

Evolutionary models of structured RNA: Text S1

Robert K. Bradley¹, Ian Holmes^{1,2,*}

1 Biophysics Graduate Group, University of California, Berkeley, CA, USA

2 Department of Bioengineering, University of California, Berkeley, CA, USA

* E-mail: indiegram@postbox.biowiki.org

Contents

1 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 $P(Y|X, \Delta T, \theta)$.

1.1 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.

1.2 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

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

State typing is as follows:

- A **Start** state begins a branch of the parse tree.
- An **Insert** state emits, but does not absorb, symbols.

¹The 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.

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

- 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 $\{\mathbf{Start}, \mathbf{Insert}, \mathbf{Match}, \mathbf{End}\}$ states of the familiar Pair HMM. Deletions are handled as a special case by states of type **Match** which absorb but do not emit symbol 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 $\phi : \text{type}(\phi) \in \{\mathbf{Insert}, \mathbf{Match}\}$ can emit nonterminal pairs (cd) , where c or d can be the null symbol, and the emitted pairs (cd) can be absorbed by states of type **Match**.

The emission weight of a nonterminal pair (cd) from the state $b : \text{type}(b) = \mathbf{Match}$, conditional on absorption of a nonterminal pair (lm) , is

$$e_b(cd|lm, \theta).$$

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

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

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 $\mathbf{B[d}\phi]$.

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

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

Terminal emission is handled by states $\phi : \text{type}(\phi) \in \{\mathbf{Match}, \mathbf{Insert}\}$, 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 $(x \text{ null})$; right single-terminal emissions are handled similarly. The weight of an emission of a terminal pair (x, y) from a state $b : \text{type}(b) = \mathbf{Match}$, 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 \mathbf{Match} are defined conditioned upon the absorbed symbols.

2 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 [?] for combining HMMs on a guide tree.

2.1 The guide tree

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

2.2 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 (parent(m) \rightarrow m); $m = 2, \dots, N$ 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 (parent(m) \rightarrow m) of the guide tree.

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

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

These states are typed as

$$\text{type}(\mathbf{a}) \in \{\text{Start}, \text{Emit}, \text{Bifurcation}, \text{Null}, \text{End}\}.$$

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 ??).

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 $\mathbf{a} \rightarrow \mathbf{b}$ is therefore

$$t(\mathbf{a}, \mathbf{b}) = \prod_{m | a_m \neq b_m} t(a_m, b_m | \theta^{(m)}), \quad (1)$$

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

$$\mathbf{b}(\mathbf{x} \mathbf{y}) = \prod_m e_{b_m} \left(x_m y_m | x_{\text{parent}(m)} y_{\text{parent}(m)}, \theta^{(m)} \right). \quad (2)$$

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

2.3 Formal grammars

Analogously to the case with two-sequence models (Section ??), 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 $P(X_1, \dots, X_n)$.

2.4 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.

2.5 Allowed transitions

Following [?], 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 \mathbf{a} ,

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

$$= \operatorname{argmax}_m \{ \operatorname{type}(a_m) \notin \{\mathbf{Wait}, \mathbf{End}\} \} . \tag{4}$$

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

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

$$\operatorname{type}(\mathbf{b}) = \mathbf{Null} \tag{5}$$

$$\operatorname{Weight}(\mathbf{a} \rightarrow \mathbf{b}) = t(\mathbf{a}, \mathbf{b}) \tag{6}$$

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

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 $b_n : \operatorname{type}(b_n) \in \{\mathbf{Start}, \mathbf{Wait}\}$. 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

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

$$\text{type}(\mathbf{b}) = \text{Emit} \quad (8)$$

$$\text{Weight}(\mathbf{a} \rightarrow \mathbf{x} \mathbf{b} \mathbf{y}) = t(\mathbf{a}, \mathbf{b}) \cdot \mathbf{b}(\mathbf{x} \mathbf{y}) \quad (9)$$

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

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

The active node n makes a transition to a state $b_n : \text{type}(b_n) = \text{Insert}$, 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 $\{m | m \triangleright n, \text{type}(a_m) \neq \text{End}\}$, 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 \mathbf{x} or \mathbf{y} is null, then (??) could represent the evolution of a single unpaired nucleotide along the subtree rooted at node n .

