particular branches.

Direct output of GFF and Wiggle annotations

XRate allows parse annotations to be written out directly in common bioinformatics file formats: GFF (a format for specifying co-ordinates of genomic features) [141] and WIG (a per-base format for quantitative data) [149].

This allows a direct link between XRate and visualization tools such as JBrowse [148], GBrowse [150], the UCSC Genome Browser [151],and Galaxy [152], allowing the results of different analyses to be displayed next to one another and/or processed in a unified framework.

GFF: Discrete genomic features GFF is a format oriented towards storing genomic features using 9 tab-delimited fields: each line represents a separate feature, with each field storing a particular aspect of the feature (e.g. identifier, start, end, etc). With XRate, a common application is using GFF to annotate an alignment with features corresponding to grammar nonterminals. For instance, using a gene-prediction grammar one could store the predicted start and end points of genes together with a confidence measure. Similarly, predicted RNA base pairs could be represented in GFF as one feature per pair, with start and end positions indicating the paired positions.

WIG: Quantitative values for each column(s) Wiggle format stores a quantitative value for a single or group of positions. This can be especially useful to summarize a large number of possibilities as a single representative value. For instance, when predicting regions of structured RNA, XRate may sum over many thousands of possible structures. We can summarize the model's results with the posterior probability that each column is involved in a base-pairing interaction.

The Dart Scheme (Darts) interpreter

Another way to use XRate, instead of running it from the command line, is to call it from the Scheme interpreter (included in DART). The compiled interpreter executable is named "darts" (for "DART Scheme"). This offers a simple yet powerful way to create parameter-fitting and genome annotation workflows. For example, a user could train a grammar on a set of alignments, then use the resulting grammar to annotate a set of test alignments.

Darts, in common with the Scheme interpreter used in XRate grammars, is implemented using Guile (GNU's Ubiquitous Intelligent Language for Extension: `http://www.gnu.org/software/guile`
Certain commonly-encountered bioinformatics objects, serializable via standard file formats and implemented as C++ classes within XRate, are exposed using Guile's "small object" (smob) mechanism. Currently, these types include Newick-format trees and Stockholm-format alignments. API calls are provided to construct these "smobs" by parsing strings (or files) in the appropriate format. The smobs may then be passed directly as parameters to XRate API calls, or may be "unpacked" into Scheme data structures for individual element access. Guile encourages sparing use of smobs; consequently, smobs are used within Darts exclusively to implement bioinformatic objects that already have a broadly-used file format (Stockholm alignments and Newick trees). In contrast, formats that are newly-introduced by XRate (grammars, alphabets and so forth) are all based on S-expressions, and so may be represented directly as native Scheme data structures.

The functions listed in Appendix D provide an interface between Scheme and XRate. Together with the functions in the XRate-scheme standard library and Scheme's native functional scripting abilities, a broad array of models and/or workflows are possible. For instance, one could estimate several sets of parameters for Nielsen-Yang models using groups of alignments, and then embed each one in a PhastCons-style phylo-HMM, finally using this model to annotate a set of alignments. While this and other workflows could be accomplished in an external framework (e.g. Make, Galaxy [152]), Darts provides an alternate way to script XRate tasks using the same language that is used to construct the grammars.

Additional supporting information

The JBrowse instance displayed in Figure 6.3 is available at
`http://jbrowse.org/poliovirus-xrate-demo-by-oscar-westesson/`.

Text S1 is available at
`http://biowiki.org/twiki/pub/Main/XrateTutorial/example_grammars.zip`
It contains example grammars referred to in the text, as well as small and large test Stockholm alignments. The alignment of *poliovirus* genomes along with the grammars used to produce Figure 6.3 are also included along with a Makefile indicating how the data was analyzed. Typing `make help` in the directory containing the Makefile will display the demonstrations available to users.

Appendix D contains tables describing the Scheme-XRate functions available in Darts.

Appendix E contains a glossary of XRate terminology.

Chapter 7

RecHMM: Detecting phylogenetic recombination

The following section contains work conducted with Ian Holmes published in PLoS Computational Biology [153].

Overview

In viral and bacterial pathogens, recombination has the ability to combine fitness-enhancing mutations. Accurate characterization of recombinant breakpoints in newly-sequenced strains can provide information about the role of this process in evolution; for example, in immune evasion. Of particular interest are situations of admixture of pathogen subspecies, recombination between whose genomes may change the apparent phylogenetic tree topology in different regions of a multiple-genome alignment.

We describe an algorithm that can pinpoint recombination breakpoints to greater accuracy than previous methods, allowing detection of both short recombinant regions and long-range multiple-crossovers. The algorithm is appropriate for analysis of fast-evolving pathogen sequences where repeated substitutions may be observed at a single site in a multiple alignment (violating the "infinite sites" assumption inherent to some other breakpoint-detection algorithms). Simulations demonstrate the practicality of our implementation for alignments of longer sequences and more taxa than previous methods.

7.1 Background

Recombination is the process by which a child inherits a mosaic of genes or sequences from multiple parents. Though most species participate in some form of genetic mixing or recombination, the mechanics by which this occurs varies greatly among them. In higher order organisms, crossing over occurs in meiosis along the parent-child relationship, whereas in bacteria, viruses, and protozoans, homologous exchange of DNA material can occur from one individual to another without the need for sexual reproduction [154]. The diversity with which recombination occurs motivates the need for different models and methods, each ideally suited to its biological situaion. We have developed a probabilistic approach to recombination detection that we believe to be superior for analyzing situations of admixture of pathogen subspecies with a high mutation/recombination ratio.

The situation we concern ourselves with has been termed *phylogenetic recombination inference* (PRI) by [155], and works by inferring phylogenetic tree topology changes over a multiple alignment. Though it has been shown that under a neutral coalescent model, the number of recombination events which will lead to a tree topology change is very small, [156] in situations of admixture following geographical separation a greater proportion of topology-changing recombinations are expected. Abandoning the infinite-sites model of sequence evolution and instead using a continuous-time Markov chain makes direct inference intractable, and so we instead employ a phylo-HMM which models an *effect* of recombination, rather than modeling the process explicitly.

While recombination detection is an interesting mathematical challenge, fast, flexible, and reliable computational methods are also motivated by a multitude of biological reasons. We see our method not as being able to answer all of these biological questions on recombination, but rather a potentially valuable tool for furthering recombination research.

- **Genome Dynamics** The two most significant factors driving change in genomes in the context of evolutionary adaptation and diversity are point mutation and recombination. The ratio between these two differs greatly among organisms; in most, recombination among subtypes is fairly rare and point mutation occurs comparatively often. Similarly to point mutation, recombination has the possibility to combine independent fitness-enhancing changes among genomes as well as disable genes. As Awadalla remarks, "recombinant genomes are known to be associated with changes in phenotype or fitness, including heightened or reduced pathogenicity or virulence" [154]. Our understanding about where, how, and why recombination occurs is comparatively primitive. We know, for example, that pathogens such as *Chlamydia trachomatis* have recombination hotspots [157], but the relevant *cis*-acting factors are unknown. The precise determination of breakpoints in recombining pathogens is crucial for higher-level downstream analyses such as that of [157], or the methods proposed by [155] and [25] in which genome-scale conclusions about recombination are made from large sets of observed breakpoint locations. We believe our method offers improved precision and flexibility as compared to other programs. Furthermore, in light of the high proportion of HIV isolates which are recombinant, it can be useful that PRI allows one to safely relax the requirement that all but one of the sequences in the alignment are 'pure' subtypes.

  As well as being an appealing scientific challenge, a better understanding of the dynamics of pathogen genome evolution might help highlight molecular processes to target in designing therapeutics, as well as opening up the possibility for genetic manipulation.

- **Phylogenetic Analysis** When performing phylogenetic analysis on a multiple sequence alignment, most methods assume that there is a unique hierarchical relationship among the taxa in question. If recombination has occurred in evolutionary history, this phylogeny reconstruction will be systematically faulty in either its topology, branch lengths or both. Incorrect trees could hinder further comparative genomic inferences made from the data [158]. In our training scheme, we estimate separate trees for all regions in the alignment, and if more sophisticated tree-inference methods ought to be used, our precise breakpoint inference allows for training trees on the alignment sections.

- **Genetic Mapping** In using genetic mapping information to locate genes associated with various phenotypes, it is vital to know the extent of genome rearrangement present. For a discussion of how this can affect microbial pathogen analysis, see [159].

Previous Related Work We give here an outline of previous methods which are related to our phylo-HMM approach. For a more thorough survey or recombination detection methods, see [154].

The rationale for phylogenetic recombination inference is motivated by the structure of the Ancestral Recombination Graph (ARG), which contains all phylogenetic and recombination histories. The underlying idea is that recombination events in the history of the ARG will, in certain cases, lead to discordant phylogenetic histories for present-day species.

Various approaches to learn the ARG directly from sequence data have been developed, such as [160] and [161]. We recognize that PRI is in a somewhat different category both in goal and approach as compared these methods, though they are motivated by the same underlying biological phenomenon. Rather than aiming to reconstruct the ARG in its entirety, our emphasis is on modeling fast-evolving organisms with the goal of accurately detecting breakpoints for biological and epidemiological study.

The most widely-used program for phylogenetic recombination detection is SimPlot [24] (on MS Windows). Recombination Identification Program (RIP), a similar program, [162] runs on UNIX machines as well as from a server on the LANL HIV Database site. This program slides a window along the alignment and plots the similarity of a query sequence to a panel of reference sequences. The window and step size are adjustable to accommodate varying levels of sensitivity. Bootscanning slides a window and performs many replicates of bootstrapped phylogenetic trees in each window, and plots the percentage of trees which show clustering between the query sequence and the given reference sequence. Bootscanning produces similar output to our program, namely a predicted partition of the alignment as well as trees for each region, but the method is entirely different.

In [27], Husmeier and Wright use a model that is similar to ours except for the training scheme. Since they have no scalable tree-optimizing heuristic, their input alignment is limited to 4 taxa so as to cover all unrooted tree topologies with only 3 HMM states, making their method intractable for larger datasets. They show they are able to convincingly detect small recombinant regions in *Neisseria* as well as simulated datasets limited to 4 taxa [27].

The recombination detection problem can be thought of as two inter-related problems: how to accurately partition the alignment and how to construct trees on each region. This property is due to the dual nature of the ARG: it simultaneously encodes the marginal tree topologies as well as where they occur in the alignment. Notice that if the solution to one sub-problem is known, the other becomes easy. If an alignment is already partitioned, simply run a tree-inference program on the separate regions and this will give the marginal trees of the sample. If the trees are known, simply construct an HMM with one tree in each state and run the forward/backward algorithm to infer breakpoints. Previous methods have used this property by assuming one of these problems to be solved and focusing on the other. For example, in Husmeier and Wright's model, there were very few trees to be tested, and so the main difficulty was partitioning the alignment, which they did with a HMM similar to ours. In SimPlot, windows (which are essentially small partitions) passed along the alignment and trees/similarity plots are constructed. This allows the program to focus on tree-construction (usually done with bootstrapped neighbor-joining) rather than searching for the optimal alignment partition.

By employing a robust probabilistic model with a novel training scheme, we find a middle ground between the heuristic approach of SimPlot [24] and the computational intractability of Husmeier and Wright's method [27], where we are essentially able to solve the recombination inference problem a whole, rather than neglecting one sub-part and focusing on the other. We use a HMM to model tree topology changes over the columns of a multiple alignment. This is done much in the same way as Husmeier and Wright, but our use of a more sophisticated tree-optimization (the structural EM heuristic) method allows searching for recombination from a larger pool of sequences. By modifying the usual EM method for estimating HMM parameters in a suitable way, we are able to simultaneously learn the optimal partitioning of the alignment as well as trees in each of these partitions. We are able to detect short recombinant regions better than previous methods for several reasons. First, we do not use any sliding windows which may be too coarse-grained to detect such small regions of differing topology. Second, our method allows each tree after EM convergence to be evaluated at every column, and so small recombinant regions are not limited by their size; they must only 'match' the topology to be detected or contribute to the tree training. By embedding trees in hidden states of an HMM, the transition matrix allows us to essentially put a prior on the number of breakpoints, as opposed to considering each column independently. Furthermore, since the counts in the E-Step are computed using all columns of the alignment, distant regions of the alignment with similar topology may contribute their signal to a single tree, whereas in a window-sliding approach each window is analyzed independently.

7.2 Results

Interpretation of the Results Since it is difficult to experimentally verify predictions of recombination, we test our methods on previously-analyzed data from earlier studies on *Neisseria* and HIV-1 as well as simulated datasets. Statistics summarizing simulations with respect to several simulation parameters are included in Figure 7.1.

When comparing real and simulated data, one must keep in mind that real data may have complications such as rate heterogeneity and structural features that are not present in simulations, which are carried out using a simple independent-sites Markov chain model of nucleotide evolution, such as the HKY85 model [66]. While this is currently the only model we use in our program, it is straightforward to extend this to other models.

In analyzing real data, when there were several alignments in the original analysis, we include only those in which we recover new breakpoints, or otherwise demonstrate our method's utility. It is implicitly assumed that in the analyses which we don't include, we came to similar or identical conclusions as the original authors.

In our analysis of simulated data, we aim to quantitatively characterize the strengths and weaknesses of the recHMM method by varying several simulation and analysis parameters. In each simulation case, ARGs (and hence K-tuples of trees) were generated with RECODON

[163], which uses a coalescent-based simulation approach (for exact simulation parameters, see Appendix F) . In keeping with the above discussion of PRI vs coalescent modeling, we filter out ARGs whose marginal trees are identical in topology using the **treecomp** program [164]. Thus, in all of the simulations, a perfect detector of topology change would find every breakpoint. After tree-simulation, we simulate alignments using Seq-Gen, which generates multiple alignments according to simple independent sites Markov chain models. The reason for this decoupling of tree and sequence simulation is that Seq-Gen allows for easier manipulation of the variables we're testing, namely length of region, divergence since recombination, and overall divergence (by way of branch-scaling.) [77].

After running our program on the simulated data to estimate parameters, recombinant regions are determined by a posterior decoding algorithm which we describe briefly in the Methods section and is fully outlined in Appendix F. (We use posterior decoding as opposed to the Viterbi algorithm since we are primarily concerned with maximizing the expected number of correct column labelings as opposed to maximizing the probability that our state path is exactly correct.) As the notion of a 'true negative', a column which was correctly annotated as a non-breakpoint, is not meaningful in this case, we instead examine positive predictive value: $\frac{TP}{TP+FP}$, where a true positive (TP) is defined as a predicted breakpoint which occurs within 10 positions of a true breakpoint. Similarly, a false positive (FP) is a predicted breakpoint which has no true breakpoint within 10 positions. In plotting sensitivity: $\frac{TP}{TP+FN}$, we define a false negative (FN) to be a true breakpoint for which we have no predicted breakpoint within 10 positions.

We vary the following parameters with regard to simulation of data, the results of which are depicted in Figure 7.1:

- **Length of recombinant region** The length of the region which has a discordant phylogenetic signal is inversely proportional to detection power. We simulated alignments with three regions: two regions of length 200 bp on either side of the variable-length region, resulting in two true breakpoints, at positions 200 and 200+length(region).

  On alignments with recombinant regions longer than 100 bp, recHMM detects a high percentage of breakpoints with few false positives. Below 100 bp the detection suffered, with the program able to detect approximately 40% of breakpoints. For any recombination-detection program, smaller regions will be harder to detect, and so high accuracy down to 100 bp is promising.

- **Taxa** With more taxa, tree estimation becomes more difficult, and so distinguishing regions having different trees becomes more challenging. Further, comparing likelihoods of two large trees becomes unreliable as the scale of the likelihood becomes larger.

  In alignments with up to 23 taxa, detection is fairly strong, but begins to taper off around 25 taxa. Still, this is a notable improvement over Husmeier and Wright's model which could only accommodate 4 species. This is practically relevant only for initial screening for recombination; once the donor species are known, the alignment can be pruned of the irrelevant taxa for more accurate breakpoint detection.

- **Divergence** We vary the overall evolutionary time since speciation among the taxa by scaling the branch lengths of the tree used to simulate the alignment. The idea is that as divergence grows and the tree becomes indistinguishable from a 'star-like' topology, the phylogenetic signal relating species becomes weaker. On the other hand, if divergence were 0, the population would appear clonal (e.g. identity along alignment columns) and recombination would be undetectable.

- **Divergence since recombination event** In varying the divergence time since recombination, we wish to quantify the idea that more recent recombination events are easier to detect, since they have closer homology to their donor genomes.

  Since directly varying the divergence since the recombination event is difficult, we instead restrict our analysis to a subset of topology-changing ARGs, namely those whose marginal trees differ by a leaf-transfer event (as opposed to a general subtree-transfer). While this may be an unlikely scenario from a pure coalescent perspective, newly emergent recombinant pathogens can be represented as leaf-transfer events. In terms of simulation, this restriction of ARGs allows us to approximate scaling all branches (and sub-branches) since recombination by scaling only terminal branches, allowing us to demonstrate the difference in detection power between 'ancient' and 'recent' recombination events.

  Divergence and divergence since recombination appear to affect detection power in a similar way. Though it is difficult to draw conclusions from so few data points, one can see a sharper dependence in the leaf-scaling case, whereas in scaling all branches, the curve is slightly gentler. This can be intuitively understood considering that the leaf-scaling is varying only the relevant part of the tree, whereas when the whole tree is scaled, the effect on the phylogeny is more evenly dispersed, resulting in a more gradual effect on detection power.

- **Number of recombinant regions** K This varies the number of topologically distinct regions in the alignment. In analyzing these alignments, the number of HMM states, k, is set to the correct value K.

  We observe that the method is relatively stable with respect to number of regions for the values we tested (2-8), provided that the number of regions is correctly specified. When this part of the model is mis-specified, the results are mixed, and we show results from simulation studies varying the model structure for fixed alignments in Figure 7.2. The $K-$varying plot in Figure 7.1 does not take into account the possibility that non-bordering regions were wrongly annotated as coming from the same state. The breakpoints of these regions would still be detected, but their tree topology would be incorrect. Thus, we re-emphasize that recHMM is primarily a breakpoint-detection tool, and that if serious inferences are to be made from the trees within each hidden state, then more sophisticated tree construction methods should be used on the separate alignment regions.

We examined the effect of the following parameters in data analysis:

- **Predicted number of recombinant regions** For certain values of K above, we vary k to see how detection power is affected when we have greater or fewer HMM states than distinct regions. We would have liked to vary k extensively for every value of K, but we as were limited by computing time, we analyzed only $k = 4, 5, 6, 7, 8$ for $K = 5$.

  From this study, we concluded that specifying more states increases sensitivity, but at a slight cost of PPV. Intuitively, if a model has too few states, discordant regions may be merged together and modeled by a consensus topology, instead of being correctly modeled as separate recombinant regions with their own tree-states. If excessively many states are used, then presumably more of the genuinely differing regions will be modeled, but also small, spurious regions of convergent mutations or rate heterogeneity could be modeled by one of the extra states, leading to falsely predicted breakpoints. In many cases, the cutoff criterion helps in filtering out small errant regions, and we see only a moderate drop in PPV in Figure 7.2 for 7 and 8 states.

- **Length of cutoff criterion** The cutoff criterion is the value of the smallest distance between breakpoints we allow in our predicted state path. For a detailed description on how this cutoff criterion informs our posterior decoding algorithm, see the Methods section and Appendix F. Simply stated, we disregard paths with breakpoints occurring within the cutoff of each other when choosing a maximal path.

  In Figure 7.1, we see that sensitivity rises as we allow for shorter region predictions (by specifying a higher cutoff value). PPV shows the opposite trend; with smaller cutoff criteria, we can be increasingly certain that any breakpoints we find are true breakpoints. The cutoff value where the two curves intersect can be thought of as a value which optimally balances sensitivity and PPV, and so in our simulated data analysis we set the cutoff to 30 bp.

In our analysis of real data, we cover a range of data sizes and types, ranging between 4 and 9 taxa, with length ranging from 700 bp to circa 10,000 bp. We find that in each case, we are able to recover the previous authors' predictions for breakpoints. In many cases, we find compelling evidence for additional, often shorter recombinant regions that the original analysis either missed completely or registered as minor 'spikes' in their plot. In each example we highlight the aspects of our method that contribute to its sensitivity, flexibility and utility. In the case where we had no additional predictions to add to a dataset, we omitted that analysis for brevity. For example, we analyzed the data from [157], but the low mutation rate enabled their simpler approach to adequately determine breakpoints. In this situation we acknowledge that our method is able, but not necessary, to analyze the data.

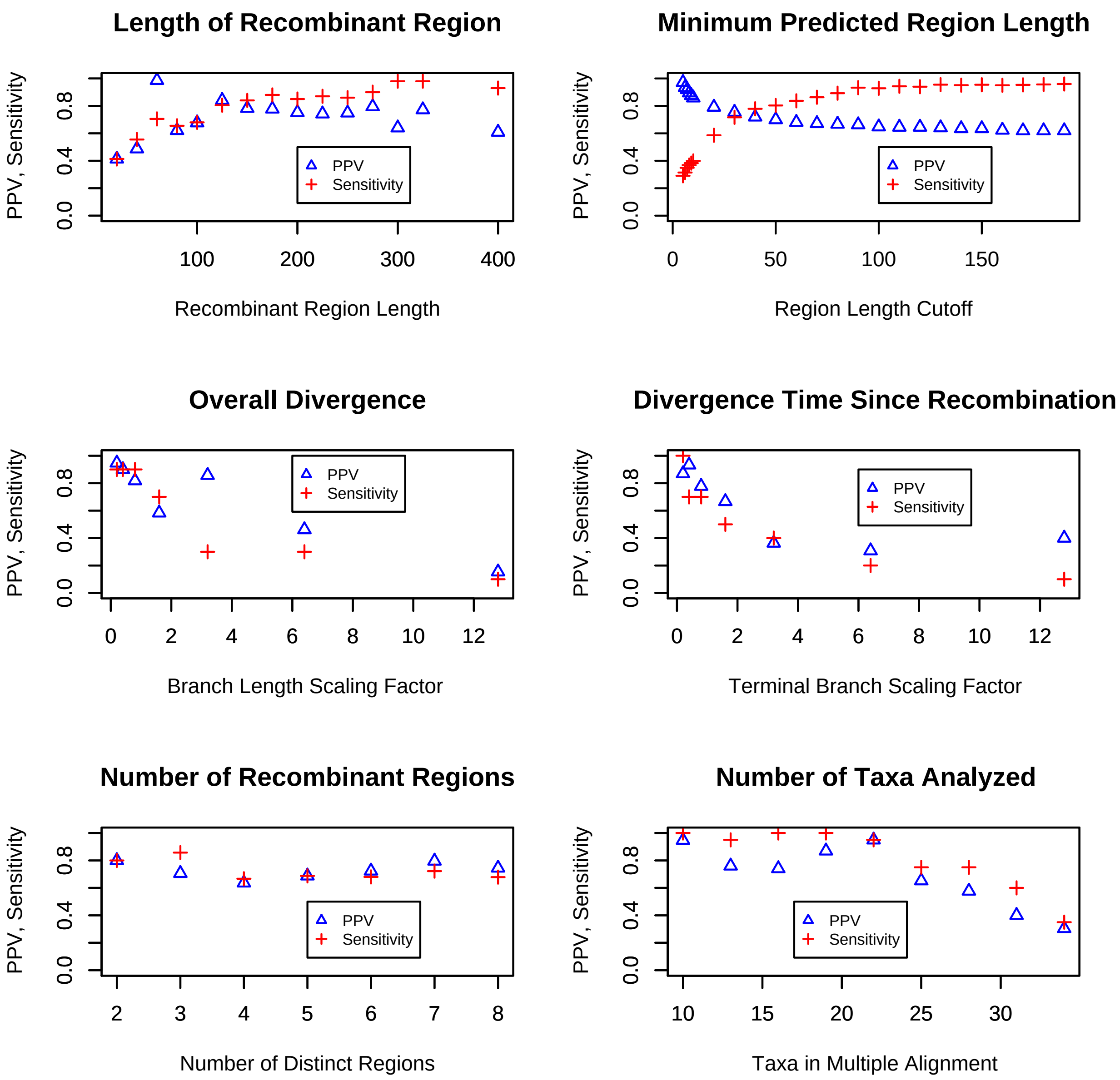


Figure 7.1: Accuracy of breakpoint detection varies as a function of simulation and inference parameters. In each case, we plot both positive predictive value (TP/(TP+FP) = PPV) and sensitivity (TP/(TP+FN)). A correctly predicted breakpoint is defined as one which occurs less than 10 bp from a true breakpoint. We observe that the overall accuracy remains high except for situations of high diversity, extremely short recombinant region (less than 50 bp), or more than 20 taxa. In several cases, we were resource-limited and only able to provide a few data points for each variable, and this is the reason for the sparseness of the plots. Each data point is the maximum-likelihood outcome of 10 independently run EM trials, each one taking on average 15 minutes for small length/taxa, though this varies as seen in Figure 7.10.

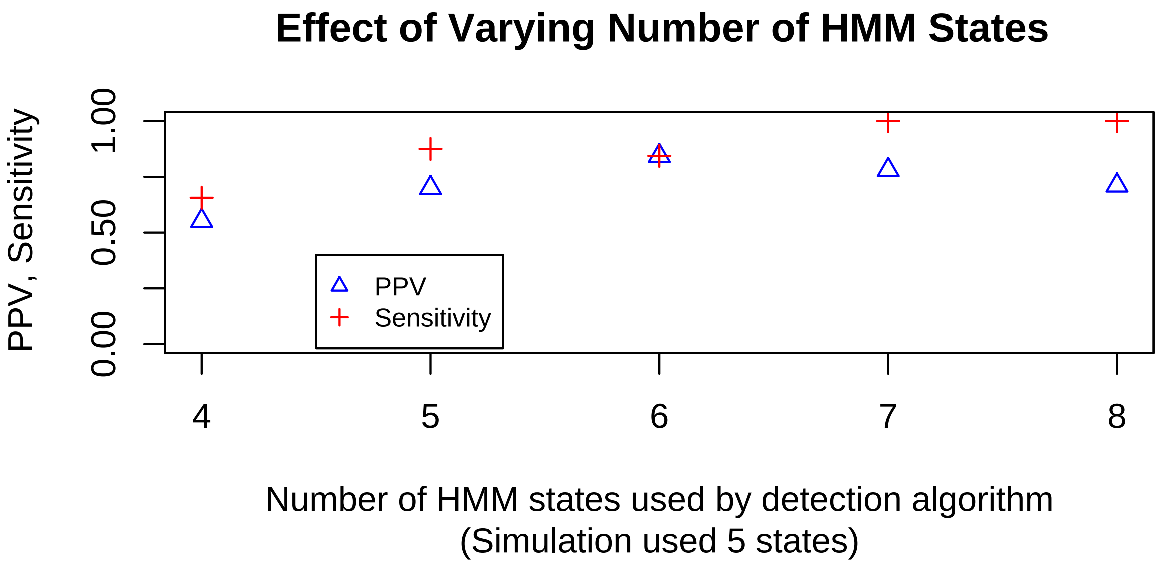


Figure 7.2: The detection power increases as more trees are added to the model. Here we analyze alignments with 5 regions, while setting our predicted number of states to various values. The sensitivity increases steadily while PPV tapers off at a fixed value.

***Neisseria* ArgF and penA genes** We used our program to analyze data from *Neisseria* data that consisted of single gene regions suspected of recombination. In these analyses, recombinant regions were quite short and we demonstrate that our method is capable of handling this situation.

In their 2001 work, Husmeier and Wright use a similar tree-topology HMM to detect recombination. Since each EM iteration involves searching over all possible tree topologies for the optimal trees for each region, they were limited to alignments of 4 taxa, where there are only 3 unrooted phylogenies [27]. As mentioned earlier, both this and window-based methods assume one part of the recombination inference problem to be solved. In this case, the method allocates one tree per HMM state, and so estimation of the trees is no longer necessary, leaving only the alignment partitioning problem to be solved. Our results on this dataset are shown in Figure 7.3. The previous predictions are shown in red dashed lines. The horizontal axis refers to the position within the alignment, and the vertical axis is partitioned according to posterior state probability of the HMM. The posterior state probability can intuitively be thought of as the probability that a certain column was generated by a certain phylogenetic tree, taking into account the model structure and all the alignment data. At each position, the posterior probabilities for the three trees must sum to one, and hence the different colors partition the vertical axis. We were able to closely replicate their results (namely the state probabilities depicted in Figure 15 of [27]).

In comparing our results to theirs, we note that our program, which does a probabilistic

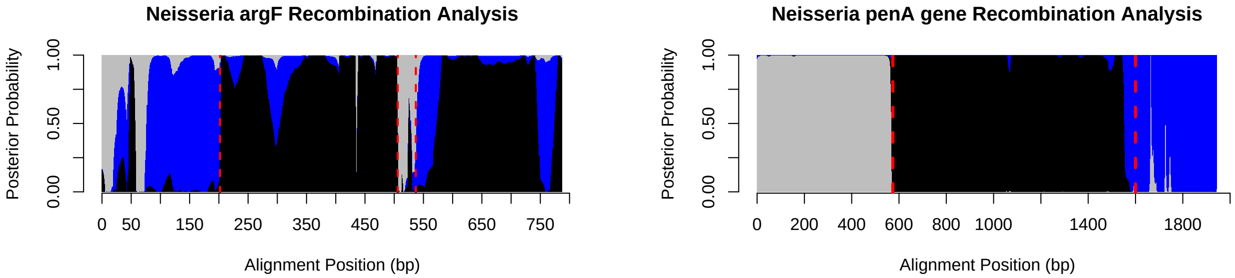


Figure 7.3: The left plot shows the analysis of *Neisseria* argF data with predictions from Husmeier and Wright, who used a similar method, [27] in red dashed lines. We confirm each of their breakpoints and are able to better characterize uncertain regions. Still, the region from 0-75 remains difficult to characterize. Different colors represent posterior probabilities of different tree-topology states in the HMM, and sharp changes in color indicates likely recombination breakpoints. The right figure shows analysis of *Neisseria* penA data, an alignment of 9 taxa of length circa 1900 bp, demonstrating our ability to analyze many taxa. We confirm with high posterior probabilities the two breakpoints previously found by Bowler *et al*, shown in red [165].

tree-updating step, rather than providing a hidden state for all possible topologies, recovers all the breakpoints of the previous study. At positions 202, 507, and 538 there are clearly points at which different colors have high posterior probabilites. In regions such as 0-50, it is difficult to make reliable inferences because with so few bases, phylogenetic tree construction is unreliable. As mentioned in the Methods section, we employ a simple length cutoff heuristic whereby all recombinant regions smaller than a certain length are removed. Though this is less sophisticated than, say, explicitly specifying a prior distribution over state paths which takes length into account, it performs reasonably well for the situations we analyzed. In considering putative crossovers, points where a tree with high posterior probability changes abruptly in favor of a different tree should be considered most closely. Also, topology changes that are extremely short could be the result of spurious convergent mutations, in which two phylogenetically distant species undergo mutations to the same base, making it seem as if they had exchanged information. Note also that our method is better able to characterize the regions 537-560 and 750-780. In [27], 537-560 is classified as “irregular”, and 750-780 shows only a spike in probability in Figure 13 of [27], and not at all in their Figure 15. We predict the 500-600 region to be composed of two separate topologies, and 750-780 to be a possibly newly characterized recombinant region, having the same topology as the 100-202 region.

