Paleogenetic Algorithms for Primordial Ribozymes: Computational Reconstruction of Ancient RNA

SUPPLEMENTARY MATERIAL

Robert K. Bradley and Ian Holmes

Revision Notes:

1. These comments are not a complete summary of the review, just a partial strategy for addressing some of the points. The review is the more authoritative guide... He is giving us the benefit of the doubt but acceptance is still very much in the balance, I think, so we need to stick at it, and take it that extra step beyond minimal compliance with his suggestions.

2. Obviously we have to do simulation tests and hopefully EvolSayer will be of some use there. One thing to be careful of is that there are certain regimes of rate/branch-length parameter space where the "trivial" outcomes identified by the reviewer (ancestral sequence is always identical to one of the descendants) are the norm. Oscar and I have been looking at this quite a bit recently (see also 2003 paper by Mossel). The regime we are probably most interested in is where the branch lengths are of comparable length and so the two slightly-longer branches can "out-vote" the shorter branch under certain conditions. This should probably be commented on in the paper: the reviewer says he

would be happy with “a small, noncomprehensive set of anecdotes” but if we can rationalize why some parameter settings are more interesting than others, then that’s all to the better.

3. I would actually be very wary of taking that “small noncomprehensive set of anecdotes” at face value. This is one area where I sense we’re being given enough rope to hang ourselves... an unsatisfactory response to the simulation issue will be unequivocal grounds for rejection. Obviously there are limits to the number of simulations we can run, but with EvolSayer’s output statistics (allowing us to filter out “interesting” cases), we can automate it somewhat.

4. I wrote out the TKF structure tree for the case $a \rightarrow b \rightarrow c$, which can be used to estimate the intermediate sequence b (e.g. for Redelings-Suchard). I think this could be useful in explaining tree transducer composition more concretely, and why is important: by inspecting these tables, the reader can see that the structure of this grammar is quite complicated, and should then be prepared for the idea that the “composition algorithm” is simply generating this table (or other tables for other phylogenies). The latex for this is in the file “tkfst.tex”, and is currently include’d at the start of the supplementary information (it should go in the main paper though, I think).

5. Can we dump out the grammar for the three-branch TKFST and include it in the Supplement? (Or the main paper?) I realize it’s huge, but this very hugeness does motivate the transducer composition algorithm, in the same way that the two-branch model (which is now included in a more human-readable form) does.

6. If we are moving the description of loopy DP to the main paper, I think it needs to be more precise. Currently it just says “we repeatedly iterate over strongly-connected components of the state sub graph”, which falls short of the level of precision/detail with which the other DP algorithms are presented. The description should encompass alter-

nate strategies (i.e. orders of visiting the states) that you have considered: randomized, Tarjan's algorithm, etc. The reader should be left understanding the pros-and-cons of each, e.g. worst-case performance or difficulty of implementation. No need to agonize about this, but be helpful to potential future developers. (Example reader: a grad student in our group who might try to implement loopy DP themselves.)

7. The descriptions of the DP algorithms do not currently include any traceback algorithms (CYK or Inside); therefore, we haven't technically described how to do ancestral reconstruction! Obviously these algorithms are almost trivial, but they're quite important. Implementing stochastic loopy-Inside traceback in tripletscfgdp.{h,cc} is a key step in transforming it into a useful tool (for example, with this extension, Indiegram would be pretty much ready-to-go for Redelings-Suchard sampling). The deterministic CYK traceback algorithm is *even more* important to present here, because implementing this algorithm is currently our main result! The paper should describe *both* these traceback algorithms. This shouldn't be that hard to do in a comprehensible, rigorous style (loopy traceback is much simpler than the algorithms for restoring eliminated states to traceback paths, which are painfully described in my 2003 paper).

8. All algorithms should be described in full, rather than giving recipes for deriving some algorithms from others. In particular, the CYK algorithm is currently summarized by statements along the lines of "replace sums over paths through the model (or equivalently, parses) with the $\max()$ operation (e.g., in equation X, sum-over-complicated-expression will be replaced with max-over-complicated-expression)". It shouldn't be too hard to write a latex command/macro that can be used (with "$\sum$" or "$\max$" as an argument) to generate complete versions of *both* algorithms.

9. The paper the reviewer cites as an example of the simplicity of missing data estimation (Rivas and Eddy (2008) PLoS CB 4:e1000172) is an IID-columns model for a multiple

alignment. When he says that this issue "may simply go away if you view the problem as a missing data problem", that is precisely what we are talking about when we discuss null cycle elimination. I've tried to explain in "tkfst.tex" why this is harder than it superficially appears:

(a) eliminating missing "empty" columns from an IID-columns model (such as in Rivas and Eddy, 2008) is easy.

(b) eliminating missing emissions from an HMM is slightly harder (c.f. "wing retraction" algorithm in HMMer source code) but general algorithms do exist.

(c) eliminating null states from an SCFG is much harder, due to null bifurcations.

10. The reviewer asks for early discussion of constraints in the context of the "rate-limiting steps of paleogenetics". We should take this opportunity to clarify that our approach is (1) formulate the model, (2) apply appropriate constraints. Our emphasis here is on (1). Cite the literature: QRNA represents an alignment constraint on a pair SCFG, Holmes-Rubin 2002 uses fold constraints on pair SCFGs, stemloc uses fold+alignment constraints, Ortheus uses an alignment constraint, etc, but none of these things can be done (in a probabilistic modeling framework at least) until you can formulate the model. For this paper, we chose to condition on structures essentially because they are easy to constrain (fold envelopes can be applied independently to each sequence). But we could equally easily have conditioned on alignments. Constraints are easy to tweak, once you have a model.

11. The reviewer also comments on the phylogeny being a given. We can note that even though we constrain the phylogeny, more realistic phylogenetic applications are possible. Given a structural phylo-alignment (tree+alignment+structure), we can calculate the likelihood; therefore, given two such structural phylo-alignments, we can say which one is more probable (describe their relative posterior probabilities, etc.). From this it

is possible to make a crude MCMC sampler, just by proposing/accepting/rejecting alignment/tree/structure changes. An *efficient* MCMC sampler needs to be able to make local topology changes and to resample the multiple alignment around a topology change; we can do this too, in principle, but it's beyond this paper.

12. As I understand it, pruning & peeling are not exactly equivalent as the reviewer claims, *except* in the special case of a reversible model. Specifically, pruning is a *restricted case* of peeling, giving you the posterior probabilities at the root node, but not any other node. Of course, if you have a reversible model, you can re-root the tree anywhere, and so the two are equivalent for reversible models only. When Felsenstein says the two approaches are the same (on p253 of his book), it is quite a vague statement: e.g. later in the same paragraph, he says that both peeling and pruning are variants of Horner's rule, and particular cases of dynamic programming. All of this is somewhat academic; basically I'm thinking the pruning/peeling distinction might yet have some pedagogic value, and we might not need to abandon it entirely.

13. We should address all the reviewer's comments about the figures if possible. I have added xfig files "stree.fig" and "stree-mutations.fig" to the repository; the latter shows an evolutionary trajectory of the TKFST, as requested by the reviewer.

14. Finally, a very minor/trivial point: would you consider a name-change for IndieGram to something beginning with "Evol"? The EvolDeeds package is EvolDoer (previously released as "tkfstalign"), EvolSayer and IndieGram. I put a few random suggestions for Evol-names at **biowiki.org/EvolDeeds**. Don't take this too seriously, of course! I'll understand if you want to stick with IndieGram, and this is obviously negligible compared to the other revisions.

Contents

1 The TKF Structure Tree model

Here we study the TKF Structure Tree on the two-branch phylogeny $a \xrightarrow{t} b \xrightarrow{t'} c$. We present the singlet, pair and triplet rule-sets separately.

We explicitly note cases where certain rule-sets can be derived "automatically" from simpler rule-sets: this motivates the development of the generic tree transducer composition algorithm.

1.1 Time-dependent probabilities

Here $n \in \{l, s\}$ denotes the type of structural element (loop or stem). $\mathbf{R}^{(n)}$ is the rate matrix and π_n the equilibrium distribution. λ_n is the insertion rate and μ_n is the deletion rate.

$$\begin{aligned}
\alpha_n(t) &= \exp(-\mu_n t) \\
\beta_n(t) &= \frac{\lambda_n(1-\exp((\lambda_n-\mu_n)t))}{\mu_n - \lambda_n \exp((\lambda_n-\mu_n)t)} \\
\gamma_n(t) &= 1 - \frac{\mu_n(1-\exp((\lambda_n-\mu_n)t))}{(1-\exp(-\mu_n t))(\mu_n - \lambda_n \exp((\lambda_n-\mu_n)t))} \\
\kappa_n(t) &= \lambda_n/\mu_n \\
M_n(i,j;t) &= \exp(\mathbf{R}^{(n)}t)_{ij}
\end{aligned}$$

Let $\alpha_n \equiv \alpha_n(t)$ and $\alpha'_n = \alpha_n(t')$; similarly $\beta'_n, \gamma'_n, M'_n$.

1.2 Grammars

Note that any of the grammars can be "downsized" to a smaller number of sequences, simply by dropping terminals, yielding alternative applications for each grammar.

For example, the Pair SCFG for $a \xrightarrow{t} b$ becomes a Single SCFG for a if sequence b is dropped by removing terminals w, x. Stochastic (loopy) traceback then can be used to sample the descendant b. Thus, we have constructed a forward-time simulator, simply by application

of the loopy Inside & stochastic traceback algorithms.

Similarly, $a \xrightarrow{t} b \xrightarrow{t'} c$ can be viewed as a pair grammar $a \xrightarrow{t+t'} c$ that can be used to sample an unknown evolutionary intermediate b. Again, this requires only the loopy Inside & stochastic traceback algorithms.

1.2.1 Singlet rules

This section contains three singlet grammars, one for each of the sequences a, b and c.

Note that Tables 2 and 3 may be deduced automatically from Table 1 if it is known that the underlying process is initially at equilibrium.

1.2.2 Pair rules

This section contains two pairwise rule-sets.

The pair grammar for $a \xrightarrow{t} b$ is the union of Tables 1, 2 and 4.

The pair grammar for $b \xrightarrow{t'} c$ is the union of Tables 2, 3 and 5.

Note that Table 5 may be deduced automatically from Table 4 if it is known that the underlying process is stationary.

1.2.3 Triplet rules

Because of its length, we have split the triplet rule-set over three tables:

Table 6: Rules for transforming $\{L_{abc}, S_{abc}\}$, after emissions to c;

Table 7: Rules for transforming $\{L_{ab*c}, S_{ab*c}\}$, after deletions on $b \xrightarrow{t'} c$;

Table 8: Rules for transforming $\{L_{a*bc}, S_{a*bc}\}$, after deletions on $a \xrightarrow{t} b$.

The triplet grammar for $a \xrightarrow{t} b \xrightarrow{t'} c$ is the union of Tables 1 through 8.

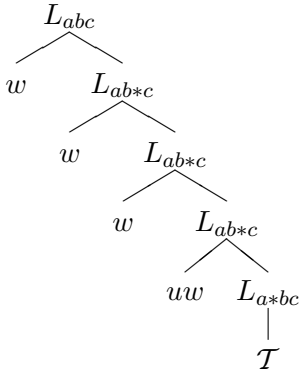
Note that Tables 6 through 8 may be deduced automatically from Tables 1 through 5, **regardless of the properties of the underlying process,** using the tree transducer composition algorithm.

1.3 Dealing with null cycles

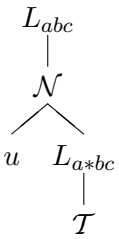
We require "loopy" versions of the dynamic programming algorithms (Inside & Outside) because we do not know of an exact way to eliminate null cycles from SCFGs.

In a probabilistic alignment model where columns are independent & identically-distributed (IID), such as (1), eliminating null/unobserved columns is straightforward. Suppose that $P(x)$ is the probability of column x, and $P(\epsilon)$ is the probability of an unobservable (empty) column; then, the probability that the next *observable* column is x is simply $\sum_{n=0}^{\infty} P(\epsilon)^n P(x) = \frac{P(x)}{1-P(\epsilon)}$, obtained by summing an infinite geometric series for the number of empty columns, n.

If the alignment model is not an IID sequence but rather a Hidden Markov model, elimination of unobserved columns is slightly more involved, but still well-understood (2). For example, consider the following parse tree for the $a \xrightarrow{t} b \xrightarrow{t'} c$ grammar:



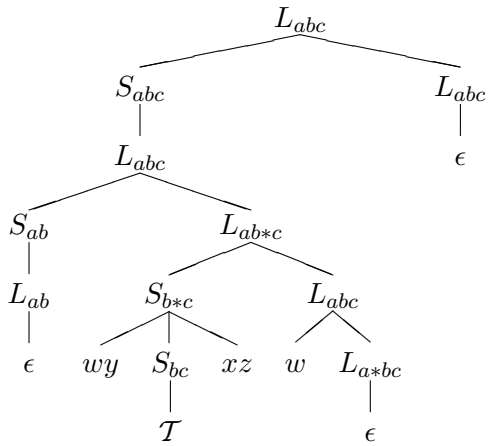
After removing unobserved w-terminals, this becomes:



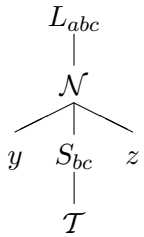
where $\mathcal{T}$ is a subtree and $\mathcal{N}$ represents the marginalized null subtree (which in this case

has a linear chain structure). Elimination of such linear chains of null states results in "effective direct transitions" between non-null states; in this case $L_{abc} \rightarrow u\ L_{a*bc}$. Such eliminations are relatively easy to handle; the effective non-null-transition matrix is obtained by summing a geometric series of null-transition matrices (2). This may be viewed as the matrix generalization of the argument for IID models, outlined above.

Thus, null-state elimination for HMMs and IID models is straightforward. In contrast, null-state elimination for SCFGs is complicated by the possibility of *bifurcating* null paths, in addition to linear ones. Consider the following parse tree:



After removing unobserved w, x-terminals and pruning null branches, this becomes:



In this case, elimination of the null states also results in an effective direct transition (from L_{abc} to S_{bc}). However, the eliminated subtree is not a linear chain of null states, but rather a bifurcating structure in which all but one of the branches generate empty sequence. This structure makes elimination more complicated.