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

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

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

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

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

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

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

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

Right emission. All terminals x in the transition $a \rightarrow x b y$ (??) are null.

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

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

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

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

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

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

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

Paired emission. There is at least one non-null terminal in both x and y in the transition $a \rightarrow x b y$ (??).

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

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

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

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

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

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

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

Bifurcations

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

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

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

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

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

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

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

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

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

Node n : $b_n : \text{type}(b_n) = \mathbf{Insert}$, \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 $\text{type}(a_m) = \mathbf{End}$

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

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

Right bifurcation. All nonterminals \mathbf{c} in the transition $\mathbf{a} \rightarrow \mathbf{c} \mathbf{b} \mathbf{d} (??)$ are null. The nonterminals \mathbf{d} are the “new” states; the nonterminals \mathbf{b} are the states which will generate the (evolved) ancestral sequence.

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

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

Node n : $b_n : \text{type}(b_n) = \mathbf{Insert}$, \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 $\text{type}(a_m) = \mathbf{End}$

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 \mathbf{c} and \mathbf{d} in the transition $\mathbf{a} \rightarrow \mathbf{c} \mathbf{b} \mathbf{d}$ (??). The nonterminals \mathbf{c} and \mathbf{d} are both “new” states; the nonterminals \mathbf{b} are the states which will generate the (evolved) ancestral sequence.

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

$$(c_m d_m) = (\mathbf{End} \mathbf{End}).$$

Node n : $b_n : \text{type}(b_n) = \mathbf{Insert}$, \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 $\text{type}(a_m) = \mathbf{End}$

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

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

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

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

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

$$= \prod_{m \mid \text{type}(a_m) \neq \mathbf{End}} t(a_m, \mathbf{End} \mid \theta^{(m)}). \tag{15}$$

The singlet transducer associated with the highest active ancestral node,

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

can make a transition to the state \mathbf{End} , forcing the entire multi-sequence model to transition to \mathbf{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:

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

Nodes $m \not\geq \hat{n}$: $a_m = \mathbf{End}$.

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

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

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

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

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.

\hat{n} and $\text{Weight}(\mathbf{a} \rightarrow \mathbf{End})$ 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 $\{m | m \geq \hat{n}\}$. The TKF Structure Tree model (Section ??) satisfies this condition. A more general approach is probably possible, but it involves summing over paths $a' : a_m \rightarrow a' \rightarrow \mathbf{End}$, handling possible bifurcations in these paths, etc.

3 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 ??, 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 ??; the generalization to N sequences is straightforward.

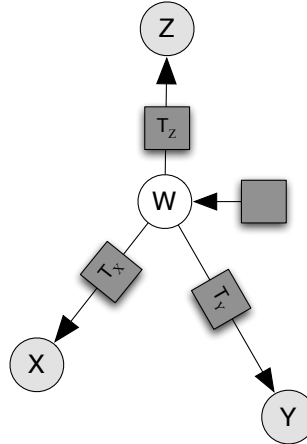


Figure 1. A star phylogeny with three (extant) leaf sequences. An ancestral sequence W evolves into three descendant sequences X , Y and Z . A singlet transducer (the horizontal gray box) emit ancestral sequence and structure and three branch transducers (the gray boxes labeled ΔT_X , ΔT_Y and ΔT_Z) 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.

3.1 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 $L_1 = |x^{(1)}|$.

3.2 Fold Envelopes

We use the fold envelope concept $[?, ?]$ to constrain the set of structures our algorithms consider. A fold envelope \mathcal{F} for a sequence x is a set of coordinate pairs satisfying

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

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

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 \mathcal{F} are sorted in the same order as coordinate pairs (i, j) are generated by the iteration $\{\text{for } i = L \text{ to } 0 \ \{\text{for } j = i \text{ to } L\}\}$.