In [165], Bowler *et al* discovered a mosaic structure in the PenA gene of *Neisseria Meningitidis* which conferred resistance to Penicillin. Analyzing a DNA multiple alignment between 9 species, they were able to manually determine estimates for recombination breakpoints. Constructing phylogenetic trees for each of the regions gave them clues as to the donors of the acquired regions, after which these predictions were experimentally verified. In contrast, our method is able to simultaneously partition and estimate the trees of a recombinant alignment. In Figure 7.3 we see that our method predicts nearly the same breakpoints with high posterior probabilities. The alignment analyzed was composed of 9 species, covering the range of virulent and commensal *Neisseria* subtypes, with length 1950 bp. This analysis demonstrates the ability of our phylo-HMM to effectively make use of alignments with relatively many taxa, a notable advantage over Husmeier *et al*'s method. For a quantitative look at how detection power varies with taxa number, see Figure 7.1. By using so many subtypes for comparison, Bowler *et al* were able to precisely determine which species were the donors and recipients of the recombinant regions, and subsequently verified these predictions in a laboratory setting. If they had been limited to 4 taxa, the analysis would have had to be repeated many times to cover all the possibilities. Biologically, the results in [165] motivate a search for recombination *within* genes implicated in resistance, in contrast to the multiple resistance gene transfer that has been previously studied, and this is a possible application of our method.

HIV-1 whole-genome scans In order to determine the effectiveness of our method on longer alignments, we analyzed several datasets of entire genomes (10,000 bp) of HIV-1 strains suspected of inter-subtype recombination. Our method is equally able to perform on the genome scale as it is on the single-gene scale. In *Neisseria* argF, one of the predicted recombinant regions was only 30 bp long, whereas in HIV they range from 100 bp to 6 kb. This is a notable advantage over sliding-window methods which have a fixed resolution to be used over the whole scan. We demonstrate here that we are able to determine breakpoints between both large and small recombinant regions, making our method a promising tool for comparative analysis of HIV and similar genomes. In analyzing data from previous studies, we recovered all the breakpoints found by the previous authors. In cases in which we found additional breakpoints, we describe them below, but otherwise we omit the plots for brevity.

HIV-1 CRF01_AE/B from Malaysia Figure 7.4 depicts our results on a new Malaysian HIV strain previously analyzed by Lau *et al* [19]. We recover all six of the breakpoints inferred by the original authors, who used a SimPlot/Bootscanning approach, and also we find two new breakpoints whose significance appears equal to those found previously. In Figure 7.4, we show for comparison the results from bootscanning, which Lau *et al* used for their inference of recombination breakpoints. Lau *et al* provided precise breakpoint positions, and these are plotted in our diagram as red dashed lines. Since bootscanning typically removes gaps from multiple alignments before analysis, the breakpoint positions do not align with Lau *et al*'s plot very well, and we provide rough mapping between plots. All six of their breakpoint predictions are well-represented in our analysis. Note the 'spike' in likelihood at around nt 5800 in Lau *et al*'s plot. This region registered as strongly recombinant in our analysis, depicted as the grey region in region nt 6415-6594. Lau *et al*'s characterization of the 1500 to 2000 region (2141 to 2856 in ours) is marked somewhat by uncertainty in the optimal tree topology; their " % trees" line wavers and is never very close to 100%,

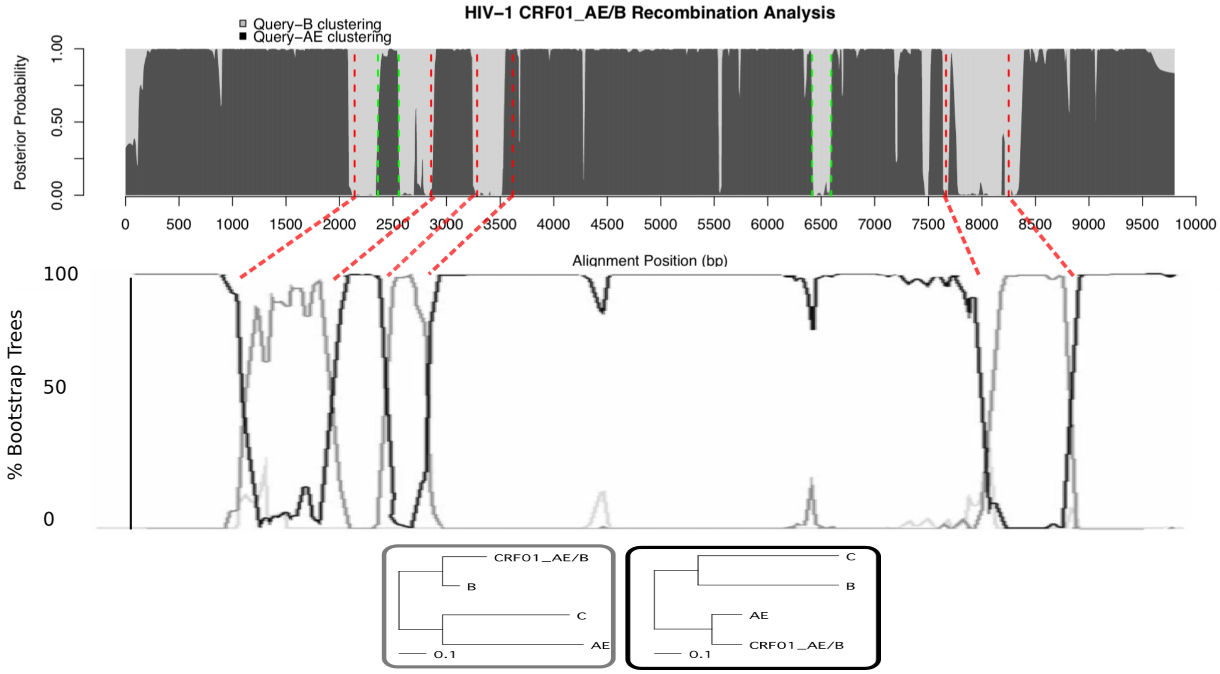


Figure 7.4: The top figure shows our analysis of the strain CRF01_AE/B Malaysian HIV-1 with our recombination phylo-HMM. We recover 6 previously predicted recombination breakpoints (red), and predict new regions in 6415-6594 and 2360-2553 (green). The grey and black regions correspond to posterior probabilities of the trees shown in the lowest figure. Previous bootscanning analysis of the same data is shown in the middle figure [19]. Since their method involved removing gaps from the alignment, we provide approximate mappings from our predictions to their plot, as the red dashed lines between the two figures. They provided precise breakpoint locations in their paper based on consensus HXB2 strain, which we plot in our figure as the vertical red lines. Note the spike in their plot that appears in our plot around 6500 as a recombinant region. The trees in the lowest figure were those trained as hidden states in our HMM; the black state clearly shows the query strain clustering with CRF_AE, whereas the gray tree shows a closer relationship with subtype B, in accordance with the previous findings.

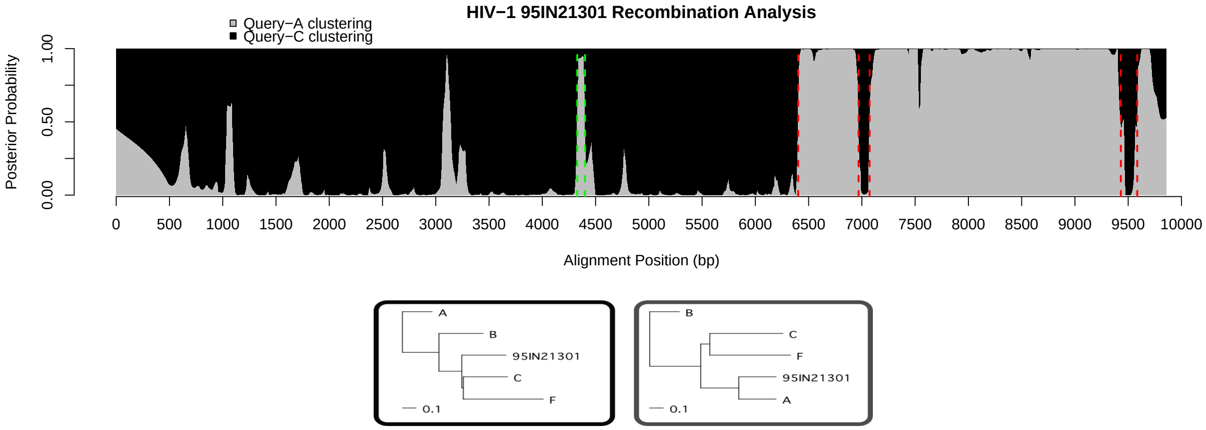


Figure 7.5: Analysis of A/C Indian HIV-1 recombinant strain 95IN21301. In the original paper [24], gaps were stripped and so mapping predictions to our plot is difficult. Instead, we show our confirmations in red, which correspond closely to the predictions seen in Figures 1 and 2 of [24]. Our new prediction of region 4328-4401 is shown in green. Trees trained as hidden HMM states are shown underneath, with their colored boxes corresponding to the colors in the plot, which in turn denote posterior probabilities of hidden states. Note that in the black tree the query sequence doesn't cluster with C, but the branch length from the (C,F) clade to the query strain is effectively zero, indicating a star-like topology in these areas.

in contrast to their inference of region 3000 to 5500, where the line remains constant and close to 100%. This uncertainty suggests that there may be additional recombination points within that region, as is more conclusively shown in our diagram. We venture that the region between nt 2141 and 2856 can be further partitioned by two more breakpoints, at nt 2360 and 2553, shown in Figure 7.4.

When using bootscanning, there is a lower limit to the size of the recombinant region that can be found, which depends closely on the size of the sliding window. The 3283-3617 region, at just over 300 nt, is clearly found, but smaller regions registered only as spikes or showed uncertainty of the region. Our method is probabilistic, and instead of defining sharp partitions of the alignment, we allow the parameter training to gradually decide which regions to use to train different trees. In our analyses, we consistently found that our program is able to find small recombinant regions better than others' methods. In this case, as the posterior probabilities become more certain of the alignment partitioning, each of the grey regions contributes its information in updating the grey tree. If a sliding window was being passed over the alignment, each region would have to 'fend for itself' in conveying its phylogenetic signal, and small regions would go undetected. Because we use information from the entire alignment during tree-optimization and sum over all possible tree-column assignments, our approach is computationally more expensive but allows collaboration between small recombinant regions, and, consequently, improved detection.

HIV-1 A/C Recombinant 95IN21301 from India We examined data from the original SimPlot paper by Lole *et al.* In this work, the authors test five newly-sequence HIV-1 strains from India, and find one of them to be recombinant [24]. We examine this strain and confirm all five of their breakpoints and offer one new prediction. SimPlot detects mosaic strains by plotting the similarity of a query strain to other subtype reference strains. The similarity is computed within a sliding window of predefined size, according to various criteria (eg Hamming distance, Jukes-Cantor distance, etc). The result is a visualization of the closest relative of different regions of the query sequence. This is similar in effect to bootscanning distance-based phylogenetic methods (eg Neighbor-joining), and suffers from many of the same pitfalls. For example, in their whole-genome analysis of strain 95IN21301, Lole *et al* used a window size of 600bp, severely limiting the resolution of recombination detection. They conclusively found breakpoints around nt 6400 and 9500 (since gaps were removed, it is difficult to determine exact breakpoint predictions from their plot alone). They then did a second, finer-scale analysis with window size 200 bp on just the *env* and *nef* genes which were suspected to be recombinant. Within each of these single-gene regions they found an additional breakpoint in which the query sequence more closely represented subtype C.

In our analysis, we confirmed all five of these breakpoints by using our method (again, gap-stripping made exact comparison somewhat limited), and our result is shown in Figure 7.5; breakpoints previously found are in red, and new predictions in green. Since we do not have to specify a window and use instead a probabilistic weighting scheme, we are able to detect large regions (eg the break at position 6402) just as well as shorter regions (eg 6969-7073, 9431-9585). Furthermore, the method uses information from the entire alignment, rather than partitioning it by windows. In this case, it's possible that attempting to train a phylogenetic tree T on the region R between nt 6969 and 7073 wouldn't have yielded conclusive results. If region R has high posterior probability of being generated by the 'black' tree that is dominant from positions 1-6402, the following M-Step will incorporate more counts from region R, and so when region R is examined, the inference algorithm recognizes that these columns 'fit' perfectly with the black topology, which corresponds to having high emission probabilities from the black HMM state. Also, a short 83 bp region is found supporting grey topology, in which 95IN21301 clusters with subtype A. This region is short, and its posterior probability never reaches 1, but a neighbor-joining tree on this section, 4328-4401 supports this clustering. In this way, our method is able to take into account information from the entire alignment, rather than defining a rigid window which can skip over small recombinant regions.

Three HIV-1 BF recombinants from Brazil We considered data from Filho *et al* [20]. Their data was composed of 10 newly sequenced strains from Brazil determined to have varying levels and structure of mosaicism, as determined by bootscan analysis. We confirm their predictions (from Figure 2 of [20]) in strains PM12313, BREPM11871, and BREPM16704 and we find several more small recombinant regions. Each of the new recombinant regions we find share breakpoints with other strains we analyzed as well as strain CRF12_BF [21], suggesting they could be hotspots for recombination activity.

As seen in Figure 7.6, strain BREPM12313 showed a clear recombinant region from nt 1322-2571, previously characterized by Filho, *et al.* Also, a region around 4700-5000 showed some evidence of recombination, having the same topology as 1322-2571. As this region's posterior probability is more 'spike-shaped,' rather than having sharp borders between colors, it is difficult to say whether or not it is an ambiguous region or a genuine recombinant. It does share one crossover point with strain BREPM11871, giving it somewhat more credibility. Performing neighbor-joining on nt 4784-4945 (eg positions where posterior probability is higher for grey) showed BREPM12313 clustering with subtype F. At the end of the genome, another possible recombinant region is seen, at around 9700. This region includes only gaps for the query sequence, and thus the inference is not reliable. Our method treats gaps as missing information, and when they are present in small numbers reliability is not affected, but in places like this where only gaps are present it can hinder the tree-inference.

Strain BREPM16704 was previously predicted to have four breakpoints, which we recovered with remarkably high posterior probabilities for the tree-states. Figure 7.7 shows our results with previous predictions in red. A new region, at 9281-9405, shows high posterior probability and is common to BREPM11871 and CRF12BF [21], making a strong case for a recombination hotspot.

In strain BREPM11871, all four breakpoints predicted by Filho, *et al* were found, as well as a new crossover region, common to BREPM 16704, at 9238-9361 (shown in green in Figure 7.8). The break previously described at nt 5462 bp was predicted by our method to be at 5277. To determine the more likely crossover point, we performed 1000 bootstrapping trials on each of the following regions: 4782-5277 (our prediction), 4782-5462 (Filho, *et al*'s prediction), and 5277-5462 (the disputed region). We found that the 5277-5462 region strongly supported BREPM11871 clustering with subtype B, with 98.2% bootstrap support. Moreover, bootstrap support for query-F clustering appears higher for our predicted region (99.9%) than the previous prediction (85.1%). We conclude that our algorithm often outperforms previous methods in accurately determining recombination breakpoints.

7.3 Discussion

Recombination is an important force driving genome evolution, and in several cases it is the primary force for diversity. As such, methods which can detect and characterize recombination events are crucial to the successful utilization of new sequence data. On the single-gene level, recombination has been shown, in at least one case, to be able to confer antibiotic resistance [165]. It could be possible that inter-subtype recombination conferring drug resistance is a common phenomenon, which could be investigated using our methods. On the multiple-gene scale, *Chlamydia trachomatis* has been shown to undergo frequent inter-subtype recombination resulting in mosaic genomes [157] which complicate subtype definition and classification. On the genome scale, HIV-1 has long been known to participate in recombination leading to several identified circulating recombinant forms (CRFs). For these

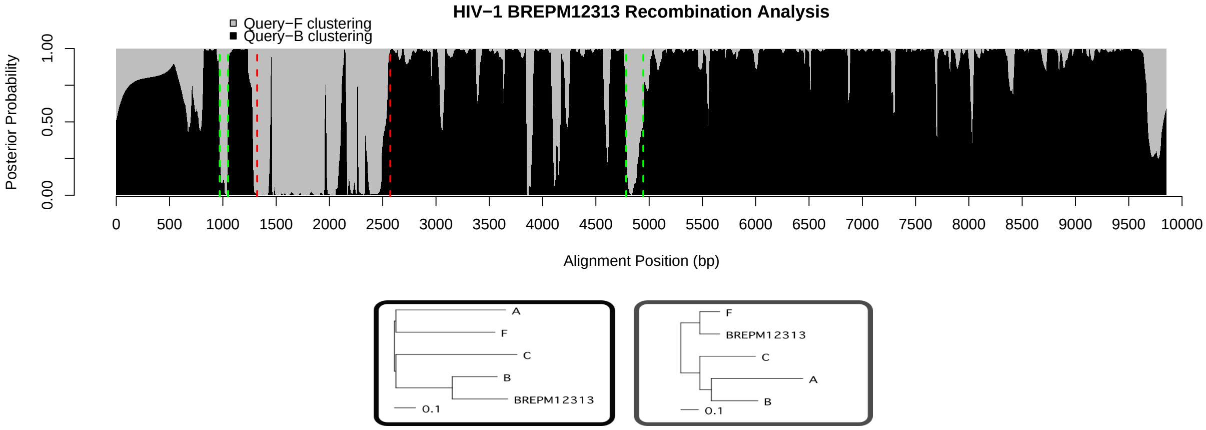


Figure 7.6: Brazilian strain BREPM12313. We confirm Filho, *et al*'s breakpoints near 1322 and 2571 (red), and predict new recombinant regions in nt 4784-4945 as well as 970-1049 (green). The second of these is short, but present in some form in all three strains analyzed here. The spike at 3851-3909 is even shorter and is not represented in the other two species, leading us to not predict it as a likely recombinant region. Trees trained in hidden states are shown below the plot.

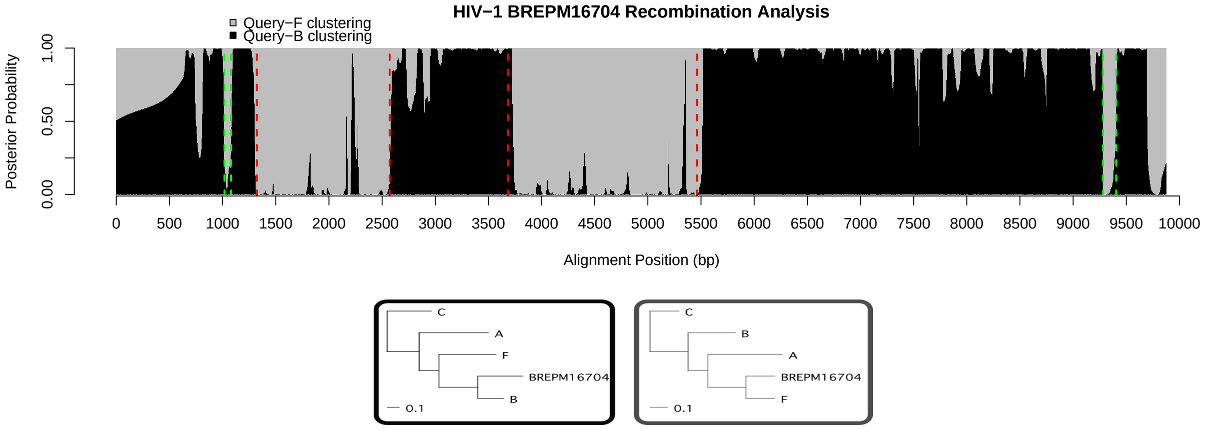


Figure 7.7: Brazilian strain BREPM16704. We confirm breakpoints near 1322, 2571, and 5462 (red) and predict recombinations in 9281-9405 and 1017-1085 (green). Trees trained in hidden states are shown below the plot.

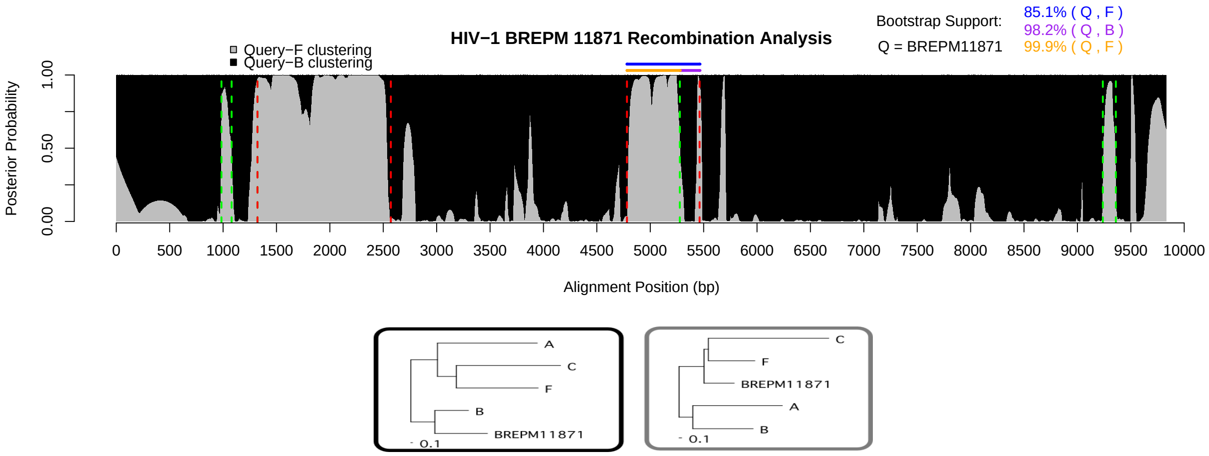


Figure 7.8: Brazilian strain BREPM11871. Confirmation of breakpoints 1322 and 2571, and 4782 (red dashed lines). We predict a region common to BREPM16704 at 9238-9361 (green). Also, the breakpoint previously estimated at 5462 (red) we propose to be at 5277 (green dashed line). In support of this, we provide bootstrapping values (1000 replicates) for the 3 different regions, indicated by horizontal colored lines above the plot. Our prediction (orange) carries the highest value, 99.9%, whereas the previous (blue) is only 85.1%, since it includes a region (purple) that strongly supports BREPM11871 clustering with subtype B, with value 98.2%. The small region at 985-1080 is difficult to confidently categorize, but its high posterior probability for clustering with F and its agreement with the other two strains lead us to suspect a recombination. Trees trained in hidden states are shown below the plot.

clinically relevant pathogens, accurate detection of recombination following introgression is important not only to guide disease treatment methods, but also for tracing the epidemiological history of the virus. In this work we present a novel method for recombination detection which we believe to be more sensitive, flexible, and robust in the aforementioned evolutionary scenario. We combine two long-standing concepts, phylogenetics and hidden Markov models, in a maximum-likelihood framework to model topology changes over an alignment of related sequences.

We present a training scheme which attempts to solve the two problems embedded within recombination detection simultaneously. We model evolution probabilistically with a continuous-time Markov chain which directs the likelihood-based tree construction algorithm [28]. Furthermore, our alignment-partitioning is handled with posterior probabilities which take into account each hidden state tree. By summing over all possible tree-column assignments and not using sharp window cutoffs, we are able to perform more precise breakpoint determination. We can adjust the specificity and sensitivity of the model with the transition matrix of the HMM, which dictates how much of a likelihood change should cause the model inference to change states. We believe this *likelihood* comparison to be superior to adjusting the size of the window because it enables distant sections of the alignment to combine their phylogenetic signal in training a hidden state of the HMM.

7.4 Methods

The goal of this study is to start with a multiple alignment of sequence data and find positions where recombination events have occurred. This is done by recovering a set of phylogenetic trees and a map that assigns a tree to each column. The points at which neighboring columns have different tree assignments will indicate possible locations of recombination events in evolutionary history.

EM for Recombination-HMM We use a hidden Markov model with tree topologies as hidden states and alignment columns as observed states. The usual method to train HMM parameters is by the specialization of the EM algorithm known as the Baum-Welch algorithm. Our transitions can be optimized in the usual way, but the emissions are more difficult since their likelihood is governed by the tree topologies in the hidden states, which are not easily optimized. For this problem, we employ an EM method for trees, developed by Friedman, *et al* [28], within our M-step for emission probabilities. Their phylogenetic inference algorithm is implemented with such improvements as simulated annealing and noise injection in SEMPHY, available from their website at http://compbio.cs.huji.ac.il/semphy/. We instead implemented our own 'bare-bones' version of this algorithm, without these improvements. To progressively assign columns to trees, at each M-step, we weight the expected counts of the tree-EM by the posterior counts of the phylo-HMM. Intuitively, this guides the tree-maximization by providing comparatively more information from regions which fit a particular tree. We give a high-level description of our method here and more detail is provided in Appendix F. Figure 7.9 gives a graphical representation of the overall task-flow. After running our program on the alignment data to estimate parameters, a state path through the HMM is computed from the final matrix of posterior probabilities. We would like this path to represent a balance between being biologically reasonable and highly probable under our model. Thus, we propose maximizing the sum of posterior state probabilities subject to a length cutoff by the following method.

Let π be a path of length m through the state space of the HMM (discounting start and end states), emitting the first m columns of the alignment, with one state per column. Let M be the total number of columns in the alignment. We say that the state path is *partial* if $m \leq M$, and *complete* if $m = M$.

Let π_n denote the n^{th} state in path π. The *score* of path π is defined as $S(\pi) = \sum_{n=1}^{m} P[\pi_n, n]$ where $P[i, m]$ is the posterior probability that column m was emitted by state i.

We say a state path π is *valid* if all its breakpoints are more than ϵ apart. That is, there exist no $m, n > 1$ with $n - m < \epsilon$ such that $\pi_{m-1} \neq \pi_m$ and $\pi_m \neq \pi_n$.

Finding the maximal valid π is solved by a simple dynamic programming procedure, outlined in Appendix F.

Since the EM algorithm has a tendency to converge to local likelihood maxima which are not global maxima, especially when initialized randomly, we ran the algorithm several times for each dataset, took the highest-likelihood result for the set of trials, and performed the above posterior decoding on the final distribution. We show plots of our program's performance when various aspects of the model and input alignment are varied in Figure 7.10.

Algorithm: Recombination Phylo-HMM Training

Input: An alignment D, an integer K specifying number of trees, a guess at the true tree topologies and transition matrix. In practice, we initialized the transition matrix to have values of .999 on the diagonal, and split the remaining .001 among the remaining columns. Transition probabilities are trained in the normal Baum-Welch manner. We noted that inference was relatively robust to the initial value of this parameter.

Output: A proposed MLE of K phylogenetic trees and the transition matrix. This determines a posterior state distribution, from which we deduce breakpoints.

E-Step $_{Outer}$: Calculate the Forward, Backward, and posterior probability arrays in the usual manner for HMMs, as in [17]. The emission probabilities for each tree-state are computed by Felsenstein's pruning algorithm [7]. Use the Forward and Backward arrays to compute the expected transitions between hidden states.

M-Step $_{Outer}$:

1. Maximize the transition probabilities to obtain a new estimate of the transition matrix.
2. Find a new set of trees, using model selection with Structural EM:

   E-Step $_{Inner}$: For all K trees, compute expected counts of hidden data $S_{ij}(a,b)$ (see below). Scale counts for tree k from column m by the posterior probability that column m was emitted from state/tree k.

   M-Step $_{Inner}$: Increase the likelihood of each tree topology with Structural EM (see below) [28].

The hidden data for each tree are defined as $S_{ij}(a,b) = P(x_i = a, x_j = b | D, (T,t))$, the number of transitions from nucleotide a to nucleotide b from node i to node j, for all pairs of nodes. For neighboring nodes, this can be computed by Elston and Stewart's Peeling algorithm [64], and for non-neighboring nodes, exact computation requires a dynamic programming routine described in the original Structural EM paper [28]. In this work, the authors showed that the likelihood contribution of an edge between two nodes can be summarized as a function of this count $S_{ij}(a,b)$.

If we arrange these summaries in a weight matrix W in which W_{ij} represents the expected likelihood contribution resulting from placing an edge from node i to node j, it is easy to see that the maximum expected likelihood tree will be the topology which maximizes the sum of its edge scores. Finding such a topology is trivial if we do not require that the tree is binary, for instance by maximum spanning tree algorithms. This (possibly non-binary) tree

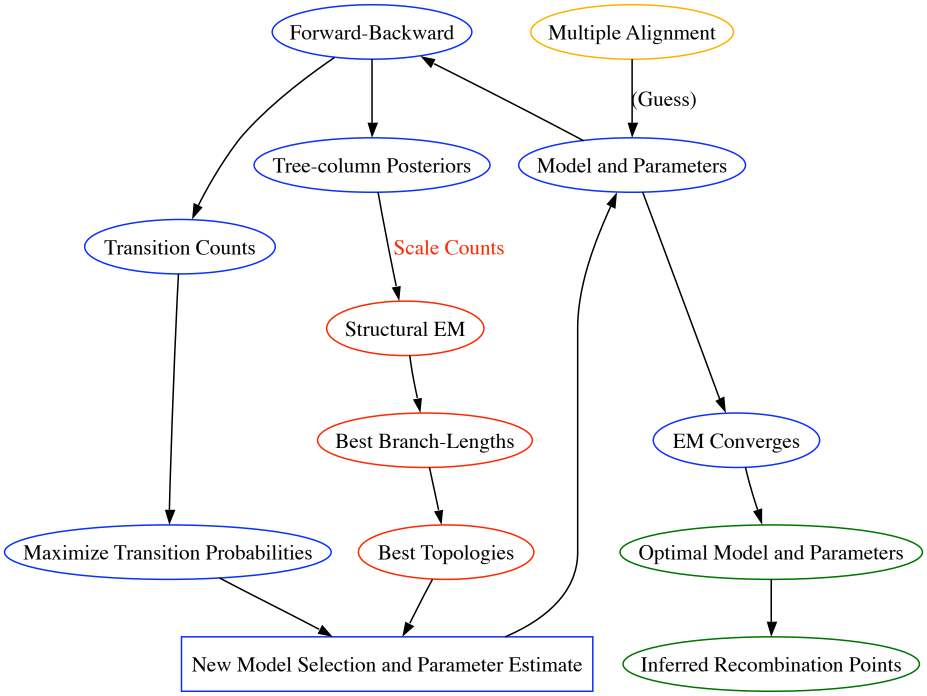


Figure 7.9: Phylo-HMM training algorithm: Input Alignment ⟹ Model Selection/Parameter Estimation ⟹ Recombination Inference

is then transformed to a binary tree by operations which do not alter the tree's likelihood. In this way, Structural EM allows for iteratively constructing higher likelihood trees by choosing the next tree which maximizes the expected likelihood based on the current tree. The reader is referred to Appendix F and the original Structural EM paper [28] for more detailed discussions of this algorithm which is crucial to our method.

In our methods, instead of allowing Structural EM to converge, we allow two iterations using the same set of transition and posterior probabilities, as a heuristic substitute for finding the true hidden tree topologies.

Possible Extensions We outline here a number of extensions which could grow directly from this work. One of the strengths of the method is its generality and flexibility, and so we believe it is ideally suited for continued development.

