A Note on Probabilistic Models over Strings: the Linear Algebra Approach

Alexandre Bouchard-Côté

August 29, 2013

Abstract

Probabilistic models over strings have played a key role in developing methods that take into consideration indels as phylogenetically informative events. There is an extensive literature on using automata and transducers on phylogenies to do inference on these probabilistic models, in which an important theoretical question is the complexity of computing the normalization of a class of string-valued graphical models. This question has been investigated using tools from combinatorics, dynamic programming, and graph theory, and has practical applications in Bayesian phylogenetics. In this work, we revisit this theoretical question from a different point of view, based on linear algebra. The main contribution is a set of results based on this linear algebra view that facilitate the analysis and design of inference algorithms on string-valued graphical models. As an illustration, we use this method to give a new elementary proof of a known result on the complexity of inference on the "TKF91" model, a well-known probabilistic model over strings. Compared to previous work, our proving method is easier to extend to other models, since it relies on a novel weak condition, triangular transducers, which is easy to establish in practice. The linear algebra view provides a concise way of describing transducer algorithms and their compositions, opens the possibility of transferring fast linear algebra libraries (for example based on GPUs), as well as low rank matrix approximation methods, to string-valued inference problems.

keywords indel, alignment, probabilistic models, TKF91, string transducers, automata, graphical models, phylogenetics, factor graphs

1 Introduction

This work is motivated by models of evolution that go beyond substitutions by incorporating insertions and deletions as phylogenetically informative events [48, 49, 35, 34, 6]. In recent years, there has been significant progress in turning these models into phylogenetic reconstruction methods that do not assume that the sequences have been aligned as a pre-processing step, and into statistical alignment methods [16, 32, 33, 28, 40, 42, 41, 3]. Joint phylogenetic reconstruction and alignment methods are appealing because they can produce calibrated confidence assessments [42], they leverage more information than pure substitution models [27], and they are generally less susceptible to biases induced by conditioning on a single alignment [54].

Model-based joint phylogenetic methods require scoring each tree in a large collection of hypothesized trees [40]. This collection of proposed trees is generated by a Markov chain Monte Carlo (MCMC) algorithm in Bayesian methods, or by a search algorithm in frequentist methods. Scoring one hypothesized tree then boils down to computing the probability of the observed sequences given the tree. This computation requires summing over all possible ancestral sequences and derivations (a refinement of the notion of alignment, localizing all the insertions, deletions and substitutions on the fixed tree that can yield the observed sequences) [23]. Since the length of the ancestral strings is a priori unknown, these summations range over a countably infinite space, creating a challenging problem. This situation is in sharp contrast to the classical setup where an alignment is fixed, in which case computing the probability of the observed aligned sequences can be done efficiently using the standard Felsenstein recursion [12].

When an alignment is not fixed, the distribution over a single branch is typically expressed as a transducer [16], and the distribution over a tree is obtained by composite transducers and automata [23]. The idea of composite transducers and automata, and their use in phylogenetic, have been introduced and subsequently developed in previous work in the framework of graph representations of transducers and combinatorial methods [23, 17, 18, 19, 30, 9]. Our contribution lies in new proofs, and a theoretical approach purely based on linear algebra. We use this linear algebraic formulation to obtain simple algebraic expressions for marginalization algorithms. In the general cases, our representation provides a slight improvement on the asymptotic worst-case running time of existing transducer algorithms [38]. We also characterize a subclass of problems, which we call *triangular problems*, where the running time can be further improved. We show, for example, that marginalization problems derived from the TKF91 model [48] are triangular. We use this to develop a new complexity analysis of the problem of inference in the TKF91 model. The bounds for the TKF91 model on star trees and perfect trees coincide with those obtained from previous work [31], but our proving method is easier to extend to other models since it only relies on the triangularity assumption rather than on the details of the TKF91 model.

Algorithms for composite phylogenetic automata and transducers have been introduced in [23, 21], generalizing the dynamic programming algorithm of [17] to arbitrary guide trees. Other related work based on combinatorics includes [46, 31, 45, 43, 4]. See [7] for an outline of the area, and [22, 40, 39] for computer implementations. Our work is most closely related to [50, 51], which extends Felsenstein's algorithm to transducers and automata. They use a different view based on partial-order graphs to represent ensemble profiles of ancestral sequences.

While the exact methods described here and in previous work are exponential in the number of taxa, they can form the basis of efficient approximation methods, for example via existing Gibbs sampling methods [25]. These methods are based on auxiliary variables, which augment the state space of the MCMC algorithm, for example with internal sequences [20]. The running time then depends on the number of taxa involved in the local tree perturbation. This number can be as small as two, but there is a trade-off between the mixing rate of the MCMC chain and the computational cost of each move [40].

Note that the matrices used in this work should not be confused with the rate matrices and their marginal transition probabilities. The latter are used in classical phylogenetic reconstruction setups where the alignment is assumed to be fixed. Our proving methods are more closely related to previous work on representation of stochastic automata network [14, 29], where interactions represent synchronization steps rather than string evolution, and therefore behave differently.

2 Formal description of the problem

2.1 Notation

We use τ to denote a rooted phylogeny, with topology $(\mathscr{V}, \mathscr{E})$.[1] The root of τ is denoted by Ω; the parent of $v \in \mathscr{V}, v \neq \Omega$, by $\mathrm{pa}(v) \in \mathscr{V}$; the branch lengths by $b(v) = b(\mathrm{pa}(v) \to v)$; and the set of leaves by $\mathscr{L} \subset \mathscr{V}$.

The models we are interested in are tree-shaped graphical models [26, 2, 1] where nodes are string-valued random variables X_v over a finite alphabet Σ (for example a set of nucleotides or amino acids). There is one such random variable for each node in the topology of the phylogeny, $v \in \mathscr{V}$, and the variable X_v can be interpreted as an hypothesized biological sequence at one point in the phylogeny.

[1] In reversible models such as the TKF91 model, an arbitrary root can be selected without changing the likelihood. In this case, the root is used for computational convenience and has no special phylogenetic meaning.