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 \mathcal{F} are then sorted in the same order as coordinate pairs (i, j) are generated by the iteration $\{\text{for } i = 0 \text{ to } L \ \{\text{for } j = L \text{ to } i\}\}$.

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

Inward and outward emission connections, specifying which subsequences are reachable from a given subsequence and state, are defined as

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

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

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

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

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

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

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.

3.3 Inside

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

The transition and emission probability `calcTransEmitProb($\mathbf{a}; \cdot$)` can be calculated recursively in the unconstrained case as

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

Null transitions are caught in the sum when the terminals $\mathbf{y} = \mathbf{z} = \text{null}$. The constrained case is handled differently: The multiple sums are replaced by a single iteration over states \mathbf{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 ??.

⁴We defined emission connections slightly differently than do [?].

```

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

```

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

```

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

```

Algorithm 2: Subroutine calcTransEmitProb() for the Inside algorithm.

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

$$\sum_{\mathbf{b} | \exists \mathbf{a} \rightarrow \mathbf{c} \mathbf{b}} \text{Weight}(\mathbf{a} \rightarrow \mathbf{c} \mathbf{b}) \alpha_{\mathbf{c}}(n_L^{(1)}, n_L^{(2)}, n_L^{(3)}) \alpha_{\mathbf{b}}(n_R^{(1)}, n_R^{(2)}, n_R^{(3)})$$

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

$$\sum_{\mathbf{b} | \exists \mathbf{a} \rightarrow \mathbf{b} \mathbf{d}} \text{Weight}(\mathbf{a} \rightarrow \mathbf{b} \mathbf{d}) \alpha_{\mathbf{b}}(n_L^{(1)}, n_L^{(2)}, n_L^{(3)}) \alpha_{\mathbf{d}}(n_R^{(1)}, n_R^{(2)}, n_R^{(3)}).$$

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

$$\alpha_{\mathbf{a}}(j_1, j_1, j_2) = t(\mathbf{a}, \text{End}) + \sum_{\mathbf{b} | \exists \mathbf{a} \rightarrow \mathbf{b}} \text{Weight}(\mathbf{a} \rightarrow \mathbf{b}) \alpha_{\mathbf{a}}(j_1, j_1, j_2), \quad (23)$$

for $0 \leq j_1 \leq L_1, 0 \leq j_2 \leq L_2, 0 \leq j_3 \leq L_3$. 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

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

where `Start` is the unique start state of the model.

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

3.4 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 (??), $\sum_{\mathbf{b} | \exists \mathbf{a} \rightarrow \mathbf{b}}$ will be replaced by $\max_{\mathbf{b} | \exists \mathbf{a} \rightarrow \mathbf{b}}$). 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 $x_{i_1+1}^{(1)} \dots x_{j_1}^{(1)}, x_{i_2+1}^{(2)} \dots x_{j_2}^{(2)}, x_{i_3+1}^{(3)} \dots x_{j_3}^{(3)}$.

3.5 Outside

The outside probability $\beta_{\mathbf{b}}(i_1, j_1, i_2)$ 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 $x_{i_1+1}^{(1)} \dots x_{j_1}^{(1)}, x_{i_2+1}^{(2)} \dots x_{j_2}^{(2)}, x_{i_3+1}^{(3)} \dots x_{j_3}^{(3)}$ which are rooted in the state \mathbf{b} . Algorithm ?? gives pseudocode for the fold-envelope version of the Outside algorithm. The subroutines $\text{calcTransEmitProb}()$,

$\text{calcLBifurcProb}()$ and $\text{calcRBifurcProb}()$ used in the Outside algorithm are defined below.

```

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