For some insight into why this gets complicated, consider the following toy grammar, with terminal u and nonterminals A, B, C:

lhs	$\rightarrow$	rhs	prob.
A	$\rightarrow$	$B\ C$	$1-p$
	$\mid$	$u\ A$	p
B	$\rightarrow$	$C\ A$	$1-q$
	$\mid$	ϵ	q
C	$\rightarrow$	$A\ B$	$1-r$
	$\mid$	ϵ	r

Now, for each $X \in \{A, B, C\}$, consider the probability $P(\epsilon|X)$ that X generates an empty sequence. These probabilities can be used for null subtree elimination; e.g. bifurcation $A \rightarrow B\ C$ is, with probability $P(\epsilon|C)$, equivalent to direct transition $A \rightarrow B$. The probabilities are, however, non-linearly coupled:

$$\begin{aligned} P(\epsilon|A) &= (1-p)\ P(\epsilon|B)\ P(\epsilon|C) \\ P(\epsilon|B) &= (1-q)\ P(\epsilon|C)\ P(\epsilon|A) + q \\ P(\epsilon|C) &= (1-r)\ P(\epsilon|A)\ P(\epsilon|B) + r \end{aligned}$$

Numerical solution of such coupled non-linear equations is, of course, possible (e.g. via Newton's method), and exact solution may be possible in some cases; but this is somewhat more involved than solving the coupled *linear* equations required to eliminate null paths from HMM-like models (which can be done using matrix algebra) and significantly harder than eliminating unobserved symbols from an IID sequence (which can be done in closed form). Loopy dynamic programming is an alternative numerical approach that is easier to implement.

TKF Structure Tree singlet rules (a)
Sequence a *terminals:* $\{u, v\}$.

lhs	$\rightarrow$	rhs	$P(a)$
L_a	$\rightarrow$	$u\ L_a$	$\kappa_l \pi_l(u)$
	$\mid$	$S_a\ L_a$	$\kappa_l \pi_l(S)$
	$\mid$	ϵ	$1 - \kappa_l$
S_a	$\rightarrow$	$u\ S_a\ v$	$\kappa_s \pi_s(uv)$
	$\mid$	L_a	$1 - \kappa_s$

Table 1: Singlet rule-set for a.

TKF Structure Tree singlet rules (b)
Sequence b *terminals:* $\{w, x\}$.

lhs	$\rightarrow$	rhs	$P(b)$
L_b	$\rightarrow$	$w\ L_b$	$\kappa_l \pi_l(w)$
	$\mid$	$S_b\ L_b$	$\kappa_l \pi_l(S)$
	$\mid$	ϵ	$1 - \kappa_l$
S_b	$\rightarrow$	$w\ S_b\ x$	$\kappa_s \pi_s(wx)$
	$\mid$	L_b	$1 - \kappa_s$

Table 2: Singlet rule-set for b.

TKF Structure Tree singlet rules (c)
Sequence c *terminals:* $\{y, z\}$.

lhs	$\rightarrow$	rhs	$P(c)$
L_c	$\rightarrow$	$y\ L_c$	$\kappa_l \pi_l(y)$
	$\mid$	$S_c\ L_c$	$\kappa_l \pi_l(S)$
	$\mid$	ϵ	$1 - \kappa_l$
S_c	$\rightarrow$	$y\ S_c\ z$	$\kappa_s \pi_s(yz)$
	$\mid$	L_c	$1 - \kappa_s$

Table 3: Singlet rule-set for c.

TKF Structure Tree pair rules ($a \xrightarrow{t} b$)
Sequence a *terminals:* $\{u, v\}$. b *Sequence terminals:* $\{w, x\}$.

lhs	$\rightarrow$	rhs	$P(a)$	$P(b\|a)$
L_{ab}	$\rightarrow$	$u\ w\ L_{ab}$	$\kappa_l \pi_l(u)$	$(1-\beta_l)\alpha_l M_l(u,w)$
	$\mid$	$w\ L_{ab}$	1	$\beta_l \pi_l(w)$
	$\mid$	$u\ L_{a*b}$	$\kappa_l \pi_l(u)$	$(1-\beta_l)(1-\alpha_l)$
	$\mid$	$S_{ab}\ L_{ab}$	$\kappa_l \pi_l(S)$	$(1-\beta_l)\alpha_l$
	$\mid$	$S_b\ L_{ab}$	1	$\beta_l \pi_l(S)$
	$\mid$	$S_a\ L_{a*b}$	$\kappa_l \pi_l(S)$	$(1-\beta_l)(1-\alpha_l)$
	$\mid$	ϵ	$1-\kappa_l$	$1-\beta_l$
S_{ab}	$\rightarrow$	$u\ w\ S_{ab}\ x\ v$	$\kappa_s \pi_s(uv)$	$(1-\beta_s)\alpha_s M_s(uv,wx)$
	$\mid$	$w\ S_{ab}\ x$	1	$\beta_s \pi_s(wx)$
	$\mid$	$u\ S_{a*b}\ v$	$\kappa_s \pi_s(uv)$	$(1-\beta_s)(1-\alpha_s)$
	$\mid$	L_{ab}	$1-\kappa_s$	$1-\beta_s$
L_{a*b}	$\rightarrow$	$u\ w\ L_{ab}$	$\kappa_l \pi_l(u)$	$(1-\gamma_l)\alpha_l M_l(u,w)$
	$\mid$	$w\ L_{ab}$	1	$\gamma_l \pi_l(w)$
	$\mid$	$u\ L_{a*b}$	$\kappa_l \pi_l(u)$	$(1-\gamma_l)(1-\alpha_l)$
	$\mid$	$S_{ab}\ L_{ab}$	$\kappa_l \pi_l(S)$	$(1-\gamma_l)\alpha_l$
	$\mid$	$S_b\ L_{ab}$	1	$\gamma_l \pi_l(S)$
	$\mid$	$S_a\ L_{a*b}$	$\kappa_l \pi_l(S)$	$(1-\gamma_l)(1-\alpha_l)$
	$\mid$	ϵ	$1-\kappa_l$	$1-\gamma_l$
S_{a*b}	$\rightarrow$	$u\ w\ S_{ab}\ x\ v$	$\kappa_s \pi_s(uv)$	$(1-\gamma_s)\alpha_s M_s(uv,wx)$
	$\mid$	$w\ S_{ab}\ x$	1	$\gamma_s \pi_s(wx)$
	$\mid$	$u\ S_{a*b}\ v$	$\kappa_s \pi_s(uv)$	$(1-\gamma_s)(1-\alpha_s)$
	$\mid$	L_{ab}	$1-\kappa_s$	$1-\gamma_s$

Table 4: Pair rule-set for $a \xrightarrow{t} b$ branch. Requires Table 1 and Table 2.

TKF Structure Tree pair rules ($b \xrightarrow{t'} c$)
Sequence b terminals: $\{w, x\}$. c Sequence terminals: $\{y, z\}$.

lhs	$\rightarrow$	rhs	$P(b)$	$P(c\|b)$
L_{bc}	$\rightarrow$	$w\ y\ L_{bc}$	$\kappa_l \pi_l(w)$	$(1-\beta'_l)\alpha'_l M'_l(w,y)$
	$\mid$	$y\ L_{bc}$	1	$\beta'_l \pi_l(y)$
	$\mid$	$w\ L_{b*c}$	$\kappa_l \pi_l(w)$	$(1-\beta'_l)(1-\alpha'_l)$
	$\mid$	$S_{bc}\ L_{bc}$	$\kappa_l \pi_l(S)$	$(1-\beta'_l)\alpha'_l$
	$\mid$	$S_c\ L_{bc}$	1	$\beta'_l \pi_l(S)$
	$\mid$	$S_b\ L_{b*c}$	$\kappa_l \pi_l(S)$	$(1-\beta'_l)(1-\alpha'_l)$
	$\mid$	ϵ	$1-\kappa_l$	$1-\beta'_l$
S_{bc}	$\rightarrow$	$w\ y\ S_{bc}\ z\ x$	$\kappa_s \pi_s(wx)$	$(1-\beta'_s)\alpha'_s M'_s(wx,yz)$
	$\mid$	$y\ S_{bc}\ z$	1	$\beta'_s \pi_s(yz)$
	$\mid$	$w\ S_{b*c}\ x$	$\kappa_s \pi_s(wx)$	$(1-\beta'_s)(1-\alpha'_s)$
	$\mid$	L_{bc}	$1-\kappa_s$	$1-\beta'_s$
L_{b*c}	$\rightarrow$	$w\ y\ L_{bc}$	$\kappa_l \pi_l(w)$	$(1-\gamma'_l)\alpha'_l M'_l(w,y)$
	$\mid$	$y\ L_{bc}$	1	$\gamma'_l \pi_l(y)$
	$\mid$	$w\ L_{b*c}$	$\kappa_l \pi_l(w)$	$(1-\gamma'_l)(1-\alpha'_l)$
	$\mid$	$S_{bc}\ L_{bc}$	$\kappa_l \pi_l(S)$	$(1-\gamma'_l)\alpha'_l$
	$\mid$	$S_c\ L_{bc}$	1	$\gamma'_l \pi_l(S)$
	$\mid$	$S_b\ L_{b*c}$	$\kappa_l \pi_l(S)$	$(1-\gamma'_l)(1-\alpha'_l)$
	$\mid$	ϵ	$1-\kappa_l$	$1-\gamma'_l$
S_{b*c}	$\rightarrow$	$w\ y\ S_{bc}\ z\ x$	$\kappa_s \pi_s(wx)$	$(1-\gamma'_s)\alpha'_s M'_s(wx,yz)$
	$\mid$	$y\ S_{bc}\ z$	1	$\gamma'_s \pi_s(yz)$
	$\mid$	$w\ S_{b*c}\ x$	$\kappa_s \pi_s(wx)$	$(1-\gamma'_s)(1-\alpha'_s)$
	$\mid$	L_{bc}	$1-\kappa_s$	$1-\gamma'_s$

Table 5: Pair rule-set for $b \xrightarrow{t'} c$ branch. Requires Table 2 and Table 3.

TKF Structure Tree triplet rules ($a \xrightarrow{t} b \xrightarrow{t'} c$): "$abc$" states (post emission at c)
Sequence a terminals: $\{u, v\}$. Sequence b terminals: $\{w, x\}$. Sequence c terminals: $\{y, z\}$.

lhs	$\rightarrow$	rhs	$P(a)$	$P(b\|a)$	$P(c\|b)$
L_{abc}	$\rightarrow$	$u\ w\ y\ L_{abc}$	$\kappa_l \pi_l(u)$	$(1-\beta_l)\alpha_l M_l(u,w)$	$(1-\beta'_l)\alpha'_l M'_l(w,y)$
	$\mid$	$y\ L_{abc}$	1	1	$\beta'_l \pi_l(y)$
	$\mid$	$w\ y\ L_{abc}$	1	$\beta_l \pi_l(w)$	$(1-\beta'_l)\alpha'_l M'_l(w,y)$
	$\mid$	$w\ L_{ab*c}$	1	$\beta_l \pi_l(w)$	$(1-\beta'_l)(1-\alpha'_l)$
	$\mid$	$u\ w\ L_{ab*c}$	$\kappa_l \pi_l(u)$	$(1-\beta_l)\alpha_l M_l(u,w)$	$(1-\beta'_l)(1-\alpha'_l)$
	$\mid$	$u\ L_{a*bc}$	$\kappa_l \pi_l(u)$	$(1-\beta_l)(1-\alpha_l)$	$1-\beta'_l$
	$\mid$	$S_{abc}\ L_{abc}$	$\kappa_l \pi_l(S)$	$(1-\beta_l)\alpha_l$	$(1-\beta'_l)\alpha'_l$
	$\mid$	$S_c\ L_{abc}$	1	1	$\beta'_l \pi_l(S)$
	$\mid$	$S_{bc}\ L_{abc}$	1	$\beta_l \pi_l(S)$	$(1-\beta'_l)\alpha'_l$
	$\mid$	$S_b\ L_{ab*c}$	1	$\beta_l \pi_l(S)$	$(1-\beta'_l)(1-\alpha'_l)$
	$\mid$	$S_{ab}\ L_{ab*c}$	$\kappa_l \pi_l(S)$	$(1-\beta_l)\alpha_l$	$(1-\beta'_l)(1-\alpha'_l)$
	$\mid$	$S_a\ L_{a*bc}$	$\kappa_l \pi_l(S)$	$(1-\beta_l)(1-\alpha_l)$	$1-\beta'_l$
	$\mid$	ϵ	$1-\kappa_l$	$1-\beta_l$	$1-\beta'_l$
S_{abc}	$\rightarrow$	$u\ w\ y\ S_{abc}\ z\ x\ v$	$\kappa_s \pi_s(uv)$	$(1-\beta_s)\alpha_s M_s(uv,wx)$	$(1-\beta'_s)\alpha'_s M'_s(wx,yz)$
	$\mid$	$y\ S_{abc}\ z$	1	1	$\beta'_s \pi_s(yz)$
	$\mid$	$w\ y\ S_{abc}\ z\ x$	1	$\beta_s \pi_s(wx)$	$(1-\beta'_s)\alpha'_s M'_s(wx,yz)$
	$\mid$	$w\ S_{ab*c}\ x$	1	$\beta_s \pi_s(wx)$	$(1-\beta'_s)(1-\alpha'_s)$
	$\mid$	$u\ w\ S_{ab*c}\ x\ v$	$\kappa_s \pi_s(uv)$	$(1-\beta_s)\alpha_s M_s(uv,wx)$	$(1-\beta'_s)(1-\alpha'_s)$
	$\mid$	$u\ S_{a*bc}\ v$	$\kappa_s \pi_s(uv)$	$(1-\beta_s)(1-\alpha_s)$	$1-\beta'_s$
	$\mid$	L_{abc}	$1-\kappa_s$	$1-\beta_s$	$1-\beta'_s$

Table 6: Triplet rule-set for "abc" states (post emission at c) on $a \xrightarrow{t} b \xrightarrow{t'} c$ tree. Requires Tables 1 through 8.