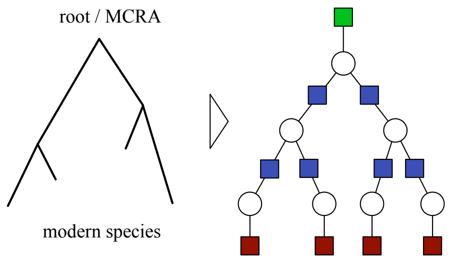


Figure 1: Factor graph construction (right) derived from a phylogenetic tree (left): in green, the unary factor at the root capturing the initial string distribution; in blue, the binary factors capturing string mutation probabilities; and in red, the unary factor at the leaves, which are indicator functions on the observed strings.

Each edge $(\mathrm{pa}(v) \to v)$ denotes evolution from one species to another, and is associated with a conditional probability $\mathbb{P}_\tau(X_v = s' | X_{\mathrm{pa}(v)} = s) = \theta_v(s, s')$, where $s, s' \in \Sigma^*$ are strings (finite sequences of characters from Σ). These conditional probabilities can be derived from the marginals of continuous time stochastic process [48, 23, 35, 34], or from a parametric family [28, 42, 40, 41]. We also denote the distribution at the root by $\mathbb{P}_\tau(X_\Omega = s) = \theta(s)$, which is generally set to the stationary distribution in reversible models or to some arbitrary initial distribution otherwise. Usually, only the sequences at the terminal nodes are observed, an event that we denote by $\mathscr{Y} = (X_v = x_v : v \in \mathscr{L})$.

We denote the probability of the data given a tree by $\mathbb{P}_\tau(\mathscr{Y})$. The computational bottleneck of model-based methods is to repeatedly compute $\mathbb{P}_\tau(\mathscr{Y})$ for different hypothesized phylogenies τ. This is the case not only in maximum likelihood estimators, but also in Bayesian methods, where a ratio of the form $\mathbb{P}_{\tau'}(\mathscr{Y})/\mathbb{P}_\tau(\mathscr{Y})$ is the main contribution to the Metropolis-Hastings ratio driving MCMC approximations.

While it is more intuitive to describe a phylogenetic tree using a Bayesian network, it is useful when developing inference algorithms to transform these Bayesian networks into equivalent factor graphs [2]. A factor graph encodes a weight function over a space $\mathscr{X}$ assumed to be countable in this work. For example, this weight function could be a probability distribution, but it is convenient to drop the requirement that it sums to one. In phylogenetics, the space $\mathscr{X}$ is a tuple of $|\mathscr{V}|$ strings, $\mathscr{X} = (\Sigma^*)^{|\mathscr{V}|}$.

A factor graph (see Figure 1 for an example) is based on a specific bipartite graph, built by letting the first bipartite component correspond to $\mathscr{V}$ (the nodes denoted by circles in the figure), and the second bipartite component, to a set of *potentials* or *factors* $\mathscr{W}$ described shortly (the nodes denoted by squares in the figure).

We put an edge in the factor graph between $w \in \mathscr{W}$ and $v \in \mathscr{V}$ if the value taken by w depends on the string at node v. Formally, a potential $w \in \mathscr{W}$ is a map taking elements of the product space, $(x_v : v \in \mathscr{V}) \in \mathscr{X}$, and returning a non-negative real number. However the value of the factors we consider only depend on a small subset of the variables at a time. More precisely, we will use two types of factors: unary factors, which depend on a single variable, and binary factors, which depend on two variables. We use the abbreviation SFG for such String-valued Factor Graph.

In order to write a precise expression for the probability of the data given a tree, $\mathbb{P}_\tau(\mathscr{Y})$, we construct a factor graph (shown in Figure 1) with three types of factors: one unary potential $w_\Omega(s) = \theta(s)$ at the root modeling the initial string distribution; binary factors $w_{v,v'}(s, s') = \theta_{v'}(s, s')$ capturing string mutation probabilities between pairs of species connected by an edge $(v \to v')$ in the phylogenetic tree; and finally, unary factors $w_v(s) = \mathbf{1}[s = x_v]$ attached to each leaf $v \in \mathscr{L}$, which are Dirac deltas on the observed strings.

From this factor graph, the likelihood can be written as:

$$\mathbb{P}_\tau(\mathscr{Y}) = \sum_{(s_v \in \Sigma^* : v \in \mathscr{V}) \in (\Sigma^*)^{|\mathscr{V}|}} w_\Omega(s_\Omega) \left(\prod_{v \in \mathscr{L}} w_v(s_v) \right) \left(\prod_{(v \to v') \in \mathscr{E}} w_{v,v'}(s_v, s_{v'}) \right).$$

For any functions $f_i : \mathcal{X}_i \to \mathbb{R}^+$ over a countable spaces $\mathcal{X}$, we let $\prod_i f_i$ denote their pointwise product, and we let $\|f\|$ denote the sum of the function over the space,

$$\|f\| = \sum_{x \in \mathcal{X}} f(x).$$

The likelihood can then be expressed as follows:

$$\mathbb{P}_\tau(\mathscr{Y}) = \left\| \prod_{w \in \mathscr{W}} w \right\|. \tag{1}$$

So far, we have seen each potential w as a black box, focussing instead on how they interact with each other. In the next section, we describe the form assumed by each individual potential.

2.2 Weighted automata and transducers

Weighted automaton (respectively, transducers) is a classical way to define a function from the space of strings (respectively, the space of pairs of strings) to the non-negative reals [44]. At a high level, weighted automata and transducers proceed by extending a simpler weight function defined over a finite state space $\mathcal{Q}$ into a function over the space of strings. This is done via a collection of paths (lists of variable length) in $\mathcal{Q}$.