```

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

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

unconstrained case as

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

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

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

$$\sum_{\mathbf{a} \mid \exists \mathbf{a} \rightarrow \mathbf{c} \mathbf{b}} \text{Weight}(\mathbf{a} \rightarrow \mathbf{c} \mathbf{b}) \beta_{\mathbf{a}}(n_O^{(1)}, n_O^{(2)}, n_O^{(3)}) \alpha_c(n_L^{(1)}, n_L^{(2)}, n_L^{(3)})$$

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

$$\sum_{\mathbf{a} \mid \exists \mathbf{a} \rightarrow \mathbf{b} \mathbf{d}} \text{Weight}(\mathbf{a} \rightarrow \mathbf{b} \mathbf{d}) \beta_{\mathbf{a}}(n_O^{(1)}, n_O^{(2)}, n_O^{(3)}) \alpha_d(n_R^{(1)}, n_R^{(2)}, n_R^{(3)})$$

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

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

The boundary condition is just

$$\beta_{\text{Start}}(0, L_1, 0) = 1. \quad (24)$$

3.6 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 \mathbf{a} in Algorithm ?? and Algorithm ?? 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 \mathbf{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 \mathbf{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 [?], 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 ??,

4 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 ?? through Table ??...”**

4.1 The single-sequence TKFST model as a singlet transducer

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

state	type	absorb	emit	$e(\bullet \text{TKFST})$
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 1. States of the singlet transducer of the TKF Structure Tree model [?]. Singlet transducers can only have states of type Start or Insert. **This is the indiegram-style transducer equivalent of the SCFG in Table ?? ...**

4.2 The two-sequence TKFST model as a branch transducer

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

4.3 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 ?? to create a model of the simultaneous evolution of several sequences.

Consider the simple model shown in Figure ?. The state of the multi-sequence model describing this model is a 3-vector $\mathbf{a} = (a_1, a_2, a_3)$, 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 .

state	type	absorb	emit	$e(\bullet \text{TKFST})$
L	Start			
I_L	Insert		(u, null)	$p(u)$
M_L	Match	(x, null)	(u, null)	$p(u x)$
D_L	Match	(x, null)	$(\text{GAP}, \text{null})$	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)	(GAP, GAP)	1
W_S	Wait			
B_i	Insert		$(L_i S_i)$	1
B	Match	$(L S)$	$(L S)$	1
B_p	Match	$(L S)$	$(L \text{End})$	1
B_e	Match	$(L \text{End})$	$(L \text{End})$	1

Table 2. States of the branch transducer of the TKF Structure Tree model [?]. States which have the same names as states of the singlet transducer in Table ?? 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 ?? ...**

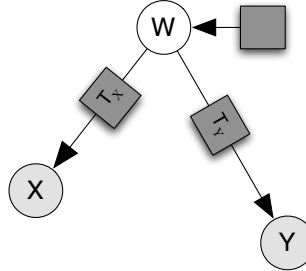


Figure 2. A simple example of transducer composition to build a multi-sequence model of two extant sequences. An ancestral sequence W evolves into two descendant sequences X and Y . A singlet transducer (the horizontal gray box) emit ancestral sequence and structure and two branch transducers (the gray 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.

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.

$$\begin{array}{lcl}
1 : & \begin{pmatrix} L \\ L \\ \mathbf{L} \end{pmatrix} & \rightarrow \begin{pmatrix} \mathbf{L} \\ W_L \\ W_L \end{pmatrix} \rightarrow \begin{pmatrix} B \\ B \\ B_p \end{pmatrix} \rightarrow \begin{pmatrix} L \\ L \\ \mathbf{L} \end{pmatrix} \begin{pmatrix} S \\ \mathbf{S} \\ \text{End} \end{pmatrix} \rightarrow \begin{pmatrix} L \\ L \\ \mathbf{L} \end{pmatrix} \begin{pmatrix} \mathbf{S} \\ W_S \\ \text{End} \end{pmatrix} \\
2 : & & \\
3 : & &
\end{array} \quad (25)$$

$$\begin{array}{lcl}
& & \rightarrow \begin{pmatrix} \mathbf{L} \\ W_L \\ W_L \end{pmatrix} \begin{pmatrix} \mathbf{S} \\ W_S \\ \text{End} \end{pmatrix} \\
& &
\end{array} \quad (26)$$