- **Sequence Evolution Model** Currently we model gaps as missing information (eg. summing over possible values). This is not realistic and may hinder phylogenetic reconstruction, and consequently recombination inference. The simplest possible next step is to treat a gap as a '5th nucleotide.' While this assumes independence among inserted and deleted residues, it has been shown to aid phylogenetic reconstructions more than treating gaps as missing characters [166]. Our code uses the HKY85 sub-

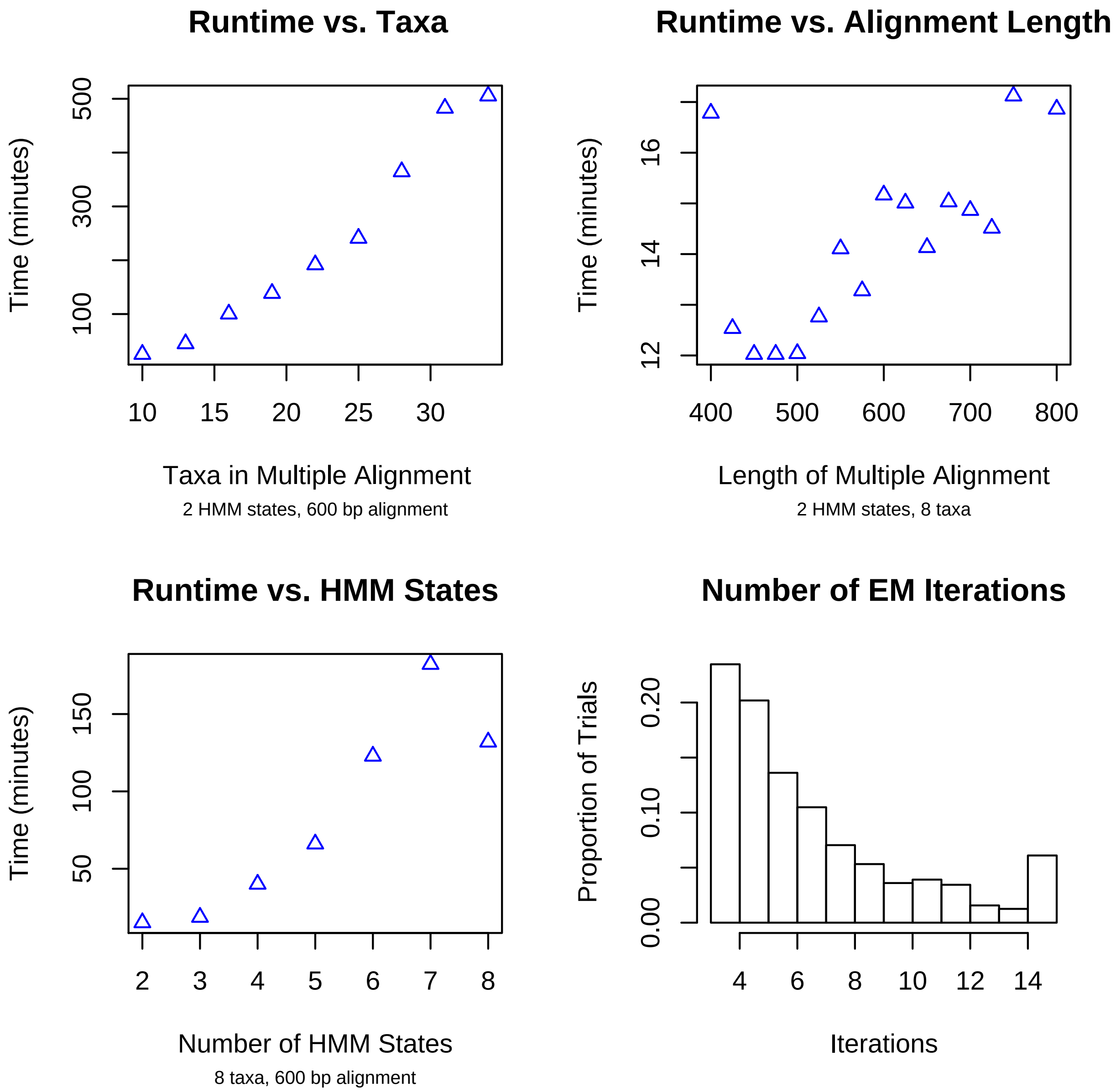


Figure 7.10: Resource use of the algorithm increases with model complexity. The algorithm converges in a reasonable number of EM steps, as seen in the lower right plot. We observed no dependence of iterations to convergence and the model complexity, and so the lower right histogram represents data concatenated from all simulation trials. The final bar in the histogram represents the proportion of trials which took 14 or more iterations to converge.

stitution model, whose matrix exponential is solved in closed form. A more general rate matrix diagonalization and exponentiation is currently only implemented in the Python prototype of our core dynamic program, which we find to be too slow (the experiments reported in this paper used a core dynamic programming algorithm implemented in C, for speed). This is, however, purely a technical issue, and modeling gaps is entirely feasible. Similar elaborations of the substitution model, such as codon evolution (if the region of interest was protein-coding) or an extra hidden state determining coding and non-coding regions, might provide more accurate modeling of large genome-scale alignments.

- **ARG-like trees and k** The method currently does not restrict the k-tuples of trees produced at each iteration. As [167] point out, not all groups of trees can fit together to produce an ARG. Restricting the tree groups would give a more conclusive answer to the epidemiological recombination question, and may even be helpful in informing the tree selection heuristic. One can imagine a simple extension to our method which attempts to learn k as well as producing consistent trees:

  At each training iteration:

  - If the trees maximized in each hidden state are not consistent:
    - First, find a set of trees in the usual manner, without regard to whether they are consistent. Then, find the best-scoring set of trees which are consistent. This is computationally intensive but not intractable, since we can enumerate ordered spanning trees for each of our hidden states from our E-Step. Once this set has been found, if the likelihood difference between the inconsistent and consistent set is deemed acceptable, accept the trees and begin a new training iteration.
    - Otherwise, if this likelihood penalty is deemed to large, we recognize that the current k is inadquate to describe the data, and so we add a new hidden state to the model, and continue training.

  A simpler way of estimating k would be to run a coarser heuristic method (eg. SimPlot) and seed the HMM with the number of states that it finds.

Sequence Data All sequence data used in this study was downloaded from public databases (GenBank and LANL HIV Database). The sequences were aligned with MUSCLE [168] with the default parameters. Gaps in the alignments were treated as missing information. Bootstrap analyses were performed with CLUSTAL W [79] with 1000 replicates and the default parameters. The GenBank identifiers for sequences used are as follows, grouped by figure: Figure 7.3: argF: X64860, X64866, X64869, X64873; penA: X59624-X59635; Figure 7.5: AF067158, AB253429, AF067159, M17451, AF005494; Figure 7.4: AB032740, AB023804, AY713408, EF495062; Figures 7.6-7.8: AF286228, AF005494, AY173956, AB098332, AY173956, DQ085867, DQ085876, DQ085872.

The source code for our programs, though still being developed, is available upon request or through CVS. For documentation, contact, and download information see
http://biowiki.org/RecHMM

Bibliography

1. Darwin C, Bynum W (2009) The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life. AL Burt.

2. Zuckerkandl E, Pauling L (1965) Molecules as documents of evolutionary history. Journal of theoretical biology 8: 357–366.

3. Needleman SB, Wunsch CD (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology 48: 443-453.

4. Holmes I (2003) Using guide trees to construct multiple-sequence evolutionary HMMs. Bioinformatics 19 Suppl. 1: i147-157.

5. Löytynoja A, Goldman N (2005) An algorithm for progressive multiple alignment of sequences with insertions. Proceedings of the National Academy of Sciences of the USA 102: 10557-62.

6. Löytynoja A, Goldman N (2008) Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. Science 320: 1632–1635.

7. Felsenstein J (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. Journal of Molecular Evolution 17: 368-376.

8. Burton P, Clayton D, Cardon L, Craddock N, Deloukas P, et al. (2007) Genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls. Nature 447: 661–678.

9. Rioux J, Xavier R, Taylor K, Silverberg M, Goyette P, et al. (2007) Genome-wide association study identifies new susceptibility loci for crohn disease and implicates autophagy in disease pathogenesis. Nature genetics 39: 596–604.

10. Thomas G, Jacobs K, Yeager M, Kraft P, Wacholder S, et al. (2008) Multiple loci identified in a genome-wide association study of prostate cancer. Nature genetics 40: 310–315.

11. Lambert J, Heath S, Even G, Campion D, Sleegers K, et al. (2009) Genome-wide association study identifies variants at clu and cr1 associated with alzheimer's disease. Nature genetics 41: 1094–1099.

12. Ng P, Henikoff S (2003) Sift: Predicting amino acid changes that affect protein function. Nucleic acids research 31: 3812–3814.

13. Adzhubei I, Schmidt S, Peshkin L, Ramensky V, Gerasimova A, et al. (2010) A method and server for predicting damaging missense mutations. Nature methods 7: 248–249.

14. Ferrer-Costa C, Gelpí J, Zamakola L, Parraga I, De La Cruz X, et al. (2005) Pmut: a web-based tool for the annotation of pathological mutations on proteins. Bioinformatics 21: 3176–3178.

15. Marini N, Thomas P, Rine J (2010) The use of orthologous sequences to predict the impact of amino acid substitutions on protein function. PLoS genetics 6: e1000968.

16. Chomsky N (1956) Three models for the description of language. IRE Transactions Information Theory 2: 113-124.

17. Durbin R, Eddy S, Krogh A, Mitchison G (1998) Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge, UK: Cambridge University Press.

18. Klosterman PS, Uzilov AV, Bendana YR, Bradley RK, Chao S, et al. (2006) XRate: a fast prototyping, training and annotation tool for phylo-grammars. BMC Bioinformatics 7.

19. Lau K, Wang B, Kamarulzaman A, Ng K, Saksena N (2007) Near full-length sequence analysis of a unique crf01 ae/b recombinant from kuala lumpur, malaysia. AIDS Research and Human Retroviruses 23: 1139-1145.

20. Filho D, Sucupira M, Casiero M, Sabino E, Diaz R, et al. (2006) Identification of two hiv type 1 circulating recombinant forms in brazil. AIDS Research & Human Retroviruses 22: 1–13.

21. Thomson M, Delgado E, Herrero I, Villahermosa M, Vázquez-de Parga E, et al. (2002) Diversity of mosaic structures and common ancestry of human immunodeficiency virus type 1 bf intersubtype recombinant viruses from argentina revealed by analysis of near full-length genome sequences. Journal of general virology 83: 107–119.

22. Kew O, Wright P, Agol V, Delpeyroux F, Shimizu H, et al. (2004) Circulating vaccine-derived polioviruses: current state of knowledge. Bulletin of the World Health Organization 82: 16–23.

23. Simon-Loriere E, Holmes E (2011) Why do rna viruses recombine? Nature Reviews Microbiology 9: 617–626.

24. Lole K, Bollinger R, Paranjape R, Gadkari D, Kulkarni S, et al. (1999) Full-length human immunodeficiency virus type 1 genomes from subtype c-infected seroconverters in india, with evidence of intersubtype recombination. Journal of virology 73: 152–160.

25. Archer J, Pinney J, Fan J, Simon-Loriere E, Arts E, et al. (2008) Identifying the important hiv-1 recombination breakpoints. PLoS computational biology 4: e1000178.

26. Baird H, Galetto R, Gao Y, Simon-Loriere E, Abreha M, et al. (2006) Sequence determinants of breakpoint location during hiv-1 intersubtype recombination. Nucleic acids research 34: 5203–5216.

27. Husmeier D, Wright F (2001) Detection of recombination in dna multiple alignments with hidden markov models. Journal of Computational Biology 8: 401–427.

28. Friedman N, Ninio M, Pe'er I, Pupko T (2001) A structural EM algorithm for phylogenetic inference. In: Lengauer T, Sankoff D, Istrail S, Pevzner P, Waterman M, editors, Proceedings of the Fifth Annual International Conference on Computational Biology. New York: Association for Computing Machinery.

29. Westesson O, Lunter G, Paten B, Holmes I (2011) An alignment-free generalization to indels of Felsenstein's phylogenetic pruning algorithm. arXiv .

30. Yang Z (1994) Estimating the pattern of nucleotide substitution. Journal of Molecular Evolution 39: 105-111.

31. Rannala B, Yang Z (1996) Probability distribution of molecular evolutionary trees: a new method of phylogenetic inference. Journal of Molecular Evolution 43: 304-311.

32. Siepel A, Haussler D (2004) Combining phylogenetic and hidden Markov models in biosequence analysis. Journal of Computational Biology 11: 413-428.

33. Yang Z, Nielsen R, Goldman N, Pedersen AM (2000) Codon-substitution models for heterogeneous selection pressure at amino acid sites. Genetics 155: 432-449.

34. Thorne JL, Goldman N, Jones DT (1996) Combining protein evolution and secondary structure. Molecular Biology and Evolution 13: 666-673.

35. Siepel A, Haussler D (2004) Phylogenetic estimation of context-dependent substitution rates by maximum likelihood. Molecular Biology and Evolution 21: 468-488.

36. Knudsen B, Hein J (1999) RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. Bioinformatics 15: 446-454.

37. Siepel A, Haussler D (2004) Computational identification of evolutionarily conserved exons. In: Bourne P, Gusfield D, editors, Proceedings of the eighth annual international conference on research in computational molecular biology, San Diego, March 27-31 2004. ACM, pp. 177-186.

38. Pedersen JS, Bejerano G, Siepel A, Rosenbloom K, Lindblad-Toh K, et al. (2006) Identification and classification of conserved RNA secondary structures in the human genome. PLoS Computational Biology 2: e33.

39. Matsen FA, Kodner RB, Armbrust EV (2010) pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. BMC Bioinformatics 11: 538.

40. Kschischang FR, Frey BJ, Loeliger HA (1998) Factor graphs and the sum-product algorithm. IEEE Transactions on Information Theory 47: 498-519.

41. Sankoff D, Cedergren RJ (1983) Simultaneous comparison of three or more sequences related by a tree. In: Sankoff D, Kruskal JB, editors, Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison, Reading, MA: Addison-Wesley, chapter 9. pp. 253-264.

42. Hein J (2001) An algorithm for statistical alignment of sequences related by a binary tree. In: Altman RB, Dunker AK, Hunter L, Lauderdale K, Klein TE, editors, Pacific Symposium on Biocomputing. Singapore: World Scientific, pp. 179-190.

43. Lee C, Grasso C, Sharlow M (2002) Multiple sequence alignment using partial order graphs. Bioinformatics 18: 452–464.

44. Mohri M, Pereira F, Riley M (2002) Weighted finite-state transducers in speech recognition. Computer Speech and Language 16: 69-88.

45. Hein J, Wiuf C, Knudsen B, Moller MB, Wibling G (2000) Statistical alignment: computational properties, homology testing and goodness-of-fit. Journal of Molecular Biology 302: 265-279.

46. Holmes I, Bruno WJ (2001) Evolutionary HMMs: a Bayesian approach to multiple alignment. Bioinformatics 17: 803-820.

47. Metzler D (2003) Statistical alignment based on fragment insertion and deletion models. Bioinformatics 19: 490-499.

48. Suchard MA, Redelings BD (2006) BAli-Phy: simultaneous Bayesian inference of alignment and phylogeny. Bioinformatics 22: 2047–2048.

49. Westesson O, Lunter G, Paten B, Holmes I (2012) Accurate reconstruction of insertion-deletion histories by statistical phylogenetics. PLoS ONE 7: e34572.

50. Searls DB, Murphy KP (1995) Automata-theoretic models of mutation and alignment. In: Rawlings C, Clark D, Altman R, Hunter L, Lengauer T, et al., editors, Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology. Menlo Park, CA: AAAI Press, pp. 341-349.

51. Bradley RK, Holmes I (2007) Transducers: an emerging probabilistic framework for modeling indels on trees. Bioinformatics 23: 3258–3262.

52. Satija R, Pachter L, Hein J (2008) Combining statistical alignment and phylogenetic footprinting to detect regulatory elements. Bioinformatics 24: 1236-1242.

53. Paten B, Herrero J, Fitzgerald S, Beal K, Flicek P, et al. (2008) Genome-wide nucleotide-level mammalian ancestor reconstruction. Genome Research 18: 1829–1843.

54. Thorne JL, Kishino H, Felsenstein J (1991) An evolutionary model for maximum likelihood alignment of DNA sequences. Journal of Molecular Evolution 33: 114-124.

55. Miklós I, Lunter G, Holmes I (2004) A long indel model for evolutionary sequence alignment. Molecular Biology and Evolution 21: 529-540.

56. Knudsen B, Miyamoto M (2003) Sequence alignments and pair hidden Markov models using evolutionary history. Journal of Molecular Biology 333: 453-460.

57. Rivas E (2005) Evolutionary models for insertions and deletions in a probabilistic modeling framework. BMC Bioinformatics 6.

58. Bradley RK, Holmes I (2009) Evolutionary triplet models of structured RNA. PLoS Comput Biol 5: e1000483.

59. Http://www.graphviz.org/.

60. Karlin S, Taylor H (1975) A First Course in Stochastic Processes. San Diego, CA: Academic Press.

61. Thorne JL, Kishino H, Felsenstein J (1992) Inching toward reality: an improved likelihood model of sequence evolution. Journal of Molecular Evolution 34: 3-16.

62. Motwani R, Raghavan P (2010) Randomized algorithms. Chapman & Hall/CRC.

63. Holmes I (2005) Accelerated probabilistic inference of RNA structure evolution. BMC Bioinformatics 6.

64. Elston RC, Stewart J (1971) A general model for the genetic analysis of pedigree data. Human Heredity 21: 523-542.

65. Jukes TH, Cantor C (1969) Evolution of protein molecules. In: Mammalian Protein Metabolism, New York: Academic Press. pp. 21-132.

66. Hasegawa M, Kishino H, Yano T (1985) Dating the human-ape splitting by a molecular clock of mitochondrial DNA. Journal of Molecular Evolution 22: 160-174.

67. Wong KM, Suchard MA, Huelsenbeck JP (2008) Alignment uncertainty and genomic analysis. Science 319: 473-6.

68. Qu X, Swanson R, Day R, Tsai J (2009) A guide to template based structure prediction. Curr Protein Pept Sci 10: 270-85.

69. Moses AM, Chiang DY, Pollard DA, Iyer VN, Eisen MB (2004) MONKEY: identifying conserved transcription-factor binding sites in multiple alignments using a binding site-specific evolutionary model. Genome Biology 5.

70. Pollard KS, Salama SR, Lambert N, Lambot M, Coppens S, et al. (2006) An RNA gene expressed during cortical development evolved rapidly in humans. Nature 443: 167–172.

71. Thompson JD, Plewniak F, Poch O (1999) A comprehensive comparison of multiple sequence alignment programs. Nucleic Acids Research 27: 2682-2690.

72. Markova-Raina P, Petrov D (2011) High sensitivity to aligner and high rate of false positives in the estimates of positive selection in the 12 Drosophila genomes. Genome Research 21: 863–874.

73. Nelesen S, Liu K, Zhao D, Linder CR, Warnow T (2008) The effect of the guide tree on multiple sequence alignments and subsequent phylogenetic analyses. Pacific Symposium on Biocomputing 2008: 25–36.

74. Liu K, Nelesen S, Raghavan S, Linder CR, Warnow T (2009) Barking up the wrong treelength: the impact of gap penalty on alignment and tree accuracy. IEEE/ACM Trans Comput Biol Bioinform 6: 7–21.

75. project consortium E (2007) Analyses of deep mammalian sequence alignments and constraint predictions for 1% of the human genome. Genome Research 17: 760–774.

76. Bradley RK, Uzilov AV, Skinner ME, Bendana YR, Barquist L, et al. (2009) Evolutionary modeling and prediction of non-coding RNAs in Drosophila. PLoS ONE 4: e6478.

77. Strope C, Abel K, Scott S, Moriyama E (2009) Biological sequence simulation for testing complex evolutionary hypotheses: indel-seq-gen version 2.0. Mol Biol Evol 26: 2581-93.

78. Edgar RC (2004) MUSCLE: a multiple sequence alignment method with reduced time and space complexity. BMC Bioinformatics 5: 113.

79. Larkin M, Blackshields G, Brown N, Chenna R, McGettigan P, et al. (2007) Clustal W and Clustal X version 2.0. Bioinformatics 23: 2947–2948.

80. Katoh K, Kuma K, Toh H, Miyata T (2005) Mafft version 5: improvement in accuracy of multiple sequence alignment. Nucleic Acids Research 33: 511–518.

81. Higgins DG, Bleasby AJ, Fuchs R (1992) CLUSTAL V: improved software for multiple sequence alignment. Computer Applications in the Biosciences 8: 189-191.

82. Cartwright RA (2005) DNA assembly with gaps (Dawg): simulating sequence evolution. Bioinformatics 21 Suppl 3: iii31-8.

83. Bradley RK, Roberts A, Smoot M, Juvekar S, Do J, et al. (2009) Fast statistical alignment. PLoS Computational Biology 5: e1000392.

84. Kamneva O, Liberles A, Ward N (2010) Genome-wide influence of indel substitutions on evolution of bacteria of the pvc superphylum, revealed using a novel computational method. Genome Biology and Evolution 2: 870-886.

85. Zhang Z, Huang J, Wang Z, WAng L, Peiji G (2011) Impact of indels on the flanking regions in structural domains. Molecular Biology and Evolution 28: 291-301.

86. Zhu L, Wang Q, Tang P, Araki H, Tian D (2009) Genomewide association between insertions/deletions and the nucleotide diversity in bacteria. Molecular Biology and Evolution 26: 2353-2361.

87. Gomez-Valero L, Latorre A, Gil R, Gadau J, Feldhaar H, et al. (2008) Patterns and rates of nucleotide substitution, insertion and deletion in the endosymbiont of ants blochmannia floridanus. Molecular Ecology 17: 4382-4392.

88. Clark AG, Eisen MB, Smith DR, Bergman CM, Oliver B, et al. (2007) Evolution of genes and genomes on the Drosophila phylogeny. Nature 450: 203-218.

89. Sinha S, Siggia E (2005) Sequence turnover and tandem repeats in cis-regulatory modules in drosophila. MBE 22.

90. Lunter G (2007) Probabilistic whole-genome alignments reveal high indel rates in the human and mouse genomes. Bioinformatics 23: 289-296.

91. Heger A, Ponting C (2008) OPTIC: orthologous and paralogous transcripts in clades. NAR 36: 267-270.

92. de la Chaux N, Messeer P, Arndt P (2007) DNA indels in coding regions reveal selective contraints on protein evolution in the human lineage. BMC Evolutionary Biology 7.

93. Wang Z, Martin J, Abubucker S, Yin Y, Gasser R, et al. (2009) Systematic analysis of insertions and deletions specific to nematode proteins and their proposed functional and evolutionary relevance. BMC Evol Biol 9.

94. Saccone S, Quan J, Mehta G, Bolze R, Thomas P, et al. (2011) New tools and methods for direct programmatic access to the dbSNP relational database. Nucleic Acids Res .

95. Beissbarth T, Speed TP (2004) GOstat: find statistically overrepresented Gene Ontologies within a group of genes. Bioinformatics 20: 1464–1465.

96. Yang Z (2007) PAML 4: phylogenetic analysis by maximum likelihood. Molecular Biology and Evolution 24: 1586-1591.

97. (2002) Initial sequencing and comparative analysis of the mouse genome. Nature .

98. Mills R, Luttig C, Larkins C, Beauchamp A, Tsui C, et al. (2006) An initial map of insertion and deletion (indel) variation in the human genome. Genome Research 16.

99. Westesson O, Barquist L, Holmes I (2012) Handalign: Bayesian multiple sequence alignment, phylogeny, and ancestral reconstruction. Bioinformatics .

100. Yang Z, dos Reis M (2011) Statistical properties of the branch-site test of positive selection. Molecular biology and evolution 28: 1217.

101. Malaspinas A, Eriksson N, Huggins P (2011) Parametric analysis of alignment and phylogenetic uncertainty. Bulletin of mathematical biology 73: 1–16.

102. Hanson-Smith V, Kolaczkowski B, Thornton J (2010) Robustness of ancestral sequence reconstruction to phylogenetic uncertainty. Molecular biology and evolution 27: 1988.

103. Varadarajan A, Bradley RK, Holmes I (2008) Tools for simulating evolution of aligned genomic regions with integrated parameter estimation. Genome Biology 9.

104. Arribas-Gil A (2010) Parameter estimation in multiple-hidden IID models from biological multiple alignment. Statistical Applications in Genetics and Molecular Biology 9: 10.

105. Lunter GA, Drummond AJ, Miklós I, Hein J (2004) Statistical alignment: Recent progress, new applications, and challenges. In: Nielsen R, editor, Statistical methods in Molecular Evolution, Springer Verlag, Series in Statistics in Health and Medicine, chapter 14. pp. 375-406.

106. Lunter G, Miklos I, Drummond A, Jensen JL, Hein J (2005) Bayesian coestimation of phylogeny and sequence alignment. BMC Bioinformatics 6: 83.

107. Fleissner R, Metzler D, von Haeseler A (2005) Simultaneous statistical multiple alignment and phylogeny reconstruction. Syst Biol 54: 548–561.

108. Redelings BD, Suchard MA (2005) Joint Bayesian estimation of alignment and phylogeny. Systematic Biology 54: 401-418.

109. Bouchard-Côté A, Jordan MI, Klein D (2009) Efficient inference in phylogenetic InDel trees. In: Advances in Neural Information Processing Systems 21 (NIPS). Vancouver, Canada.

110. Holmes I (2007) Phylocomposer and Phylodirector: Analysis and Visualization of Transducer Indel Models. Bioinformatics 23: 3263-3264.

111. Holmes I, Rubin GM (2002) An Expectation Maximization algorithm for training hidden substitution models. Journal of Molecular Biology 317: 757-768.

112. Gelfand A (2000) Gibbs sampling. Journal of the American Statistical Association 95: 1300–1304.

113. Lunter G (2007) HMMoC–a compiler for hidden Markov models. Bioinformatics 23: 2485–2487.

114. Leonard C, Spellman M, Riddle L, Harris R, Thomas J, et al. (1990) Assignment of intrachain disulfide bonds and characterization of potential glycosylation sites of the type 1 recombinant human immunodeficiency virus envelope glycoprotein (gp120) expressed in chinese hamster ovary cells. Journal of Biological Chemistry 265: 10373.

115. Frazer K, Ballinger D, Cox D, Hinds D, Stuve L, et al. (2007) A second generation human haplotype map of over 3.1 million snps. Nature 449: 851–861.

116. Altshuler D, Lander E, Ambrogio L, Bloom T, Cibulskis K, et al. (2010) A map of human genome variation from population scale sequencing. Nature 467: 1061–1073.

117. Wheeler D, Srinivasan M, Egholm M, Shen Y, Chen L, et al. (2008) The complete genome of an individual by massively parallel dna sequencing. Nature 452: 872–876.

118. Siepel A, Bejerano G, Pedersen JS, Hinrichs AS, Hou M, et al. (2005) Evolutionarily conserved elements in vertebrate, insect, worm, and yeast genomes. Genome Research 15: 1034-1050.

119. Abecasis A, Vandamme A, Lemey P (2009) Quantifying differences in the tempo of human immunodeficiency virus type 1 subtype evolution. Journal of virology 83: 12917–12924.

120. Loeb D, Swanstrom R, Everitt L, Manchester M, Stamper S, et al. (1989) Complete mutagenesis of the hiv-1 protease. Nature 340: 397–400.

121. Rennell D, Bouvier S, Hardy L, Poteete A, et al. (1991) Systematic mutation of bacteriophage t4 lysozyme. Journal of molecular biology 222: 67.

122. Markiewicz P, Kleina L, Cruz C, Ehret S, Miller J, et al. (1994) Genetic studies of the lac repressor. xiv. analysis of 4000 altered escherichia coli lac repressors reveals essential and non-essential residues, as well as" spacers" which do not require a specific sequence. Journal of molecular biology 240: 421.

123. Wei Q, Wang L, Wang Q, Kruger W, Dunbrack Jr R (2010) Testing computational prediction of missense mutation phenotypes: functional characterization of 204 mutations of human cystathionine beta synthase. Proteins: Structure, Function, and Bioinformatics 78: 2058–2074.

124. Westesson O, Holmes I (2012) Developing and applying heterogeneous phylogenetic models with xrate. PLoS ONE 7: e36898.

125. Goldman N, Thorne J, Jones D, et al. (1996) Using evolutionary trees in protein secondary structure prediction and other comparative sequence analyses. Journal of Molecular Biology 263: 196–208.

126. Thorne J, Goldman N, Jones D (1996) Combining protein evolution and secondary structure. Molecular Biology and Evolution 13: 666–673.

127. Siepel A, Haussler D (2004) Combining phylogenetic and hidden markov models in biosequence analysis. Journal of Computational Biology 11: 413–428.

128. Adachi J, Hasegawa M (1996) Model of amino acid substitution in proteins encoded by mitochondrial dna. Journal of Molecular Evolution 42: 459–468.

129. Kimura M (1980) A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. Journal of Molecular Evolution 16: 111-120.

130. Meyer IM, Durbin R (2004) Gene structure conservation aids similarity based gene prediction. Nucleic Acids Research 32: 776-783.

131. Eddy SR (1998) Profile hidden Markov models. Bioinformatics 14: 755-763.

132. Garber M, Guttman M, Clamp M, Zody M, Friedman N, et al. (2009) Identifying novel constrained elements by exploiting biased substitution patterns. Bioinformatics .

133. Slater GSC, Birney E (2005) Automated generation of heuristics for biological sequence comparison. BMC Bioinformatics 6: 31.

134. Birney E, Durbin R (1997) Dynamite: a flexible code generating language for dynamic programming methods used in sequence comparison. In: Gaasterland T, Karp P, Karplus K, Ouzounis C, Sander C, et al., editors, Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology. Menlo Park, CA: AAAI Press, pp. 56-64.

135. Drummond AJ, Rambaut A (2007) BEAST: Bayesian evolutionary analysis by sampling trees. BMC Evolutionary Biology 7.

136. Knudsen B, Hein J (2003) Pfold: RNA secondary structure prediction using stochastic context-free grammars. Nucleic Acids Research 31: 3423-3428.

137. Heger A, Ponting CP, Holmes I (2009) Accurate estimation of gene evolutionary rates using XRATE, with an application to transmembrane proteins. Molecular Biology and Evolution 26: 1715–1721.

138. Ayres D, Darling A, Zwickl D, Beerli P, Holder M, et al. (2011) Beagle: an application programming interface and high-performance computing library for statistical phylogenetics. Systematic Biology .

139. The Stockholm file format. `http://www.cgb.ki.se/cgb/groups/sonnhammer/Stockholm.html`.

140. The Newick file format. `http://evolution.genetics.washington.edu/phylip/newicktree.html`.

141. GFF: an exchange format for gene-finding features. Webpage at http://www.sanger.ac.uk/Software/GFF/.

142. Saitou N, Nei M (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. Molecular Biology and Evolution 4: 406-425.

143. Penn O, Stern A, Rubinstein ND, Dutheil J, Bacharach E, et al. (2008) Evolutionary modeling of rate shifts reveals specificity determinants in hiv-1 subtypes. PLoS Computational Biology 4: e1000214.

144. Pedersen JS, Meyer IM, Forsberg R, Simmonds P, Hein J (2004) A comparative method for finding and folding RNA secondary structures within protein-coding regions. Nucleic Acids Research 32: 4925-4923.