TKF Structure Tree triplet rules ($a \xrightarrow{t} b \xrightarrow{t'} c$): "$ab * c$" states (post deletion on $b \xrightarrow{t'} c$ branch)
Sequence a terminals: $\{u, v\}$. Sequence b terminals: $\{w, x\}$. Sequence c terminals: $\{y, z\}$.

lhs	$\rightarrow$	rhs	$P(a)$	$P(b\|a)$	$P(c\|b)$
L_{ab*c}	$\rightarrow$	$u\ w\ y\ L_{abc}$	$\kappa_l \pi_l(u)$	$(1-\beta_l)\alpha_l M_l(u,w)$	$(1-\gamma'_l)\alpha'_l M'_l(w,y)$
	$\mid$	$y\ L_{abc}$	1	1	$\gamma'_l \pi_l(y)$
	$\mid$	$w\ y\ L_{abc}$	1	$\beta_l \pi_l(w)$	$(1-\gamma'_l)\alpha'_l M'_l(w,y)$
	$\mid$	$w\ L_{ab*c}$	1	$\beta_l \pi_l(w)$	$(1-\gamma'_l)(1-\alpha'_l)$
	$\mid$	$u\ w\ L_{ab*c}$	$\kappa_l \pi_l(u)$	$(1-\beta_l)\alpha_l M_l(u,w)$	$(1-\gamma'_l)(1-\alpha'_l)$
	$\mid$	$u\ L_{a*bc}$	$\kappa_l \pi_l(u)$	$(1-\beta_l)(1-\alpha_l)$	$1-\gamma'_l$
	$\mid$	$S_{abc}\ L_{abc}$	$\kappa_l \pi_l(S)$	$(1-\beta_l)\alpha_l$	$(1-\gamma'_l)\alpha'_l$
	$\mid$	$S_c\ L_{abc}$	1	1	$\gamma'_l \pi_l(S)$
	$\mid$	$S_{bc}\ L_{abc}$	1	$\beta_l \pi_l(S)$	$(1-\gamma'_l)\alpha'_l$
	$\mid$	$S_b\ L_{ab*c}$	1	$\beta_l \pi_l(S)$	$(1-\gamma'_l)(1-\alpha'_l)$
	$\mid$	$S_{ab}\ L_{ab*c}$	$\kappa_l \pi_l(S)$	$(1-\beta_l)\alpha_l$	$(1-\gamma'_l)(1-\alpha'_l)$
	$\mid$	$S_a\ L_{a*bc}$	$\kappa_l \pi_l(S)$	$(1-\beta_l)(1-\alpha_l)$	$1-\gamma'_l$
	$\mid$	ϵ	$1-\kappa_l$	$1-\beta_l$	$1-\gamma'_l$
S_{ab*c}	$\rightarrow$	$u\ w\ y\ S_{abc}\ z\ x\ v$	$\kappa_s \pi_s(uv)$	$(1-\beta_s)\alpha_s M_s(uv,wx)$	$(1-\gamma'_s)\alpha'_s M'_s(wx,yz)$
	$\mid$	$y\ S_{abc}\ z$	1	1	$\gamma'_s \pi_s(yz)$
	$\mid$	$w\ y\ S_{abc}\ z\ x$	1	$\beta_s \pi_s(wx)$	$(1-\gamma'_s)\alpha'_s M'_s(wx,yz)$
	$\mid$	$w\ S_{ab*c}\ x$	1	$\beta_s \pi_s(wx)$	$(1-\gamma'_s)(1-\alpha'_s)$
	$\mid$	$u\ w\ S_{ab*c}\ x\ v$	$\kappa_s \pi_s(uv)$	$(1-\beta_s)\alpha_s M_s(uv,wx)$	$(1-\gamma'_s)(1-\alpha'_s)$
	$\mid$	$u\ S_{a*bc}\ v$	$\kappa_s \pi_s(uv)$	$(1-\beta_s)(1-\alpha_s)$	$1-\gamma'_s$
	$\mid$	L_{abc}	$1-\kappa_s$	$1-\beta_s$	$1-\gamma'_s$

Table 7: Triplet rule-set for "$ab * c$" states (post deletion on $b \xrightarrow{t'} c$ branch) on $a \xrightarrow{t} b \xrightarrow{t'} c$ tree. Requires Tables 1 through 8.

TKF Structure Tree triplet rules ($a \xrightarrow{t} b \xrightarrow{t'} c$): "$a * bc$" states (post deletion on $a \xrightarrow{t} b$ branch)
Sequence a terminals: $\{u, v\}$. Sequence b terminals: $\{w, x\}$. Sequence c terminals: $\{y, z\}$.

lhs	$\rightarrow$	rhs	$P(a)$	$P(b\|a)$	$P(c\|b)$
L_{a*bc}	$\rightarrow$	$u\ w\ y\ L_{abc}$	$\kappa_l \pi_l(u)$	$(1-\gamma_l)\alpha_l M_l(u,w)$	$\alpha'_l M'_l(w,y)$
	$\mid$	$w\ y\ L_{abc}$	1	$\gamma_l \pi_l(w)$	$\alpha'_l M'_l(w,y)$
	$\mid$	$w\ L_{ab*c}$	1	$\gamma_l \pi_l(w)$	$1-\alpha'_l$
	$\mid$	$u\ w\ L_{ab*c}$	$\kappa_l \pi_l(u)$	$(1-\gamma_l)\alpha_l M_l(u,w)$	$1-\alpha'_l$
	$\mid$	$u\ L_{a*bc}$	$\kappa_l \pi_l(u)$	$(1-\gamma_l)(1-\alpha_l)$	1
	$\mid$	$S_{abc}\ L_{abc}$	$\kappa_l \pi_l(S)$	$(1-\gamma_l)\alpha_l$	α'_l
	$\mid$	$S_{bc}\ L_{abc}$	1	$\gamma_l \pi_l(S)$	α'_l
	$\mid$	$S_b\ L_{ab*c}$	1	$\gamma_l \pi_l(S)$	$1-\alpha'_l$
	$\mid$	$S_{ab}\ L_{ab*c}$	$\kappa_l \pi_l(S)$	$(1-\gamma_l)\alpha_l$	$1-\alpha'_l$
	$\mid$	$S_a\ L_{a*bc}$	$\kappa_l \pi_l(S)$	$(1-\gamma_l)(1-\alpha_l)$	1
	$\mid$	ϵ	$1-\kappa_l$	$1-\gamma_l$	1
S_{a*bc}	$\rightarrow$	$u\ w\ y\ S_{abc}\ z\ x\ v$	$\kappa_s \pi_s(uv)$	$(1-\gamma_s)\alpha_s M_s(uv,wx)$	$\alpha'_s M'_s(wx,yz)$
	$\mid$	$w\ y\ S_{abc}\ z\ x$	1	$\gamma_s \pi_s(wx)$	$\alpha'_s M'_s(wx,yz)$
	$\mid$	$w\ S_{ab*c}\ x$	1	$\gamma_s \pi_s(wx)$	$1-\alpha'_s$
	$\mid$	$u\ w\ S_{ab*c}\ x\ v$	$\kappa_s \pi_s(uv)$	$(1-\gamma_s)\alpha_s M_s(uv,wx)$	$1-\alpha'_s$
	$\mid$	$u\ S_{a*bc}\ v$	$\kappa_s \pi_s(uv)$	$(1-\gamma_s)(1-\alpha_s)$	1
	$\mid$	L_{abc}	$1-\kappa_s$	$1-\gamma_s$	1

Table 8: Triplet rule-set for "$a * bc$" states (post deletion on $a \xrightarrow{t} b$ branch) on $a \xrightarrow{t} b \xrightarrow{t'} c$ tree. Requires Tables 1 through 8.

2 Two-sequence transducer models

We here give formal definitions of two-sequence models and the state machines which generate the factored probability distribution $P(X, Y|\Delta T) = P(X) \cdot P(Y|X, \Delta T)$, where the marginal $P(X)$ is generated by a **singlet transducer** and the conditional $P(Y|X, \Delta T)$ by a **branch transducer**. We refer to the branch transducer as θ, so the conditional distribution is more precisely $P(Y|X, \Delta T, \theta)$.

2.1 Formal Grammars

There is a close relationship between formal grammars and the singlet and branch transducer abstract state machines. By labeling the nonterminals of a regular or stochastic context-free grammar (SCFG) as states of an abstract machine, allowing these states to absorb and emit appropriate terminal symbols,[1], and carefully assigning transition and emission weights, we can create a state machine which generates the same language, with the same distribution of weights, as that produced by the original grammar.[2] We therefore use the terms "nonterminal" and "state" interchangeably and refer to the "parse tree" generated by singlet and branch transducers.

Phrased more precisely, there is an isomorphism between the singlet and branch transducers of two-sequence models and Pair SCFGs. Practically speaking, this means that for every joint distribution $P(X, Y|\Delta T)$ generated by a singlet and branch transducers, there exists a Pair SCFG which generates the same distribution.

[1] The relationship between formal grammars and abstract state machines is more transparent in the Mealy machine view, but the Moore machine view turns out to be more useful for our purposes.

[2] We speak of weights associated with a transition rather than probabilities in order to allow for more general models.

2.2 States, transitions and emissions

A singlet transducer has states of type `Start` and `Insert`. Each state $\phi \in \Phi$ of the branch transducer has type

$$\mathrm{type}(\phi) \in \{\texttt{Start}, \texttt{Insert}, \texttt{Match}, \texttt{Wait}, \texttt{End}\}\,.$$

State typing is as follows:

- A `Start` state begins a branch of the parse tree.
- An `Insert` state emits, but does not absorb, symbols.
- A `Match` state absorbs and (possibly) emits symbols.
- A `Wait` state is a null state which allows a branch transducer to "pause" while it waits for an input symbol from another transducer. In the two-sequence case, the input symbols is emitted by the singlet transducer generating the ancestral sequence.
- An `End` state ends a branch of the parse tree.

The names of state types are similar to the $\{\texttt{Start}, \texttt{Insert}, \texttt{Match}, \texttt{End}\}$ states of the familiar Pair HMM. Deletions are handled as a special case by states of type `Match` which absorb but do not emit symbols pairs. Only states of type `Match` can absorb symbol pairs (x, y).

Bifurcations in the grammar, when considered as emission of nonterminals, can be handled analogously to terminal emission. States $\phi : \mathrm{type}(\phi) \in \{\texttt{Insert}, \texttt{Match}\}$ can emit nonterminal pairs $(c\,d)$, where c or d can be the null symbol, and the emitted pairs $(c\,d)$ can be absorbed by states of type `Match`.

The emission weight of a nonterminal pair $(c\,d)$ from the state $b : \mathrm{type}(b) = \texttt{Match}$, conditional on absorption of a nonterminal pair $(l\,m)$, is

$$e_b(c\,d|l\,m, \theta)\,.$$

The functions $\mathrm{emit}()$ and $\mathrm{absorb}()$ are defined to return the emission or absorption of a particular state,

$$\mathrm{emit}(\phi) := \begin{cases} (x, y) & x, y = \text{terminal or null} \\ (c\, d) & c, d = \text{nonterminal or null.} \end{cases}$$

$$\mathrm{absorb}(\phi) := \begin{cases} (x, y) & x, y = \text{terminal or null} \\ (c\, d) & c, d = \text{nonterminal or null.} \end{cases}$$

We frequently use the notation ${}^{uv}\phi^{xy}$ to indicate that a state of type `Match` absorbs a symbol pair (u, v) and emits a pair (x, y). The notation for bifurcations is slightly different: A bifurcation state which left-emits a nonterminal d and makes a transition to a state ϕ with weight 1 is written as $\mathtt{B}[\mathtt{d}\,\phi]$.

A transition between states a and b of the branch transducer θ has a weight

$$t(a, b|\theta) = t(a \to b|\theta).$$

Terminal emission is handled by states $\phi : \mathrm{type}(\phi) \in \{\mathtt{Match}, \mathtt{Insert}\}$, which emit symbol pairs (x, y), where x or y can be the null symbol. In this paired-emission perspective, left single-terminal emissions x are represented as $(x\,\mathrm{null})$; right single-terminal emissions are handled similarly. The weight of an emission of a terminal pair (x, y) from a state $b : \mathrm{type}(b) = \mathtt{Match}$, conditioned on absorption of a terminal pair (u, v), is

$$e_b(x, y|u, v, \theta)\,.$$

Recall that the emission weights of states of type `Match` are defined conditioned upon the absorbed symbols.

3 Multiple-sequence transducer models

We can use our two-sequence models to construct a model of many sequence related by a guide phylogenetic tree. The guide tree specifies the (conjectured) phylogenetic relationship of all sequences. A singlet transducer, which emits, but does not absorb, symbols, lies at the root of the guide tree and serves as a generative model of the ancestral sequence. A branch transducer represents the evolution of an ancestral sequence into a single descendant sequence, that is, the action of evolution along the single-branch tree (Ancestor $\rightarrow$ Descendant). To represent the evolution of an ancestral sequence into many descendant sequences (whose phylogenetic relationship is specified by the guide tree), we place a branch transducer on each branch of the guide tree to create a multiple-sequence model.

If the branch grammar has no bifurcations and only left or right emissions are allowed, then the language generated is a regular string language and the corresponding jointly normalized abstract state machine is an HMM. In this simplest case our formalism for creating a multiple-sequence model reduces to that given by (2) for combining HMMs on a guide tree.

3.1 The guide tree

The nodes of the tree are labeled $1, \ldots, N$ in the order reached by any preorder depth-first traversal of the tree. The length of each branch $(\mathrm{parent}(m) \rightarrow m)$ is given by the evolutionary time t_m. To specify ancestor-descendant relationships, we introduce notation: $m \rhd n$ ($m \not\rhd n$) means node m is descended from (not descended from) node n, and $m \trianglerighteq n$ ($m \not\trianglerighteq n$) means node m is descended from or identical to (not descended from and not identical to) node n.

3.2 States, transitions and emissions

The multiple-sequence model is formed by the composition/intersection of $(N-1)$ branch transducers such that there is a branch transducer on each branch $(\mathrm{parent}(m) \rightarrow m)$; $m =$

$2, \ldots, N$ of the guide tree and a singlet transducer at the root node. Our framework allows for the placement of different branch transducers, with a unique set of nonterminals (state space) Φ and allowed transitions between states and corresponding weights, on each branch. $\theta^{(m)}$ denotes the branch transducer governing evolution along the branch $(\mathrm{parent}(m) \to m)$ of the guide tree.

States of the multiple-sequence model are represented by as N-dimensional vectors $\boldsymbol{a}$,

$$\boldsymbol{a} = \begin{pmatrix} a_1 \\ \vdots \\ a_N \end{pmatrix} .$$

These states are typed as

$$\mathrm{type}(\boldsymbol{a}) \in \{\texttt{Start}, \texttt{Emit}, \texttt{Bifurcation}, \texttt{Null}, \texttt{End}\} .$$

State typing is as follows:

- A `Start` state begins a branch of the parse tree of one or more of the N sequences on the phylogenetic tree.
- An `Emit` state emits symbols (terminals) to one or more of the N sequences.
- A `Bifurcation` state corresponds to a bifurcation in the parse tree of one or more of the N sequences.
- A `Null` state corresponds to any non-`End` state which represents neither an emission or bifurcation.
- An `End` state ends a branch of the parse tree.

States are typed according to the transition by which they are reachable (details of the typing are given in Section 3.5).