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

$$\begin{array}{lcl}
1 : & \begin{pmatrix} L \\ L \\ \mathbf{L} \end{pmatrix} & \rightarrow \begin{pmatrix} L \\ \mathbf{L} \\ W_L \end{pmatrix} \rightarrow \begin{pmatrix} L \\ B_i \\ B \end{pmatrix} \rightarrow \begin{pmatrix} L \\ L \\ \mathbf{L} \end{pmatrix} \begin{pmatrix} \text{End} \\ S \\ \mathbf{S} \end{pmatrix} \rightarrow \begin{pmatrix} \mathbf{L} \\ W_L \\ W_L \end{pmatrix} \begin{pmatrix} \text{End} \\ \mathbf{S} \\ W_S \end{pmatrix} \\
2 : & & \\
3 : & &
\end{array} \quad (27)$$

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

$$\begin{array}{lcl}
1 : & \begin{pmatrix} \mathbf{S} \\ W_S \\ W_S \end{pmatrix} & \rightarrow \begin{pmatrix} B_e \\ B_e \\ B_e \end{pmatrix} \rightarrow \begin{pmatrix} \text{End} \\ \text{End} \\ \text{End} \end{pmatrix} \begin{pmatrix} L \\ L \\ \mathbf{L} \end{pmatrix} \rightarrow \begin{pmatrix} \text{End} \\ \text{End} \\ \text{End} \end{pmatrix} \begin{pmatrix} \text{End} \\ \text{End} \\ \text{End} \end{pmatrix} \\
2 : & & \\
3 : & &
\end{array} \quad (28)$$

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

$$\kappa_1 = \lambda_1 / \mu_1 \quad (29)$$

$$\alpha_1 = \exp(-\mu_1 t) \quad (30)$$

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

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

and similarly for stem sequences [?].

The TKFST model on a two-branch phylogeny

While the idea of composing conditionally-normalized models on a tree is intuitive, the resulting models can quickly become too complex to deal with by hand, necessitating an algorithm for automated model construction. We constructed the model corresponding to TKFST on the simple the two-branch phylogeny $a \xrightarrow{t} b \xrightarrow{t'} c$. The results, shown below, make clear that by-hand model construction quickly becomes impractical as model complexity or tree size increases.

We explicitly note cases where certain sets of grammar rules can be derived “automatically” from simpler sets, motivating the development of our generic tree transducer composition algorithm.

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 $\beta'_n, \gamma'_n, M'_n$.

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 traceback then can be used to sample the descendant b . This corresponds to a forward-time simulator when we apply the 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

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

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

Table 3. Singlet rule-set for a .

evolutionary intermediate b . Again, this requires only the Inside & stochastic traceback algorithms.

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

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

Pair rules. This section contains two pairwise rule-sets.

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

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

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

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

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

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

Table ??: 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 ?? through ??.

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

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

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

Table 4. Singlet rule-set for b .*TKF Structure Tree singlet rules (c)**Sequence c terminals: $\{y, z\}$.*

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

Table 5. Singlet rule-set for c .