We start with the case of automata (in the terminology of SFG introduced in the previous section, unary potentials). First, we designate one state in the state space $\mathcal{Q}$ to be the *start state* and one to be the *end state* (we need only consider one of each without loss of generality). We define a *path* as a sequence of states $q_i \in \mathcal{Q}$ and *emissions* $\hat{\sigma}_i \in \hat{\Sigma} = \Sigma \cup \{\epsilon\}$, starting at the start state and ending at the stop state: $p = (q_0, \hat{\sigma}_1, q_1, \hat{\sigma}_2, q_2, \ldots, \hat{\sigma}_n, q_n)$. Here the set of possible emissions $\hat{\Sigma}$ is the set of characters extended with the *empty emission* ϵ.[2]

Let us now assign a non-negative number to each triplet formed by a transition (pair of state $q, q' \in \mathcal{Q}$) and emission $\hat{\sigma} \in \hat{\Sigma}$.[3] We call this map $w : \hat{\Sigma} \times \mathcal{Q}^2 \to [0, \infty)$, a parameter specified by the user.[4]

Next, we extend the input space of w in two steps. First, if w takes a path p as input, we define $w(p)$ as the product of the weights of the consecutive triplets $(q_i, \hat{\sigma}_{i+1}, q_{i+1})$ in p. Second, if w takes a string $s \in \Sigma^*$ as input, we define $w(s)$ as the sum of the weights of the paths p emitting s, where a path p emits a string s if the list of characters in p, omitting ϵ, is equal to s.

For transducers (binary potentials), we modify the above definition as follows. First, emissions become pairs of symbols in $\hat{\Sigma} \times \hat{\Sigma}$. Second, paths are similarly extended to emit pairs of symbols at each transition, $p = (q_0, (\hat{\sigma}_1, \hat{\sigma}'_1), q_1, (\hat{\sigma}_2, \hat{\sigma}'_2), q_2, \ldots, (\hat{\sigma}_n, \hat{\sigma}'_n), q_n)$. Finally a path emits a pair of strings (s, s') if the concatenation of the first characters of the emissions, $\hat{\sigma}_1, \hat{\sigma}_2, \ldots, \hat{\sigma}_n$, is equal to s after removing the ϵ symbols, and similarly for s' with the second characters of each emission, $\hat{\sigma}'_1, \hat{\sigma}'_2, \ldots, \hat{\sigma}'_n$. For example, $p = (q_0, (\mathrm{a}, \epsilon), q_1, (\mathrm{b}, \mathrm{c}), q_2, (\epsilon, \epsilon), q_n)$ emits the pair of strings (ab, c).

The *normalization* of a transducer or automata is the sum of all the valid paths' weights, $Z = \|w\|$. We set the normalization to positive infinity when the sum diverges.

We now have reviewed all the ingredients required to formalize the problem we are interested in: **Problem description:** Computing Equation (1), where each $w \in \mathscr{W}$ is a weighted automaton or transducer organized into a tree-shaped SFG as defined in Section 2.1.

[2] Note that we use the convention of using $\hat{\sigma}$ when the character is an element of the extended set of characters $\hat{\Sigma}$.

[3] In technical terms, we use the Mealy model.

[4] The inverse problem, learning the values for this map is also of interest [24], but here we focus on the forward problem.

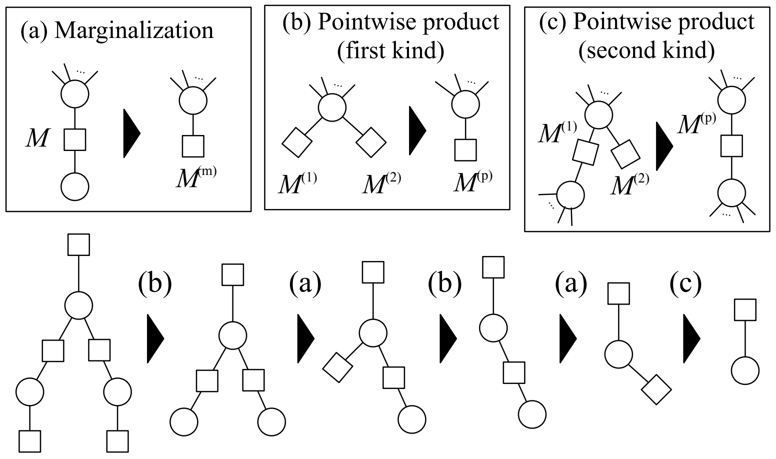

(a) Marginalization
M
M(m)
(b) Pointwise product (first kind)
M(1)
M(2)
M(p)
(c) Pointwise product (second kind)
M(1)
M(2)
M(p)
(b)
(a)
(b)
(a)
(c)

Figure 2: Top: The tree graph transformations used in the elimination algorithm. Bottom: An example of the application of these three operations on a graph induced by a phylogenetic tree.

3 Background

In the previous section, we have shown how the normalization of a single factor is defined (how to compute it in practice is a another issue, that we address in Section 5). In this section, we review how to transform the problem of computing the normalization of a full factor graph, Equation (1), into the problem of finding the normalization of a single unary factor. This is done using the elimination algorithm [2], a classical method used to break complex computations on graphical models into a sequence of simpler ones.

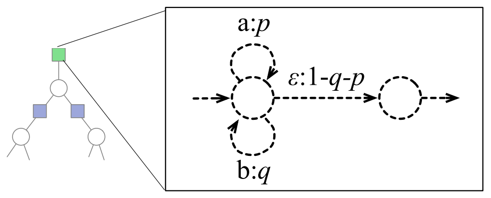
The elimination algorithm is typically applied to find the normalization of factor graphs with finite-valued or Gaussian factors. Since our factors take values in a different space, the space of strings, it is useful in this section to take a different perspective on the elimination algorithm: instead of viewing the algorithm as exchanging messages on the factor graph, we view it as applying transformation on the topology of the graph. These operations alter the normalization of individual potentials, but, as we will show, they keep the normalization of the whole factor graph invariant, just as in standard applications of the elimination algorithm.

The elimination algorithm starts with the initial factor graph, and eliminates one leaf node at each step (either a variable node or a factor node, there must be at least one of the two types). This process creates a sequence of factor graphs with one less nodes at each step, until there are only two nodes left (one variable and one unary factor). While the individual factors are altered by this process, the global normalization of each intermediate factor graph remains unchanged.

We show in Figure 2 an example of this process, and a classification of the operations used to simplify the factor graph. We call the operations corresponding to a factor eliminations a *pointwise product* (Figure 2(b,c)), and the operations corresponding to variable eliminations, *marginalization* (Figure 2(a)). We further define two kinds of pointwise products: those where the variable connected to the eliminated factor is also connected to a binary factor (the first kind, shown in Figure 2(b)), and the others (second kind, shown in Figure 2(c)).