Transitions and emissions of the multiple-sequence model are defined in terms of the transitions and emissions of the branch transducers at each node as well as the singlet transducer at the root node. The weight of a transition $\boldsymbol{a} \rightarrow \boldsymbol{b}$ is therefore

$$t(\boldsymbol{a}, \boldsymbol{b}) = \prod_{m|a_m \neq b_m} t\left(a_m, b_m | \theta^{(m)}\right) , \tag{1}$$

and the weight of an emission $(\boldsymbol{x}\, \boldsymbol{y})$ from a state $\boldsymbol{b}$ is

$$_{\boldsymbol{b}}(\boldsymbol{x}\, \boldsymbol{y}) = \prod_{m} e_{b_m}\left(x_m\, y_m | x_{\mathrm{parent}(m)}\, y_{\mathrm{parent}(m)}, \theta^{(m)}\right) . \tag{2}$$

We frequently will not explicitly write out the conditional dependence on the absorbed terminals $(x_{\mathrm{parent}(m)}\, y_{\mathrm{parent}(m)})$, but the reader should keep in mind that in general emission weights will depend on the absorbed symbols.

3.3 Formal grammars

Analogously to the case with two-sequence models (Section 2.1), there exists a one-to-one mapping between the multiple-sequence models generated by our model-construction algorithm and multi-sequence SCFGs. In other words, given a singlet and branch transducers of a two-sequence model, as well as a guide tree relating the extant sequences, there exists a corresponding multi-sequence SCFG which generates the same joint probability distribution $P(X_1, ..., X_n)$.

3.4 Constructing the state graph

As described in the paper, we need a way to efficiently construct the state graph of the multi-sequence model, where the state graph consists of a list of accessible states and the possible transitions between them. This state graph can be constructed by an uninformed depth-first

search, where at each step of the search we obtain the possible child nodes by applying one of the following possible transitions of the multi-sequence model:

1. **Null transition**: The state of a single branch transducer is updated, with no terminal emission or bifurcation.

2. **Terminal emission**: A state makes a transition to a `Insert` state. The emitted symbol is passed down the guide tree to all descendant branch transducers, which transition to states of type `Match`.

3. **Bifurcation**: A state makes a transition to a special `Insert` state which emits a new branch of the parse tree. The emitted symbol is passed down the guide tree to all descendant branch transducers, which transition to states of type `Match`.

4. **End transition**: The singlet transducer associated with the root sequence can transition to the `End` state, signaling that this parse tree is finished.

Each of these transitions is explained in detail in the following section.

3.5 Allowed transitions

Following (2), we let transitions of the multi-sequence model begin at the active node and cascade down the guide tree as appropriate. Unless defined otherwise, node n is the active node of the multi-sequence model with state $\boldsymbol{a}$,

$$n = n(\boldsymbol{a}) \tag{3}$$

$$= \mathrm{argmax}_m \left\{ \mathrm{type}(a_m) \notin \{\texttt{Wait}, \texttt{End}\} \right\} . \tag{4}$$

Each possible allowed transition is obtained by making a valid change (updating) the state of the singlet or one or more of the branch transducers of the multi-sequence model.

Null Transitions

$$\begin{pmatrix} a_1 \\ \vdots \\ a_N \end{pmatrix} \rightarrow \begin{pmatrix} b_1 \\ \vdots \\ b_N \end{pmatrix}$$

$$\text{type}(\boldsymbol{b}) = \texttt{Null} \tag{5}$$

$$\text{Weight}(\boldsymbol{a} \rightarrow \boldsymbol{b}) = t(\boldsymbol{a}, \boldsymbol{b}) \tag{6}$$

$$= t\left(a_n, b_n | \theta^{(n)}\right). \tag{7}$$

This transition updates the state of the branch transducer at the active node n of the guide tree, leaving the rest unchanged, with no corresponding terminal emission or bifurcation in the grammar. Nodes other than the active node do not change state, $a_m = b_m \, \forall \, m \neq n$, and the only allowable transitions of this form are to states $b_n : \text{type}(b_n) \in \{\texttt{Start}, \texttt{Wait}\}$. Transitions to states of type `Insert` or `Match` result in emissions, and transitions to the end state `End` are handled as a special case.

Terminal Emission

$$\begin{pmatrix} a_1 \\ \vdots \\ a_N \end{pmatrix} \rightarrow \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} \begin{pmatrix} b_1 \\ \vdots \\ b_N \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

$$\text{type}(\boldsymbol{b}) = \texttt{Emit} \tag{8}$$

$$\text{Weight}(\boldsymbol{a} \rightarrow \boldsymbol{x}\,\boldsymbol{b}\,\boldsymbol{y}) = t(\boldsymbol{a}, \boldsymbol{b}) \cdot {}_{\boldsymbol{b}}(\boldsymbol{x}\,\boldsymbol{y}) \tag{9}$$

$$= \left[\prod_{m | a_m \neq b_m} t\left(a_m, b_m | \theta^{(m)}\right) \right] \cdot \left[\prod_m e_{b_m}\left(x_m\, y_m | \theta^{(m)}\right) \right], \tag{10}$$

where we are defining states with no emissions to emit $(\text{null}\,\text{null})$ with weight 1, $e_{b_m}\left(x_m\, y_m | \theta^{(m)}\right) = 1$ if $(x_m\, y_m) = (\text{null}\,\text{null})$ and $\text{emit}(b_m) = \text{null}$.

The active node n makes a transition to a state $b_n : \text{type}(b_n) = \texttt{Insert}$, emitting a terminal symbol pair $\text{emit}(b_n) = (x_n\, y_n)$. This symbol pair is passed down the guide tree to descendant nodes $\{m | m \rhd n, \text{type}(a_m) \neq \texttt{End}\}$, forcing them to make a transition from states of type `Wait` to states of type `Match`, which can absorb terminal pairs (x, y). Qualitatively, this transition and emission could represent the evolution of two paired nucleotides along the subtree rooted at node n of the complete guide tree. If one of $\boldsymbol{x}$ or $\boldsymbol{y}$ is null, then (8) could represent the evolution of a single unpaired nucleotide along the subtree rooted at node n.

Left emission. All terminals $\boldsymbol{y}$ in the transition $\boldsymbol{a} \rightarrow \boldsymbol{x}\,\boldsymbol{b}\,\boldsymbol{y}$ (8) are null.

Nodes $m \not\trianglerighteq n$: $a_m = b_m$.

$(x_m\, y_m) = (\text{GAP}\,\text{null})$.

Node n: $b_n : \text{type}(b_n) = \texttt{Insert}$, $\exists$ a transition $a_n \rightarrow b_n$ in the branch transducer $\theta^{(n)}$.

$(x_n\, y_n) = \text{emit}(b_n)$.

Nodes $m \rhd n$: Either $a_m = b_m$, $\text{emit}(b_{\text{parent}(m)}) = \text{null}$ or $\text{type}(a_m) = \texttt{End}$

or $\text{emit}(b_{\text{parent}(m)}) = \text{absorb}(b_m) = (x_{\text{parent}(m)}\,\text{null})$.

$$(x_m\, y_m) = \begin{cases} (\text{GAP}\,\text{null}) & \text{emit}(b_m) = \text{null} \\ \text{emit}(b_m) & \text{emit}(b_m) \neq \text{null} \end{cases}$$

Right emission. All terminals $\boldsymbol{x}$ in the transition $\boldsymbol{a} \rightarrow \boldsymbol{x}\,\boldsymbol{b}\,\boldsymbol{y}$ (8) are null.

Nodes $m \not\trianglerighteq n$: $a_m = b_m$.

$(x_m\, y_m) = (\text{null}\, \text{GAP})$.

Node n: $b_n :$ $\text{type}(b_n) = \texttt{Insert}$, $\exists$ a transition $a_n \to b_n$ in the branch transducer $\theta^{(n)}$.

$(x_n\, y_n) = \text{emit}(b_n)$.

Nodes $m \triangleright n$: Either $a_m = b_m$, $\text{emit}(b_{\text{parent}(m)}) = \text{null}$ or $\text{type}(a_m) = \texttt{End}$

or $\text{emit}(b_{\text{parent}(m)}) = \text{absorb}(b_m) = (\text{null}\, y_{\text{parent}(m)})$.

$$(x_m\, y_m) = \begin{cases} (\text{null}\, \text{GAP}) & \text{emit}(b_m) = \text{null} \\ \text{emit}(b_m) & \text{emit}(b_m) \neq \text{null}. \end{cases}$$

Paired emission. There is at least one non-null terminal in both $\boldsymbol{x}$ and $\boldsymbol{y}$ in the transition $\boldsymbol{a} \to \boldsymbol{x}\, \boldsymbol{b}\, \boldsymbol{y}$ (8).

Nodes $m \not\trianglerighteq n$: $a_m = b_m$.

$(x_m\, y_m) = (\text{GAP}\, \text{GAP})$.

Node n: $b_n :$ $\text{type}(b_n) = \texttt{Insert}$, $\exists$ a transition $a_n \to b_n$ in the branch transducer $\theta^{(n)}$.

$(x_n\, y_n) = \text{emit}(b_n)$.

Nodes $m \triangleright n$: Either $a_m = b_m$, $\text{emit}(b_{\text{parent}(m)}) = \text{null}$ or $\text{type}(a_m) = \texttt{End}$

or $\text{emit}(b_{\text{parent}(m)}) = \text{absorb}(b_m) = (x_{\text{parent}(m)}\, y_{\text{parent}(m)})$.

$$(x_m\, y_m) = \begin{cases} (\text{GAP}\, \text{GAP}) & \text{emit}(b_m) = \text{null} \\ \text{emit}(b_m) & \text{emit}(b_m) \neq \text{null}. \end{cases}$$

Bifurcations

$$\begin{pmatrix} a_1 \\ \vdots \\ a_N \end{pmatrix} \to \begin{pmatrix} c_1 \\ \vdots \\ c_N \end{pmatrix} \begin{pmatrix} b_1 \\ \vdots \\ b_N \end{pmatrix} \begin{pmatrix} d_1 \\ \vdots \\ d_N \end{pmatrix}$$

$$\text{type}(\boldsymbol{b}) = \texttt{Bifurcation} \tag{11}$$

$$\text{Weight}(\boldsymbol{a} \rightarrow \boldsymbol{c}\,\boldsymbol{b}\,\boldsymbol{d}) = t(\boldsymbol{a}, \boldsymbol{b}) \cdot {}_{\boldsymbol{b}}(\boldsymbol{c}\,\boldsymbol{d}) \tag{12}$$

$$= \left[\prod_{m|\, a_m \neq b_m} t\left(a_m, b_m | \theta^{(m)}\right) \right] \cdot \left[\prod_m e_{b_m}\left(c_m\, d_m | \theta^{(m)}\right) \right]. \tag{13}$$

where we are defining the emission weight of the `End` nonterminal to be 1, $e_{b_m}\left(c_m\, d_m | \theta^{(m)}\right) = 1$ if $c_m = \texttt{End}$ or $d_m = \texttt{End}$.

Bifurcations are handled similarly to terminal emission. The active node n can undergo a bifurcation by making a transition $a_n \rightarrow b_n$, $b_n : \text{type}(b_n) = \texttt{Insert}$, emitting a pair of nonterminals $\text{emit}(b_n) = (c_n\, d_n)$. Descendant nodes $\{m | m \rhd n, \text{type}(a_m) \neq \texttt{End}\}$ are forced to make a transition from states of type `Wait` to states of type `Match` which can absorb nonterminal pairs $(c\, d)$. All emissions are pairwise, so left and right bifurcations are represented as pairs $(\boldsymbol{c}\,\boldsymbol{d})$ where either $\boldsymbol{c}$ or $\boldsymbol{d}$ is null. If $\boldsymbol{d}$ is null, then (11) could represent the insertion and subsequent evolution of a new RNA stem-loop structure.

Left bifurcation. All nonterminals $\boldsymbol{d}$ in the transition $\boldsymbol{a} \rightarrow \boldsymbol{c}\,\boldsymbol{b}\,\boldsymbol{d}$ (11) are null. The nonterminals $\boldsymbol{c}$ are the "new" states (for example, corresponding to a newly formed stem); the nonterminals $\boldsymbol{b}$ are the states which will generate the (evolved) ancestral sequence.

Nodes $m \not\rhd n$: $a_m = b_m$.

$(c_m\, d_m) = (\texttt{End}\, \text{null})$.

Node n: $b_n : \text{type}(b_n) = \texttt{Insert}$, $\exists$ a transition $a_n \rightarrow b_n$ in the branch transducer $\theta^{(n)}$.

$(c_n\, d_n) = \text{emit}(b_n)$.

Nodes $m \rhd n$: Either $a_m = b_m$, $\text{emit}(b_{\text{parent}(m)}) = \text{null}$ or $\text{type}(a_m) = \texttt{End}$

or $\text{emit}(b_{\text{parent}(m)}) = \text{absorb}(b_m)$

$(c_m\, d_m) = \text{emit}(b_m)$.

Right bifurcation. All nonterminals $\boldsymbol{c}$ in the transition $\boldsymbol{a} \rightarrow \boldsymbol{c}\,\boldsymbol{b}\,\boldsymbol{d}$ (11) are null. The nonterminals $\boldsymbol{d}$ are the "new" states; the nonterminals $\boldsymbol{b}$ are the states which will generate the (evolved) ancestral sequence.

Nodes $m \not\trianglerighteq n$: $a_m = b_m$.

$(c_m\, d_m) = (\text{null}\, \texttt{End})$.

Node n: $b_n : \text{type}(b_n) = \texttt{Insert}$, $\exists$ a transition $a_n \rightarrow b_n$ in the branch transducer $\theta^{(n)}$.

$(c_n\, d_n) = \text{emit}(b_n)$.

Nodes $m \triangleright n$: Either $a_m = b_m$, $\text{emit}(b_{\text{parent}(m)}) = \text{null}$ or $\text{type}(a_m) = \texttt{End}$

or $\text{emit}(b_{\text{parent}(m)}) = \text{absorb}(b_m)$

$(c_m\, d_m) = \text{emit}(b_m)$.

Paired bifurcation There is at least one non-null nonterminal in both $\boldsymbol{c}$ and $\boldsymbol{d}$ in the transition $\boldsymbol{a} \rightarrow \boldsymbol{c}\,\boldsymbol{b}\,\boldsymbol{d}$ (11). The nonterminals $\boldsymbol{c}$ and $\boldsymbol{d}$ are both "new" states; the nonterminals $\boldsymbol{b}$ are the states which will generate the (evolved) ancestral sequence.

Nodes $m \not\trianglerighteq n$: $a_m = b_m$.

$(c_m\, d_m) = (\texttt{End}\, \texttt{End})$.