TKF Structure Tree pair rules ($a \xrightarrow{t} b$)				
Sequence a terminals: $\{u, v\}$.			Sequence b terminals: $\{w, x\}$.	
lhs	\rightarrow	rhs	$P(a)$	$P(b a)$
L_{ab}	\rightarrow	$u \ w \ L_{ab}$	$\kappa_l \pi_l(u)$	$(1 - \beta_l) \alpha_l M_l(u, w)$
		$w \ L_{ab}$	1	$\beta_l \pi_l(w)$
		$u \ L_{a*b}$	$\kappa_l \pi_l(u)$	$(1 - \beta_l)(1 - \alpha_l)$
	$ $	$S_{ab} \ L_{ab}$	$\kappa_l \pi_l(S)$	$(1 - \beta_l) \alpha_l$
		$S_b \ L_{ab}$	1	$\beta_l \pi_l(S)$
		$S_a \ L_{a*b}$	$\kappa_l \pi_l(S)$	$(1 - \beta_l)(1 - \alpha_l)$
		ϵ	$1 - \kappa_l$	$1 - \beta_l$
	\rightarrow	$u \ w \ S_{ab} \ x \ v$	$\kappa_s \pi_s(uv)$	$(1 - \beta_s) \alpha_s M_s(uv, wx)$
		$w \ S_{ab} \ x$	1	$\beta_s \pi_s(wx)$
		$u \ S_{a*b} \ v$	$\kappa_s \pi_s(uv)$	$(1 - \beta_s)(1 - \alpha_s)$
L_{a*b}	\rightarrow	$u \ w \ L_{ab}$	$\kappa_l \pi_l(u)$	$(1 - \gamma_l) \alpha_l M_l(u, w)$
		$w \ L_{ab}$	1	$\gamma_l \pi_l(w)$
		$u \ L_{a*b}$	$\kappa_l \pi_l(u)$	$(1 - \gamma_l)(1 - \alpha_l)$
	$ $	$S_{ab} \ L_{ab}$	$\kappa_l \pi_l(S)$	$(1 - \gamma_l) \alpha_l$
		$S_b \ L_{ab}$	1	$\gamma_l \pi_l(S)$
		$S_a \ L_{a*b}$	$\kappa_l \pi_l(S)$	$(1 - \gamma_l)(1 - \alpha_l)$
		ϵ	$1 - \kappa_l$	$1 - \gamma_l$
	\rightarrow	$u \ w \ S_{ab} \ x \ v$	$\kappa_s \pi_s(uv)$	$(1 - \gamma_s) \alpha_s M_s(uv, wx)$
		$w \ S_{ab} \ x$	1	$\gamma_s \pi_s(wx)$
		$u \ S_{a*b} \ v$	$\kappa_s \pi_s(uv)$	$(1 - \gamma_s)(1 - \alpha_s)$
S_{a*b}	$ $	L_{ab}	$1 - \kappa_s$	$1 - \gamma_s$

Table 6. Pair rule-set for $a \xrightarrow{t} b$ branch. Requires Table ?? and Table ??.

TKF Structure Tree pair rules ($b \xrightarrow{t'} c$)

Sequence b terminals: $\{w, x\}$. *Sequence c terminals:* $\{y, z\}$.

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

Table 7. Pair rule-set for $b \xrightarrow{t'} c$ branch. Requires Table ?? and Table ??.

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

Table 8. Triplet rule-set for “*abc*” states (post emission at *c*) on $a \xrightarrow{t} b \xrightarrow{t'} c$ tree. Requires Tables ?? through ??.

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

Table 9. 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 ?? through ??.

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

Table 10. 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 ?? through ??.

5 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.

5.1 Automated grammar construction

We implemented our model construction algorithm on the star phylogeny (Figure ??) 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 a multi-sequence model of three extant sequences (Figure ??). It removes the useless windback `Null` states described in the paper and introduces effective direct transitions caused by bifurcations with possibly-empty children. The package `ComposedTreeTransducer::TripletSCFG` transforms a multi-sequence model created by `ComposedTreeTransducer::FourWayComposedTT` into the corresponding jointly-normalized three-sequence SCFG (Section ??) 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.

5.2 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.

6 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 [?]. 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 [?] from the European rRNA database [?]. 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 [?]. (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 ?? (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 ??, right).

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

7 Terminological asides

This subsection contains some asides about terminology, and can be omitted on a first reading of this paper. The intent of this subsection is to highlight connections to related fields in molecular evolution and computer science.