145. Watts J, Dang K, Gorelick R, Leonard C, Bess J, et al. (2009) Architecture and secondary structure of an entire hiv-1 rna genome. Nature .

146. Col format: http://colformat.kvl.dk/. URL `COLFormat:http://colformat.kvl.dk/`.

147. Zuker M (1989) Computer prediction of RNA structure. Methods in Enzymology 180: 262-288.

148. Skinner ME, Uzilov AV, Stein LD, Mungall CJ, Holmes IH (2009) JBrowse: a next-generation genome browser. Genome Res 19: 1630–1638.

149. Wiggle track format: https://cgwb.nci.nih.gov/goldenpath/help/wiggle.html. URL `https://cgwb.nci.nih.gov/goldenPath/help/wiggle.html`.

150. Stein L, Mungall C, Shu S, Caudy M, Mangone M, et al. (2002) The generic genome browser: a building block for a model organism system database. Genome Research 12: 1599-1610.

151. Kent WJ, Sugnet CW, Furey TS, Roskin KM, Pringle TH, et al. (2003) The human genome browser at UCSC. Genome Research 12: 996-1006.

152. Goecks J, Nekrutenko A, Taylor J, Afgan E, Ananda G, et al. (2010) Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol 11: R86.

153. Westesson O, Holmes I (2009) Accurate detection of recombinant breakpoints in whole-genome alignments. PLoS Comput Biol 5: e1000318.

154. Awadalla P (2003) The evolutionary genomics of pathogen recombination. Nature Reviews Genetics 4: 50–60.

155. Minin V, Dorman K, Fang F, Suchard M (2007) Phylogenetic mapping of recombination hotspots in human immunodeficiency virus via spatially smoothed change-point processes. Genetics 175: 1773–1785.

156. Hein J, Schierup M, Wiuf C (2005) Gene genealogies, variation and evolution: a primer in coalescent theory. Oxford University Press, USA.

157. Gomes J, Bruno W, Nunes A, Santos N, Florindo C, et al. (2007) Evolution of chlamydia trachomatis diversity occurs by widespread interstrain recombination involving hotspots. Genome research 17: 50–60.

158. Kedzierska A, Husmeier D (2006) A heuristic bayesian method for segmenting dna sequence alignments and detecting evidence for recombination and gene conversion. Statistical Applications in Genetics and Molecular Biology 5: 1–32.

159. Anderson T, Haubold B, Williams J, Estrada-Franco J, Richardson L, et al. (2000) Microsatellite markers reveal a spectrum of population structures in the malaria parasite plasmodium falciparum. Molecular Biology and Evolution 17: 1467–1482.

160. Song Y, Hein J (2005) Constructing minimal ancestral recombination graphs. Journal of Computational Biology 12: 147–169.

161. Minichiello M, Durbin R (2006) Mapping trait loci by use of inferred ancestral recombination graphs. The American Journal of Human Genetics 79: 910–922.

162. Siepel A, Korber B (1995) Scanning the database for recombinant hiv-1 genomes. Human retroviruses and AIDS .

163. Arenas M, Posada D (2007) Recodon: coalescent simulation of coding dna sequences with recombination, migration and demography. BMC bioinformatics 8: 458.

164. Puigbò P, Garcia-Vallvé S, McInerney J (2007) Topd/fmts: a new software to compare phylogenetic trees. Bioinformatics 23: 1556–1558.

165. Bowler L, Zhang Q, Riou J, Spratt B (1994) Interspecies recombination between the pena genes of neisseria meningitidis and commensal neisseria species during the emergence of penicillin resistance in n. meningitidis: natural events and laboratory simulation. Journal of bacteriology 176: 333–337.

166. McGuire G, Denham M, Balding D (2001) Models of sequence evolution for dna sequences containing gaps. Molecular Biology and Evolution 18: 481–490.

167. Hudson R, Kaplan N (1985) Statistical properties of the number of recombination events in the history of a sample of dna sequences. Genetics 111: 147–164.

168. Edgar RC (2004) Muscle: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Research 32: 1792-1797.

169. Posada D, Buckley T (2004) Model selection and model averaging in phylogenetics: advantages of akaike information criterion and bayesian approaches over likelihood ratio tests. Systematic biology 53: 793–808.

170. Murray K, Barton D (2003) Poliovirus cre-dependent vpg uridylylation is required for positive-strand rna synthesis but not for negative-strand rna synthesis. Journal of virology 77: 4739–4750.

171. Goodfellow I, Chaudhry Y, Richardson A, Meredith J, Almond J, et al. (2000) Identification of a cis-acting replication element within the poliovirus coding region. Journal of virology 74: 4590–4600.

172. Barton D, O'Donnell B, Flanegan J (2001) 5 cloverleaf in poliovirus rna is a cis-acting replication element required for negative-strand synthesis. The EMBO journal 20: 1439–1448.

173. GOODFELLOW I, KERRIGAN D, EVANS D (2003) Structure and function analysis of the poliovirus cis-acting replication element (cre). Rna 9: 124–137.

174. Martínez-Salas E, Ramos R, Lafuente E, de Quinto S (2001) Functional interactions in internal translation initiation directed by viral and cellular ires elements. Journal of General Virology 82: 973–984.

175. Clever J, Sassetti C, Parslow T (1995) Rna secondary structure and binding sites for gag gene products in the 5'packaging signal of human immunodeficiency virus type 1. Journal of virology 69: 2101–2109.

176. Alford R, Honda S, Lawrence C, Belmont J (1991) Rna secondary structure analysis of the packaging signal for moloney murine leukemia virus. Virology 183: 611–619.

177. Han J, Townsend H, Jha B, Paranjape J, Silverman R, et al. (2007) A phylogenetically conserved rna structure in the poliovirus open reading frame inhibits the antiviral endoribonuclease rnase l. Journal of virology 81: 5561–5572.

178. Merino E, Wilkinson K, Coughlan J, Weeks K (2005) Rna structure analysis at single nucleotide resolution by selective 2'-hydroxyl acylation and primer extension (shape). Journal of the American Chemical Society 127: 4223–4231.

179. St George J (2003) Gene therapy progress and prospects: adenoviral vectors. Gene Therapy 10: 1135–1141.

180. Roth J, Cristiano R (1997) Gene therapy for cancer: What have we done and where are we going? Journal of the National Cancer Institute 89: 21–39.

181. Bainbridge J, Smith A, Barker S, Robbie S, Henderson R, et al. (2008) Effect of gene therapy on visual function in leber's congenital amaurosis. New England Journal of Medicine 358: 2231–2239.

182. Mancuso K, Mauck M, Kuchenbecker J, Neitz M, Neitz J (2010) A multi-stage color model revisited: implications for a gene therapy cure for red-green colorblindness. Retinal Degenerative Diseases : 631–638.

183. Yu M, Poeschla E, Wong-Staal F, et al. (1994) Progress towards gene therapy for hiv infection. Gene therapy 1: 13.

184. Flotte T, Carter B, et al. (1995) Adeno-associated virus vectors for gene therapy. Gene therapy 2: 357.

185. Wu Z, Asokan A, Samulski R (2006) Adeno-associated virus serotypes: vector toolkit for human gene therapy. Molecular therapy 14: 316–327.

186. Dong J, Fan P, Frizzell R (1996) Quantitative analysis of the packaging capacity of recombinant adeno-associated virus. Human gene therapy 7: 2101–2112.

187. Perabo L, Endell J, King S, Lux K, Goldnau D, et al. (2006) Combinatorial engineering of a gene therapy vector: directed evolution of adeno-associated virus. The journal of gene medicine 8: 155–162.

188. Fisher K, Jooss K, Alston J, Yang Y, Haecker S, et al. (1997) Recombinant adeno-associated virus for muscle directed gene therapy. Nature medicine 3: 306–312.

189. Maheshri N, Koerber J, Kaspar B, Schaffer D (2006) Directed evolution of adeno-associated virus yields enhanced gene delivery vectors. Nature biotechnology 24: 198–204.

190. Do CB, Brudno M, Batzoglou S (2004). PROBCONS: Probabilistic consistency-based multiple alignment of amino acid sequences. Submitted.

191. Holmes I (2000) A DART tutorial. Berkeley Drosophila Genome Project, LSA Room 539, UC Berkeley. A tutorial for probabilistic methods and hidden Markov models, presented with the aid of the author's software package implementing many common HMM algorithms. Available from http://www.fruitfly.org/˜ihh/.

192. Documentation of Stockholm alignment file format: `http://biowiki.org/StockholmFormat`.

193. Felsenstein J (1973) Maximum-likelihood estimation of evolutionary trees from continuous characters. American Journal of Human Genetics 25: 471-492.

194. Van Dyk D (2000) Nesting em algorithms for computational efficiency. Statistica Sinica 10: 203–226.

Part III

Appendices

Appendix A

Experimental connections

Experimental data provide a unique, orthogonal lens for interpreting the models presented in Parts I and II. Studies which provide reliable annotation (e.g. testing whether a gene or RNA structure has the predicted functional role) serve to allow evaluation of evolutionary predictive methods (e.g. XDecoder). Others measure short-scale evolutionary patterns, often on the scale of hours (e.g. a single viral replication cycle) or days (e.g. multiple passages in cell culture). Such studies provide a complementary viewpoint to evolutionary modeling on longer time scales. Evolution occurs via two principal components - the mechanism for change and the selective pressures determining whether this change will rise to fixation in the population. Short-scale studies primarily provide information on the former, and together with retrospective phylogenetic studies we can tease apart the relative contribution of the latter. For instance, it is known that transitions (mutations among pyrimidines and purines) are more common than transversions (mutations between these groups), and that this can be explained by a combination of biochemical (transversions are a "smaller" change chemically, and so more likely to happen in the course of genome replication) and selective effects (transversions change the encoded amino acid more often, which is more likely to be deleterious). This appendix describes three connections to experiments in viruses, an especially attractive system for experimental evolutionary work due to their fast evolutionary rates, short generation times, and small, well-characterized genomes.

In Section A.1, we describe a fine-scale map of single-replication recombination rates in the coding region of *poliovirus*, which we compare to natural recombinants analyzed with recHMM. In contrast to residue substitution, the relative contributions of mechanistic and selective factors determining recombination patterns are poorly understood. This study, along with ongoing characterization of naturally-occurring recombinants, serves to further our understanding of the interplay between mechanistic and selective forces contributing to recombination patterns.

In Section A.2, we describe a whole-genome, single-base resolution RNA structure probing experiment which, combined with XDecoder predictions, was used to identify and verify a novel functional RNA secondary structure in *poliovirus.* The evolutionary predictions of XDecoder provide an ideal complement to the chemically-based probing experiments, with the former providing a functional perspective and the latter a structural one. The novel structure was selected for having strong signal in both forms of data.

In Section A.3 we describe the use of `handalign` and ProtPal to reconstruct an ancestral form of *Adeno-associated virus* (AAV) for applications to directed evolution and gene therapy. Though directed evolution of extant AAV has been successful, it is thought that ancestral forms possessed even greater evolvability. By reconstructing and synthesizing specific ancestors in AAV's history, we hope to generate new forms with desirable properties, as well as highlight reconstruction as a powerful technique in synthetic biology and directed evolution.

A.1 Recombination map via high-throughput sequencing in *poliovirus*

The following section contains unpublished work in progress conducted jointly with Joseph DeRisi, Charles Runckel, and Raul Andino.

Part II Chapter 7 focused on detecting recombination on long time scales (e.g. decades) using phylogenetic data. Often the precise recombination breakpoint is not known, either because there are too few substutions near the breakpoint, or there are too many and the phylogenetic relationships have been obscured. This, combined with the (relatively) few recombinants which have been isolated from the field, makes determining genomic breakpoint distributions quite difficult. Without reliable estimates of breakpoint distributions, it is nearly impossible to begin to determine the genomic features which affect local variations in recombination rate.

Recent advances in gene synthesis provide opportunities to investigate these questions in a controlled laboratory setting. Using a synthetic form of *poliovirus* "tagged" with SNPs every 18 bases, co-infecting with wild-type (WT) and sequencing the progeny after one replication cycle allows detection of recombination simply by looking for sequence reads containing both WT and tagged sequences. The location of recombination can then be pinpointed within a window of 18 bases. By counting recombinant and non-recombinant reads covering each window and normalizing by the size (typically 18, though some deviation from this was necessary to preserve the amino acid sequence), the number of recombinations per base and replication cycle can be computed for each region, resulting in the map in Figure Figure A.1. Millions of reads were recovered, allowing for sufficient coverage to identify approximately 57,000 recombination breakpoints which in turn allowed for high confidence in estimating local recombination rates.

This experiment allows us to see which recombinant forms are produced in a single selection-independent replication cycle; that is, the extent to which breakpoints in a particular region are mechanistically favored to occur. This is entirely different than phylogenetic investigation of recombination (e.g. by recHMM), where the forms which occur in nature may be extremely unlikely from a mechanistic perspective, but have been selected for by other pressures. What is striking is that there are notable similarities between this experimentally-determined recombination map and the few naturally-occurring *poliovirus* recombinants that have been isolated. Figure A.1 shows local recombination rates for each of the 18-base tiles, with the most prominent spike in rate occurring near the RNAse-L element, a large RNA structure in the 3' end of the polio genome. Initial analysis of circulating vaccine-derived *poliovirus* forms shows that several circulating recombinants have a breakpoint near the RNAse-L element (Figure A.2). While these isolates are thought to be epidemiologically independent (e.g. each resulting from independent infection with the Sabin vaccine strain), additional analysis is needed to confidently determine whether these recombination events are shared or separate.

This correspondence between the selection-free map and naturally-occurring forms suggests that recombination is strongly influenced by mechanistic, rather than selective forces. However, significantly more data, especially naturally-occurring forms, is needed to verify this apparent pattern. Focusing on the laboratory-generated recombinants, we used a probabilistic model for the recombination process to determine the sequence features responsible for modulating local recombination rates. The model we chose was a discrete-time, discrete-state partially hidden Markov model. Changes in state (WT or tagged sequence) could occur at any genomic position, but this state was only observable at SNP positions, which were distributed at roughly every 18^{th} base throughout the coding region. Using an EM algorithm, we estimated maximum likelihood parameters for a variety of models incorporating homopolymers, GC content, known RNA structure, SHAPE reactivity quantiles (e.g. high, medium, and low SHAPE reactivity), and various combinations of these features. Each model has a collection of features that classifies each genome position. For instance, the **Uniform** model states that positions with a mismatch between tagged and WT forms have one recombination probability, and the remaining positions (between tags) have another, for a total of two parameter combinations. Note that for models with many features, not all will be used since not all combinations are present (or in some cases even possible - for instance one cannot have sequence identity on a mismatch position). The models' features are listed below.

testModel: Randomly-selected positions in the genome chosen to have a particular null "feature", used to verify that random model building did not result in better model fit.

Uniform: Positions on a mismatch (e.g. SNP tag) given a separate rate.

3'/5' Sequence ID: Mismatch within 5 bases 3' (/5') of the recombination point, mismatches.

3'/5' Homopolymer: Homopolymer (length 3 or more) contained in 5 bases 3' (/5') of recombination point, and mismatch positions.

3'/5' GC rich-only: Positions with 100% GC content 5' or 3' of the recombination point, as well as mismatches.

3'/5' GC rich+identity: Positions with 100% GC content 5' or 3' of the recombination point, as well as mismatches, identity, and homopolymers.

RNA structure predictions: Positions with XDecoder posterior pairing probability above 0.75, 3' homopolymers, 3' sequence identity, and mismatch.

Known RNA structures: Positions near a known structured element.

SHAPE 10-category: 10 quantiles of SHAPE reactivity.

SHAPE 2-category: 2 quantiles of SHAPE reactivity ("low" and "high").

Using the Bayesian Information Criterion for model comparison [169]), the relative fit of these models was determined (Figure A.4). Duplicates of certain models (e.g. 3' Homopolymer) were used to probe the sensitity of the parameter estimation method. Across multiple datasets and replicates, GC content and RNA structure repeatedly emerged as strong factors raising the local recombination rate. A synthetic construct with increased GC content (40 synonymous SNPs, 12% increase in GC content) was constructed to test this hypothesis, and had a 7.4-fold increase in recombination rate across the modified region. In our parameter estimates, shown in Figure A.3, GC content was predicted to raise the recombination rate by a factor of 4.68.

Increased sequencing surveillance in regions where recombinant forms are causing disease, accurate high-throughput computational methods, and novel laboratory techniques provide a promising outlook for advancing our knowledge of this potent evolutionary force. Especially appealing is the sort of hybrid computational/experimental inquiry which has been initiated with this project: large datasets can be generated by sophisticated biochemical means, and later analyzed with mathematical tools, producing predictions which can then be tested back in the laboratory.

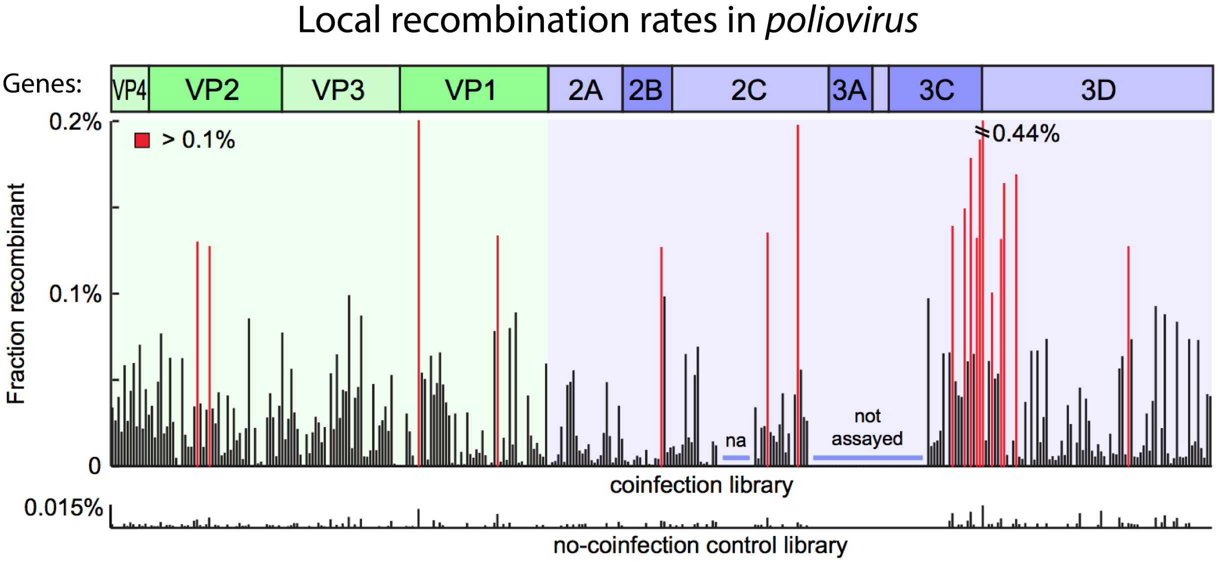


Figure A.1: By counting sequence reads with recombinant and non-recombinant tags, the recombination fraction on each region between tags can be estimated across the *poliovirus* genome.

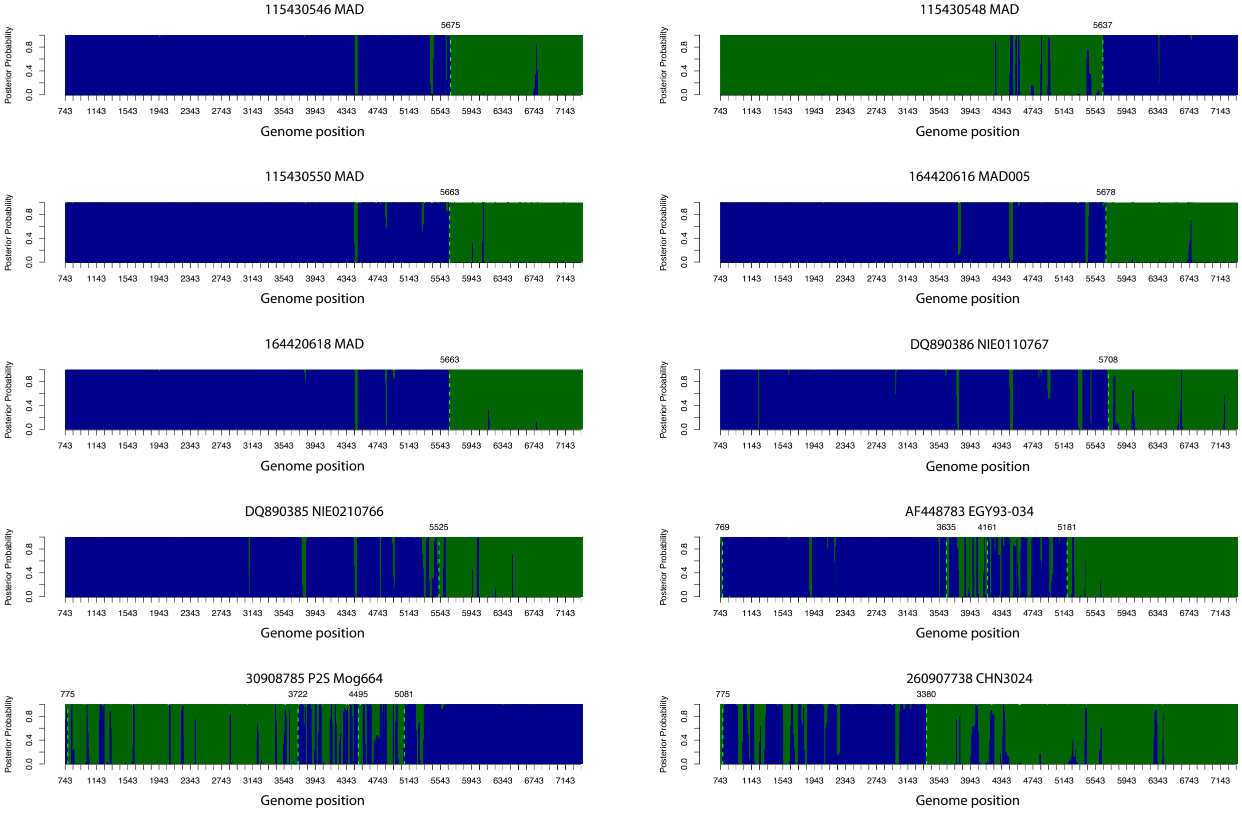


Figure A.2: Circulating vaccine-derived *poliovirus* forms analyzed with recHMM. Labeled green dashed lines indicate predicted breakpoints, and accession numbers and strain names (where available) appear above the plots. Many breakpoints lie near the RNAse-L element (at 5748 in *poliovirus* 1), consistent with the "hotspot" seen in Figure A.1 in that region. More isolated recombinants, together with epidemiological data concerning their independence (e.g. resulting from separate innoculations with the Sabin vaccine strain) are necessary to further investigate this phenomenon.

A.2 Probing RNA secondary structure via SHAPE in *poliovirus*

The following section contains unpublished work in progress conducted jointly with Cecily Burrill and Raul Andino.

The functional roles of RNA are gradually being uncovered. No longer thought of as a passive intermediate between DNA and proteins, it is especially important in RNA viruses, whose genomes exist either in single- or double-stranded RNA form. In particular, the secondary structure (e.g. stems, loops, etc) is known to play major roles in the viral repli-

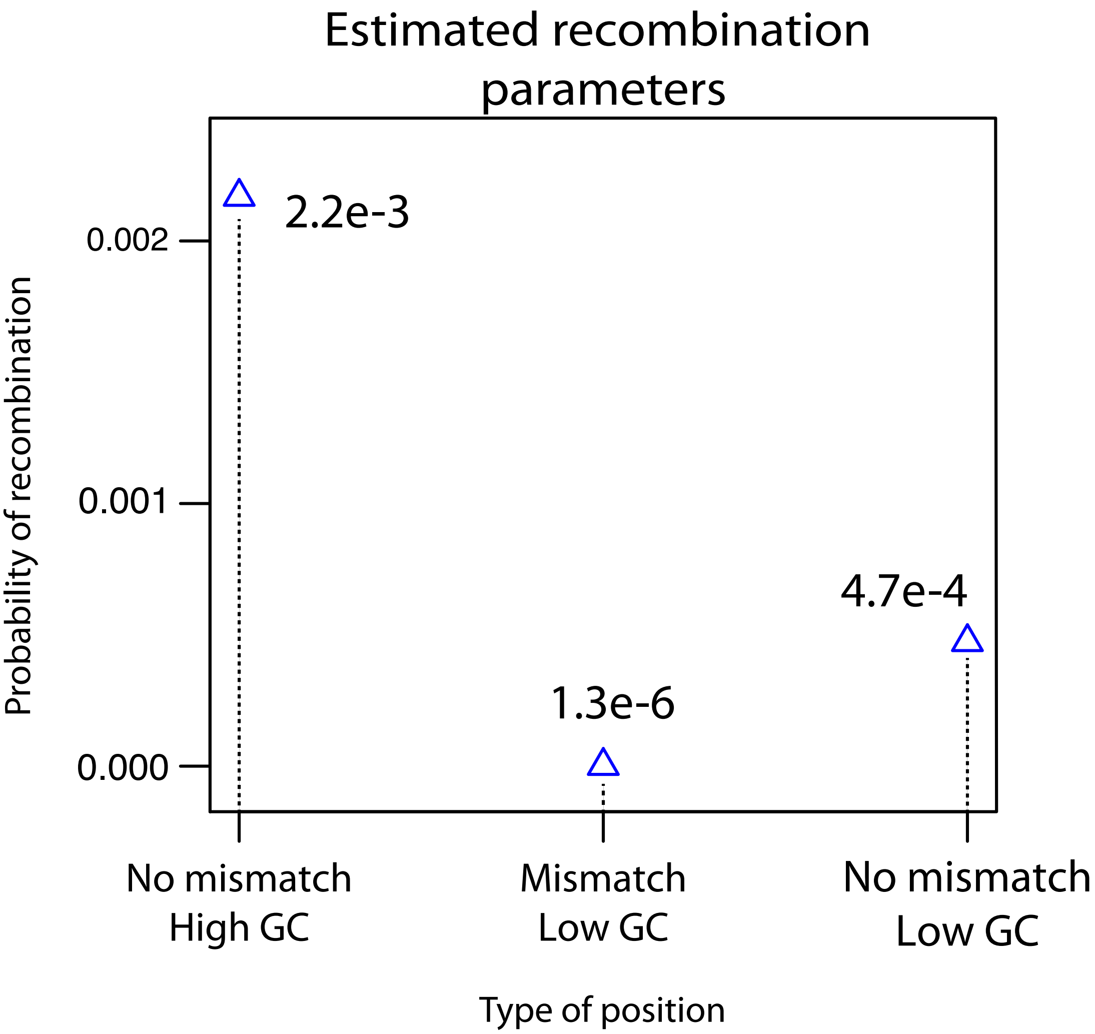


Figure A.3: Using probabilistic models for the recombination and read generation process, parameters based on sequence features were estimated using the EM algorithm. High GC content away from SNP tags was predicted to increase the recombination rate by a factor of 4.68 (from 4.7×10^{-4} to 2.2×10^{-3}. Recombination near mismatches was predicted as much less probable, consistent with hypothesized mechanisms for viral recombination.

cation cycle, including genome replication [170–173], initiation of translation [174] packaging [175, 176], and immune evasion [177].

In Part II Chapter 6 we presented a grammar for predicting RNA secondary structure overlapping protein-coding sequence on the basis of nucleotide substitution histories. We summarized this as pairing probabilities - the sum over all posterior probabilities corresponding to possible pairings involvinga given column. This quantity, the tendency of a base to bind to another part of the genome, can be measured by complementary laboratory techniques using modification chemistry. Selective 2'-hydroxyl acylation analyzed by primer extension (SHAPE) preferentially modifies the backbone of RNA in flexible (e.g. loop) regions. By measuring the relative frequency of this modification and normalizing via controls, a single-base resolution measure of reactivity (or tendency to not form a pair) can be computed for each position across the genome [178]. Similar to obtaining a raw DNA sequence, the positional reactivity measurements (or even the complete RNA structure itself) says little about the sequence's functional role. These must currently be verified by labor-intensive experimental techniques. Thus, XDecoder's predictions provide a complementary lens: while not as biochemically meaningful (indeed, the consensus structure predicted by XDecoder often contains many non-canonical base pairs), the pairing probabilities, based on evolutionary conservation, are likely to indicate functional roles. Combining these two sources of data allowed us to identify a probable novel RNA structure in *poliovirus* and verify its functionality using traditional mutagenesis/cell culture techniques.

Figure Figure A.5 shows SHAPE reactivity for each position in the genome alongside XDecoder's pairing probabilities. Figure A.6 shows the two forms of data around the CRE. Pairing probability here is high, but surprisingly SHAPE reactivity appears no lower than normal. Figure A.7 shows the two forms of data in the RNAse-L region where each is highly confident and the correspondence is very close. The novel structure identified is in the region 6700-7100 of *poliovirus 1*, shown in Figure A.8. Both SHAPE and XDecoder had strong predictions in this region (low SHAPE reactivity and high pairing probability). After making synonymous mutations via standard mutagenesis techniques, the growth rate and plaque size of the resulting virus was markedly reduced, leading us to believe that this region's structure plays an important, though as yet unknown function in the viral life cycle.

A.3 Reconstructing an ancestral *Adeno-associated virus* sequence for synthesis

The following section contains unpublished work in progress conducted jointly with David Schaffer and John Weinstein.

Though many viruses cause disease, there are others which are either benign or even useful to humans. Often those viruses which have co-evolved with their hosts exclusively for long periods are able to infect and replicate without causing disease symptoms. Those which have recently emerged (or re-emerged) due to cross-species transitions often have the most severe symptoms, as in the case of *ebolavirus*, HIV, or influenza.

One virus which has had significant success in use as a gene therapy vector is *Adeno-associated virus* (AAV). Gene therapy is the use of DNA as a therapeutic agent, often with the goal of expressing a particular gene within a host [179]. Cancer [180], retinal disease [181], colorblindness [182], and even HIV [183] have been treated with gene therapy. AAV, so named because it requires the presence of a helper virus (typically*Adenovirus*), causes no detectable disease phenotype and is robust to insertion of entire genes into its genome [184]. These genes are then expressed in whichever cells the virus infects, which can be of great use in treating diseases which are manifestations of malfunctioning genes in the patient. The different strains of AAV possess different properties and tissue specificities. For instance, AAV8 and AAV9 are able to infect the liver, whereas only AAV5 is able to infect photoreceptor cells [185]. Directed evolution uses gene shuffling and mutagenesis is used to generate a diverse library around a starting sequence, which is then subject to selection (e.g. particular tissues) to generate new variants with the desired properties [186]. This approach has been successful in creating novel AAV possessing properties useful for gene therapy [187–189]. However, extant forms may occupy local optima of the fitness landscape, hindering the scale of possible evolutionary steps.

In this project, we endeavored to predict and synthesize the AAV sequence ancestral to those strains with desireable properties, with the rationale that the progenitor sequence, may possess heightened "evolvability" since it gave rise to the myriad extant forms. Given proper evolutionary pressures, it is conceivable that the observed strains, as well as other forms with additional properties, could be derived through directed evolution of this ancestral form.