Let $\mathscr{W}$ denote a set of factors, and let $\mathscr{W}'$ denote a new set of factors obtained by applying one of the three operations described above. For the elimination algorithm to be correct, the following invariant should hold [2]:

$$\left\| \prod_{w\in\mathscr{W}} w \right\| = \left\| \prod_{w'\in\mathscr{W}'} w' \right\|.$$



$$M_{\mathrm{a}} = \begin{pmatrix} p & 0 \\ 0 & 0 \end{pmatrix}$$

$$M_{\mathrm{b}} = \begin{pmatrix} q & 0 \\ 0 & 0 \end{pmatrix}$$

$$M_{\epsilon} = \begin{pmatrix} 0 & 1-p-q \\ 0 & 0 \end{pmatrix}$$

Figure 3: An initial geometric length distribution unary factor. On the left, we show the standard graph-based automaton representation, where states are dashed circles (to differentiate them from nodes in SFGs), and arcs are labelled by emissions and weights. Zero weight arcs are omitted. The start and stop state are indicated by inward and outward arrows, respectively on the first and second state here. On the right, we show the indexed matrix representation of the same unary potential.

4 Indexed matrices

In this section, we describe the representation that forms the basis of our normalization method and establish the main properties of this representation. We use the TKF91 model [48] as a running example to illustrate the construction.

4.1 Unary factors

We assume without loss of generality that the states of the automata are labelled by the integers $1, \ldots, K$, and that state 1 is the start state, and K, the stop state.

We represent an automaton as a character-indexed collection, M, of $K \times K$ matrices, where K is the size of the state space of the automaton, and the index runs over the extended characters:

$$M = \left(M_{\hat{\sigma}} \in \mathbb{M}_K \left(\mathbb{R}^+ \right) \right)_{\hat{\sigma} \in \hat{\Sigma}}$$

Each entry $M_{\hat{\sigma}}(k, k')$ encodes the weight of transitioning from states k to k' while emitting $\hat{\sigma} \in \hat{\Sigma}$.

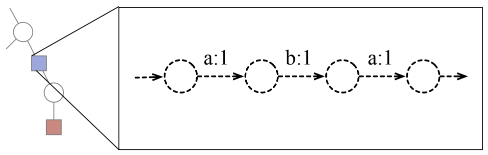
An indexed collection M specifies a corresponding weight function $w_M : \Sigma^* \to [0, \infty)$ as follows: let p denote a valid path, $p = q_0, \hat{\sigma}_1, q_1, \hat{\sigma}_2, q_2, \ldots, \hat{\sigma}_n, q_n$, and set

$$w_M(p) = \phi \left(\prod_i^n M_{\hat{\sigma}_i} \right),$$

where ϕ selects the entry $(1, K)$ corresponding to the product of weights starting from the initial and ending in the final state, $\phi(M) = M(1, K)$. The product denotes a matrix multiplication over the matrices indexed by the emissions in p, in the order of occurrence.

In the TKF91 model for example, an automaton is needed to model the geometric root distribution, $\theta(s)$. We have in Figure 3 the classical state diagram as well as the indexed matrix representation, in the case where the length distribution has mean $1/(1-q-p)$, and the proportion of 'a' symbols versus 'b' symbols is p/q.

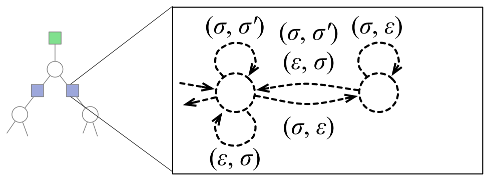
Another important example of an automaton is one that gives a weight of one to a single observed string, and zero otherwise: $w_v(s) = \mathbf{1}[s = x_v]$. We show the state diagram and the indexed matrix representation in Figure 4. Note that the indexed matrices are triangular in this case, a property that will have important computational consequence in Section 6.



$$M_{\mathrm{a}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad M_{\mathrm{b}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$M_{\epsilon} = \mathbf{0}$$

Figure 4: A unary factor encoding a string indicator function. In this example, only the string 'aba' is accepted (i.e. the string 'aba' is the only string emitted with positive weight).



$$\begin{aligned} M_{\sigma,\sigma'} &= \frac{\alpha\lambda}{\mu}\begin{pmatrix} (1-\beta) & 0 \\ (1-\gamma) & 0 \end{pmatrix} P_{\sigma,\sigma'} \\ M_{\sigma,\epsilon} &= \frac{\lambda(1-\alpha)}{\mu}\begin{pmatrix} 0 & (1-\beta) \\ 0 & (1-\gamma) \end{pmatrix} \\ M_{\epsilon,\sigma} &= \pi_\sigma \begin{pmatrix} \beta & 0 \\ \gamma & 0 \end{pmatrix} \end{aligned}$$

Figure 5: A binary factor encoding a TKF91 model marginal.

4.2 Binary factors

We now turn to the weighted transducers, which we viewed as a collection of matrices as well, but this time with an index running over pairs of symbols $(\hat{\sigma}, \hat{\sigma}') \in \hat{\Sigma}$:

$$M = \left(M_{\hat{\sigma},\hat{\sigma}'} \in \mathbb{M}_K\left(\mathbb{R}^+\right) \right)_{\hat{\sigma},\hat{\sigma}' \in \hat{\Sigma}}.$$

Note that in the context of transducers, epsilons need to be kept in the basic representation, otherwise transducers could not give positive weights to pairs of strings where the input has a different length than the output.

In phylogenetics, transducers represent the key component of the probabilistic model: the probability of a string given its parent, $w_{v,v'}(s,s') = \theta_{v'}(s,s')$. For example, in the TKF91 model, this probability is determined by three parameters: an insertion rate $\lambda > 0$, a deletion rate $\mu > 0$, and a substitution rate matrix Q. Given a branch of length t, the probability of all derivations between s and s' can be marginalized using the transducer shown in Figure 5, where the values of α, β and P are obtained by solving a system of differential equations yielding [48], for all $\sigma, \sigma' \in \Sigma$:

$$\begin{aligned} \alpha(t) &= \exp(-\mu t) \\ \beta(t) &= \frac{\lambda(1-\exp((\lambda-\mu)t))}{\mu - \lambda\exp((\lambda-\mu)t)} \\ \gamma(t) &= 1 - \frac{\mu\beta(t)}{\lambda(1-\alpha(t))} \\ P_t(\sigma,\sigma') &= \big(\exp(tQ)\big)(\sigma,\sigma'), \end{aligned}$$

and π_σ is the stationary distribution of the process with rate matrix Q.[5] While the standard description of the TKF91 model uses three states, our formalism allows us to express it with two states, with a state diagram shown in Figure 5. Note also that for simplicity we have left out the description of the 'immortal link' [48], but it can be handled without increasing the number of states. This is done by introducing an extra symbol $\# \notin \hat{\Sigma}$ flanking the end of the sequence.