7.1 Phylogenetic grammars

We here describe some relationships to grammar models in current use in molecular evolution.

Given a singlet transducer, a phylogenetic tree, and a mapping from tree to branch transducers, the transducer composition algorithm returns a corresponding multi-sequence SCFG which models the joint probability distribution $P(X_1, \dots, X_{n+m})$.

Several related bioinformatics grammars can be represented within this multi-SCFG framework, allowing for substantial re-use of prior modeling theory:

- String transducers (which are similar to HMMs) can be viewed as special cases of parse-tree transducers, without the facility to model long-distance correlations (base-pairs) or bifurcated structures.
- Phylo-HMMs and phylo-SCFGs [?] can be viewed as special cases of (respectively) string transducers and parse-tree transducers. (Phylo-HMMs and phylo-SCFGs typically model substitutions but

not indels, and so can be viewed as compositions of a richly-featured singleton transducer with a restricted class of branch transducers.)

- Sakakibara [?] has described “Pair HMMs on tree structures” that are closely related to our parse-tree transducers (though without the composition algorithms for scoring multiple alignments).
- Sakakibara has also described pair stochastic tree-adjoining grammars, or TAGs [?]. TAGs are a formal generalization of SCFGs to mildly context-sensitive grammars [?, ?, ?]. We speculate that the conditionally-normalized transducer analogues of such Pair-TAGs may encompass the phylogenetic model of covariant RNA substitutions and indels on pseudoknotted structures described by [?], just as parse-tree transducers encompass phylo-SCFGs.

7.2 Statistical Alignment

The term **Statistical Alignment** was introduced by Hein [?] to describe DP algorithms for phylogenetic grammars. Although this term has been adopted by others in molecular evolution [?, ?, ?, ?], we’ve avoided it (except in this section) for the following four reasons: (i) Several RNA alignment methods already use another probability distribution that’s completely different to (though equally as valid as) the phylogenetic probability distribution over alignments: namely, the Boltzmann probability distribution over secondary structures [?, ?, ?]. (ii) Many alignment tools make use of prior and posterior distributions over pairwise alignments [?, ?, ?, ?, ?, ?, ?, ?]; all of these are arguably “statistical” alignment tools as well. (iii) Some of the above-cited methods use sum-over-pairs alignment metrics, taking the expectation of such metrics over pairwise alignment posteriors, and finding the multiple alignment that maximizes this expectation [?, ?, ?]; others do progressive multiple alignment using pair grammars [?, ?, ?]. Speculatively, such pair-based heuristics may (in some sense) approximate probabilistic alignment inference with true phylogenetic likelihoods; similar convergences have, at least, been demonstrated between other pair-based heuristics [?]. Thus, even using Hein’s strict definition, it’s possible that a broad range of methods systematically *approximate* “Statistical Alignment”; this cautions against using the term too exclusively. (iv) Even more broadly, many alignment tools make use of statistics to estimate alignment “significance”. In this frequentist approach, probability distributions for alignment scores (particularly Gumbel distributions) are estimated for random sequences. The tail integrals of these distributions then yield P-values for high-scoring database search results [?, ?, ?]. Thus, while the term “Statistical Alignment” has a quite

specific technical meaning in molecular evolution (a meaning which certainly encompasses the algorithms we describe here), the broader bioinformatics literature may have competing claims to the term. In any case, the goal of this work is statistical *reconstruction* of RNA protosequences, rather than alignment *per se*.

7.3 Parse trees and stacks

Grammars are abstract models for languages; finite-state automata are the corresponding abstract models for those languages' parsers. Most of this paper uses the grammatical view of RNA structure evolution; in this short section we digress briefly to mention the automata-theoretic view of these models.

The two non-local features that set SCFGs apart from HMMs are bifurcations and paired emissions [?]. In RNA structure analysis, these features are used to model (respectively) branched-loops and base-pairs.