Node n: $b_n : \text{type}(b_n) = \texttt{Insert}$, $\exists$ a transition $a_n \rightarrow b_n$ in the branch transducer $\theta^{(n)}$.

$(c_n\, d_n) = \text{emit}(b_n)$.

Nodes $m \triangleright n$: Either $a_m = b_m$, $\text{emit}(b_{\text{parent}(m)}) = \text{null}$ or $\text{type}(a_m) = \texttt{End}$

or $\text{emit}(b_{\text{parent}(m)}) = \text{absorb}(b_m)$

$(c_m\, d_m) = \text{emit}(b_m)$.

This paired-bifurcation is included for completeness–for example, it could be used to model symmetric loops–but it increases the complexity of grammar parsing.

Transition to `End`

$$\begin{pmatrix} a_1 \\ \vdots \\ a_N \end{pmatrix} \rightarrow \texttt{End}$$

$$\text{Weight}(\boldsymbol{a} \rightarrow \texttt{End}) = t(\boldsymbol{a}, \texttt{End}) \tag{14}$$

$$= \prod_{m|\,\text{type}(a_m)\neq\texttt{End}} t\left(a_m, \texttt{End}|\theta^{(m)}\right). \tag{15}$$

The singlet transducer associated with the highest active ancestral node,

$$\hat{n} = \text{argmin}_m \left\{\text{type}(a_m) \in \{\texttt{Start}, \texttt{Insert}\}\right\} \tag{16}$$

can make a transition to the state `End`, forcing the entire multi-sequence model to transition to `End`.[3] If the branch transducer does not permit inserted bifurcations then $\hat{n} = 1$ always, but this is generically not true for a more general grammar (for example, see the TKF91 Structure Tree model).

We require:

Nodes $m \not\trianglerighteq \hat{n}$: $a_m = \texttt{End}$.

Node $\hat{n}$: $\text{type}(a_{\hat{n}}) \in \{\texttt{Start}, \texttt{Insert}\}$

$\exists$ a transition $a_{\hat{n}} \rightarrow \texttt{End}$.

Nodes $m \triangleright \hat{n}$: $\text{type}(a_m) = \texttt{Wait}$

$\exists$ a transition $a_m \rightarrow \texttt{End}$.

In many probabilistic models, the transition from a state of type `Wait` to the `End` state has weight 1 conditional on absorbing the `End` symbol (called ϵ in formal grammar theory), but we here allow for a more general contribution to the total weight $\mathbb{F}$ of the transition.

[3] The node $\hat{n}$ which initiates the transition to `End` is the root of the greatest active subtree of the whole guide tree (called such because $a_m = \texttt{End}\,\forall\, m \not\trianglerighteq \hat{n}$).

$\hat{n}$ and $\mathrm{Weight}(\boldsymbol{a} \rightarrow \texttt{End})$ are so defined in order to ensure that there exists a direct path along which the end symbol can be passed down the tree. The grammar should be designed such that if the singlet transducer at node $\hat{n}$ can make a transition to `End`, then so can all machines at $\{m|m \trianglerighteq \hat{n}\}$. The TKF Structure Tree model (Section 5) satisfies this condition. A more general approach is probably possible, but it involves summing over paths $a' : a_m \rightarrow a' \rightarrow \texttt{End}$, handling possible bifurcations in these paths, etc.

4 Inference with multi-sequence SCFGs

The model construction algorithm described above creates a composed state machine which generates the joint distribution $P(X, Y, Z, W)$. As described in Section 3.3, there exists a corresponding multi-sequence SCFG which generates the same probability distribution. It follows that we can perform inference on our composed model with the well-known Sankoff algorithms.

As described in the paper, we can use the CYK algorithm directly, but the Inside and Outside algorithms fail for generic multi-sequence models due to the presence of looping `Null` transitions. We here give algorithms for inference in the case of no looping `Null` transitions and explain how to modify them to create **loopy DP** iterative algorithms for our models.

We here give versions for the model of three extant sequences show in Figure 1; the generalization to N sequences is straightforward.

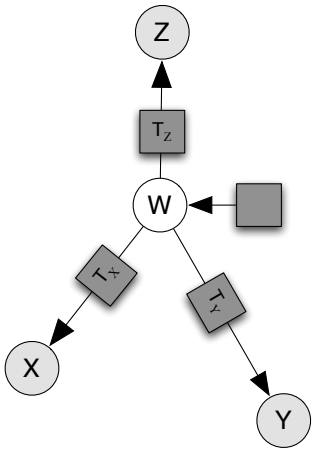


Figure 1: A star phylogeny with three (extant) leaf sequences. An ancestral sequence W evolves into three descendant sequences X, Y and Z. A singlet transducer (the horizontal gray box) emit ancestral sequence and structure and three branch transducers (the gray boxes labeled ΔT_X, ΔT_Y and ΔT_Z) mutate it according to the specified multi-sequence model. Gray nodes correspond to observed data and white nodes unobserved data. If the branch transducers are time-reversible, then this star phylogeny with three leaves is the neighborhood of any interior node in a (binary) phylogenetic tree, from which it follows that evaluating the likelihood function on this star phylogeny is sufficient for a sampling algorithm on an arbitrary phylogeny.

4.1 Notation

The 3 (ungapped) sequences to be aligned are $x^{(1)}$, $x^{(2)}$ and $x^{(3)}$ (we refer to them by an index rather than as X, Y and Z for notational convenience). The i^{th} symbol of sequence $x^{(1)}$ is written $x_i^{(1)}$ and the length of the same sequence is $L_1 = |x^{(1)}|$.

4.2 Fold Envelopes

We use the fold envelope concept (3; 4) to constrain the set of structures our algorithms consider. A fold envelope $\mathscr{F}$ for a sequence x is a set of coordinate pairs satisfying

$$\mathscr{F} \subset \{(i,j) : 0 \leq i \leq j \leq L\} \ . \tag{17}$$

We consider a subsequence $x_{i+1} \ldots x_j$ only if the corresponding coordinate pair $(i,j) \in \mathscr{F}$. The unconstrained fold envelope has set equality in (17).

We use two orderings for sequences in the fold envelopes. An inside-outside ordering is used for the iteration in the Inside algorithm: Subsequences are ordered such that each successive subsequence contains all previous subsequences in the fold envelope. More precisely, subsequences in $\mathscr{F}$ are sorted in the same order as coordinate pairs (i,j) are generated by the iteration $\{\,\mathrm{for}\, i = L \,\mathrm{to}\, 0 \ \{\mathrm{for}\, j = i \,\mathrm{to}\, L\}\,\}$.

The Outside algorithm uses an outside-inside ordering, where subsequences starting at a fixed i are reverse-sorted with respect to the inside-outside ordering described above. Subsequences in $\mathscr{F}$ are then sorted in the same order as coordinate pairs (i,j) are generated by the iteration $\{\,\mathrm{for}\, i = 0 \,\mathrm{to}\, L \ \{\mathrm{for}\, j = L \,\mathrm{to}\, i\}\,\}$.

We frequently refer to subsequences by their index in the fold envelope. The m^{th} subsequence in $\mathscr{F}$ is labeled m and corresponds to the coordinate pair (i_m, j_m). The index of a pair $(i,j) \in \mathscr{F}$ is $n[i,j]$.

Inward and outward emission connections, specifying which subsequences are reachable

from a given subsequence and state, are defined as

$$c_{in}\left(\boldsymbol{b}; m\right) = n\left[i_m + \Delta_{\boldsymbol{b}}^L, j_m - \Delta_{\boldsymbol{b}}^R\right] \tag{18}$$

$$c_{out}\left(\boldsymbol{b}; m\right) = n\left[i_m - \Delta_{\boldsymbol{b}}^L, j_m + \Delta_{\boldsymbol{b}}^R\right] \tag{19}$$

The emission connection is undefined if the corresponding subsequence is not in the fold envelope.[4] Inward, outward-left and outward-right bifurcation connections are defined as

$$b_{in}(n) = \{(n_L, n_R) : i_{n_L} = i_n,\, j_{n_L} = i_{n_R},\, j_{n_R} = j_n\} \tag{20}$$

$$b_{out,L}(n) = \{(n_O, n_L) : i_{n_L} = i_{n_O},\, j_{n_L} = i_n,\, j_n = j_{n_O}\} \tag{21}$$

$$b_{out,R}(n) = \{(n_O, n_R) : i_n = i_{n_O},\, j_n = i_{n_R},\, j_{n_R} = j_{n_O}\} \tag{22}$$

We generally write out explicit subsequence coordinate pairs (i, j) when their usage will make mathematical formulae clearer and fold-envelope labels n when writing pseudocode. An implementation will rely on such an iterator over subsequences in the fold envelope.

4.3 Inside

The inside probability $\alpha_{\boldsymbol{a}}(i_1, j_1, i_2, j_2, i_3, j_3)$ is the summed probability of subsequences $x_{i_1+1}^{(1)} \ldots x_{j_1}^{(1)}$, $x_{i_2+1}^{(2)} \ldots x_{j_2}^{(2)}$, $x_{i_3+1}^{(3)} \ldots x_{j_3}^{(3)}$ under all paths through the model which are rooted in state $\boldsymbol{a}$. Algorithm 1 gives pseudocode for the fold-envelope version of the Inside algorithm. The subroutines $\mathrm{calcTransEmitProb}()$, $\mathrm{calcLBifurcProb}()$ and $\mathrm{calcRBifurcProb}()$ used in the Inside algorithm are defined below.

The transition and emission probability $\mathrm{calcTransEmitProb}(\boldsymbol{a}; \cdot)$ can be calculated recur-

[4] We defined emission connections slightly differently than do (3).

Input: sequences $x^{(1)}, x^{(2)}, x^{(3)}$
initialization;
foreach $n^{(1)} \in \mathscr{F}^{(1)}$ **do** `/* inside-outside sorted */`
  foreach $n^{(2)} \in \mathscr{F}^{(2)}$ **do** `/* inside-outside sorted */`
    foreach $n^{(3)} \in \mathscr{F}^{(3)}$ **do** `/* inside-outside sorted */`
      foreach *state* $\boldsymbol{a}$ **do**
        bifurcProb $\leftarrow 0$;
        foreach $\left(n_L^{(1)}, n_R^{(1)}\right) \in b_{in}\left(n^{(1)}\right)$ **do**
          foreach $\left(n_L^{(2)}, n_R^{(2)}\right) \in b_{in}\left(n^{(2)}\right)$ **do**
            foreach $\left(n_L^{(3)}, n_R^{(3)}\right) \in b_{in}\left(n^{(3)}\right)$ **do**
              bifurcProb += `calcLBifurcProb` $(\boldsymbol{a}; \cdot)$;
              bifurcProb += `calcRBifurcProb` $(\boldsymbol{a}; \cdot)$;
            end
          end
        end
        $\alpha_{\boldsymbol{a}}\left(n^{(1)}, n^{(2)}, n^{(3)}\right) \leftarrow$ `calcTransEmitProb` $\left(\boldsymbol{a}; n^{(1)}, n^{(2)}, n^{(3)}\right)$;
        $\alpha_{\boldsymbol{a}}\left(n^{(1)}, n^{(2)}, n^{(3)}\right)$ += bifurcProb ;
        store $\alpha_{\boldsymbol{a}}\left(n^{(1)}, n^{(2)}, n^{(3)}\right)$;
      end
    end
  end
end
return $\alpha_{\boldsymbol{a}}\left(n[0, L_1], n[0, L_2], n[0, L_3]\right)$;

Algorithm 1: The constrained Inside algorithm for three sequences $x^{(1)}, x^{(2)}, x^{(3)}$. States $\boldsymbol{a}$ in the iteration over states are sorted in Inside fill order with `Emit` states first, then `Null` states in reverse topological order.

sively in the unconstrained case as

$$\sum_{y_1 \in \{x^{(1)}_{i_1+1}, \text{null}\}} \sum_{z_1 \in \{x^{(1)}_{j_1}, \text{null}\}} \sum_{y_2 \in \{x^{(2)}_{i_2+1}, \text{null}\}} \sum_{z_2 \in \{x^{(2)}_{j_2}, \text{null}\}} \sum_{y_3 \in \{x^{(3)}_{i_3+1}, \text{null}\}} \sum_{z_3 \in \{x^{(3)}_{j_3}, \text{null}\}}$$
$$\left[\sum_{\boldsymbol{b} \mid \exists\, \boldsymbol{a} \to \boldsymbol{y}\boldsymbol{b}\boldsymbol{z}} \text{Weight}(\boldsymbol{a} \to \boldsymbol{y}\,\boldsymbol{b}\,\boldsymbol{z}) \alpha_{\boldsymbol{b}}(i_1 + |y_1|, j_1 - |z_1|, i_2 + |y_2|, j_2 - |z_2|, i_3 + |y_3|, j_3 - |z_3|) \right].$$

Null transitions are caught in the sum when the terminals $\boldsymbol{y} = \boldsymbol{z} = \text{null}$. The constrained

case is handled differently: The multiple sums are replaced by a single iteration over states $\boldsymbol{b}$ which connect subsequences $n^{(1)}, n^{(2)}, n^{(3)}$ to others in the fold envelopes. Pseudocode for the constrained calculation is given in Algorithm 2.

Input: state $\boldsymbol{a}, n^{(1)}, n^{(2)}, n^{(3)}$, intermediate Inside matrix α
emitProb $\leftarrow 0$;
foreach $\boldsymbol{b} : \exists\, \boldsymbol{a} \rightarrow \boldsymbol{b}$ **do**
  emitProb += `Weight` $(\boldsymbol{a} \rightarrow \boldsymbol{b})\ \alpha_{\boldsymbol{b}}\left(n^{(1)}, n^{(2)}, n^{(3)}\right)$;
end
foreach $\boldsymbol{b} : \exists\, \boldsymbol{a} \rightarrow \boldsymbol{y}\,\boldsymbol{b}\,\boldsymbol{z}$ **do**
  if $c_{in}\left(\boldsymbol{b}; n^{(1)}\right) \notin \mathscr{F}^{(1)}$ *or* $c_{in}\left(\boldsymbol{b}; n^{(2)}\right) \notin \mathscr{F}^{(2)}$ *or* $c_{in}\left(\boldsymbol{b}; n^{(3)}\right) \notin \mathscr{F}^{(3)}$ **then** next;
  emitProb += `Weight` $(\boldsymbol{a} \rightarrow \boldsymbol{y}\,\boldsymbol{b}\,\boldsymbol{z})\ \alpha_{\boldsymbol{b}}\left(c_{in}\left(\boldsymbol{b}; n^{(1)}\right), c_{in}\left(\boldsymbol{b}; n^{(2)}\right), c_{in}\left(\boldsymbol{b}; n^{(3)}\right)\right)$;
end
return emitProb;

Algorithm 2: Subroutine $\mathrm{calcTransEmitProb}()$ for the Inside algorithm.