[5]The rate matrix Q comes from a substitution model (we will assume the Jukes-Cantor model for simplicity, but all our results carry over to the General Time Reversible models [13]).

5 Operations on indexed matrices

In this section, we give a closed-form expression for all the operations described in Section 2. We start with the simplest operations, epsilon-removal and normalization, and then cover marginalization and pointwise products.

5.1 Unary operations

While epsilon symbols are convenient when defining new automata, some of the algorithms in the next section assume that there are no epsilon transitions of positive weights. Formally, a factor M is called *epsilon-free* if $M_\epsilon = 0$. Fortunately, converting an arbitrary unary factor into an epsilon-free factor, can be done using the well known generalization of the geometric series formula to matrices. But in order to apply this result to our situation, we need to assume that in all positive weight path, the last emission is never an epsilon. Fortunately, this can be done without loss of generality by adding a boundary symbol at the end of each string. We assume this thorough the rest of this paper and get:

Proposition 1. *Let M denote a unary factor such that each of the eigenvalues λ_i of M_ϵ satisfies $|\lambda_i| < 1$. Then the transformed factor $M^{(\mathrm{f})}$ defined as:*

$$\begin{aligned} M^{(\mathrm{f})}_{\hat\sigma} &= \begin{cases} 0 & \textit{if } \hat\sigma = \epsilon \\ M^*_\epsilon M_{\hat\sigma} & \textit{otherwise} \end{cases} \\ M^* &= (1-M)^{-1} \end{aligned}$$

is epsilon-free and assigns the same weights to strings, i.e. $w_M(s) = w_{M^{(\mathrm{f})}}(s)$ for all $s \in \Sigma^*$.

We show a proof of this elementary result to illustrate the conciseness of our indexed matrices notation:

Proof. Note first that the condition on the eigenvalues implies that $(1 - M)$ is invertible.

To prove equivalence of the weight functions w_M and $w_{M^{(\mathrm{f})}}$, let $s \in \Sigma^*$ be a string of length N. We need to show that the epsilon-removed automaton $M^{(\mathrm{f})}_\sigma$ assigns the same weights $w_{M^{(\mathrm{f})}}(s)$ to strings as the original automaton $M_{\hat\sigma}$:

$$\begin{aligned} w_M(s) &= \phi\left(\sum_{\hat s \rightsquigarrow s}\prod_{\hat\sigma\in\hat s} M_{\hat\sigma}\right) \\ &= \phi\left(\sum_{(k_1,\dots,k_N)\in\mathbb{N}^N} M_\epsilon^{k_1} M_{s_1} M_\epsilon^{k_2} M_{s_2}\cdots M_\epsilon^{k_N} M_{s_1}\right) \\ &= \phi\left(\prod_{n=1}^{N}\left(\sum_{m=0}^{\infty} M_\epsilon^m\right) M_{s_n}\right) \\ &= \phi\left(\prod_{n=1}^{N} M^{(\mathrm{f})}_{s_n}\right) \\ &= w_{M^{(\mathrm{f})}}(s). \end{aligned}$$

Here we used the nonnegativity of the weights, which implies that if the automaton has a finite normalization, then the infinite sums above are absolutely convergent, and can therefore be rearranged. □

We also need the following elementary result on the normalizer of arbitrary automata.

Proposition 2. *Let M denote a unary potential, and define the matrix:*

$$\tilde{M} = \sum_{\hat{\sigma}\in\hat{\Sigma}} M_{\hat{\sigma}},$$

with eigenvalues λ_i. Then the normalizer Z_M of the unary potential is given by:

$$Z_M = \begin{cases} \phi\left(\tilde{M}^*\right) & \text{if for all } i, |\lambda_i| < 1 \\ +\infty & \text{otherwise,} \end{cases}$$

where $\phi(M) = M(1, K)$.

Proof. Note first that for all indexed matrices M, we have the following identity:

$$\sum_{(\hat{\sigma}_1,\dots,\hat{\sigma}_N)\in\hat{\Sigma}^L} \prod_{n=1}^{N} M_{\sigma_n} = \left(\sum_{\sigma\in\hat{\Sigma}} M_{\hat{\sigma}}\right)^N = \tilde{M}^N.$$

Therefore by decomposing the set of all paths by their path lengths N, we obtain:

$$\begin{aligned} Z_M &= \phi\left(\sum_{N=0}^{\infty} \sum_{(\hat{\sigma}_1,\dots,\hat{\sigma}_N)\in\hat{\Sigma}^L} \prod_{n=1}^{N} M_{\sigma_n}\right) \\ &= \phi\left(\sum_{N=0}^{\infty} \tilde{M}^N\right). \end{aligned}$$

The infinite sum in the last line converges to M^* if and only if the eigenvectors satisfy $|\lambda_i| < 1$, and diverges to $+\infty$ otherwise since the weights are non-negative.

□

5.2 Binary operations

We now describe how the operations of marginalization and pointwise product can be efficiently implemented using our framework. We begin with the marginalization operation.

The marginalization operation that we first focus on, shown in Figure 2, takes as input a binary factor M, and returns a unary factor $M^{(\mathrm{m})}$. A precondition of this operation is that one of the nodes connected to M should have no other potentials attached to it. We use the variable σ' for the values of that node. This node is eliminated from the factor graph after applying the operation. The other node is allowed to be attached to other factors, and its values are denoted by σ.

Proposition 3. *If M is a binary factor, then the unary factor $M^{(\mathrm{m})}$ defined as:*

$$M^{(\mathrm{m})}_{\hat{\sigma}} = \sum_{\hat{\sigma}'\in\hat{\Sigma}} M_{\hat{\sigma},\hat{\sigma}'}$$

satisfies $w_{M^{(\mathrm{m})}}(s) = \sum_{s'\in\Sigma^*} w_M(s, s')$ for all $s \in \Sigma^*$.