We utilized `handalign` to build an ancestral MSA and tree relating the relevant AAV strains. By superimposing the known 3D structure of the gene of interest (*cap*), we were able to examine the structural context of regions of alignment and/or reconstruction uncertainty. There were several areas where the alignment was uncertain, often corresponding with unstructured loop regions on the capsid exterior. While accurate reconstruction of such regions is very difficult, the structural leniency and fast evolutionary behavior led us to believe that small reconstruction errors in these regions would be unlikely to render our synthesized form unviable. Figure A.9 shows the MSA visualized along with the 3D structure, with both colored according to the percent identity of the corresponding alignment column.

Reconstruction uncertainty (conditional on an alignment) was also present at several regions of the predicted sequence. Even if a "perfect" reconstruction of the indel history is available, reconstructing individual characters may not be possible with 100% confidence when multiple substitutions have occurred at a given position, or substitutions occur simultaneously on sister branches. Fortunately, we were able to tailor the synthesis procedure to mirror this uncertainty. Using specialized synthesis techniques, the ancestral library can be designed to have different amino acids represented at particular positions proportional to their reconstruction probability. Figure A.10 shows the reconstruction probability of individual amino acids, with the height of each character proportional to its probability. Figure A.11 shows the reconstruction probability of individual codons: the vertical space occupied by each codon is proportional to its probability. The amino acid coded for is indicated using three-letter abbreviations, and the nucleotides are indicated with colors green, red, orange, and blue (legend at top right).

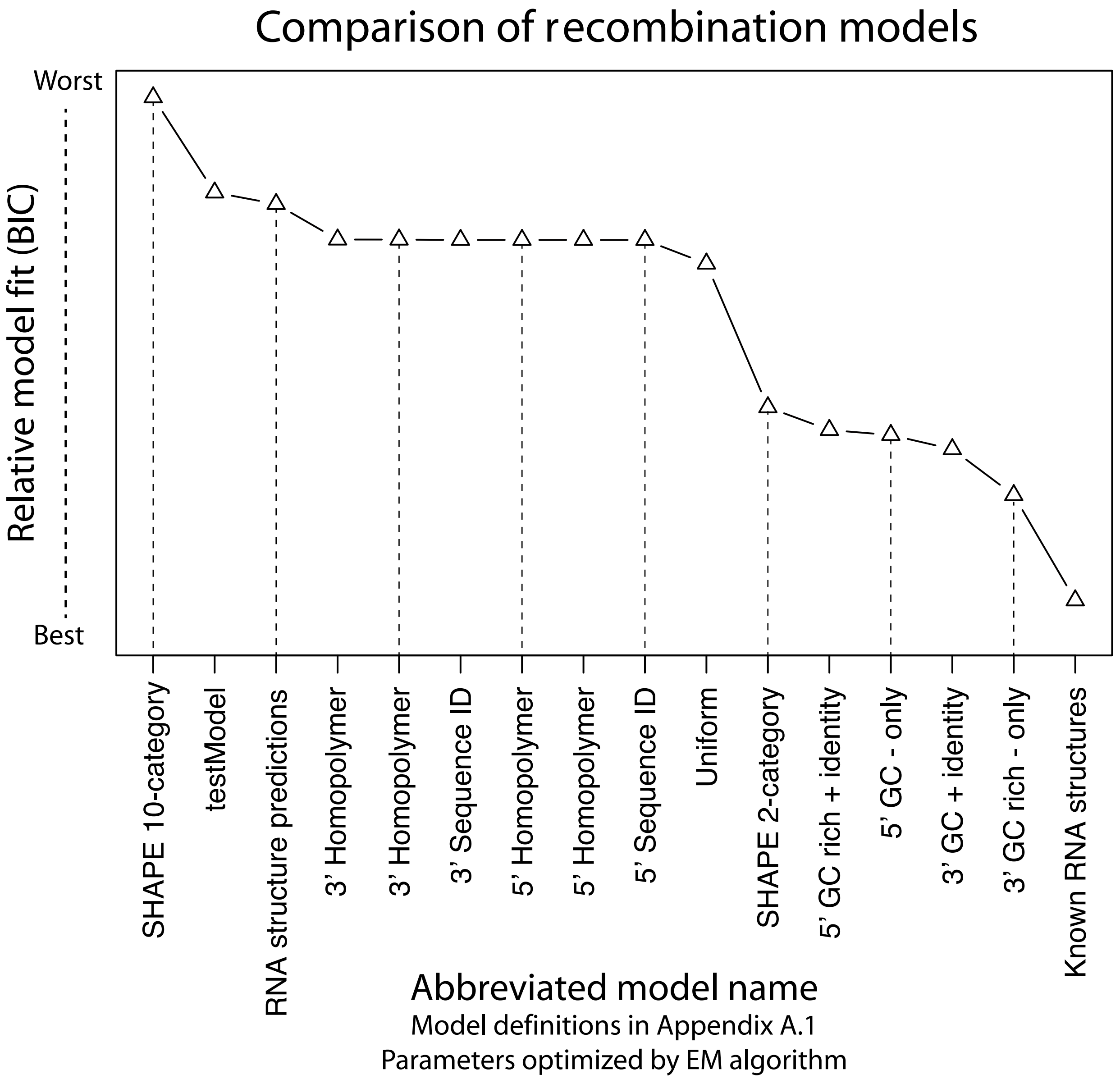


Figure A.4: In using Bayesian model comparison statistics for relative model fit evaluation, GC and RNA structure-aware models had the best fit. Homopolymers, previously thought to contribute to recombination via promoting polymerase stalling events, appear to have a negligible effect according to our modeling. Interestingly, SHAPE-based models (data taken from study described in Appendix A.1) did better than homopolymers but not as well as only using the known structures. Using parameters for each of 10 quantiles of SHAPE reactivity did worse than only having 2 categories, since the BIC penalizes additional parameters which don't significantly improve model fit.

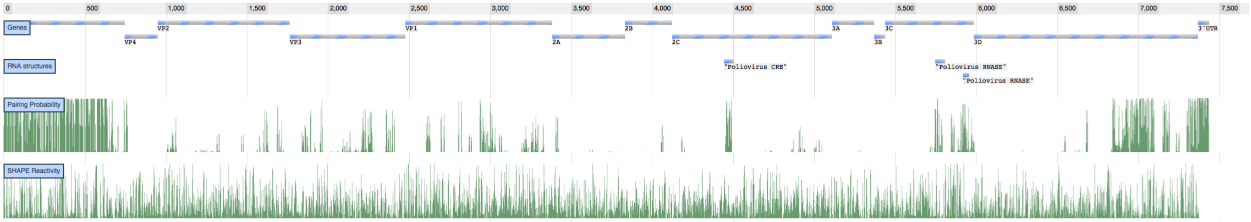


Figure A.5: Pairing probability and SHAPE measurements genome-wide in *poliovirus*. SHAPE reactivities are quite noisy, whereas only a few regions are predicted by XDecoder to harbor conserved structures. The three known structural regions - the 5' UTR, CRE element, and RNAse-L element, are all predicted with high pairing probability. A novel structure was proposed in the region 6700-7100.

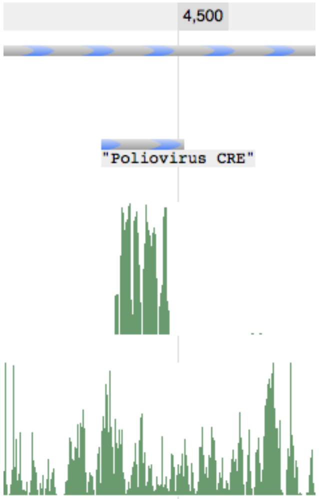


Figure A.6: The *Cis*-acting replication element (CRE) in *poliovirus* is a well-known structured region. This is predicted with high probability by XDecoder, lending confidence to our predicted structure in the 3' end of the 3D gene. Surprisingly, SHAPE reactivity in the CRE region does not appear to be lower than average, suggesting that perhaps the CRE does not fold into its hairpin structure during all parts of the replication cycle.

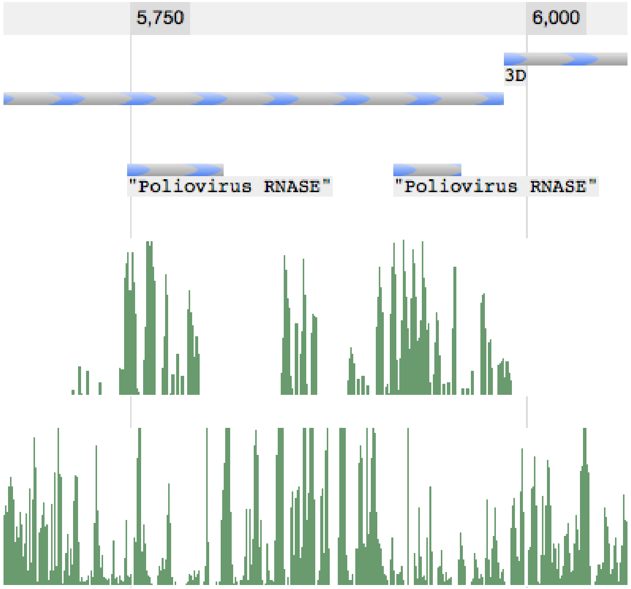


Figure A.7: The RNAse-L structure element in *poliovirus* is predicted by XDecoder with high probability, and lower than average SHAPE reactivity is also observed in this region.

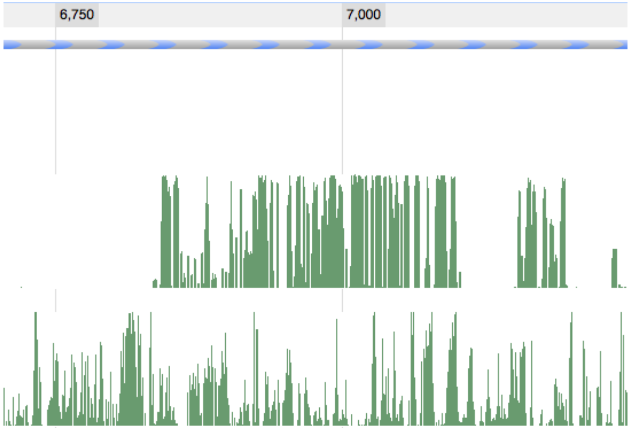
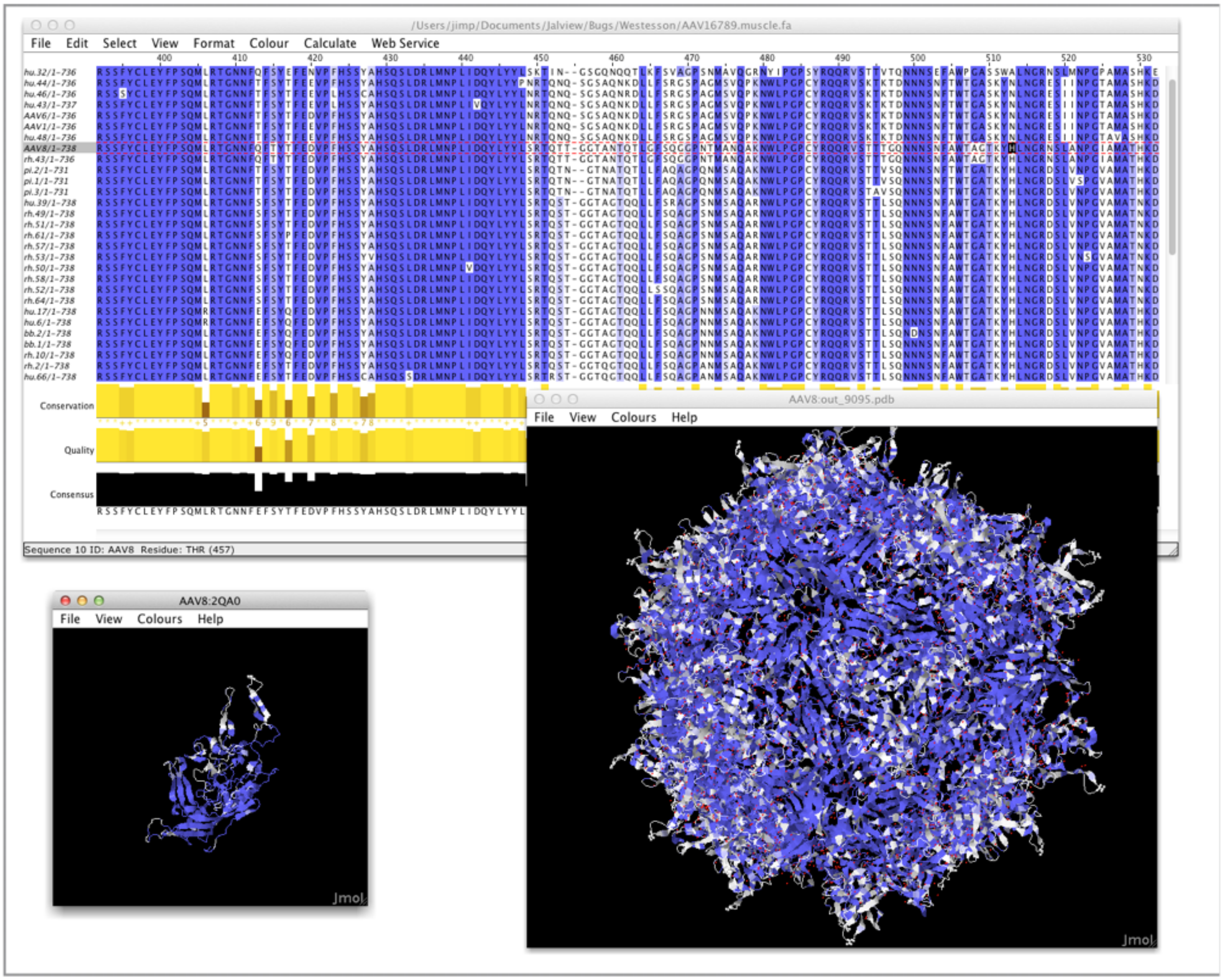


Figure A.8: The novel predicted structure possesses both high pairing probability and lower than average SHAPE reactivity, leading us to believe that this is a thermodynamically stable and biologically functional structured element.



```
hu.32/1-736   RSSFYCLEYFPSQMLRTGNNFQFSYEFENVPFHSSYAHSQSLDRLMNPLIDQYLYYLSKTIN--GSGQNQQTLKFSVAGPSNMAVQGRNYIPGPSYRQQRVSTTVTQNNNSEFAWPGASSWALNGRNSLMNPGPAMASHKE
hu.44/1-736   RSSFYCLEYFPSQMLRTGNNFTFSYTFEEVPFHSSYAHSQSLDRLMNPLIDQYLYYPNRTQNQ-SGSAQNKDLLFSRGSPAGMSVQPKNWLPGPCYRQQRVSKTKTDNNNSNFTWTGASKYNLNGRESIINPGTAMASHKD
hu.46/1-736   RSSSYCLEYFPSQMLRTGNNFTFSYTFEEVPLHSSCAHSQSLDRLMNPLIDQYLYYLNRTQNQ-SGSAQNRDLLFSRGSPAGMSVQPKNWLPGPCYRQQRVSKTKTDNNNSNFTWTGASKYNLNGRESIINPGTAMASHKD
hu.43/1-737   RSSFYCLEYFPSQMLRTGNNFTFSYTFEEVPLHSSYAHSQSLDRLMNPLIVQYLYYLNRTQNQ-SGSAQNKDLLFSRGSPAGMSVQPKNWLPGPCYRQQRVSKTKTDNNNSNFTWTGASKYNLNGRESIINPGTAMASHKD
AAV6/1-736    RSSFYCLEYFPSQMLRTGNNFTFSYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLNRTQNQ-SGSAQNKDLLFSRGSPAGMSVQPKNWLPGPCYRQQRVSKTKTDNNNSNFTWTGASKYNLNGRESIINPGTAMASHKD
AAV1/1-736    RSSFYCLEYFPSQMLRTGNNFTFSYTFEEVPFHSSYAHSQSLDRLMNPLIDQYLYYLNRTQNQ-SGSAQNKDLLFSRGSPAGMSVQPKNWLPGPCYRQQRVSKTKTDNNNSNFTWTGASKYNLNGRESIINPGTAMASHKD
hu.48/1-736   RSSFYCLEYFPSQMLRTGNNFTFSYTFEEVPFHSSYAHSQSLDRLMNPLIDQYLYYLNRTQNQ-SGSAQNKDLLFSRGSPAGMSVQPKNWLPGPCYRQQRVSKTKTDNNNSNFTWTGASKYNLNGRESIINPGTAVASHKD
AAV8/1-738    RSSFYCLEYFPSQMLRTGNNFQFTYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQTT-GGTANTQTLGFSQGGPNTMANQAKNWLPGPCYRQQRVSTTTGQNNNSNFAWTAGTKYHLNGRNSLANPGIAMATHKD
rh.43/1-736   RSSFYCLEYFPSQMLRTGNNFQFTYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQTT-GGTANTQTLGFSQGGPNTMANQAKNWLPGPCYRQQRVSTTTGQNNNSNFAWTAGTKYHLNGRNSLANPGIAMATHKD
pi.2/1-731    RSSFYCLEYFPSQMLRTGNNFTFSYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQTN--GTNATQTLLFAQAGPQNMSAQAKNWLPGPCYRQQRVSTTVSQNNNSNFTWTGATKYHLNGRDSLVNPGVAMATHKD
pi.1/1-731    RSSFYCLEYFPSQMLRTGNNFTFSYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQTN--GTNATQTLLFAQAGPQNMSAQAKNWLPGPCYRQQRVSTTVSQNNNSNFTWTGATKYHLNGRDSLVSPGVAMATHKD
pi.3/1-731    RSSFYCLEYFPSQMLRTGNNFTFSYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQTN--GTNATQTLLFAQAGPQNMSAQAKNWLPGPCYRQQRVSTAVSQNNNSNFTWTGATKYHLNGRDSLVNPGVAMATHKD
hu.39/1-738   RSSFYCLEYFPSQMLRTGNNFSFSYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLFSRAGPSNMSAQARNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATNKD
rh.49/1-738   RSSFYCLEYFPSQMLRTGNNFSFSYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLFSQAGPSNMSAQARNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATNKD
rh.51/1-738   RSSFYCLEYFPSQMLRTGNNFSFSYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLFSQAGPSNMSAQARNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATNKD
rh.61/1-738   RSSFYCLEYFPSQMLRTGNNFSFSYPFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLFSQAGPSNMSAQARNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATNKD
rh.57/1-738   RSSFYCLEYFPSQMLRTGNNFSFSYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLFSQAGPSNMSAQARNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATNKD
rh.53/1-738   RSSFYCLEYFPSQMLRTGNNFSFSYTFEDVPFHSSYVHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLFSQAGPSNMSAQARNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNSGVAMATNKD
rh.50/1-738   RSSFYCLEYFPSQMLRTGNNFSFSYTFEDVPFHSSYAHSQSLDRLMNPLVDQYLYYLSRTQST-GGTAGTQQLLFSQAGPSNMSAQARNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATNKD
rh.58/1-738   RSSFYCLEYFPSQMLRTGNNFSFSYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLFSQAGPSNMSAQARNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATNKD
rh.52/1-738   RSSFYCLEYFPSQMLRTGNNFSFSYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLSSQAGPSNMSAQARNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATNKD
rh.64/1-738   RSSFYCLEYFPSQMLRTGNNFSFSYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLFSQAGPSNMSAQARNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATNKD
hu.17/1-738   RSSFYCLEYFPSQMRRTGNNFEFSYQFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLFSQAGPNNMSAQAKNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATHKD
hu.6/1-738    RSSFYCLEYFPSQMRRTGNNFEFSYQFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLFSQAGPNNMSAQAKNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATHKD
bb.2/1-738    RSSFYCLEYFPSQMLRTGNNFEFSYQFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLFSQAGPNNMSAQAKNWLPGPCYRQQRVSTTLSQNDNSNFAWTGATKYHLNGRDSLVNPGVAMATHKD
bb.1/1-738    RSSFYCLEYFPSQMLRTGNNFEFSYQFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLFSQAGPNNMSAQAKNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATHKD
rh.10/1-738   RSSFYCLEYFPSQMLRTGNNFEFSYQFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTAGTQQLLFSQAGPNNMSAQAKNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATHKD
rh.2/1-738    RSSFYCLEYFPSQMLRTGNNFEFSYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYLSRTQST-GGTQGTQQLLFSQAGPANMSAQAKNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATHKD
hu.66/1-738   RSSFYCLEYFPSQMLRTGNNFEFSYTFEDVPFHSSCAHSQSSDRLMNPLIDQYLYYLSRTRST-GGTQGTQQLLFSQAGPANMSAQAKNWLPGPCYRQQRVSTTLSQNNNSNFAWTGATKYHLNGRDSLVNPGVAMATHKD

Consensus     RSSFYCLEYFPSQMLRTGNNFEFSYTFEDVPFHSSYAHSQSLDRLMNPLIDQYLYYL

Sequence 10 ID: AAV8  Residue: THR (457)
```

Figure A.9: Visualizing alignment and structures in JalView . At top, an alignment of AAV variants colored according to percent identity. The 3D structures of the CAP protein (left) and its arrangement into a viral capsid (right) are correspondingly colored. The JalView program allows for two-way interactive investigation: mousing over a residue in the AAV8 sequence highlights this residue in both structures, and clicking a residue in either structure highlights the corresponding alignment column. Position 513 is currently highlighted - the AAV8 residue is black in the alignment, and the molecule is shown in both structures on the largest protruding loop.

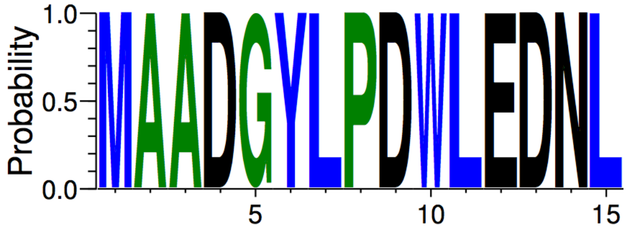


Figure A.10: A distribution of predicted ancestral amino acid sequences; each character's height is proportional to its probability.

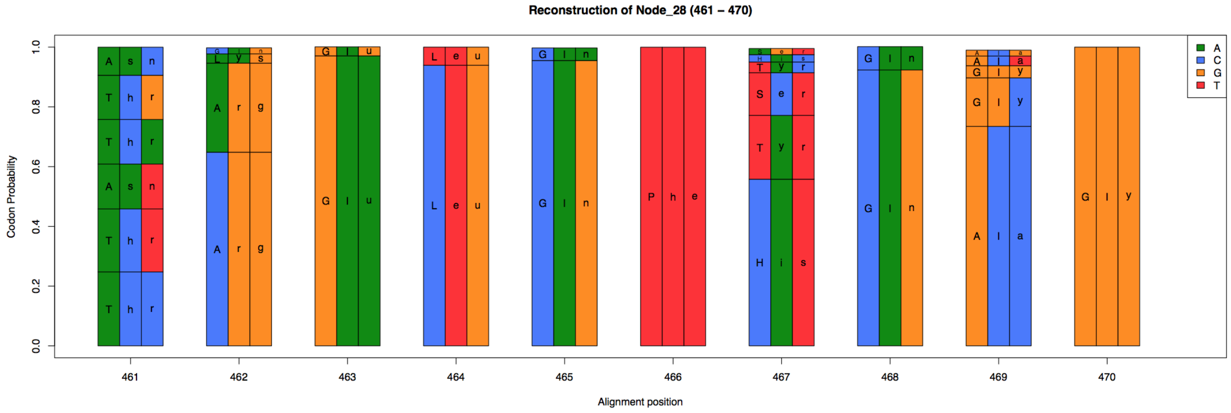


Figure A.11: A distribution of predicted ancestral DNA sequences (designated by colors green (A), blue (C), orange (G), and red (T)), with amino acids displayed using 3-letter abbreviations. The vertical space occupied by each codon is proportional to its probability. Some columns (e.g. 461) display high synonymous variability, while in others non-synonymous changes are common (e.g. 467).

Appendix B

ProtPal simulation study and OPTIC data analysis

Methods

Simulation parameters and setup

Data generation Our simulation study is comprised of alignments simulated using 5 different indel rates (0.005, 0.01, 0.02, 0.04, and 0.08 indels per unit time), each with 3 different substitution rates (0.5, 1, and 2 expected substitutions per unit time) and 100 replicates. Time is defined such that a sequence evolving for time t with substitution rate r is expected to accumulate rt subsitutions per site. We employed an independent third-party simulation program, indel-seq-gen, specifically designed to generate realistic protein evolutionary histories [77]. indel-seq-gen is capable of modeling an empirically-fitted indel length distribution, rate variation among sites, and a neighbor-aware distribution over inserted sequences allowing for small local duplications. Since the indel and substitution model used by indel-seq-gen are separate from (and richer than) those used by ProtPal, ProtPal has no unfair advantage in this test.

indel-seq-gen v2.0.6 was run with the following command:

```
cat guidetree.tree| indel-seq-gen -m JTT -u xia --num_gamma_cats 3 -a 0.372 --branch_scal
r/b --outfile simulated_alignment.fa --quiet --outfile_format f -s 10000 --write_anc
```

The above command uses the "JTT" substition model, the "xia" indel fill model (based on neighbor effects, estimated from E coli k-12 proteins [77]), and 3 gamma-distributed rate categories with shape 0.372. Branch lengths are scaled by the substitution rate for simulation rate r, normalized by the inverse of indel-seq-gen's underlying substitution rate ($b = 1.2$) so as to adhere to the above definition of evolutionary "time". Similarly, indel rates, which are set in the guide tree file `guidetree.tree`, are scaled by $\frac{b}{r}$ so that $t\lambda^*$ insertions/deletions are expected over time t for rate λ^*.

The root mean squared error (RMSE) for each error distribution was computed as follows:

$$RMSE = \sqrt{\sum_{replicates} (\frac{\hat{\lambda}^*_{\hat{H}}}{\lambda^*} - 1)^2} \tag{B.1}$$

The true tree was made available to all programs which can utilize a tree (ProtPal, PRANK, MUSCLE), representing the use case in which the true tree is known (e.g. via the species tree) but the true alignment is unknown. We ran simulations on three different phylogenies: a tree of twelve sequenced *Drosophila* genomes [88] and trees from the mammalian and amniotic clades of the OPTIC database. We here report results for the *Drosophila* tree, which we empirically observe to show trends consistent with, but more pronounced than, those of the mammalian and amniotic trees. The clearer trends may be due to the *Drosophila* tree being larger than the other trees (12 taxa), or having a diverse range of branch lengths (0.001 - 0.59 expected substitutions/site, at the genome-wide average rate). The simulation data, reconstructions, and analysis scripts are available from `http://biowiki.org/~oscar/simulation_reconstruction.tar`.

Alignment We investigated several multiple alignment tools [5, 78–80, 83, 190] in combination with alignment-conditioned reconstruction methods. Programs were run with their default settings, with the exception of PRANK and MUSCLE. To specify ancestral inference, the guide tree, and "insertions opening forever", PRANK used the extra options "`-writeanc -t <treefile> +F`". PRANK's `-F` option allows insertions to match characters at alignments closer to the root. This can be a useful heuristic safeguard when an incorrect tree may produce errors in subtree alignments that cannot be corrected at internal nodes closer to the root. Since the true guide tree is provided to PRANK, it is safe to treat insertions in a strict phylogenetic manner via the `+F` option. For computational efficiency, ProtPal was provided with a CLUSTALW guide alignment. Any alignment of the sequences can be used as a guide, and we chose CLUSTALW for its general poor performance, so that ProtPal would gain no unfair advantage by the information contained in the guide alignment. MUSCLE was provided the guide tree with the additional option "`-usetree <treefile>`".

Muscle v3.6

```
MUSCLE -in unaligned.fa -out aligned.fa -usetree guidetree.tree
```

PRANK v.080820

```
PRANK -d=unaligned.fa -noxml -realbranches -writeanc -o=output_directory
-t=guidetree.tree +F
```

Clustal v2.03

```
clustalw -INFILE=unaligned.fa -OUTFILE=aligned.fa
```

ProbCons v1.12

```
probcons unaligned.fa > aligned.fa
```

FSA v1.08

```
fsa unaligned.fa > aligned.fa
```

MAFFT v6.818b

```
mafft unaligned > aligned.fa
```

Imputing indel histories The ancestral reconstruction programs ProtPal and PRANK were used to directly impute indel histories. The remaining tools were augmented to reconstruction tools by post-processing their MSAs using the maximum parsimony algorithm described in [89], with the ambiguous cases described therein (e.g. where a column of characters could be equally parsimoniously explained by a deletion on one child branch or an insertion on the other) resolved by a uniformly random choice from the possible solutions. Indel rates were estimated by counting indel events in MAP reconstructed histories:

$$\hat{\theta}_{\hat{H}} = \text{argmax}_{\theta'} P(\theta'|\hat{H}, S, T) = \text{argmax}_{\theta'} P(\hat{H}, S|T, \theta') \tag{B.2}$$

where the latter step assumes a flat prior, $P(\theta') = \text{const}$.

This statistic is not without its problems. For one thing, we use an initial guess of θ to estimate $\hat{H}$. Furthermore, for an unbiased estimate, we should sum over all histories, rather than conditioning on the MAP reconstructed history. This summing over histories would, however, require multiple expensive calculations of $P(S|T, \theta)$, where conditioning on $\hat{H}$ requires only one such calculation. We further justify our benchmark of parameter estimates conditioned on a MAP-reconstructed history by noting that this the *de facto* method employed by large-scale genomics studies focusing on indels [84–87].

As well as imputing indel rates from reconstructed histories, we also tried using the `lambda.pl` program in the DAWG package [82], which estimates indel rates from MSAs directly (without attempting reconstruction).

Estimating substitution rates Substitution rates were estimated for each inferred alignment using XRate's built-in EM algorithm and the following simple rate matrix. Given an equilibrium distribution over amino acid characters, with π_i defining the proportion of character i, the rate of character i mutating to j is set to $r\pi_j$ where r is the only free rate parameter. XRate's estimate of r is taken to be the average substitution rate of the MSA.

By using indel-seq-gen's branch-scale option and changing the indel rate parameters accordingly, we are able to modulate the substitution and indel rates independently in the data generation step. This true substitution rate and the rate inferred by XRate are then directly comparable.

Additional figures

In addition to estimating indel rates for all genes in the OPTIC set, we performed various other analyses which were left out of the main text for reasons of space limitations. We provide figures those displaying results here.

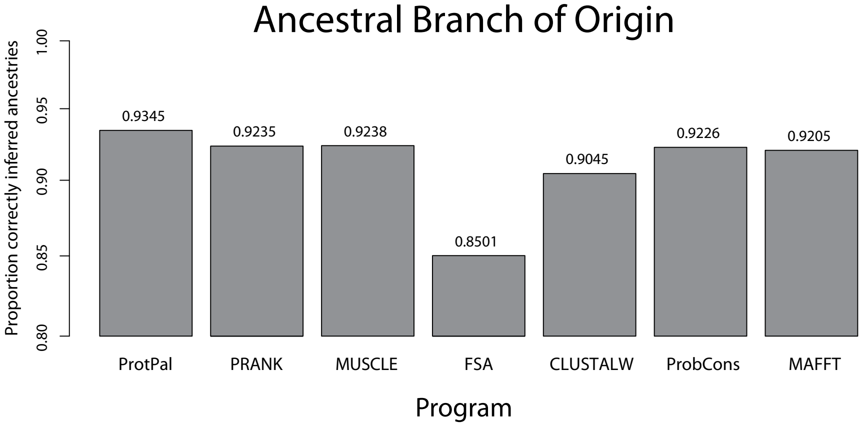


Figure B.1: ProtPal correctly reconstructs the age of more extant residues than any other program tested. The y-axis shows the proportion of extant residues whose point of origin on the phylogenetic tree was correctly pinpointed by the reconstruction. The branch of origin was found by taking the tree node closest to the root containing a non-gap reconstructed character. All programs except FSA are in the 92%-94% range, owing to the fact that many columns (especially at low indel rates) are devoid of indels, making inference of origin trivial (as these columns' origin is pre-root).