The left-bifurcation probability $\mathrm{calcLBifurcProb}\left(\boldsymbol{a}; n_L^{(1)}, n_R^{(1)}, n_L^{(2)}, n_R^{(2)}, n_L^{(3)}, n_R^{(3)}\right)$ is

$$\sum_{\boldsymbol{b}|\,\exists\,\boldsymbol{a}\rightarrow\boldsymbol{c}\boldsymbol{b}} \mathrm{Weight}(\boldsymbol{a} \rightarrow \boldsymbol{c}\,\boldsymbol{b})\, \alpha_{\boldsymbol{c}}\left(n_L^{(1)}, n_L^{(2)}, n_L^{(3)}\right)\ \alpha_{\boldsymbol{b}}\left(n_R^{(1)}, n_R^{(2)}, n_R^{(3)}\right)$$

and the right-bifurcation probability $\mathrm{calcRBifurcProb}\left(\boldsymbol{a}; n_L^{(1)}, n_R^{(1)}, n_L^{(2)}, n_R^{(2)}, n_L^{(3)}, n_R^{(3)}\right)$ is

$$\sum_{\boldsymbol{b}|\,\exists\,\boldsymbol{a}\rightarrow\boldsymbol{b}\boldsymbol{d}} \mathrm{Weight}(\boldsymbol{a} \rightarrow \boldsymbol{b}\,\boldsymbol{d})\, \alpha_{\boldsymbol{b}}\left(n_L^{(1)}, n_L^{(2)}, n_L^{(3)}\right)\ \alpha_{\boldsymbol{d}}\left(n_R^{(1)}, n_R^{(2)}, n_R^{(3)}\right).$$

The boundary condition gives the probability of a subsequence of length 0,

$$\alpha_{\boldsymbol{a}}(j_1, j_1, j_2, j_2, j_3, j_3) = t(\boldsymbol{a}, \texttt{End}) + \sum_{\boldsymbol{b}|\,\exists\,\boldsymbol{a}\rightarrow\boldsymbol{b}} \mathrm{Weight}(\boldsymbol{a} \rightarrow \boldsymbol{b})\, \alpha_{\boldsymbol{a}}(j_1, j_1, j_2, j_2, j_3, j_3)\,, \quad (23)$$

for $0 \le j_1 \le L_1, 0 \le j_2 \le L_2, 0 \le j_3 \le L_3$. We are assuming that there are no cycles of `Null` states as well as no bifurcations which can result in no emissions. The termination condition is

$$P\left(x^{(1)}, x^{(2)}, x^{(3)}\right) = \alpha_{\mathrm{Start}}(0, L_1, 0, L_2, 0, L_3)\,,$$

where Start is the unique start state of the model.

We write the DP algorithms using the $\mathrm{Weight}(\cdot)$ notation in order to preserve generality: In our formalism, a transition $a \rightarrow x\, b$ has $\mathrm{emit}(b) = x$, but the more common convention is to have $\mathrm{emit}(a) = x$. The difference will show up only in the value assigned to $\mathrm{Weight}(a \rightarrow x\, b)$, so our DP algorithms can be used in both cases.

4.4 CYK

The CYK algorithm can be obtained from the Inside algorithm by replacing sums over paths through the model (or equivalently, parses) with the $\max()$ operation (e.g., in (23), $\sum_{\boldsymbol{b}|\, \exists\, \boldsymbol{a} \rightarrow \boldsymbol{b}}$ will be replaced by $\max_{\boldsymbol{b}|\, \exists\, \boldsymbol{a} \rightarrow \boldsymbol{b}}$). The CYK probabilities for indices $(i_1, j_1, i_2, j_2, i_3, j_3)$ then represent the probability of the most likely path through the model for subsequences $x^{(1)}_{i_1+1} \ldots x^{(1)}_{j_1}$, $x^{(2)}_{i_2+1} \ldots x^{(2)}_{j_2}$, $x^{(3)}_{i_3+1} \ldots x^{(3)}_{j_3}$.

4.5 Outside

The outside probability $\beta_{\boldsymbol{b}}(i_1, j_1, i_2, j_2, i_3, j_3)$ is the summed probability of the sequences $x^{(1)}$, $x^{(2)}$, $x^{(3)}$ under all paths through the model which are rooted in the start state of the model, excluding all paths for the subsequences $x^{(1)}_{i_1+1} \ldots x^{(1)}_{j_1}$, $x^{(2)}_{i_2+1} \ldots x^{(2)}_{j_2}$, $x^{(3)}_{i_3+1} \ldots x^{(3)}_{j_3}$ which are rooted in the state $\boldsymbol{b}$. Algorithm 3 gives pseudocode for the fold-envelope version of the Outside algorithm. The subroutines $\mathrm{calcTransEmitProb}()$, $\mathrm{calcLBifurcProb}()$ and $\mathrm{calcRBifurcProb}()$ used in the Outside algorithm are defined below.

The transition and emission probability $\mathrm{calcTransEmitProb}()$ can be calculated recursively

Input: sequences $x^{(1)}, x^{(2)}, x^{(3)}$, CYK matrix α
initialization;
foreach $n^{(1)} \in \mathscr{F}^{(1)}$ **do** `/* outside-inside sorted */`
  foreach $n^{(2)} \in \mathscr{F}^{(2)}$ **do** `/* outside-inside sorted */`
    foreach $n^{(3)} \in \mathscr{F}^{(3)}$ **do** `/* outside-inside sorted */`
      foreach *state* $\boldsymbol{b}$ **do**
        bifurcProb $\leftarrow 0$;
        foreach $\left(n_O^{(1)}, n_L^{(1)}\right) \in b_{out,L}\left(n^{(1)}\right)$ **do**
          foreach $\left(n_O^{(2)}, n_L^{(2)}\right) \in b_{out,L}\left(n^{(2)}\right)$ **do**
            foreach $\left(n_O^{(3)}, n_L^{(3)}\right) \in b_{out,L}\left(n^{(3)}\right)$ **do**
              bifurcProb += `calcLBifurcProb` $(\boldsymbol{b}; \cdot)$;
            end
          end
        end
        foreach $\left(n_O^{(1)}, n_R^{(1)}\right) \in b_{out,R}\left(n^{(1)}\right)$ **do**
          foreach $\left(n_O^{(2)}, n_R^{(2)}\right) \in b_{out,R}\left(n^{(2)}\right)$ **do**
            foreach $\left(n_O^{(3)}, n_R^{(3)}\right) \in b_{out,R}\left(n^{(3)}\right)$ **do**
              bifurcProb += `calcRBifurcProb` $(\boldsymbol{b}; \cdot)$;
            end
          end
        end
        $\beta_{\boldsymbol{b}}\left(n^{(1)}, n^{(2)}, n^{(3)}\right) \leftarrow$ `calcTransEmitProb` $\left(\boldsymbol{b}; n^{(1)}, n^{(2)}, n^{(3)}\right)$;
        $\beta_{\boldsymbol{b}}\left(n^{(1)}, n^{(2)}, n^{(3)}\right)$ += bifurcProb ;
        store $\beta_{\boldsymbol{b}}\left(n^{(1)}, n^{(2)}, n^{(3)}\right)$;
      end
    end
  end
end

Algorithm 3: The constrained Outside algorithm for three sequences $x^{(1)}, x^{(2)}, x^{(3)}$. States $\boldsymbol{a}$ in the iteration over states are sorted in Outside fill order with `Emit` states first, then `Null` states in topological order.

in the unconstrained case as

$$\sum_{y_1 \in \{x_{i_1}^{(1)}, \text{null}\}} \sum_{z_1 \in \{x_{j_1+1}^{(1)}, \text{null}\}} \sum_{y_2 \in \{x_{i_2}^{(2)}, \text{null}\}} \sum_{z_2 \in \{x_{j_2+1}^{(2)}, \text{null}\}} \sum_{y_3 \in \{x_{i_3}^{(3)}, \text{null}\}} \sum_{z_3 \in \{x_{j_3+1}^{(3)}, \text{null}\}}$$
$$\left[\sum_{\boldsymbol{a} \mid \exists\, \boldsymbol{a} \to \boldsymbol{y}\boldsymbol{b}\boldsymbol{z}} \text{Weight}(\boldsymbol{a} \to \boldsymbol{y}\,\boldsymbol{b}\,\boldsymbol{z})\, \beta_{\boldsymbol{a}}(i_1 - |y_1|, j_1 + |z_1|, i_2 - |y_2|, j_2 + |z_2|, i_3 - |y_3|, j_3 + |z_3|) \right].$$

As with the Inside algorithm, an efficient implementation of the fold-envelope constraints requires a different treatment, given in Algorithm 4.

The left-bifurcation probability $\mathrm{calcLBifurcProb}\left(\boldsymbol{b}; n_O^{(1)}, n_L^{(1)}, n_O^{(2)}, n_L^{(2)}, n_O^{(3)}, n_L^{(3)}\right)$ is

$$\sum_{\boldsymbol{a} \mid \exists\, \boldsymbol{a} \to \boldsymbol{c}\boldsymbol{b}} \mathrm{Weight}(\boldsymbol{a} \to \boldsymbol{c}\,\boldsymbol{b})\, \beta_{\boldsymbol{a}}\left(n_O^{(1)}, n_O^{(2)}, n_O^{(3)}\right)\, \alpha_{\boldsymbol{c}}\left(n_L^{(1)}, n_L^{(2)}, n_L^{(3)}\right)$$

and the right-bifurcation probability $\mathrm{calcRBifurcProb}\left(\boldsymbol{b}; n_O^{(1)}, n_R^{(1)}, n_O^{(2)}, n_R^{(2)}, n_O^{(3)}, n_R^{(3)}\right)$ is

$$\sum_{\boldsymbol{a} \mid \exists\, \boldsymbol{a} \to \boldsymbol{b}\boldsymbol{d}} \mathrm{Weight}(\boldsymbol{a} \to \boldsymbol{b}\,\boldsymbol{d})\, \beta_{\boldsymbol{a}}\left(n_O^{(1)}, n_O^{(2)}, n_O^{(3)}\right)\, \alpha_{\boldsymbol{d}}\left(n_R^{(1)}, n_R^{(2)}, n_R^{(3)}\right)$$

Input: state $\boldsymbol{b}, n^{(1)}, n^{(2)}, n^{(3)}$, intermediate Outside matrix β
emitProb $\leftarrow 0$;
foreach $\boldsymbol{a} : \exists\, \boldsymbol{a} \to \boldsymbol{b}$ **do**
  emitProb += `Weight` $(\boldsymbol{a} \to \boldsymbol{b})\ \beta_{\boldsymbol{a}}\left(n^{(1)}, n^{(2)}, n^{(3)}\right)$;
end
foreach $\boldsymbol{a} : \exists\, \boldsymbol{a} \to \boldsymbol{y}\,\boldsymbol{b}\,\boldsymbol{z}$ **do**
  if $c_{out}\left(\boldsymbol{b}; n^{(1)}\right) \notin \mathscr{F}^{(1)}$ *or* $c_{out}\left(\boldsymbol{b}; n^{(2)}\right) \notin \mathscr{F}^{(2)}$ *or* $c_{out}\left(\boldsymbol{b}; n^{(3)}\right) \notin \mathscr{F}^{(3)}$ **then** next;
  emitProb += `Weight` $(\boldsymbol{a} \to \boldsymbol{y}\,\boldsymbol{b}\,\boldsymbol{z})\ \beta_{\boldsymbol{a}}\left(c_{out}\left(\boldsymbol{b}; n^{(1)}\right), c_{out}\left(\boldsymbol{b}; n^{(2)}\right), c_{out}\left(\boldsymbol{b}; n^{(3)}\right)\right)$;
end
return emitProb;

Algorithm 4: Subroutine $\mathrm{calcTransEmitProb}()$ for the Outside algorithm.

The boundary condition is just

$$\beta_{\mathrm{Start}}(0, L_1, 0, L_2, 0, L_3) = 1\,. \tag{24}$$

4.6 Loopy DP

The standard Inside and Outside algorithms fail on grammars with 1) cycles of `Null` states, or 2) bifurcations with empty children because they are incapable of performing a probabilistic summation over `Null` cycles (empty children of bifurcations can give rise to effective cycles of

`Null` states). Phrased qualitatively, the standard Inside and Outside fail because the iterations over states $\boldsymbol{a}$ in Algorithm 1 and Algorithm 3 can correspond to, at most, single `Null` cycles rather than the infinitely many which are possible.

While we believe that an efficient analytic summation is impossible, we can use an iterative approach to approximate an exact result to high precision. A brute-force approach is to just repeatedly iterate over all states $\boldsymbol{a}$ until convergence is reached, but this is computationally-costly. A much more efficient algorithm exists. The key insight is that only `Null` states which are strongly connected (reachable from each other in the state graph by either other `Null` states or bifurcations with empty children) can contribute to any possible `Null` cycle. It follows that instead of repeatedly iterating over all states $\boldsymbol{a}$, it suffices to decompose the state (sub) graph of `Null` states and bifurcation states with possibly-empty children into its strongly-connected components and repeatedly iterate over these connected components rather than the entire graph. Decomposition of a graph into its strongly-connected components can be done in linear time (5), so the overall complexity of inference is still determined by the DP iterations themselves. If we iterate over each connected component 5 times, then upper bounds on the complexity of inference are $O(5 \cdot A \cdot L^{2n})$ and $O(5 \cdot A \cdot L^{3n})$ in space and time for a model with A states of n extant sequences. For the star phylogeny of Figure 1,

5 The TKF Structure Tree model as a transducer

We can cast the TKF91 Structure Tree model as a transducer. **Why is this important/useful? e.g. "This means we can automatically deduce rules like those shown in Tables 6 through Table 8..."**

5.1 The single-sequence TKFST model as a singlet transducer

The states of the singlet transducer are shown in Table 9. Allowed transitions between these states are shown in the paper.

state	type	absorb	emit	$e(\bullet \mid \mathrm{TKFST})$
L	`Start`			
I_L	`Insert`		(x, null)	$p(x)$
S	`Start`			
I_S	`Insert`		(x, y)	$p(x, y)$
B	`Insert`		$(L\,S)$	1

Table 9: States of the singlet transducer of the TKF Structure Tree model (4). Singlet transducers can only have states of type `Start` or `Insert`. **This is the indiegram-style transducer equivalent of the SCFG in Table 1 ...**

5.2 The two-sequence TKFST model as a branch transducer

The states of the branch transducer are shown in Table 10. Allowed transitions between these states are shown in the paper.