Proof. We first note that for any matrices $A_{\hat{t}}, \hat{t} \in \hat{\Sigma}^*$ with non-negative entries, we have

$$\sum_{t\in\Sigma^*} \sum_{N=0}^{\infty} \sum_{\hat{t}\rightsquigarrow_N t} A_{\hat{t}} = \sum_{N=0}^{\infty} \sum_{\hat{t}\in\hat{\Sigma}^N} A_{\hat{t}},$$

where we write $\hat{s} \rightsquigarrow_L s$ if $\hat{s}$ has length L and $\hat{s} \rightsquigarrow s$.

We now fix $s \in \Sigma$. Applying the result above, with

$$A_{\hat{s}'} = \sum_{\hat{s} \rightsquigarrow_N s} \prod_{n=1}^{N} M_{\hat{s}_n, \hat{s}'_n},$$

for all $\hat{s}'$, and where $N = |\hat{s}'|$, we get:

$$\begin{aligned}
\sum_{s' \in \Sigma^*} w_M(s, s') &= \phi\left(\sum_{s' \in \Sigma^*} \sum_{N=0}^{\infty} \sum_{\hat{s} \rightsquigarrow_N s} \sum_{\hat{s}' \rightsquigarrow_N s'} \prod_{n=1}^{N} M_{\hat{s}_n, \hat{s}'_n} \right) \\
&= \phi\left(\sum_{s' \in \Sigma^*} \sum_{N=0}^{\infty} \sum_{\hat{s}' \rightsquigarrow_N s'} A_{\hat{s}'} \right) \\
&= \phi\left(\sum_{N=0}^{\infty} \sum_{\hat{s}' \in \hat{\Sigma}^N} A_{\hat{s}'} \right) \\
&= \phi\left(\sum_{N=0}^{\infty} \sum_{\hat{s} \rightsquigarrow_N s} \prod_{n=1}^{N} \sum_{\hat{\sigma} \in \hat{\Sigma}} M_{\hat{s}_n, \hat{\sigma}} \right) \\
&= w_{M^{(\mathrm{m})}}(s).
\end{aligned}$$

□

Next, we move to the pointwise multiplication operations, where the epsilon transitions need a special treatment. In the following, we use the notation $\{A|B|C|\dots\}$ to denote the tensor product of the matrices A, B, C, $\dots$.[6]

Proposition 4. *If $M^{(1)}$ and $M^{(2)}$ are two epsilon-free unary factors, then the unary factor $M^{(\mathrm{p})}$ defined as:*

$$M^{(\mathrm{p})}_{\alpha} = \left\{ M^{(1)}_{\sigma} \middle| M^{(2)}_{\sigma} \right\} \tag{2}$$

is epsilon-free and satisfies:

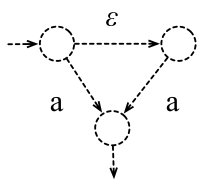
$$w_{M^{(1)}}(s) \cdot w_{M^{(2)}}(s) = w_{M^{(\mathrm{p})}}(s) \tag{3}$$

for all $s \in \Sigma^*$.

Proof. Clearly, $M^{(\mathrm{p})}_{\epsilon}$ is equal to the zero matrix, so $M^{(\mathrm{p})}$ is epsilon-free. To show that it satisfies Equation 3, we first note that since $M^{(i)}$ is epsilon free,

$$\sum_{\hat{s} \rightsquigarrow s} \prod_{\hat{\sigma} \in s} M^{(i)}_{\hat{\sigma}} = \prod_{\sigma \in s} M^{(i)}_{\sigma},$$

[6]Note that we avoided the standard tensor product notation $\otimes$ because of a notation conflict with the automaton and transducer literature, in which $\otimes$ denotes multiplication in an abstract semi-ring (the generalization of normal multiplication, $\cdot$ used here). The operator $\otimes$ is also often overloaded to mean the product or concatenation of automata or transducers, which is not the same as the *pointwise* product as defined here.



$$M_{\mathrm{a}} = \begin{pmatrix} 0 & 0 & \alpha \\ 0 & 0 & \alpha \\ 0 & 0 & 0 \end{pmatrix}$$

$$M_{\epsilon} = \begin{pmatrix} 0 & \alpha & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Figure 6: Counter-example used to illustrate that the epsilon-free condition in Proposition 4 is necessary. Here α is an arbitrary constant in the open interval $(0, 1)$.

for $i \in \{1, 2\}$. Hence:

$$\begin{aligned} w_{M^{(1)}}(s) \cdot w_{M^{(2)}}(s) &= \phi\left(\sum_{\hat{s} \rightsquigarrow s} \prod_{\hat{\sigma} \in s} M^{(1)}_{\hat{\sigma}}\right) \phi\left(\sum_{\hat{s}' \rightsquigarrow s} \prod_{\hat{\sigma}' \in s'} M^{(1)}_{\hat{\sigma}'}\right) \\ &= \phi\left(\prod_{\sigma \in s} M^{(1)}_{\sigma}\right) \phi\left(\prod_{\sigma \in s} M^{(2)}_{\sigma}\right) \\ &= \phi\left(\prod_{\sigma \in s} \left\{ M^{(1)}_{\sigma} \middle| M^{(2)}_{\sigma} \right\}\right) \\ &= w_{M^{(\mathrm{p})}}(s), \end{aligned}$$

for all $s \in \Sigma^*$. □

Note that the epsilon-free condition is necessary. Consider for example the unary potential M over $\Sigma = \{\mathrm{a}\}$ shown in Figure 6. We claim that this factor M (which has positive epsilon transitions) does not satisfy the conclusion of Proposition 4. Indeed, for $s = \mathrm{a}$, we have that $(w_M(s))^2 = \alpha^4 + 2\alpha^3 + \alpha^2$, but $w_{M^{(\mathrm{p})}}(s) = \alpha^4 + \alpha^2$, where $M^{(\mathrm{p})}$ is defined as in Equation 2.