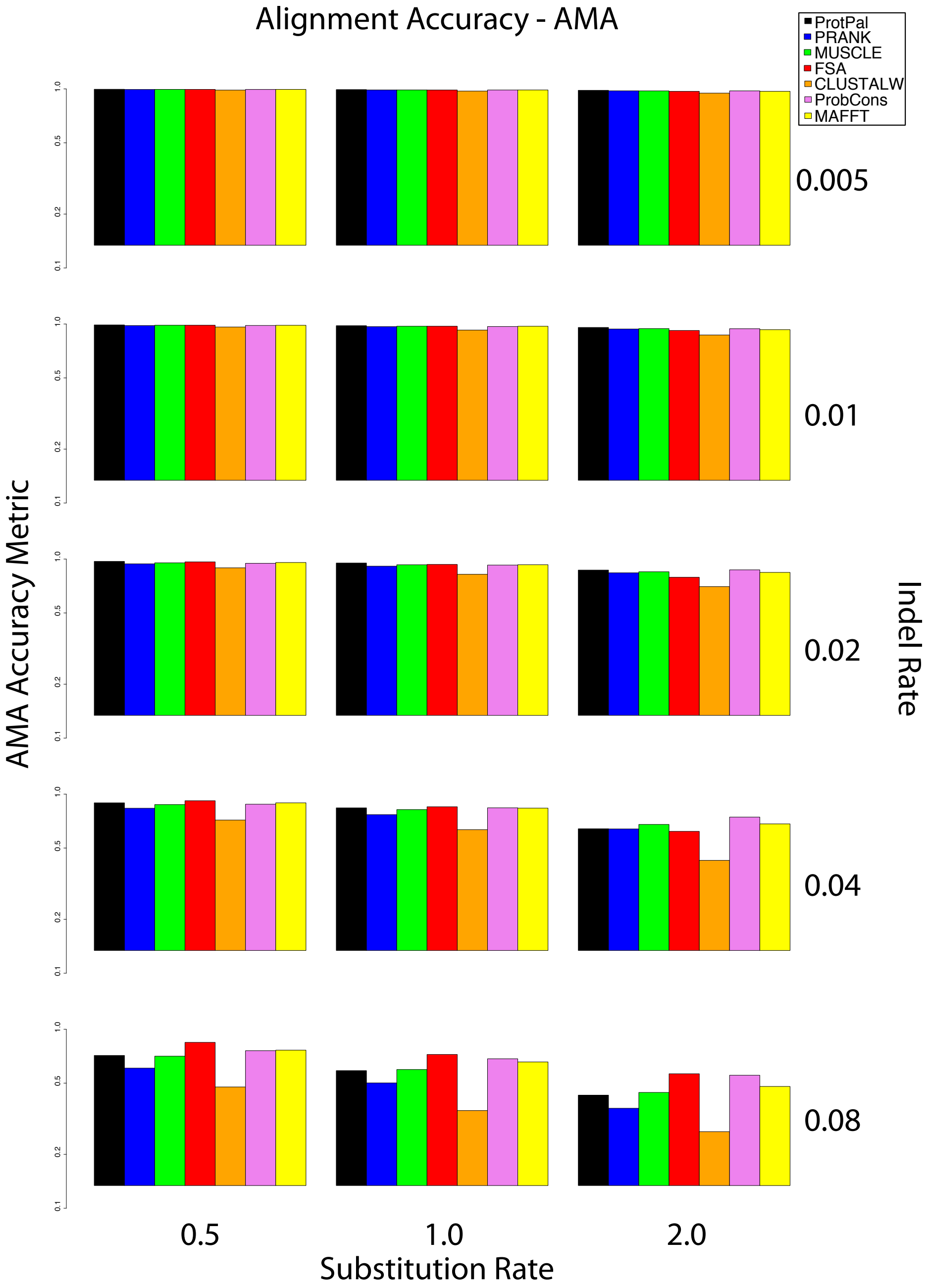


Figure B.2: Cross-comparison of AMA scores and rate estimation accuracy reveals that using a single metric to assess alignment accuracy can be unreliable. AMA scores were computed for each programs alignment of only leaf sequences using **cmpalign** from the DART package [18, 191]. AMA scores are comparable across programs until higher indel rates, where FSA performs best—contrasting with Figure 3.1 and Figure 3.2. MUSCLE's accurate deletion rate measurements at high rates and the low corresponding AMA scores suggest a "cancellation of biases".

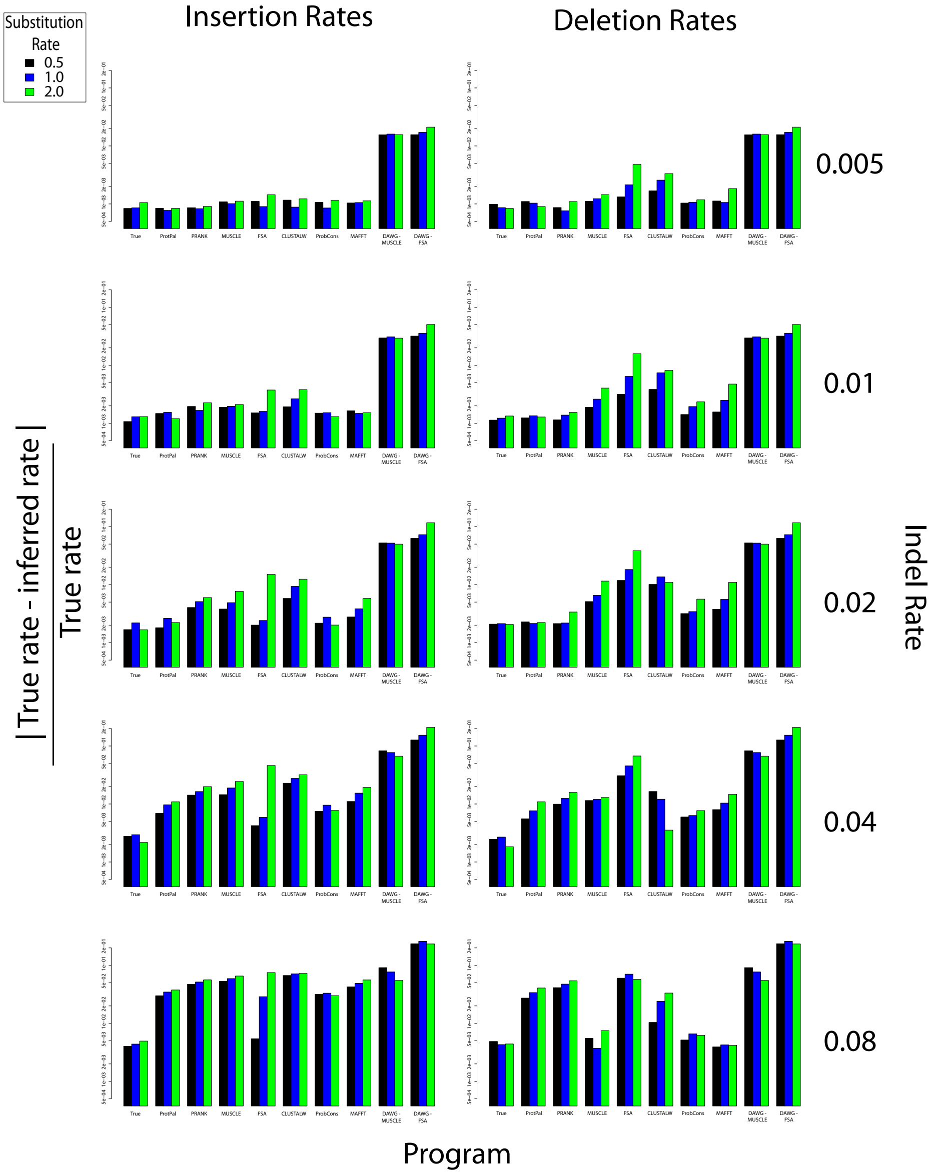


Figure B.3: Most programs are relatively robust to variations in the simulated substitution rate, as evidenced by the benchmark data grouped according to substitution rate. Accuracy of rate estimation is plotted as $|true-inferred|$ on the y-axis, with bars grouped by program for each indel rate and 3-tuple of substitution rates. Higher substitution rates often lead to higher error, presumably because they obscure homologies, making it more difficult to distinguish substitutions from indels. FSA appears more sensitive to increased substitutions than other programs - at indel rate 0.02, FSA's insertion rates are as accurate as ProtPal's at 0.5 and 1.0 substitutions per site, whereas at the highest substitution rate (2.0), its error exceeds that of CLUSTALW.

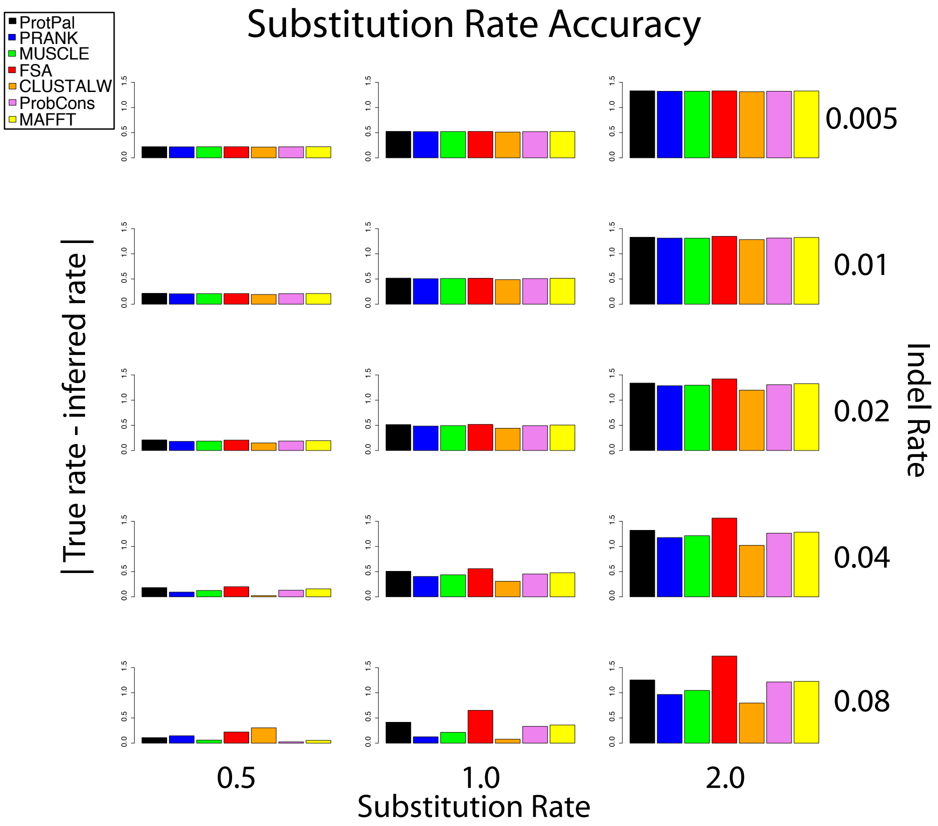


Figure B.4: Substitution rates estimated from multiple alignments display comparable accuracy across methods.

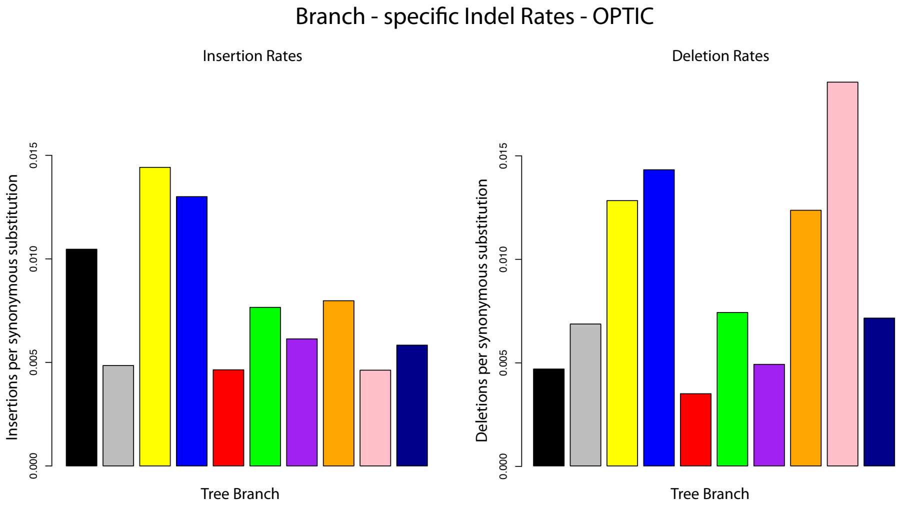


Figure B.5: Reconstruction allows for estimation of branch-specific indel rates, revealing possibly interesting signals of evolution. Indel rates were averaged over all alignments, using the species tree shown in Figure B.6. The human branch (*Euarchontoglires* - H.*sapiens*) appears to have experienced unusually many insertions. The *Amniota* - *Australophenids* (pink) branch has a higher deletion than insertion rate, though it is difficult to distinguish an insertion on this branch from a deletion on the *Amniota* - G.*gallus* (navy) branch. All other branches are comparable between insertions and deletions. Each bar is colored according to branches in Figure B.6.

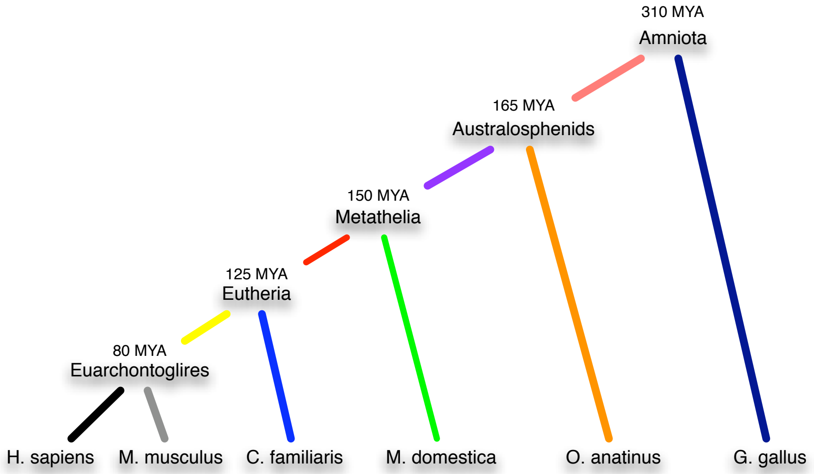


Figure B.6: The phylogenetic tree used for analysis of OPTIC data, colored to inform the branch-specific Figure B.5.

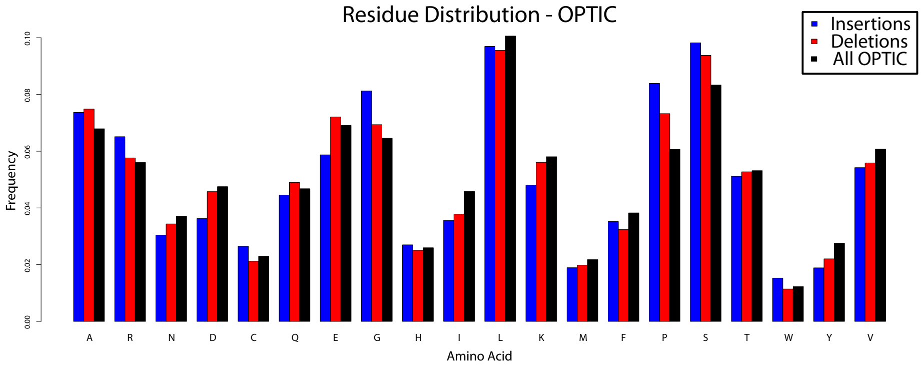


Figure B.7: Distributions over amino acids are highly non-uniform, and differ between insertions, deletions, and the overall distribution seen in OPTIC. Inserted, deleted, and all sequences were separately pooled across all OPTIC genes reconstructed and amino acid distributions were computed for each.

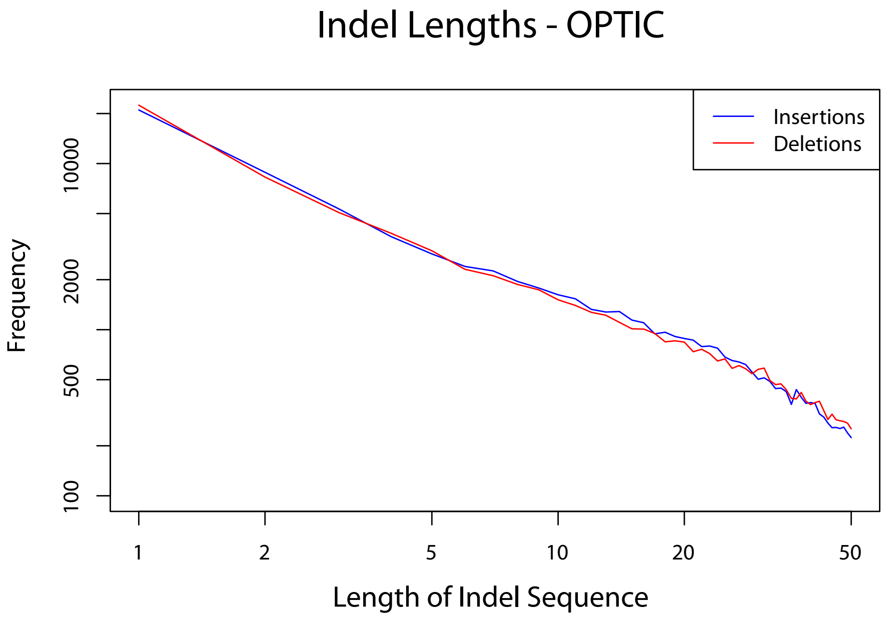


Figure B.8: Lengths of inserted and deleted sequences are similarly distributed, in contrast to the conclusions of previous studies, such as [93], which found that deletions were longer relative to insertions in C. *elegans* sequence data. While this may represent a genuine difference in the evolution of human and worm genomes, it is likely that the use of deletion-biased aligners (MUSCLE and CLUSTALW) affected their conclusions.

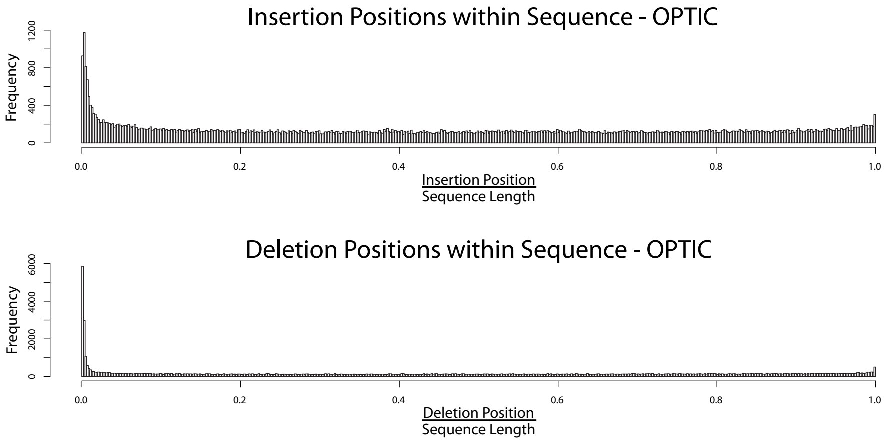


Figure B.9: Indels are highly non-uniform in their distribution across genes: we see a 6-fold enrichment for insertions within the N-terminal 1% of the protein sequence, and a 1.4-fold increase within the C-terminal 1%. There is an 14-fold enrichment in deletions within the N-terminal 1% of the protein sequence, and a 1.8-fold increase within the C-terminal 1%. Indel locations are normalized by gene length to enable combining data across all OPTIC genes analyzed. This may be a mix of genuine biology (e.g. indels occur more often near the ends of genes) and artifacts (annotation errors are more likely to occur at the ends of genes).

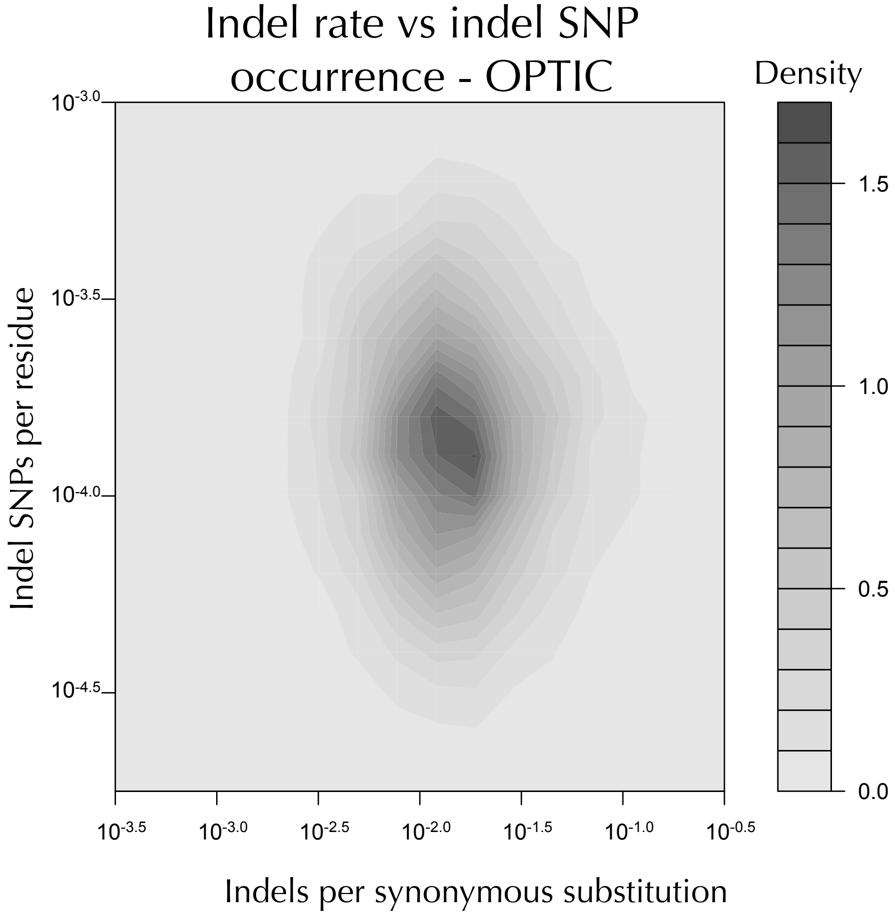


Figure B.10: Visualizing the number of indel SNPs per residue (using only human sequence) against the evolutionary indel rate (computed across the *Amniote* clade) shows no correlation.

Appendix C

Additional phenotype prediction figures

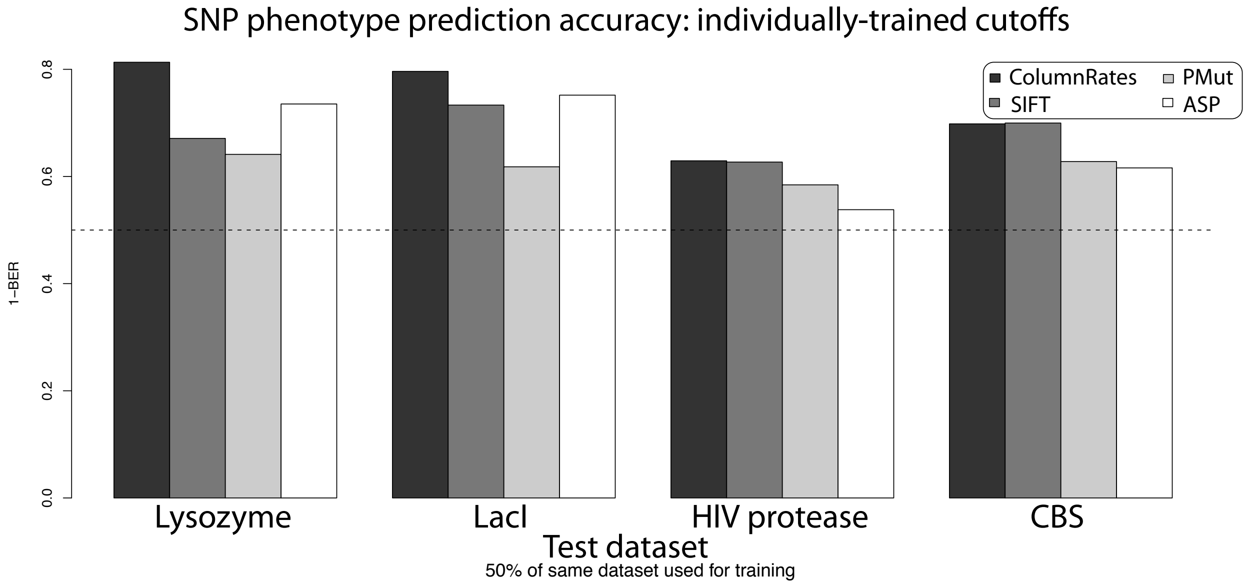
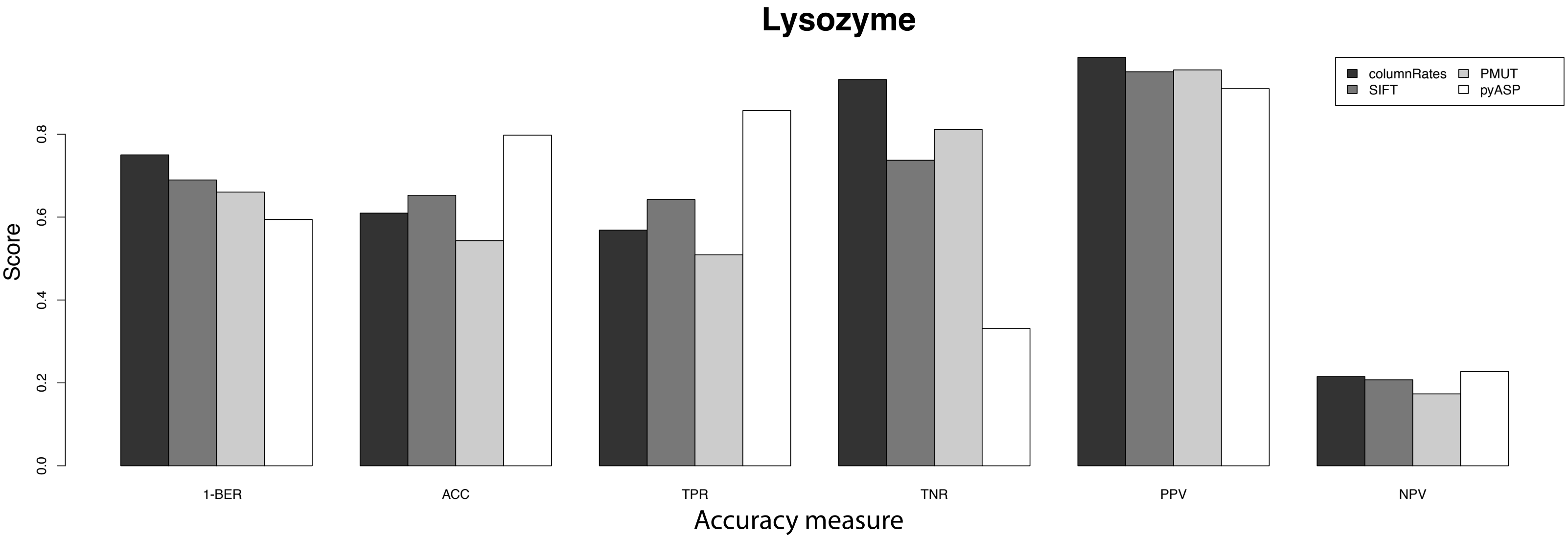
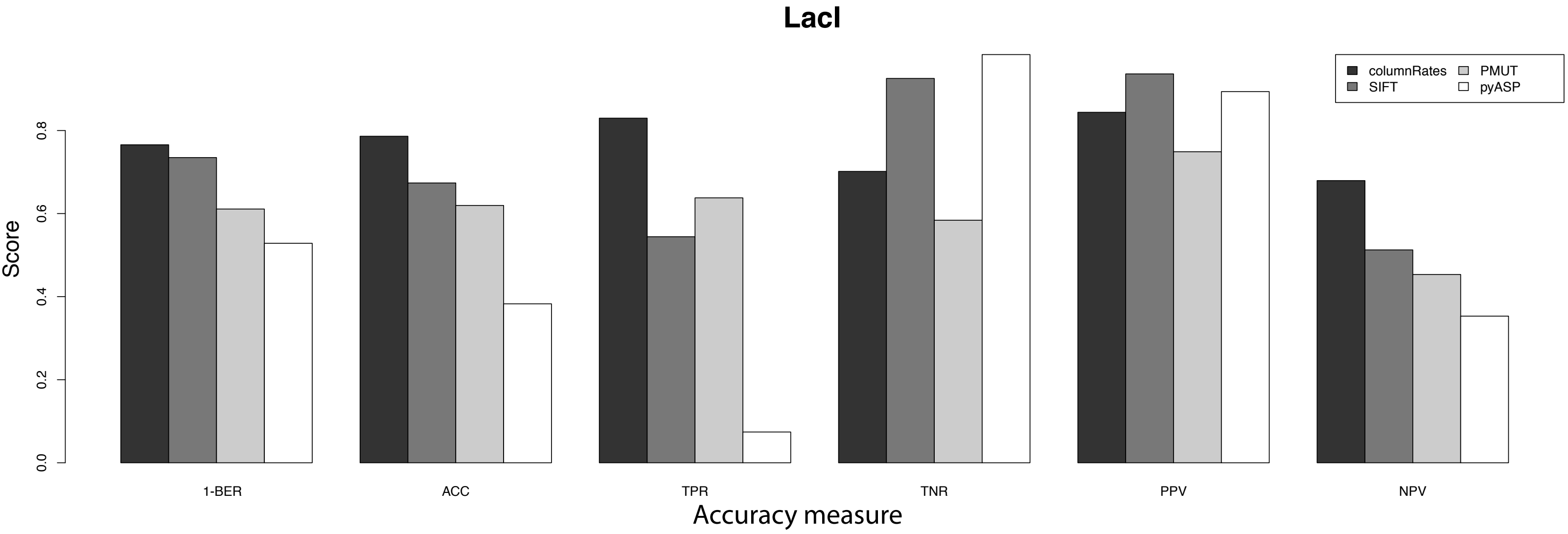
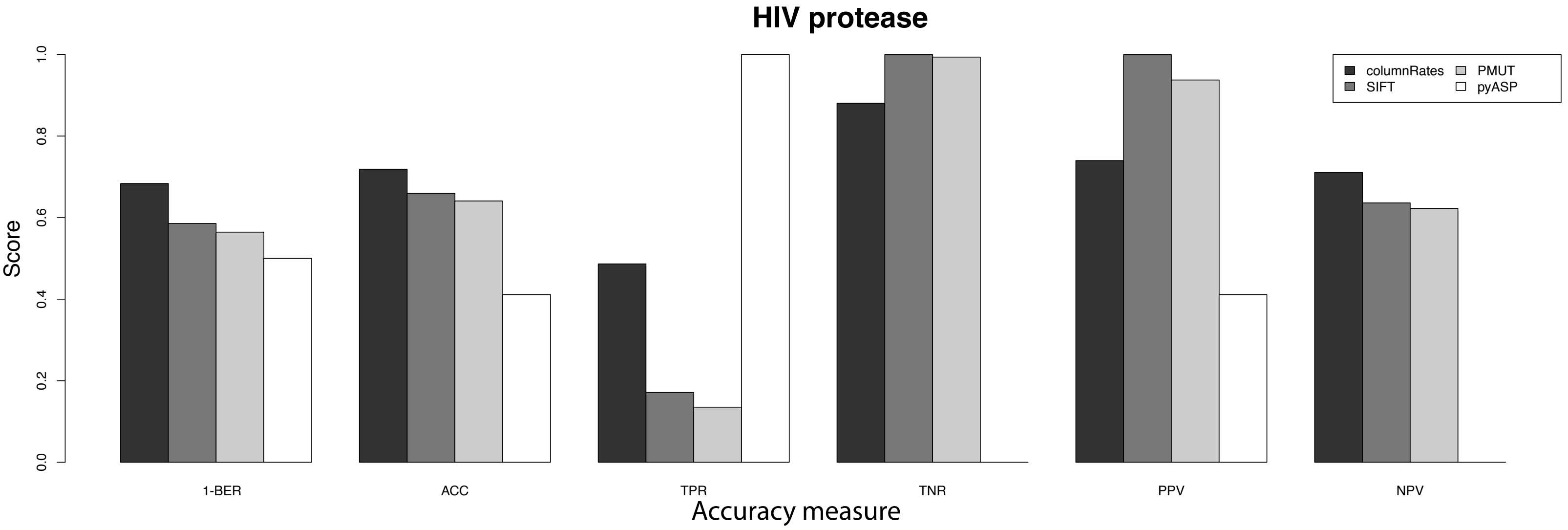