5.3 The multi-sequence TKFST model as a composite transducer

We can use the state graph construction algorithm described in the paper and detailed in Section 3.4 to create a model of the simultaneous evolution of several sequences.

state	type	absorb	emit	$e(\bullet \mid \mathrm{TKFST})$
L	`Start`			
I_L	`Insert`		(u, null)	$p(u)$
M_L	`Match`	(x, null)	(u, null)	$p(u\|x)$
D_L	`Match`	(x, null)	$(\mathrm{GAP}, \mathrm{null})$	1
W_L	`Wait`			
S	`Start`			
I_S	`Insert`		(u, v)	$p(u, v)$
M_S	`Match`	(x, y)	(u, v)	$p(u, v\|x, y)$
D_S	`Match`	(x, y)	$(\mathrm{GAP}, \mathrm{GAP})$	1
W_S	`Wait`			
B_i	`Insert`		$(L_i\, S_i)$	1
B	`Match`	$(L\, S)$	$(L\, S)$	1
B_p	`Match`	$(L\, S)$	$(L$ `End`$)$	1
B_e	`Match`	$(L$ `End`$)$	$(L$ `End`$)$	1

Table 10: States of the branch transducer of the TKF Structure Tree model (4). States which have the same names as states of the singlet transducer in Table 9 are the branch equivalent of the corresponding singlet states (e.g. a `Match` state might be the branch equivalent of an `Insert` state). States L_i and S_i are the `Start` states of a sub-model (not shown) identical in structure to the singlet transducer. They are used to insert a new stem-loop structure. **This is the indiegram-style transducer equivalent of the SCFG in Table 4 ...**

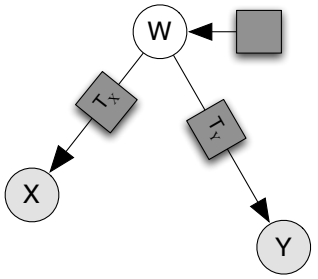


Figure 2: A simple example of transducer composition to build an multi-sequence model of two extant sequences. An ancestral sequence W evolves into two descendant sequences X and Y. A singlet transducer (the horizontal gray box) emit ancestral sequence and structure and two branch transducers (the gray boxes labeled ΔT_X and ΔT_Y) mutate it according to the specified multi-sequence model. Gray nodes correspond to observed data and white nodes unobserved data.

Consider the simple model shown in Figure 2. The state of the multi-sequence model describing this model is a 3-vector $\boldsymbol{a} = (a_1, a_2, a_3)$, where a_1 is the state of the singlet transducer generating the ancestral sequence W and a_2 and a_3 are the states of the branch transducers evolving W into extant sequences X and Y.

We can show some of the allowed transitions of this multi-sequence model. The state of the branch transducer associated with the active node n is shown in bold.

Stem creation. A stem is created at the root node W (corresponding to a bifurcation in the singlet transducer a_1) and survives in the sequence X at node 2 but is deleted in the sequence Y at node 3.

$$\begin{array}{ll} 1: & \left(\begin{array}{c} L \\ L \\ \mathbf{L} \end{array}\right) \rightarrow \left(\begin{array}{c} \mathbf{L} \\ W_L \\ W_L \end{array}\right) \rightarrow \left(\begin{array}{c} B \\ B \\ B_p \end{array}\right) \rightarrow \left(\begin{array}{c} L \\ L \\ \mathbf{L} \end{array}\right) \left(\begin{array}{c} S \\ \mathbf{S} \\ \texttt{End} \end{array}\right) \rightarrow \left(\begin{array}{c} L \\ L \\ \mathbf{L} \end{array}\right) \left(\begin{array}{c} \mathbf{S} \\ W_S \\ \texttt{End} \end{array}\right) \\ 2: & \\ 3: & \end{array} \tag{25}$$

$$\rightarrow \left(\begin{array}{c} \mathbf{L} \\ W_L \\ W_L \end{array}\right) \left(\begin{array}{c} \mathbf{S} \\ W_S \\ \texttt{End} \end{array}\right) \tag{26}$$

Stem insertion. A stem sequence is inserted in sequence X at node 2.

$$\begin{array}{ll} 1: & \left(\begin{array}{c} L \\ L \\ \mathbf{L} \end{array}\right) \rightarrow \left(\begin{array}{c} L \\ \mathbf{L} \\ W_L \end{array}\right) \rightarrow \left(\begin{array}{c} L \\ B_i \\ B \end{array}\right) \rightarrow \left(\begin{array}{c} L \\ L \\ \mathbf{L} \end{array}\right) \left(\begin{array}{c} \texttt{End} \\ S \\ \mathbf{S} \end{array}\right) \rightarrow \left(\begin{array}{c} \mathbf{L} \\ W_L \\ W_L \end{array}\right) \left(\begin{array}{c} \texttt{End} \\ \mathbf{S} \\ W_S \end{array}\right) \\ 2: & \\ 3: & \end{array} \tag{27}$$

Stem termination. All stem sequences are ended by (possibly empty) loop sequences.

$$
\begin{array}{ll}
1: & \begin{pmatrix} \mathbf{S} \\ W_S \\ W_S \end{pmatrix} \rightarrow \begin{pmatrix} B_e \\ B_e \\ B_e \end{pmatrix} \rightarrow \begin{pmatrix} \texttt{End} \\ \texttt{End} \\ \texttt{End} \end{pmatrix} \begin{pmatrix} L \\ L \\ \mathbf{L} \end{pmatrix} \rightarrow \begin{pmatrix} \texttt{End} \\ \texttt{End} \\ \texttt{End} \end{pmatrix} \begin{pmatrix} \texttt{End} \\ \texttt{End} \\ \texttt{End} \end{pmatrix}
\end{array}
\tag{28}
$$

The functions $\alpha(t)$, $\beta(t)$ and $\gamma(t)$ are parametrized by the insertion and deletion rates of the Structure Tree model. They are defined for loop sequences as

$$\kappa_1 = \lambda_1/\mu_1 \tag{29}$$

$$\alpha_1 = \exp\left(-\mu_1 t\right) \tag{30}$$

$$\beta_1 = \frac{\lambda_1\left(1-\exp\left((\lambda_1-\mu_1)t\right)\right)}{\mu_1-\lambda_1\exp\left((\lambda_1-\mu_1)t\right)} \tag{31}$$

$$\gamma_1 = 1 - \frac{\mu_1\left(1-\exp\left((\lambda_1-\mu_1)t\right)\right)}{\left(1-\exp(-\mu_1 t)\right)\left(\mu_1-\lambda_1\exp\left((\lambda_1-\mu_1)t\right)\right)} \tag{32}$$

and similarly for stem sequences (4).

6 Software

We have written software tools in Perl and C++ implementing much of the theory described here. All tools are available from `http://biowiki.org/dart` as part of the DART software package for sequence analysis.

6.1 Automated grammar construction

We implemented our model construction algorithm on the star phylogeny (Figure 1) in a set of Perl scripts. Given a singlet transducer modeling ancestral sequences and structures and a branch transducer modeling structural evolution, the scripts generate C++ code to create the corresponding jointly-normalized SCFG. All possible models of structural evolution which can be represented by a Pair SCFG are permitted as input to the scripts, allowing for flexible and automated model design.

Given files describing the singlet and branch transducers, including weights of all transitions which may be functions of evolutionary distance, the package ComposedTreeTransducer::FourWayComposedTT can automatically generate the state graph and transition matrix of an multi-sequence model of three extant sequences (Figure 1). It removes the useless wind-back `Null` states described in the paper and introduces effective direct transitions caused by bifurcations with possibly-empty children. The package ComposedTreeTransducer::TripletSCFG transforms an multi-sequence model created by ComposedTreeTransducer::FourWayComposedTT into the corresponding jointly-normalized three-sequence SCFG (Section 3.3) and generates C++ code to build the model.

Example singlet and branch transducers files are provided for a simple Pair HMM model, a simple Pair SCFG model and the full TKF Structure Tree model.

6.2 Reconstruction of ancestral structures

The program INDIEGRAM can perform maximum-likelihood inference on the three-sequence SCFGs automatically generated by the ComposedTreeTransducer::TripletSCFG package. Complete or no structural information for the three extant sequences can be supplied as input.

7 Reconstruction of covariant substitution histories

Our XRATE program is a multiple-alignment analysis tool that estimates rate parameters for a broad class of models, including covariant RNA substitution models (6). It can also be used to predict consensus secondary structures for alignments. We implemented ancestral sequence reconstruction in XRATE, including posterior probabilities that the reconstructions are correct.

Given a multiple sequence alignment and a phylogenetic tree, XRATE estimates maximum-likelihood values of rate and probability parameters by Expectation Maximization. During parameter estimation, any unspecified ancestral sequences are summed out using Felsenstein's peeling algorithm. The new feature is that the program can now find the ancestral sequence that has the highest posterior probability, contingent on the maximum-likelihood estimates of the model parameters. During the parameter estimation and ancestral reconstruction steps, the secondary structure may be specified, or it may be summed out as a latent variable.

For the ancestral reconstruction experiments described here, we assumed that alignment, phylogeny and secondary structure were known, but that ancestral sequences were unknown.

A simple computational experiment demonstrates the need for covariant substitution models. We first used XRATE to estimate maximum-likelihood parameters for a covariant model of RNA base-pair substitution. These rates were estimated from ribosomal RNA alignments derived by (7) from the European rRNA database (8). We also fit a "naive" general reversible point-substitution model to these alignments, again using XRATE. All subsequent results are implicitly conditioned on these maximum-likelihood rate estimates. Next, we used the companion program SIMGRAM to generate 5000 random alignments (including ancestral sequence), simulating on a 75-taxon phylogeny from an RNA gene family in the RFAM database (9). (Specifically, we chose the type-I Hammerhead ribozyme, one of the 5% most populous RFAM families. We repeated this experiment with other RFAM phylogenies, but the general trends reported here were not dependent on the choice of tree.) We stripped the ancestral sequence out of the simulated alignments (leaving the true structure annotation intact), then re-estimated the ancestral

sequence with XRATE, using both the (correct) covariant substitution model and the (naive) non-covariant point-substitution model. The imputed ancestral sequences were compared to the true sequences (known from the simulation), and the model-derived posterior probabilities were compared to the empirical accuracy of the corresponding reconstructed sequence.

Figure 3 (left) shows the results of these comparisons. When using the same (covariant) model for simulation and reconstruction, the posterior probabilities are an excellent unbiased estimate of the frequency with which the model reconstructions are correct. However, the naive point-substitution model, which does not include covariant substitutions at base-paired sites, systematically under-estimates these probabilities. In our simulation, the ancestral error rate for the naive model (4.9%) was significantly higher than that of the covariant model (1.7%). Furthermore, 74% of incorrect base-pairs predicted by the naive model were non-canonical (i.e. not AU, CG, GC, UA, GU or UG), compared to only 2% of incorrect predictions by the covariant model.

We also performed cross-validation (hold-one-out) experiments. Starting from RFAM alignments whose secondary structure is labeled “Published” (i.e. experimentally validated), we estimated phylogenetic trees using the neighbor-joining algorithm with the Jukes-Cantor model, then removed individual sequences and attempted to reconstruct them using XRATE. (This is possible because the substitution models are time-reversible, so we can treat any node as ancestral.) In this experiment, the error rates for base-pair reconstructions again differed by a few percent, though both error rates were higher than in the simulation test (covariant model: 11%, non-covariant model: 16%). Of the incorrectly-reconstructed base-pairs, the non-covariant model again predicted more non-canonical pairs (40%) than the covariant model (17%), especially at lower confidence levels (Figure 4, right).

Posterior probability estimates for both models were systematically higher in the cross-validation test than they were in the simulation (Figure 3, right). The presence of this trend in both models suggests that it may be due to differences between the European rRNA align-

ments (from which the rates were estimated) and the RFAM alignments (which were used in the cross-validation test). This is confirmed by our empirical observations of covariant rate matrices estimated from the two alignment datasets: base-pairs in the RFAM alignments are more frequently non-canonical, and appear to evolve faster (relative to single-stranded regions), than base-pairs in the rRNA alignments (for details, see `biowiki.org/RnaModels`). Repeating the cross-validation experiment with rates estimated from the RFAM alignments, we find the covariant model predicted slightly fewer non-canonical base-pairs (14% rather than 17%) and produced slightly more accurate posterior probability estimates (data not shown). Although this is no longer a strict cross-validation (since parameters were estimated from the test dataset), it nevertheless emphasizes the value of correct models.

Accurate estimates of posterior probabilities are likely to be important: sub-optimal predictions are a significant piece of the reconstruction puzzle (**? ? ?**). Ancestral reconstruction by probabilistic modeling does not yield a single definitive sequence, but rather a probability distribution over sequences. Combinatorial synthesis by degenerate primer assembly has been advocated as a means of sampling from this distribution (**?**). Nucleotide-perfect reconstructions may not always be possible: it may be more productive to aim for accurate confidence estimates (including sub-optimal predictions) than to focus exclusively on getting everything right the first time.

Based on these tests, we conclude that deep phylogenetic reconstructions of ribosomes, and other RNAs, will require covariant substitution models that take account of RNA secondary structure. We may reasonably deduce that indel reconstructions will, similarly, need to take account of RNA structure. These results, therefore, strongly motivate the development of multi-sequence models for RNA structure, such as the TKF Structure Tree.

8 Multiple alignment and RNA folding

Previously, we developed an evolutionary model for RNA structure, called the TKF Structure Tree (TKFST). The model is simplistic compared to heuristic SCFGs that have been developed for RNA bioinformatics applications (10; 11; 9; 12; 7; 13). Nevertheless it is a full phylogenetic model that, unlike those other models, preserves RNA structural integrity under indels and structural changes as well as substitutions. It is therefore an illustrative example for designing an RNA reconstruction framework.

The TKF Structure Tree is based on the TKF model (14), the first continuous-time stochastic model for indels. Critical evaluation of the TKF model on the BAliBASE protein alignment benchmark was crucial to the development of the transducer theory, identifying strengths and weaknesses of TKF while stimulating development of a more general framework (15). We sought to use the same approach with the TKFST model.

We emphasize that this is not just a method for simultaneous alignment and folding of RNA. Our intention here is a critical assessment of the realism of TKFST as an evolutionary model, **not** simply an implementation of the algorithm of (16), of which many exist (17; 12; 7; 18; 19; 20; 21; 22). An alignment benchmark is a powerful way to reveal potential flaws in a molecular evolutionary model, as was shown with the TKF model (15). Since our purpose is a controlled comparison between TKFST and a richer reference grammar, we report only results for our STEMLOC tool, which is one of the best-performing constrained-Sankoff tools (22). More importantly, the use of STEMLOC allows us to control for the constraint heuristics, using the same constraints for the TKFST grammar as we do for the reference grammar, as described below.