We now turn to pointwise products of the second kind, where an automaton is pointwise multiplied with a transducer. The difference is that all epsilons cannot be removed from a transducer: without emissions of the form $(\sigma, \epsilon), (\epsilon, \sigma)$, transducers would not have the capacity to model insertions and deletions. Fortunately, as long as the automaton $M^{(2)}$ is epsilon free, pointwise multiplications of the second kind can be implemented as follows:

Proposition 5. *If $M^{(2)}$ is epsilon free and $M^{(1)}$ is any transducer, then the transducer $M^{(\mathrm{p})}$ defined by:*

$$M^{(\mathrm{p})}_{\hat{\sigma},\hat{\sigma}'} = \begin{cases} \left\{ M^{(1)}_{\hat{\sigma},\hat{\sigma}'} \middle| M^{(2)}_{\hat{\sigma}} \right\} & \textit{if } \hat{\sigma} \neq \epsilon \\ \left\{ M^{(1)}_{\hat{\sigma},\hat{\sigma}'} \middle| I \right\} & \textit{otherwise} \end{cases}$$

satisfies

$$w_{M^{(1)}}(s, s') \cdot w_{M^{(2)}}(s) = w_{M^{(\mathrm{p})}}(s, s')$$

for all $s, s' \in \Sigma^*$.

Proof. Note first that since $M^{(2)}$ is epsilon-free, we have

$$\prod_{\alpha \in s} M^{(2)}_{\alpha} = \prod_{n=1}^{N} \begin{cases} I & \text{if } \hat{s}_n = \epsilon \\ M^{(2)}_{\hat{s}_n} & \text{otherwise} \end{cases}$$

for all $N \geq |s|$, $\hat{s} \rightsquigarrow_N s$. Moreover, for all matrix A_i, $\phi(\sum_i A_i) = \sum_i \phi(A_i)$, hence:

$$\begin{aligned}
w^{(1)}(s,s') \cdot w^{(2)}(s) &= \phi\left(\sum_{N=0}^{\infty} \sum_{\hat{s} \rightsquigarrow_N s} \sum_{\hat{s}' \rightsquigarrow_N s} \prod_{n=1}^{N} M^{(1)}_{\hat{s}_n, \hat{s}'_n}\right) \phi\left(\prod_{\sigma \in s} M^{(2)}_{\sigma}\right) \\
&= \sum_{N=0}^{\infty} \sum_{\hat{s} \rightsquigarrow_N s} \sum_{\hat{s}' \rightsquigarrow_N s} \phi\left(\prod_{n=1}^{N} M^{(1)}_{\hat{s}_n, \hat{s}'_n}\right) \phi\left(\prod_{n=1}^{N} \left\{ \begin{array}{ll} I & \text{if } \hat{s}_n = \epsilon \\ M^{(2)}_{\hat{s}_n} & \text{otherwise} \end{array} \right.\right) \\
&= \sum_{N=0}^{\infty} \sum_{\hat{s} \rightsquigarrow_N s} \sum_{\hat{s}' \rightsquigarrow_N s} \phi\left(\prod_{n=1}^{N} M^{(\mathrm{p})}_{\hat{s}_n, \hat{s}'_n}\right) \\
&= w_{M^{(\mathrm{p})}}(s,s').
\end{aligned}$$

□

Now that we have a simple expression for all of the operations required by the elimination algorithm on string-valued graphical models, we turn to the problem of analyzing the time complexity of exact inference in SFGs.

6 Complexity analysis

In this section, we start by demonstrating the gains in efficiency brought by our framework in the general case, that is without assuming any structure on the potentials. We then show that by adding one extra assumption on the factors, triangularity, we get further efficiency gains. We conclude the section by giving several examples where triangularity holds in practice.

6.1 General SFGs

Here we compare the local running-time of the indexed matrix representation with previous approaches based on graph algorithms [38]. By local, we mean that we do the analysis for a single operation at the time. We present global complexity analyses for some important special cases in the next section. The results here are stated in terms of K, the size of an individual transducer, but note that in sequence alignment problems, K grows in terms of L because of potentials at the leaves of the tree. This is explained in more details in the next section.

Normalization: Previous work has relied on generalizations of shortest path algorithms to Conway semirings [10], which run in time $O(K^3)$. In contrast, since our algorithm is expressed in terms of a single matrix inversion, we achieve a running time of $O(K^\omega)$, where $\omega < 2.3727$ is the exponent of the asymptotic complexity of matrix multiplication [53]. Note that even though the size of the original factors' state spaces may be small (for example of the order of the length of the observed sequences), the size of the intermediate factors created by the tensor products used in the elimination algorithm makes K grow quickly.

Epsilon-removal: Classical algorithms are also cubic [37]. Since we perform epsilon removal using one matrix multiplication followed by one matrix inversion, we achieve the same speedup of $O(K^{3-\omega})$. Note however that this operation does not preserve sparsity in the general case, both with the classical and the proposed method. This issue is addressed in the next section.

Pointwise products: The operation corresponding to pointwise products in the previous transducer literature is the *intersection* operation [11, 38]. In this case, the complexity of our algorithm is the same as previous work, but our algorithm is easier to implement when one has access to a matrix library. Moreover, tensor products preserve sparsity. More precisely, we use the following simple results: if A has k nonzero entries, and B, l nonzero entries, then forming the tensor product takes time kl. We assume throughout this section that matrices are stored in a sparse representation.

Note that these running times scale linearly in $|\Sigma|$ for operations on unary factors, and quadratically in $|\Sigma|$ for operations on binary factors.

6.2 Triangular SFGs

When executing the elimination algorithm on SFGs, the intermediate factors produced by pointwise multiplications are fed into the next operations, augmenting the size of the matrices that need to be stored. We show in this section that by making one additional assumption, part of this growth can be managed using sparsity and properties of tensor products.

We will cover the following two SFG topologies: the star SFG, and the perfect binary SFG, shown in Figure 7. We denote the number of leaves by L, and we let N and c be bounds on the sizes of the unary potentials at the leaves and the binary potentials respectively (by a size N potential M, we mean that the matrices M_σ have size N by N). For example, in the phylogenetics setup, N is equal to an upper bound on the lengths of the sequences being analyzed, L is the number of taxa under study, and $c = 2$ in the case of the TKF91 model. We start our discussion with the star SFG.