A stochastic pushdown automaton can also model these features. Such an automaton is essentially an HMM with a LIFO stack (LIFO = "Last In, First Out").

The pushdown automaton can parse two sequences simultaneously, aligning them and their parse trees. It handles bifurcations and paired emissions by pushing symbols onto its stack: a bifurcation by pushing a nonterminal, and a paired emission by pushing a terminal. The pushed terminals and nonterminals are popped off the stack when the machine reaches its **End** state.

The conditionally-normalized version of a pushdown automaton is a pushdown transducer: a machine that absorbs *and* parses an input sequence (or absorbs an already-parsed input sequence) and emits a (parsed) output sequence. This is the machine that we describe as a parse-tree transducer.

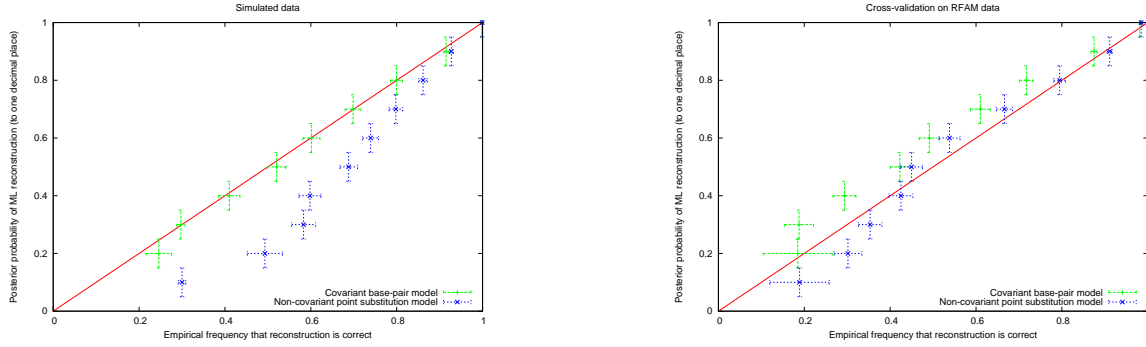


Figure 3. Accuracy of confidence estimates for base-pair reconstruction in simulations (left) and cross-validations on real data (right). Reconstructions were performed using XRATE [?], using both covariant base-pair substitution models and naive, non-covariant, single-base point-substitution models. 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.

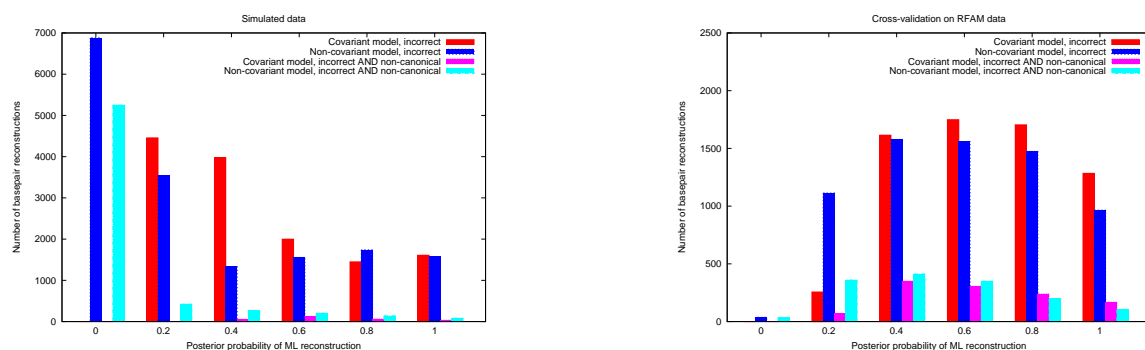


Figure 4. Proportions of reconstructed base-pairs that are incorrect (and the subset of those that are non-canonical) in simulations (left) and cross-validations on real data (right). Base-pair reconstructions that are incorrect AND non-canonical might potentially disrupt a synthetic reconstructed structure. 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 ?? 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.