We first modified STEMLOC to allow the user to specify the Pair SCFG that it uses for progressive alignment. We then used our program EVOLDOER, an implementation of TKFST for pairwise alignment, to generate Pair SCFGs corresponding to the TKFST model at timepoints $\{0.01, 0.1, 0.2, 0.3, 0.4\}$, using loop and stem substitution rate matrices estimated with XRATE

from ribosomal RNA alignments (23; 7) and TKFST parameters $\{\lambda_n, \mu_n\}$ estimated previously (4).

We then compared the TKFST grammars to STEMLOC's native grammar on the BRalibaseII benchmark of reference alignments. The native grammar in STEMLOC, as described in (12), is richer than TKFST: excluding the substitution model, it has 14 free parameters (compared to TKFST's 4), uses an affine gap penalty and explicitly models structural features such as multi-branched loops, symmetric/asymmetric bulges, and minimum loop lengths. Unlike TKFST, the STEMLOC grammar is structurally unambiguous: a one-to-one mapping exists from structures to parse trees. We therefore expected STEMLOC to perform significantly better than TKFST at both alignment and structure prediction.

	U5	g2intron	rRNA	tRNA	TPP_riboswitch
STEMLOC grammar	**82.6** / **83.7**	74.2 / 74.8	**92.6** / **92.8**	93.2 / 93.9	**76.8** / **80.7**
TKFST grammar	81.6 / 81.7	**75.4** / **75.0**	91.4 / 92.6	**94.6** / **94.4**	62.4 / 73.7

Table 11: Percentage sensitivity and positive predictive value (Sensitivity/PPV) for pairwise nucleotide-level alignments in the BRalibaseII benchmark. Sensitivity is defined as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$ and PPV is defined as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$ where TP is the number of true positives (i.e. correctly aligned residue pairs), FN is the number of false negatives (i.e. residue pairs that should have been aligned but weren't) and FP is the number of false positives (i.e. residue pairs that were incorrectly aligned). These statistics are summed over all pairs of sequences in the multiple alignment; therefore, "Sensitivity" for pairwise residue alignments is equivalent to the Sum of Pairs Score, or SPS (24). Sensitivity measures the proportion of the true alignment that was found (i.e. included in the test alignment); in contrast, PPV measures the proportion of the test alignment that was correct (i.e. included in the true alignment). "g2intron" is the RFAM entry `Intron_gpII`, containing domains V and VI of the Group II intron.

Table 11 shows the alignment accuracy of the STEMLOC and TKFST grammars on the BRalibaseII test sets, reporting sensitivity and positive predictive value (PPV) at the level of pairwise site alignments. Surprisingly, TKFST and STEMLOC are closely comparable, with no clear winner except on the TPP riboswitch alignment. One possible (partial) explanation is that the affine gap penalty is not helping all that much: the mean gap length in the STEMLOC grammar ranges from 1.3 to 3.5 bases (in loops) and 1.4 to 1.7 basepairs (in stems), so the

improvement over a linear gap penalty is small. (The mean gap lengths are specified as ranges because STEMLOC's native grammar set includes four distance settings; mean gap length is a function of distance.)

	U5	g2intron	rRNA	tRNA	TPP_riboswitch
STEMLOC grammar	74.9 / **73.9**	64.3 / 56.7	51.0 / 59.0	74.0 / 76.4	57.1 / 63.3
TKFST grammar	37.9 / 68.0	42.1 / 63.8	37.4 / 66.5	70.9 / **88.3**	36.2 / **77.2**
STEMLOC + PFOLD	**90.5** / 66.2	**88.2** / 63.2	77.2 / 64.4	88.5 / 70.6	**72.3** / 74.7
TKFST + PFOLD	80.8 / 60.7	84.5 / **64.1**	**80.6** / **66.9**	**88.6** / 71.1	59.8 / 63.9

Table 12: Percentage sensitivity and positive predictive value (Sensitivity/PPV) for predicted basepairs in the BRalibaseII benchmark. The first two rows of the table represent the structure-prediction accuracy of a progressive alignment algorithm, using the TKFST and stemloc-native grammars; this progressive algorithm makes an early commitment to structures, so the base-pair accuracy is poor, especially for TKFST. The latter two rows of the table represent the structure-prediction accuracy of the PFOLD grammar (XRATE's re-implementation), conditioned on the alignments inferred using the TKFST and stemloc-native grammars. "g2intron" is the RFAM entry `Intron_gpII`, containing domains V and VI of the Group II intron. Sensitivity is defined as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$ and PPV is defined as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$ where TP is the number of true positives (i.e. correctly predicted basepairs), FN is the number of false negatives (i.e. basepairs that should have been predicted but weren't) and FP is the number of false positives (i.e. predicted basepairs that were incorrect). Sensitivity measures the proportion of the true structure that was found (i.e. included in the test structure); in contrast, PPV measures the proportion of the test structure that was correct (i.e. included in the true structure).

Table 12 shows the sensitivity and PPV of the two grammars at structure prediction. Here, the difference is clear: STEMLOC's native grammar, with its richer model of RNA structure, is the clear winner in sensitivity, although TKFST's PPV is very good. We suspected that some of TKFST's poor sensitivity might be due to the way that STEMLOC performs progressive alignment: the structure of the initial pairwise seed alignment is propagated to all subsequently-added rows, so that the first-guessed structure is "locked in". Thus, structure prediction for multiple alignment tends to be not much better than for pairwise alignment. To test this, we re-estimated rate parameters and secondary structure using the PFOLD grammar (11), re-parameterized on rRNA as described above, while fixing the multiple sequence alignment to the value estimated in the previous step. The structure prediction accuracy was much better

when combined with a post-processing structure prediction step, and both grammars were comparable (Table 12), as might be expected from their similar alignment quality (Table 11).

Together, these results suggest that while TKFST is a reasonably good model of alignments, it is a weak model for RNA structure. We anticipate that improvements should be possible by reviewing other comparisons of SCFGs at structure prediction, such as the study by (25).

References

[1] Rivas E, Eddy S (2008) Probabilistic phylogenetic inference with insertions and deletions. PLoS Computational Biology 4:e1000172.

[2] Holmes I (2003) Using guide trees to construct multiple-sequence evolutionary HMMs. Bioinformatics 19 Suppl. 1:i147–157.

[3] Holmes I, Rubin GM (2002) Pairwise RNA structure comparison using stochastic context-free grammars. Pacific Symposium on Biocomputing, 2002.

[4] Holmes I (2004) A probabilistic model for the evolution of RNA structure. BMC Bioinformatics 5.

[5] Tarjan R (1972) Depth first search and linear graph algorithms. SIAM Journal on Computing 1:146–160.

[6] 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.

[7] Dowell RD, Eddy SR (2006) Efficient pairwise RNA structure prediction and alignment using sequence alignment constraints. BMC Bioinformatics 7:400.

[8] Wuyts J, Perrière G, Van De Peer Y (2004) The European ribosomal RNA database. Nucleic Acids Research 32:D101–103.

[9] Griffiths-Jones S, Bateman A, Marshall M, Khanna A, Eddy SR (2003) Rfam: an RNA family database. Nucleic Acids Research 31:439–441.

[10] Eddy SR, Durbin R (1994) RNA sequence analysis using covariance models. Nucleic Acids Research 22:2079–2088.

[11] Knudsen B, Hein J (1999) RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. Bioinformatics 15:446–454.

[12] Holmes I (2005) Accelerated probabilistic inference of RNA structure evolution. BMC Bioinformatics 6.

[13] 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.

[14] Thorne JL, Kishino H, Felsenstein J (1991) An evolutionary model for maximum likelihood alignment of DNA sequences. Journal of Molecular Evolution 33:114–124.

[15] Holmes I, Bruno WJ (2001) Evolutionary HMMs: a Bayesian approach to multiple alignment. Bioinformatics 17:803–820.

[16] Sankoff D (1985) Simultaneous solution of the RNA folding, alignment, and protosequence problems. SIAM Journal of Applied Mathematics 45:810–825.

[17] Mathews DH, Turner DH (2002) Dynalign: an algorithm for finding the secondary structure common to two RNA sequences. Journal of Molecular Biology 317:191–203.

[18] Torarinsson E, Havgaard J, Gorodkin J (2007) Multiple structural alignment and clustering of RNA sequences. Bioinformatics 23:926–932.

[19] Xu X, Ji Y, Stormo G (2007) RNA Sampler: a new sampling based algorithm for common RNA secondary structure prediction and structural alignment. Bioinformatics 23:1883–1891.

[20] Lindgreen S, Gardner P, Krogh A (2007) MASTR: multiple alignment and structure prediction of non-coding RNAs using simulated annealing. Bioinformatics 23:3304–3311.

[21] Kiryu H, Tabei Y, Kin T, Asai K (2007) Murlet: a practical multiple alignment tool for structural RNA sequences. Bioinformatics 23:1588–1598.

[22] Bradley RK, Pachter L, Holmes I (2008) Specific alignment of structured RNA: Stochastic grammars and sequence annealing. Bioinformatics .

[23] Wuyts J, Rijk PD, Peer YVd, Winkelmans T, Wachter RD (2001) The European Large Subunit Ribosomal RNA Database. Nucleic Acids Research 29:175–7.

[24] Thompson JD, Plewniak F, Poch O (1999) A comprehensive comparison of multiple sequence alignment programs. Nucleic Acids Research 27:2682–2690.

[25] Dowell RD, Eddy SR (2004) Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. BMC Bioinformatics 5:71.

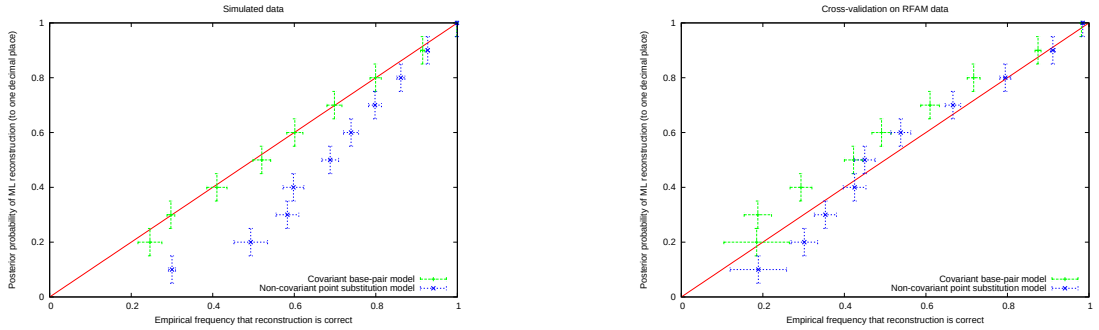


Figure 3: Accuracy of confidence estimates for base-pair reconstruction in simulations (left) and cross-validations on real data (right). Reconstructions were performed using XRATE (6), using both covariant base-pair substitution models and naive, non-covariant, single-base point-substitution models. The plots show the relationship between the posterior probability that a model-based base-pair reconstruction is correct (vertical axis), and the empirical probability that reconstructions with that posterior probability are actually correct (horizontal axis). If the model's posterior probability estimates are correct, these points should lie on a straight diagonal line. Left: simulation test on the hammerhead ribozyme phylogeny. A covariant model (estimated from rRNA alignments) was used to generate the simulated alignments on the given phylogeny; the same model was then used to reconstruct ancestral sequence and obtain confidence estimates. As expected, the model accurately estimates the probability that the reconstructed base-pair is correct. A naive, non-covariant model (the general reversible model, estimated from the same alignments) was also used for reconstruction and confidence estimates. This model *under*-estimates the probability of correct reconstructions, since its equilibrium frequency has a higher entropy than the covariant model: i.e. it assigns probability more evenly across the sixteen possible base-pairs (as opposed to the covariant model, which strongly favors the six canonical base-pairs). Right: cross-validation ("hold-one-out") test using RFAM alignments with published (verified) structures. Here, the covariant model slightly *over*-estimates the probability of correctness. The rRNA alignments (from which the model rates were estimated) have fewer non-canonical base-pairs than the RFAM alignments, so that the covariant model entropy is slightly *lower* than the empirical base-pair distribution; this may explain the discrepancy. The naive model's confidence estimates are correct for high-accuracy reconstructions, over-estimates for medium-accuracy reconstructions, and under-estimates for low-accuracy reconstructions.

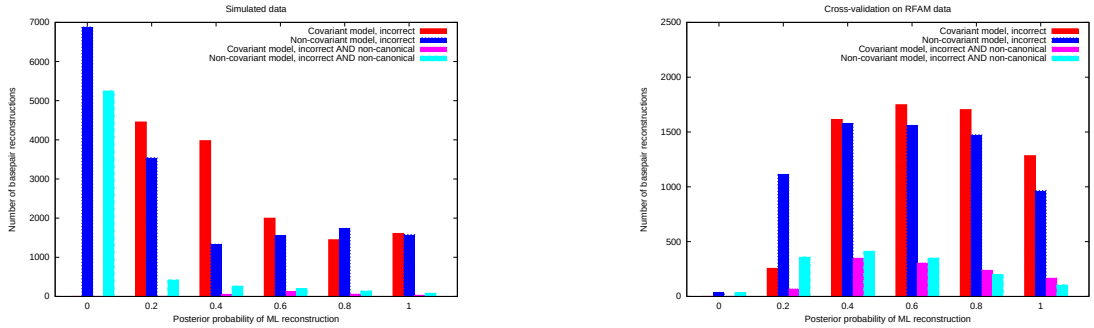


Figure 4: Proportions of reconstructed base-pairs that are incorrect (and the subset of those that are non-canonical) in simulations (left) and cross-validations on real data (right). Base-pair reconstructions that are incorrect AND non-canonical might potentially disrupt a synthetic reconstructed structure. While the covariant model does not always get more base-pairs correct than the naive model, it predicts fewer non-canonical base-pairs. The sharpest difference between the models are in the lowest posterior-probability categories, corresponding to low-confidence predictions (e.g. sequences on deep branches, or base-pairs with few homologues). In this category, the covariant model also makes far fewer errors than the naive model. See main text and caption to Figure 3 for details of simulation and cross-validation procedures. RFAM families with verified structures tended to be less phylogenetically diverse, with roughly half the total branch length (on average), than the tree used for our simulations; this may explain the relative paucity of low posterior-probability errors in the right-hand plot, compared to the left.