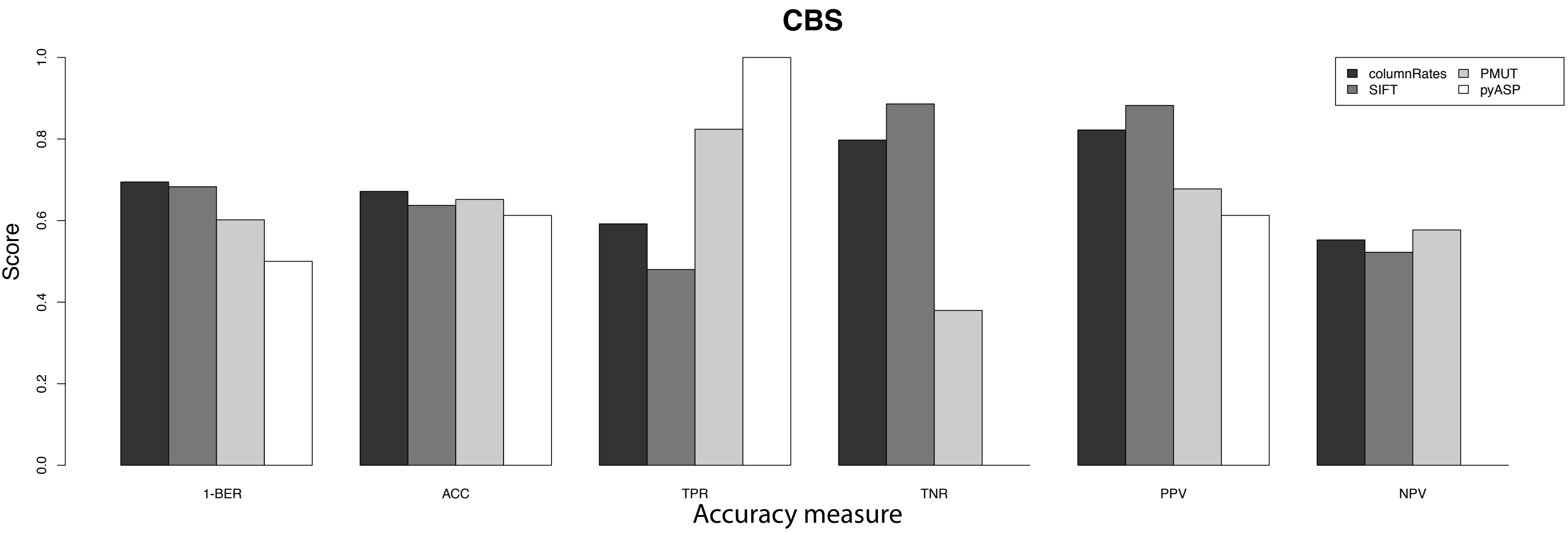
Figure C.1: When cutoffs for ColumnRates and ASP are optimized independently for each dataset (using 50% of data), ASP performs notably better, with accuracy between 60 and 80%. ColumnRates changes only a small amount, suggesting that it is capable of reliably analyzing genes which have no pre-existing training data. The dashed line shows a BER score of 0.5, the expected accuracy when randomly assigning SNPs as deleterious or neutral.

Here we show all the accuracy statistics from [123] for each of the four datasets. Each plot is titled with the dataset used (e.g. Lysozyme, HIV protease, etc) and the groups of bars each correspond to a particular statistic labeled under the group. Within each group of bars, the colors correspond to programs used for inference. The statistics are Balanced Error Rate (BER), Accuracy (ACC), True Positive Rate (TPR), Positive Predictive Value (PPV), True Negative Rate (TNR), and Negative Predictive Value (NPV), defined in Chapter 6, Materials and Methods.


Lysozyme
■ columnRates
■ SIFT
□ PMUT
□ pyASP
Score
0.0
0.2
0.4
0.6
0.8
1-BER
ACC
TPR
TNR
PPV
NPV
Accuracy measure


LacI
columnRates
SIFT
PMUT
pyASP
Score
0.0
0.2
0.4
0.6
0.8
1-BER
ACC
TPR
TNR
PPV
NPV
Accuracy measure


HIV protease
Score
0.0
0.2
0.4
0.6
0.8
1.0
columnRates
SIFT
PMUT
pyASP
1-BER
ACC
TPR
TNR
PPV
NPV
Accuracy measure


CBS
Score
0.0
0.2
0.4
0.6
0.8
1.0
columnRates
SIFT
PMUT
pyASP
1-BER
ACC
TPR
TNR
PPV
NPV
Accuracy measure

Appendix D

Table of DART-Scheme functions

The following list of Scheme functions, natively implemented within selected DART programs (including XRate) when compiled with GNU Guile, is only complete to the date of publication. A more up-to-date list may be found at `http://biowiki.org/DartSchemeFunctions`

Darts functions for working with trees

Scheme function	Effect
(newick-from-string x)	Create a tree-smob from a Newick-format string x
(newick-from-file x)	Create a tree-smob from a file x
(newick-from-stockholm x)	Create a tree-smob from the tree encoded within alignment-smob x
(newick-to-file x y)	Write tree-smob x to file y in Newick format
(newick-ancestor-list x)	List of all ancestors in the tree-smob x
(newick-leaf-list x)	List of all leaves in the tree-smob x
(newick-branch-list x)	List of all branches in the tree-smob x
(newick-unpack x)	Converts a tree-smob x into a Scheme data structure

Darts functions for working with alignments

Scheme function	Effect
(stockholm-from-string x)	Create an alignment-smob from a Stockholm-format string x
(stockholm-from-file x)	Create an alignment-smob from a Stockholm-format file x
(stockholm-to-file x y)	Write alignment-smob x to Stockholm-format alignment file y
(stockholm-column-count x)	Return the number of columns in alignment-smob x
(stockholm-unpack x)	Converts an alignment-smob x into a Scheme data structure

Functions for working with grammars

Scheme function	Effect
(xrate-validate-grammar x)	Validate the syntax of XRate grammar x
(xrate-validate-grammar-with-alignment x y)	Validate the syntax of XRate grammar x, using alignment-smob y to expand macro constructs
(xrate-estimate-tree x y)	Use XRate grammar y to estimate a tree for alignment-smob x
(xrate-annotate-alignment x y)	Use XRate grammar y to annotate alignments-smob x
(xrate-train-grammar x y)	Train XRate grammar y on the list of alignment-smobs y

Miscellaneous functions

Scheme function	Effect
(dart-log x)	Logging directive; equivalent to "`-log x`" at the command line
(discrete-gamma-medians alpha beta K)	Returns the median rates of K equal-probability bins of the gamma distribution
(discrete-gamma-means alpha beta K)	Returns the mean rates of K equal-probability bins of the gamma distribution
(ln-gamma k)	Calculates the gamma function, $\Gamma(k) = \int_0^\infty e^{-x} x^{k-1} dx$
(gamma-density x alpha beta)	Calculates the gamma probability density, $\beta^\alpha \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$
(incomplete-gamma x alpha beta)	Calculates the incomplete gamma function, i.e. the integral of the gamma density up to x
(incomplete-gamma-inverse p alpha beta)	Calculates the inverse of the incomplete gamma function

Appendix E

Glossary of XRate terminology

Within the glossary descriptions, *italicized phrases* refer to other glossary terms.

Alignment: See *multiple sequence alignment.*

Alphabet: The set of single-character tokens (symbols) from which sequences are constituted. The alphabet is defined in the grammar file; only one alphabet may be defined per grammar file. (Usually the alphabet is DNA, RNA or protein. Sometimes the alphabet is extended to include an explicit gap character.) An alphabet may optionally include a *complement* mapping, as well as specification of degenerate (ambiguous) characters.

Ancestral reconstruction: The use of XRate to reconstruct the sequences at ancestral nodes of a *phylogenetic tree*, given a *grammar*, a *multiple sequence alignment* and a *parse tree.* This occurs after *tree estimation*, *training* and *annotation.*

Annotation: The use of XRate to apply a *grammar* to a *multiple sequence alignment* and *phylogenetic tree*, so as to impute the optimal *parse tree* and mark up the alignment with the co-ordinates of selected features (associated with particular *nonterminals* in the parse tree), or generate other annotation including GFF and WIGgle files. This occurs after *tree estimation* and *training*, but prior to *ancestral reconstruction.* Can also refer to a specific part of a *transformation rule* that generates annotations.

Bifurcation: A *transformation rule* that generates two *nonterminals.* Bifurcation rules have the form

```
(transform (from (A)) (to (B C)))
```

where `A`, `B` and `C` are nonterminals.

Chain: A substitution rate matrix (the name comes from "continuous-time Markov chain"). The states of the substitution process are N-mers augmented with an optional hidden variable. That is, the state space of a chain consists of *state-tuples* of the form $(s_1, s_2, \ldots, s_N, h)$ with $N \geq 1$, where s_1 through s_N represent *alphabet* symbols (which will be observed in the final *multiple sequence alignment*) and h is an optional *hidden state* which can take on a finite set of single-character values specific to this chain. Each of the N alphabet symbols, s_1 through s_N, is associated with a unique *pseudoterminal.* Examples of valid chain state spaces include the set of all nucleotides; the set of all codons; and the set of all tuples (A, H) where A is an amino acid and $H \in \{F, S\}$ is a hidden binary variable taking values F (for fast) or S (for slow).

Complement: An order-2 permutation on the *tokens* of an *alphabet.* (Typically only used for DNA or RNA alphabets.)

Emission: A *transformation rule* that generates some *pseudoterminals* (and thus, some alignment columns); or the set of pseudoterminals (or alignment columns) generated by such a rule. In XRate, emission rules have the form $A \to x_1 \ldots x_L\ A{*}\ x_{L+1} \ldots x_{L+R}$ where $A, A{*}$ are paired *nonterminals* (whose names differ only by the final asterisk) and $x_1 \ldots x_{L+R}$ are pseudoterminals. ($A{*}$ is referred to as the *post-emit nonterminal.*) Any numbers L, R of pseudoterminals can appear to the left and right of the $A{*}$, as long as $L + R > 0$. If $L = 0$ and $R > 0$, the rule is a *right-emission*; if $L > 0$ and $R = 0$, the rule is a *left-emission.* The pseudoterminals $x_1 \ldots x_{L+R}$ must comprise (any permutation of) the full set of pseudoterminals for a given *substitution chain.* Each pseudoterminal may optionally be prefixed with a tilde character (~) to indicate that it should be *complemented* in the final alignment (used to generate reverse strands in double-stranded models). For example, if `C1`, `C2` and `C3` are the three pseudoterminals of a codon chain, `A` is an emission nonterminal and `A*` is the corresponding post-emission nonterminal, valid emission rules could include

```
(transform (from (A)) (to (C1 C2 C3 A*)) (prob (...)))
```

and

```
(transform (from (A)) (to (~C3 A* ~C2 ~C1)) (prob (...)))
```

Grammar: The contents of a grammar file: *chains*, *nonterminals*, *transformation rules*, and *alphabet.* (The alphabet is specified in a separate part of the file from the rest of the grammar, and so is sometimes omitted from this definition.)

Grammar symbol: A symbol that is either a *nonterminal* or a *pseudoterminal.*

HMM: Hidden Markov Model. An *SCFG* that is also a *regular grammar.* See also *phylo-HMM.*

Hidden state: In the context of XRate, this term is ambiguous (see *state*). In this article, it is used mostly to refer to the final element of a *state-tuple* in a *chain.* However, in the context of *HMM* theory, it refers to what we call a *nonterminal.*

Hybrid chain: A mapping from tree branches to substitution rate matrices (*chains*) where the instantaneous rate matrix may vary from one branch to another. This may be used to implement lineage-dependent selection, or other models which are heterogeneous with respect to the tree.

Initial distribution: The initial probability distribution over states in a substitution *chain*.

Left-emission: See *emission*.

Left-regular: A *grammar* is left-regular if it contains no *bifurcations* and its *emissions* are all *left-emissions*.

Macro: A construct that is expanded by the XRate grammar preprocessor and may be used to implement redundant or repetitive grammar models; e.g. *grammars* with a large number of similar *transformation rules* sharing the same probability parameter, or substitution *chains* whose *mutation rules* all share the same rate parameter.

Multiple sequence alignment: The raw data on which XRate operates, and which constitutes its input and output. XRate cannot align sequences, but assumes that they have been pre-aligned using an external alignment program. Alignments must be converted to Stockholm format [192] before supplying them to XRate. The alignment may include a *phylogenetic tree* (using the Stockholm syntax for specifying this); if no tree is provided, XRate's *tree estimation* routines can be used to find one.

Mutation rule: A single element in the rate matrix of a substitution *chain*.

Nonterminal: A *grammar symbol* that may be transformed, by application of *transformation rules*, into other nonterminals or pseudoterminals. In XRate, a nonterminal must be exclusively associated with (that is, appear on the left-hand side of) either *emission* rules, *transition* rules or *bifurcation* rules.

Parameter: A named parameter in a grammar. May be a *probability parameter* or a *rate parameter*.

Parametric model: A *grammar* whose *transformation rules* or *mutation rules* (or both) are specified as functions of the grammar's *parameters*, rather than as direct numerical values.

Parse tree: A tree structure corresponding to the derivation of a multiple sequence alignment from a *grammar*. Each tree node is labeled with a *grammar symbol*: the root node is labeled with the *start nonterminal*, internal nodes are labeled with *nonterminals*, and the leaves are labeled with *pseudoterminals*. Not to be confused with a *phylogenetic tree*.

PGroup: A set of *probability parameters* collectively representing a probability distribution over a finite set of events. Following training, probability parameters constituting a PGroup will be normalized to sum to 1.

Phylogenetic tree: The evolutionary tree describing the relationship between sequences in a multiple alignment. XRate uses the Stockholm format for alignments, which allows the tree to be included as an annotation of the alignment. If no tree is provided, XRate's *tree estimation* routines can be used to find one.

Phylo-grammar: See *phylo-SCFG.*

Phylo-HMM: A *phylo-SCFG* that uses a *regular grammar.* A phylo-HMM is an HMM whose *emissions* generate alignment columns by evolving *substitution chains* on a phylogenetic tree.

Phylo-SCFG: A phylogenetic *SCFG*: a member of the general class of *grammars* implemented by XRate. A phylo-SCFG is an SCFG whose *emissions* generate alignment columns by evolving *substitution chains* on a phylogenetic tree.

Post-emit nonterminal: See *emission.*

Production rule: See *transformation rule.*

Probability parameter: A dimensionless *parameter* that generally takes a value between 0 and 1, and so can occur in the probability part of a *transformation rule* (or as a multiplying factor in the rate part of a *mutation rule*). Probability parameters are declared in *PGroups.*

Pseudocounts: A set of nonnegative counts that specifies a Dirichlet prior distribution over a *PGroup.*

Pseudoterminal: A *grammar symbol* that is generated via an *emission* and cannot be further modified by subsequent *transformation rules.* In a *parse tree*, a pseudoterminal serves as a placeholder for an alignment column. Pseudoterminals occur in groups associated with a particular *substitution chain.* In the generative interpretation of the model, alignment columns are generated using the *initial distribution* and *mutation rules* of the chain, applied on the phylogenetic tree associated with the alignment.

Rate parameter: A nonnegative parameter that has units of "inverse time" (i.e. rate), and so can occur in the rate part of a *mutation rule.* Rate parameters can be declared individually.

Regular grammar: A *grammar* is regular if it is either *left-regular* or *right-regular*; that is, it contains no *bifurcations* and its *emissions* are all either *left-emissions* or *right-emissions.* A regular grammar is equivalent to an *HMM.*

Right-emission: See *emission.*

Right-regular: A *grammar* is right-regular if it contains no *bifurcations* and its *emissions* are all *right-emissions.*

SCFG: Stochastic Context-Free Grammar. See also *phylo-SCFG.*

Start nonterminal: The first *nonterminal* declared or used in a grammar. In the generative interpretation of the model, this is the initial *grammar symbol* to which transformation rules are applied. It is also the label of the root node in the *parse tree*.

State: In the context of a *phylo-grammar*, this term is ambiguous: it can refer either to a *state-tuple* in a *chain*, or (for phylo-HMMs) a *nonterminal* in a *grammar*. For the most part in this paper, and exclusively in this glossary, we use it in the former sense.

State space: The set of possible *state-tuples* in a *chain*.

State-tuple: A tuple of the form $(s_1, s_2, \ldots, s_N, h)$ representing a single state in a *chain*, where s_1 through s_N represent *alphabet* symbols and h is an optional *hidden state*.

Substitution chain: A continuous-time finite-state Markov chain over *state-tuples*. See *chain*.

Substitution model: See *substitution chain*.

Terminal: See *token*.

Token: An *alphabet* symbol. (Also called a *terminal*.)

Training: The use of XRate to estimate a grammar's *parameters*, *mutation rule* rates and *transformation rule* probabilities, given a (set of) multiple alignments. This occurs after *tree estimation* and prior to *annotation* or *ancestral reconstruction*.

Transformation rule: A probabilistic rule that describes the transformation of a *nonterminal* symbol into a sequence of zero or more *grammar symbols*. (Also called a *production rule*.) A transformation rule may be an *emission*, a *transition* or a *bifurcation*.

Transition: A *transformation rule* that generates exactly one nonterminal (and no pseudoterminals). Transition rules have the form

```
(transform (from (A)) (to (B)) (prob (...)))
```

where `A` and `B` are nonterminals.

Tree: In the context of a *phylo-grammar*, this term is ambiguous: it can mean a *parse tree* (which explains the "horizontal", i.e. spatial, structure of an alignment) or a *phylogenetic tree* (which explains the "vertical", i.e. temporal, structure).

Tree estimation: The use of XRate to estimate a *phylogenetic tree* for a *multiple sequence alignment*, given a *grammar*. This occurs prior to *training*, *annotation* or *ancestral reconstruction*.

Appendix F

RecHMM simulation and methodological details

F.1 Additional simulation results

In analyzing actual biological data, there is no easy way to verify the predicted recombination breakpoints, so to investigate the accuracy, reliability, and limits of our method, we applied our algorithms to data simulated by **Recodon**, [163]. We produced several synthetic datasets of varying lengths, number of taxa, number of recombination events, and structure of marginal trees. The general simulation parameters used were the following: recombination rate: 2×10^{-8} , mutation rate: 3×10^{-4}, {A,C,G,T} frequencies: {.3 .2 .2 .3}, transition/transversion ratio: 2.0.

Taxa number presented more of a limitation in actual data than in simulated data; on synthetic datasets we obtained conclusive results on alignments with up to 30 sequences. This is most likely due to the simplicity of the generated data; when modeling evolution on real data, simple Markov chain approaches greatly simplify the process, and we sum out gaps as missing information, whereas in the simulated data there were no gaps. Our simulated analyses were very accurate, usually if a breakpoint was detected, it was accurate within 10 positions. With marginal trees which differed only in subtle ways, such as branch-length or deep branching patterns, our programs rarely detected changes, whereas with changes at the leaves detection was near very strong. More HMM states than recombinant regions wasn't problematic, whereas having too few HMM states led to inaccurate or missed breakpoint predictions. We believe this to be the limiting factor in our method; it was uncommon to see a decisive model which employed more than 4 trees.

Many Taxa Intuitively, our methods will become less powerful the more taxa are included in the alignment, since phylogenetic tree estimation becomes more and more difficult. Note that there are many 'spikes' in the posterior distribution in positions 500-600 Figure

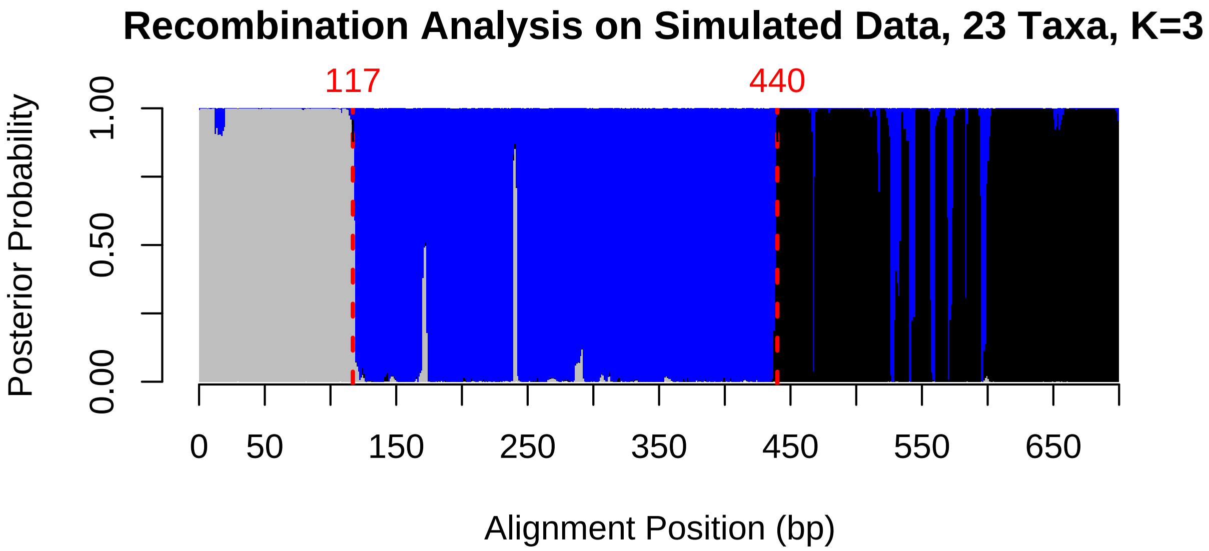


Figure F.1: Synthetic alignment of size 700 × 23, with two breakpoints (shown in red). Both breakpoints are topology-changing and both are detected with great accuracy, although there is some uncertainty in region 530-600.

F.1, signifying uncertainty of inference, and in Figure F.2 the posteriors are uninformative. Still, the program is often able to do quite well even with many taxa, such as in Figure F.3, where the posteriors are somewhat noisy but it is still clear where the change happens. The ability to include many taxa in a recombination analysis is very valuable, especially when the involved species are unknown. The reason for the decay in inference is that a group of phylogenetic trees with many taxa will have very similar likelihoods, and thus differentiating between them in order to partition an alignment is a difficult task.

Ancient Recombination In situations where recombination has occurred long ago, 'deep' in the phylogenetic tree, an entire subtree is transferred. This type of recombination is typically not detected very well by our method, since the ideal phylogenetic trees differ in their branching near the root, as opposed to near the leaves. This leads to the inference algorithm modeling the two regions by a single topology, since the likelihoods of the actual different trees is relatively similar. Practically speaking, this is not as clinically important as detecting recent recombinations. In a typical inter-subtype recombination analysis, the reference strains are taken to be 'pure' in their subtype, and the search for recombination only concerns the present-day clinical species. In the case when a few taxa in a large tree (i.e. 2 taxa in a 30-taxa tree) are transferred, this is usually detected. This could arise when analyzing a group of clinical isolates against a set of reference strains where two of the isolates had undergone similar recombination events. Figure F.4 and F.5 show an exam-

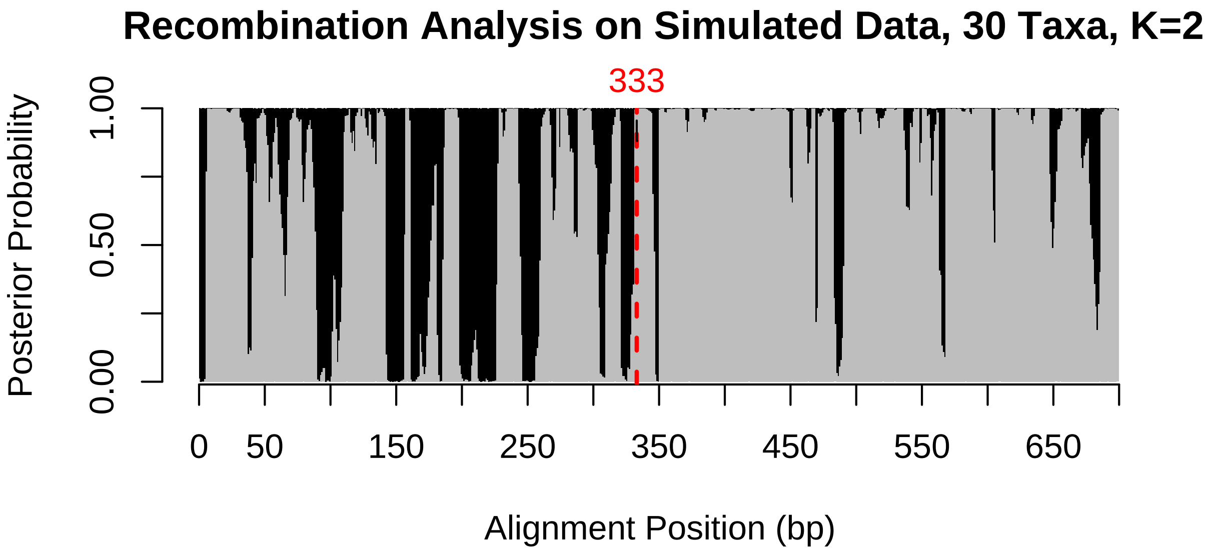


Figure F.2: Synthetic alignment of size 700 × 30, with one breakpoint (red). The entire posterior distribution is uncertain and no inference can be made. There is somewhat less ‘noise‘ in the signal after position 333, but other than this there is little to be learned from the results.

ple where an ancient recombination is missed and a recent leaf-changing topology change is clearly detected.

Too many HMM states When the number of tree-states is larger than the number of distinct regions in the alignment, this is typically not a problem, as seen in Figure F.6. One of the trees either remains at low posterior probability or only appears for very small regions. In this way, the training algorithm recognizes that it has an ‘extra’ tree that it doesn’t need to accurately model the data. In some cases, however, this extra state can be employed to model a more highly-diverged region which has a different optimal topology, but which is not actually a recombinant region. In Figure F.6, the blue tree topology remains at low probability except for two very short regions, and the breakpoint is still well-predicted.

Too few HMM states When the number of recombinant regions is more than the number of tree-states in the HMM, this has mixed effects. In Figures F.7 and F.8, the dataset actually had 5 breakpoints, and each region was topologically distinct, but only 4 resp. 6 trees were used to train the model. Sometimes topology shifts are detected, even if the this region does not have its own tree to train, but in general the model has a difficult time decisively detecting breakpoints. Adding more states, as Figure F.8 attempts, does not appear to enable detecting more regions, leading us to believe that number of distinct-topology regions of the alignment is the limiting factor in our methods. We are investigating more

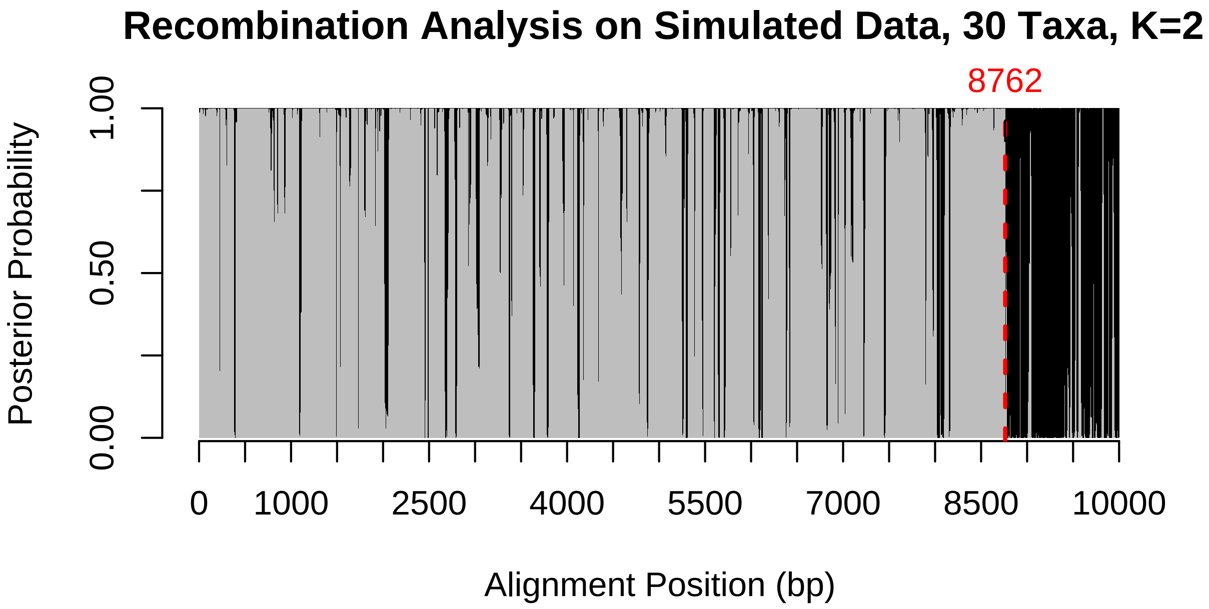


Figure F.3: Synthetic alignment of size 10,000 × 30, with one breakpoint (shown in red). The recombination point at 8762 is found, despite having to compare trees with 30 taxa. Longer alignments tend to have a clearer/stronger phylogenetic signal, which may be why this recombination was detected while the breakpoint in Figure F.2 was not.

robust initialization and training methods which might enable training many more states. The requirement that the different tree topologies fit together to form an ARG somewhat constrains the trees to look rather similar. If we relax this requirement, trees can differ by more than 1 subtree prune and regraft move (the operation of detaching an edge and reattaching it somewhere else in the tree), which leads to more radically distinct topologies. When we simulate alignments with these sorts of topologies, we are more readily able to detect 5 or more regions, as shown in Figure F.9.

Small regions Window-sliding methods typically perform badly in detecting small regions (eg less than 200 bp), whereas we are able to reliably detect small regions if they have distant supporting regions, and often even if they don't. In our method, all the information in the alignment is combined to construct trees, allowing distant but similar regions to collaborate in training trees which improves detection. If the short regions have a topology in common with a longer region in the alignment, they are typically detected very well, because their tree was mostly trained elsewhere in the alignment. If they represent a unique topology,

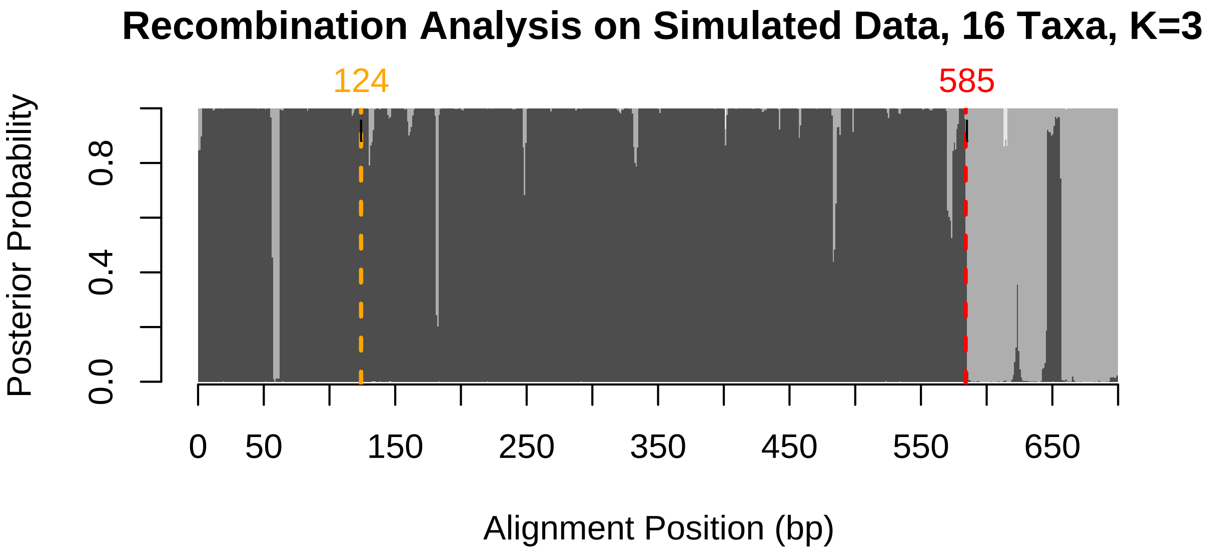


Figure F.4: Synthetic alignment of size 700 × 16, with two breakpoints. Breakpoint at 124 (orange) changes the topology near the root of the tree, whereas breakpoint 585 (red) changes tree topology near the leaves (see topologies in Figure F.5).

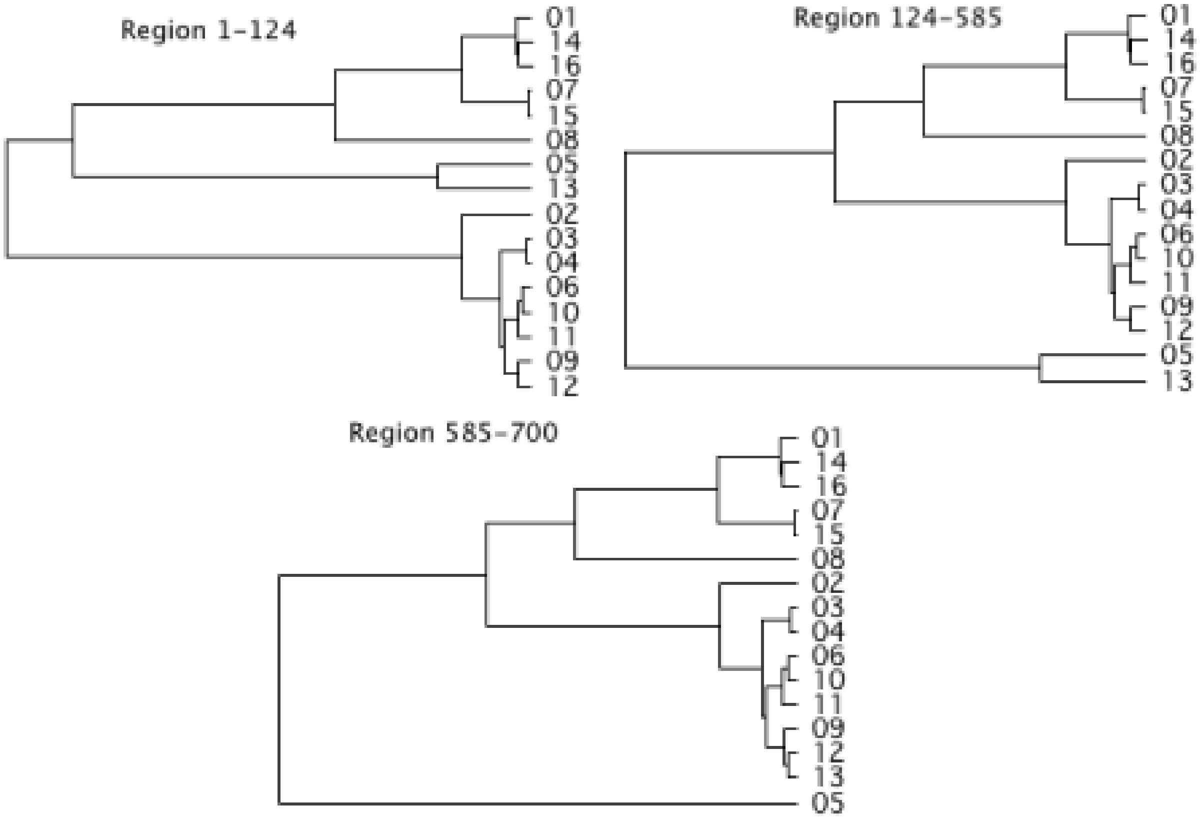


Figure F.5: Tree topologies for different regions in Figure F.4. The transformation between trees (1-124) and (125-585) involves a subtree transfer deep in the tree and so the breakpoint at position 124 goes undetected. In contrast, the difference between (124-585) and (585-700) involves transferring leaf 13, which is more readily detectable.

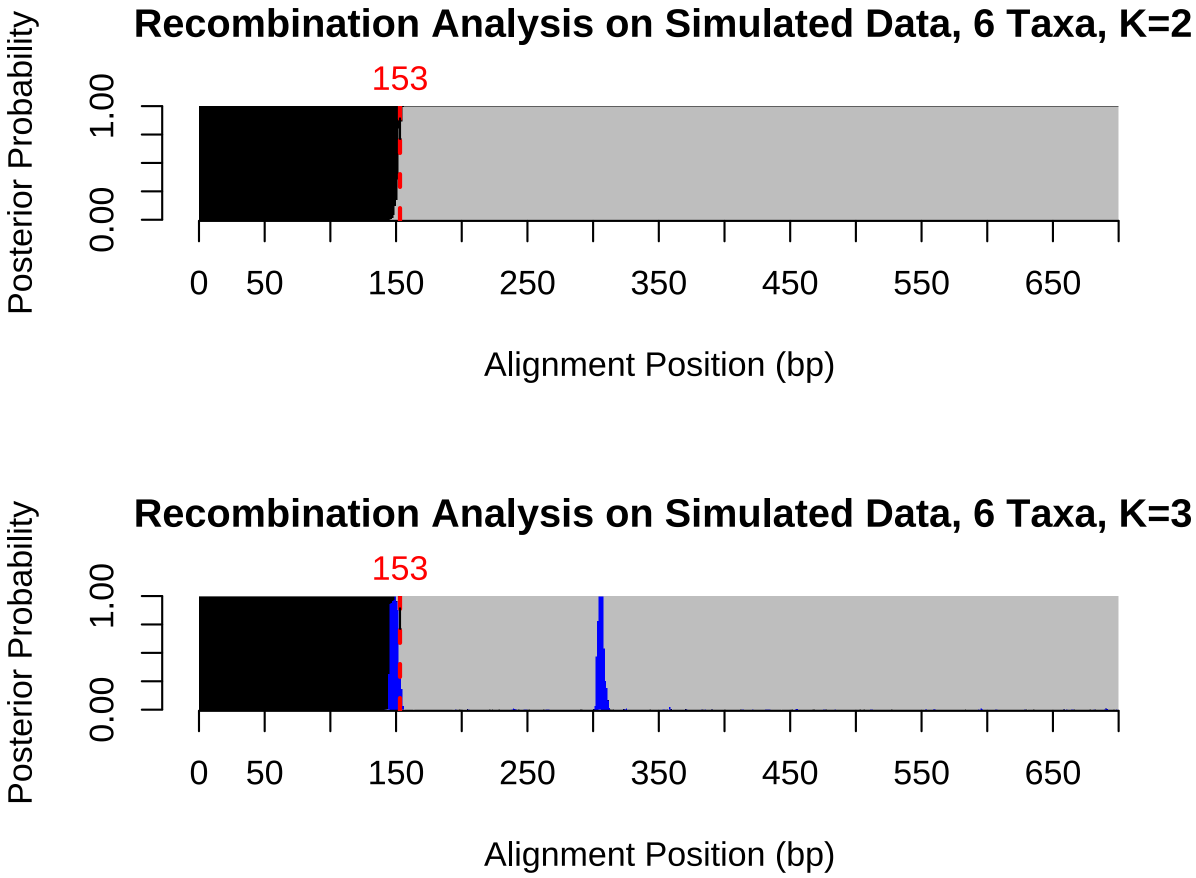


Figure F.6: When K=3, but there are only 2 recombinant regions, the third tree is unused except for a very short region near the crossover point, and a spike near bp 300.

it is difficult to construct a tree on such a small region, and inference is limited. If there are several small regions of a unique topology, however, they can combine their information to make detection more feasible. Figure F.10 shows results on an alignment with four small recombination regions which are all detected accurately. The small regions at 1000-1100 and 1600-1700 pool together their phylogenetic signal in order to train the black tree topology.

F.2 Methodological details

Here we describe the notation we use throughout this paper and the parameters we wish to learn from the data. In general we follow the notation of [17] and [28], extending it

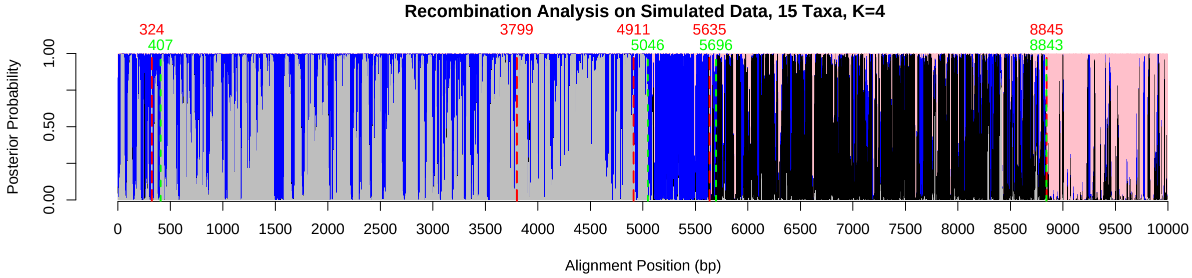


Figure F.7: When K=4, but there are actually 6 recombinant regions, inference is hampered. There are too many distinct topologies to train and the model is unable to accurately partition the alignment, leading to one breakpoint undetected and one (4911) predicted poorly. The predicted breakpoints (green) which are close to the actual simulated breakpoints (red) are labeled above the alignment. Those regions which are predicted have somewhat inconclusive posterior distributions, making inference difficult.

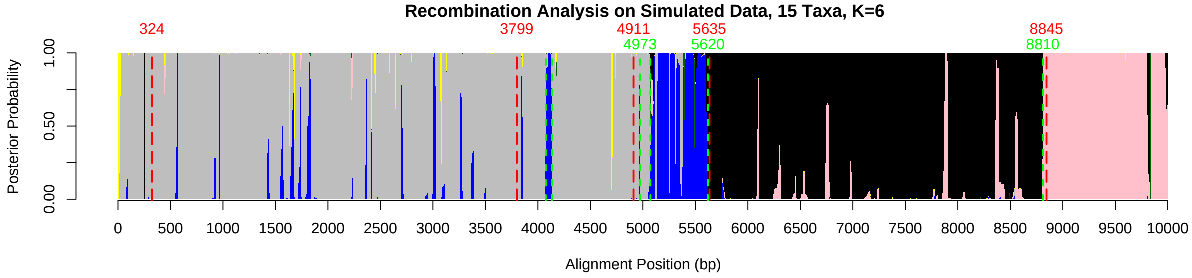


Figure F.8: Using the same data as Figure F.7, but setting K=6, we still get 2/5 breakpoints undetected. The training algorithm has a difficult time employing more than 4 trees in the HMM. In contrast to Figure F.7, the posteriors are quite decisive and the predicted breakpoints are reasonably accurate.

when necessary. While at present we must specify K, the number of trees in the model, an eventual goal is to be able to learn K from the data, allowing the training routine find the best number of states.

- An ancestral recombination graph (ARG) is a labeled directed acyclic graph in which nodes correspond to species and edges correspond to evolutionary relationships. Leaf nodes are assumed to be present-day species, and are taken to be observed, whereas ancestral nodes are hidden data. Edges can be solid or dashed, where a solid line indicates that all of the child's genetic material evolved directly from the parent's. A leaf under a dashed line represents a recombinant child, in which some of their sequence material came from one parent and some from the other. If the position of recombination is assumed to be known, then an ARG is able to convey all evolutionary

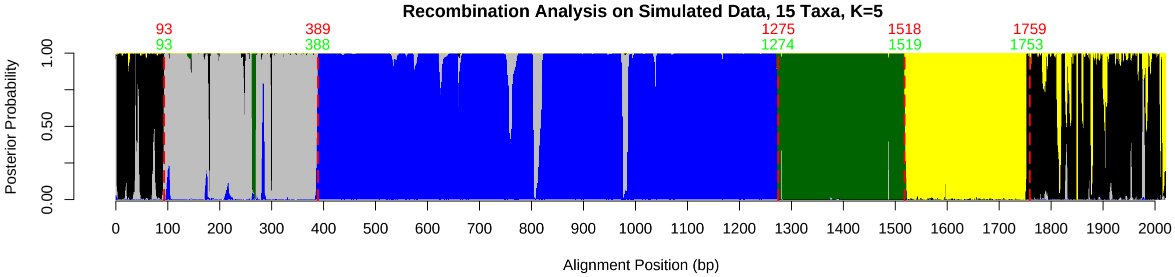


Figure F.9: Using data generated with arbitrary tree topologies, rather than ones that fit together to form an ARG, we are able to train 5 distinct trees, and locate the breakpoints with great accuracy. Relaxing the ARG requirement allows for radically different tree topologies which are easier for the program to detect. This alignment contained 6 distinct regions, but the first and last were somewhat similar, and here they are detected as being the same. Since these two are not neighboring, all breakpoints are detected well.

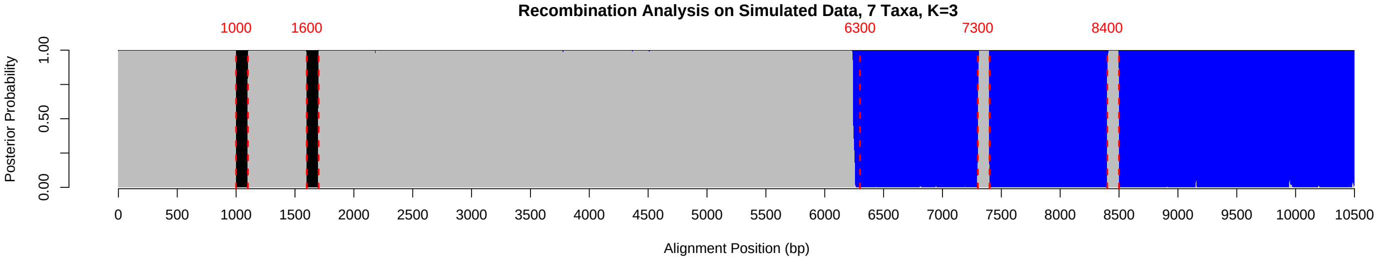


Figure F.10: Small recombinant regions. In addition to the main breakpoint at position 6300, this alignment contains four small crossover regions of size 100 bp (one label per small region, for clarity). The regions at 7300 and 8400 are of the same topology as the large grey region in the first half of the alignment, and so they are detected remarkably well (all four breaks accurate within 4 positions). The regions at 1000-1100 and 1600-1700 contain their own topology which appears nowhere else in the alignment, and so their information is contributed only to the black tree. This allows the black tree to be trained especially for those regions, leading to very close detection (accurate within 2 bp).

information of a group of present-day taxa [156]. In contrast to a phylogenetic *tree*, this *graph* contains nodes that have more than one parent.

- A marginal tree is a phylogenetic tree arising from an ARG by way of realizing a set of dotted edges such that each leaf node has exactly one parent, and internal nodes without children are removed. It is assumed that each position of the alignment can be described by single marginal tree.

- D is a multiple sequence alignment, taken as input. This is an array of N biological sequences, each of length M in which each column is assumed to have evolved from a single ancestral character [28], except when gaps are present. Gaps are treated as missing information. We use the terms 'position' and 'column' interchangeably as our model generates one column per position.

- $\{(T,t)_k\}$ is a set of phylogenetic tree topologies T and branch-length vectors t for each of the K hidden states. Strictly speaking, $\{(T,t)_k\}$ is not a *parameter* of our model, but it actually is part of the architecture of the model, since it models how sequences in the multiple alignment came to their present state via a certain evolutionary pathway. This is a subtle point since $\{(T,t)_k\}$ dictates θ^e, which is a parameter matrix of the HMM. Thus, whenever we re-estimate $\{(T,t)_k\}$, we are performing model selection as opposed to parameter estimation.

- θ^s is a stochastic matrix of size $K \times K$ in which the k,l entry indicates the probability of transitioning from tree k to tree l.

- θ^e is a stochastic matrix of size $K \times M$ where N is the number of sequences in the multiple alignment and M is the length of the alignment. $\theta^e_{k,m}$ represents the probability that tree k emitted column m of the alignment. We recognize that our choice of tree topologies defines our emission matrix in the following way:

$$\theta^e_{k,m} = P(D_m|(T,t)_k) \tag{F.1}$$

  Where D_m is the m^{th} column of the alignment data and $P(D_m|T_k,t_k)$ is the *marginal likelihood* of tree k producing D_m, which can be computed by Felsenstein's pruning algorithm [193].

- For simplicity and clarity of notation, we will henceforth refer to the set $\{(T,t)_k\}$, θ^e, and θ^s as θ or the "parameter set." We keep in mind that $\{(T,t)_k\}$ is actually part of the model structure, but since we are updating these three objects at each iteration, it makes intuitive sense to refer to them as a unit. When describing an algorithm such as EM where the parameter set is iteratively improved, $\theta^{(n)}$ refers to the parameter set at the n^{th} iteration.

- $P(D|\theta)$ represents the likelihood of the observed alignment data under our model and a particular parameter set θ.

- With $X = \{x_1, x_2...x_M\}$ representing the hidden data (in this case an assignment of phylogenetic trees to columns of our alignment), $P(D, X|\theta)$ is the probability of the fully observed data under the model, given by the equation

  $$P(D, X|\theta) = \theta^s_{start,x_1} \prod_{i=1}^{M} \theta^e_{x_i,D_i} \theta^s_{x_i,x_{i+1}} \tag{F.2}$$

  Further, the hidden and observed data are related by the property that

  $$P(D|\theta) = \sum_x P(D, x|\theta) \tag{F.3}$$

  Where x ranges over all M^K possible sequence assignments to the hidden nodes.

- $P(x_m = k|, D, \theta)$ denotes the posterior probability of a particular state emitting a certain column, conditioned on the parameters and alignment data. For details on the computation of this quantity, see [17].

- $P_{a \to b}(t)$ provides an underlying evolutionary model specifying the probability that symbol a evolved into b under time t, for $a, b \in \Sigma, t \geq 0$, as well as a prior distribution over Σ. This is assumed to be part of the evolutionary process and in this work we do not try to refine the model as we estimate the previous parameters.

Baum-Welch Algorithm The special case of the EM algorithm applied to HMMs is known as the Baum-Welch algorithm [17]. This takes advantage of the tree-like structure of HMMs, using the sum-product algorithm to efficiently calculate expected hidden data statistics. In describing the Baum-Welch method, we introduce the following terms, following the notation of Durbin *et al* [17]:

The Forward Probability (F)

Defined as $F[m, k] = P(D_{1...m}, x_m = k|\theta)$, this represents the probability of the observed data up to and including column m, requiring that tree k produced column m. The Forward probability gives us a simple way to calculate the probability of the observed set of columns under a particular parameter set:

$$P(D|\theta) = \sum_k F[k, M] \theta^s_{k,end} \tag{F.4}$$

This is intuitively understood as the probability of the sequence up to M (i.e. the whole sequence), summed out over all possible states for the last position, and then accounting for the transition into the End state.

The Backward Probability (B)

Defined as $B[m, k] = P(D_{m+1,m+2...M}|x_m = k, \theta)$, this is analogous to the Forward probability, but starting at the end of the sequence. Note that here we compute a conditional as opposed to a joint probability (although both are conditioned on θ).

Posterior Probabilities

Using these Forward and Backward probability arrays, we can now calculate the *posterior probability* of a certain hidden state assignment given the observed alignment columns. We use the following property:

$$\begin{aligned} & P(x_m = k|D,\theta) \\ = & \frac{P(D, x_m = k|\theta)}{P(D|\theta)} \\ = & \frac{P(D_{1\ldots m}, x_m = k|\theta)P(D_{m+1\ldots M}|x_m = k,\theta)}{P(D|\theta)} \\ = & \frac{F[m,k]B[m,k]}{P(D|\theta)} \end{aligned} \tag{F.5}$$

Where $P(D|\theta)$ is computed by way of equation (F.4). These probabilities are useful for both inference and training. First, in searching for optimal parameters, the posteriors are what allow us to build different trees on different parts of an alignment. At each iteration we use these posterior probabilities to weight hidden data counts in the tree-optimization step. Second, once we are satisfied with the parameters and are attempting to assign trees to columns, the posteriors can be of great use. The quantity $P(x_m = k|D,\theta)$ takes into account the HMM structure, whose transition probabilities put a restrictive prior probability distribution on the number of recombination crossovers. This not only allows for a more realistic interpretation, but also allows us to adjust the sensitivity and specificity of the algorithm by controlling the entries of θ^s.

Expected Counts

The Forward and Backward calculations also enable simple computation of expected hidden event counts, namely the number of transitions $k \to l$ and emissions of string σ from state k. In this application, we only estimate the transition counts, and use Structural EM to deal with the emission maximization. To get the expected number of transitions $k \to l$, we use:

$$\begin{aligned} & E\left[\#k \to l\right] \\ = & \sum_{m=1}^{M} P(x_m = k, x_{m+1} = l|D,\theta) \\ = & \frac{F[m,k]\theta^s_{k,l}\theta^e_{l,D_{m+1}}B[m+1,l]}{P(D|\theta)} \end{aligned} \tag{F.6}$$

With these expected counts in hand, updating the parameter θ^s is trivial:

$$\hat{\theta}^s_{k,l} = \frac{E\left[\#k \to l\right]}{\sum_{l' \in \Sigma} E\left[\#k \to l'\right]} \tag{F.7}$$

Structural EM Structural EM is an iterative model selection scheme aimed at finding the most likely phylogenetic tree model for a multiple alignment [28]. It follows the prototypical EM structure in that it alternates between estimating hidden data and maximizing the model until a critical point in the likelihood surface is reached. In the E-Step, the tree topology and branch lengths are held constant and hidden statistics are estimated, namely the character transition counts between nodes. In the M-Step, a new model (a phylogenetic tree) is selected and its branch lengths chosen based on the estimated hidden statistics. This is a useful method as it makes maximum likelihood phylogenetics computationally feasible with long sequences and many taxa. Further, its estimation step allows for differential column-weighting which allows us to progressively "focus" a tree on particular alignment region.

Structural EM Steps We give a high-level description and the reader is referred to the original Structural EM paper for more detail [28]

Algorithm: Structural EM (SEM)

Input: An $N \times M$ gapless multiple alignment of biological sequences

Output: A phylogenetic tree (T, t) such that $P(D|(T, t))$ is a critical point of the likelihood surface

E-Step: Compute expected hidden data counts $S_{i,j}(a, b)$, the expected number of transitions of symbol a to symbol b from node i to node j.

M-Step $_{Branches}$: With these counts, find the branch length $\hat{t}_{i,j}$ for each (i, j) pair (not necessarily an edge in the tree) which would maximize the contribution of edge (i, j) to the tree's likelihood function.

M-Step $_{Topology}$: From this matrix of likelihood contributions, construct a maximally-scoring topology with the branch lengths chosen from the previous step.

Incorporation into Phylo-HMM Training We use SEM in training our phylo-HMM to estimate a better and better set of trees for our model at each M-step. At each iteration we perform K independent executions of SEM model selection, where during the E-Step of each one, the counts $S^{(k)}_{i,j}(a, b)$ are weighted by the posterior probabilities of the different alignment columns having been generated by a particular tree. More precisely, while maximizing the k^{th} tree:

$$S^{(k)}_{i,j}(a, b) =$$

$$\sum_m P(x_m = k|D, \theta) P(x_i[m] = a, x_j[m] = b|D_m, T, t)$$

Where $P(x_m = k|, D, \theta)$ denotes the posterior probability of the m^{th} column being produced by the k^{th} tree given the observed data, as computed in equation (F.5). This posterior-weighting scheme is a common phylo-HMM training strategy. Intuitively, if we strongly believe that a column was generated by a particular tree, we would like the next update of that tree to have access to comparatively more information from that column than if that column-tree pair was unlikely. The higher a column is weighted for a particular tree, the more closely that tree will come to 'fitting' that column perfectly after the SEM step. In the extreme case where $P(x_m = k|D, \theta) = 1$ for certain columns and 0 for others, this is equivalent to performing Structural EM on only the columns for which $P(x_m = k|D, \theta) = 1$. As the EM training progresses, the hope is that the posterior distribution will become more and more certain of which trees go with which columns.

The above modified EM-algorithm is unusual in that the M-step is another EM. This is known as a nested EM, covered by [194]. This somewhat bends the rules of EM which state that there must be a reliable MLE for the model once the hidden data has been estimated. In this case, we note that EM typically increases drastically in likelihood in the first few iterations and then increases very slowly, so it is important to set the number of 'inner' iterations carefully. If the EM$_{inner}$ is allowed to go until it converges, then $P(D|\theta^{(n)})$ is guaranteed to increase monotonically with each iteration of EM$_{outer}$, but the speed of this convergence will be greatly compromised. If the EM$_{inner}$ is too restrained, irregular behaviour of $P(D|\theta^{(n)})$ may be observed, and convergence of EM$_{outer}$ is not guaranteed. Ideally, some middle ground would be reached in which the EM$_{inner}$ is allowed to run sufficiently so that EM$_{outer}$ increases in likelihood with every (or almost every) step [194]. In our programs, the EM$_{inner}$ is set to run 2 iterations of Structural EM which appears to be adequate.

Algorithm: Posterior Decoding

Definitions: Let π be a path of length m through the state space of the HMM (discounting start and end states), emitting the first m columns of the alignment, with one state per column. Let M be the total number of columns in the alignment. We say that the state path is *partial* if $m \leq M$, and *complete* if $m = M$.

Let π_n denote the n^{th} state in path π. The *score* of path π is defined as $S(\pi) = \sum_{n=1}^{m} P[\pi_n, n]$ where $P[i, m]$ is the posterior probability that column m was emitted by state i.

We say a state path π is *valid* if all its breakpoints are more than ϵ apart. That is, there exist no $m, n > 1$ with $n - m < \epsilon$ such that $\pi_{m-1} \neq \pi_m$ and $\pi_m \neq \pi_n$.

Let $\Pi(i, m, e)$ be the set of all valid partial state paths of length m, whose last e columns were emitted from state i. (That is, for $\pi \in \Pi(i, m, e)$ and $m - e < n \leq m$, we must have $\pi_n = i$.) We then define $U[i, m, e] = \max_{\pi \in \Pi(i,m,e)} S(\pi)$ to be the highest score of any such path, computed recursively as follows.

Input: A $k \times M$ matrix of posterior state-column probabilities, $P[i, m]$, and a minimum breakpoint separation ϵ.

Output: A complete valid state path π through the HMM such that the score $S(\pi)$ is maximal. Note that a state path uniquely determines a set of breakpoints $\{m : \pi_m \neq \pi_{m+1}\}$.

Recursion:

```
for $i = 1$ to $k$:
for $m = 1$ to $M$:
for $e = 1$ to $\min(m, \epsilon)$:
```

$$U[i,m,e] = P[i,m] + \max \left(\begin{array}{rl} 0 & \text{if } m = 1 \\ U[i,m-1,e] & \text{if } m > 1 \\ U[i,m-1,e-1] & \text{if } m > 1 \text{ and } e > 1 \\ \max_{i'} U[i',m-1,\epsilon] & \text{if } m > \epsilon \text{ and } e = 1 \\ \max_{i'} U[i',m-1,m-1] & \text{if } 1 < m \leq \epsilon \text{ and } e = 1 \end{array} \right)$$

Final score $U_{\text{final}} = \max_i U[i, M, 1]$

The score of the maximally-scoring path is U_{final}. The path having this score can be recovered by a straightforward traceback.

The terms in the max expression correspond to the possible incoming paths, and can be intuitively understood in the following way:

0 if $m = 1$: for the first column in the alignment, there is no incoming path. This term initializes the dynamic program.

$U[i, m-1, e]$ if $m > 1$: the path stays in the same state as the previous column. Since the incoming path was in state i for at least e steps, the current path must also have been in state i for at least e steps.

$U[i, m-1, e-1]$ if $m > 1$ and $e > 1$: the path stays in the same state as the previous column. Since the incoming path was in state i for at least $e-1$ steps, the current path must have been in state i for at least e steps.

$\max_{i'} U[i', m-1, \epsilon]$ if $m > \epsilon$ and $e = 1$: the path changes state outside of the first ϵ columns of the alignment. To prevent breakpoints being closer than ϵ, this is only allowed to happen if the incoming path was in the same state for ϵ steps.

$\max_{i'} U[i', m-1, m-1]$ if $1 < m \leq \epsilon$ and $e = 1$: the path changes state within the first ϵ columns of the alignment. To prevent breakpoints being closer than ϵ, this is only allowed to happen if the incoming path was in the same state for all $m-1$ of the previous columns.