In Figure 7, we show the result of the first two steps of the elimination algorithm.[7] The operations involved in this first step are pointwise products of the second kind. The intermediate factors produced, $M^{(1)}, \ldots, M^{(L)}$ are each of size cN. The second step is a marginalization, which does not increase the size of the matrices. We therefore have:

$$Z = \phi\left(\sum_{\hat\sigma \in \hat\Sigma} \left\{ \left(M_\epsilon^{(1)}\right)^* M_{\hat\sigma}^{(1)} \middle| \ldots \middle| \left(M_\epsilon^{(L)}\right)^* M_{\hat\sigma}^{(L)} \right\}\right)^* . \quad (4)$$

Computing the right-hand-side naively would involve inverting an $(cN)^L$ by $(cN)^L$ dense matrix—even if the matrices $M_{\hat\sigma}^{(l)}$ are sparse, there is no a priori guarantee for $\left(M_\epsilon^{(l)}\right)^*$ to be sparse as well. This would lead to a slow running time of $(cN)^{L\omega}$. In the rest of this section, we show that this can be considerably improved by using properties of tensor products and an extra assumptions on the factors:

Definition 6. *A factor, transducer or automaton is called* triangular *if its states can be ordered in such a way that all of its indexed matrices are upper triangular. A SFG is triangular if all of its leaf potentials are triangular.*

Note that we do not require that all the factors of SFG be triangular. In the case of the TKF91 model for example, only the factors at the leaves are triangular. This is enough to prove the main proposition of this section:

Proposition 7. *If a star-shaped or perfect binary triangular SFG has L unary potentials of size N and binary potentials of size c then the normalizer of the SFG can be computed in time $(cN)^L$.*

In order to prove this result, we need the following standard properties [29]:

Lemma 8. *Let $A^{(l)}$ be $m \times k$ matrices, and $B^{(l)}$ be $k \times n$ matrices. We have:*

$$\left\{ A^{(1)}B^{(1)} \middle| \ldots \middle| A^{(L)}B^{(L)} \right\} = \left\{ A^{(1)} \middle| \ldots \middle| A^{(L)} \right\} \left\{ B^{(1)} \middle| \ldots \middle| B^{(L)} \right\}.$$

[7] We omit the factor at the root since its effect on the running time is a constant independent of L and N.

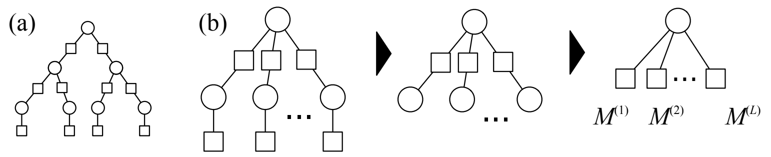


Figure 7: (a) The binary topology. (b) The star topology with the first two steps of the elimination algorithm on that topology. Note that in both cases we ignore the root factor, which only affect the running time by a constant independent of L and N.

Lemma 9. *If $A^{(l)}$ are invertible, then:*

$$\left\{\left(A^{(1)}\right)^{-1}\middle|\dots\middle|\left(A^{(L)}\right)^{-1}\right\} = \left\{A^{(1)}\middle|\dots\middle|A^{(L)}\right\}^{-1}.$$

Lemma 10. *If A is a $m \times k$ invertible matrix, and B is a $k \times n$ matrix such that $A - B$ is invertible, then $\left(A^{-1}B\right)^* = A(A-B)^{-1}$.*

The next lemma states that in a sense triangular factors are contagious:

Lemma 11. *If a factor is triangular, then marginalizing it or making it epsilon-free creates a triangular potential. Moreover, the outcome of a pointwise product (either of the first or second kind) is guaranteed to be triangular whenever at least one of its input automaton or transducer is triangular.*

Proof. For marginalization and pointwise products, the result follows directly from Proposition 3, 4, and 5. For epsilon-removal, it follows from Proposition 1 and the fact that the inverse of a triangular matrix is also triangular. □

We can now prove Proposition 7:

Proof. Note first that by Lemma 11, the matrices $M_\epsilon^{(i)}$ is upper triangular. Since the intermediate factors represent probabilities, $w_{M^{(l)}}(s) = \mathbb{P}_\tau(X_\Omega = s, \mathscr{Y})$, we have that $M_\epsilon^{(i)}(j,j) < 1$. It follows that the eigenvalues of $M_\epsilon^{(i)}$ are strictly smaller than one, so Proposition 1 can be applied to $M^{(i)}$ for each $i \in \{1,\dots,L\}$.

Next, by applying Proposition 2, we get:

$$Z = \phi\left(\sum_{\hat\sigma\in\hat\Sigma}\left\{\left(M_\epsilon^{(1)}\right)^* M_{\hat\sigma}^{(1)}\middle|\dots\middle|\left(M_\epsilon^{(L)}\right)^* M_{\hat\sigma}^{(L)}\right\}\right)^*.$$

This expression can be simplified using Lemma 8 9, and 10 to get:

$$\begin{aligned} Z &= \phi\left(\left\{\left(M_\epsilon^{(1)}\right)^*\middle|\dots\middle|\left(M_\epsilon^{(L)}\right)^*\right\}\sum_{\hat\sigma\in\Sigma}\left\{M_{\hat\sigma}^{(1)}\middle|\dots\middle|M_{\hat\sigma}^{(L)}\right\}\right)^* \\ &= \phi\left(\left\{\bar M_\epsilon^{(1)}\middle|\dots\middle|\bar M_\epsilon^{(L)}\right\}\left(\left\{\bar M_\epsilon^{(1)}\middle|\dots\middle|\bar M_\epsilon^{(L)}\right\} - \sum_{\hat\sigma\in\Sigma}\left\{M_{\hat\sigma}^{(1)}\middle|\dots\middle|M_{\hat\sigma}^{(L)}\right\}\right)^{-1}\right) \\ &= \phi\left(A(A-B)^{-1}\right), \end{aligned} \tag{5}$$

where $\bar M = I - M$, $A = \left\{\bar M_\epsilon^{(1)}\middle|\dots\middle|\bar M_\epsilon^{(L)}\right\}$, and $B = \sum_{\hat\sigma\in\Sigma}\left\{M_{\hat\sigma}^{(1)}\middle|\dots\middle|M_{\hat\sigma}^{(L)}\right\}$.

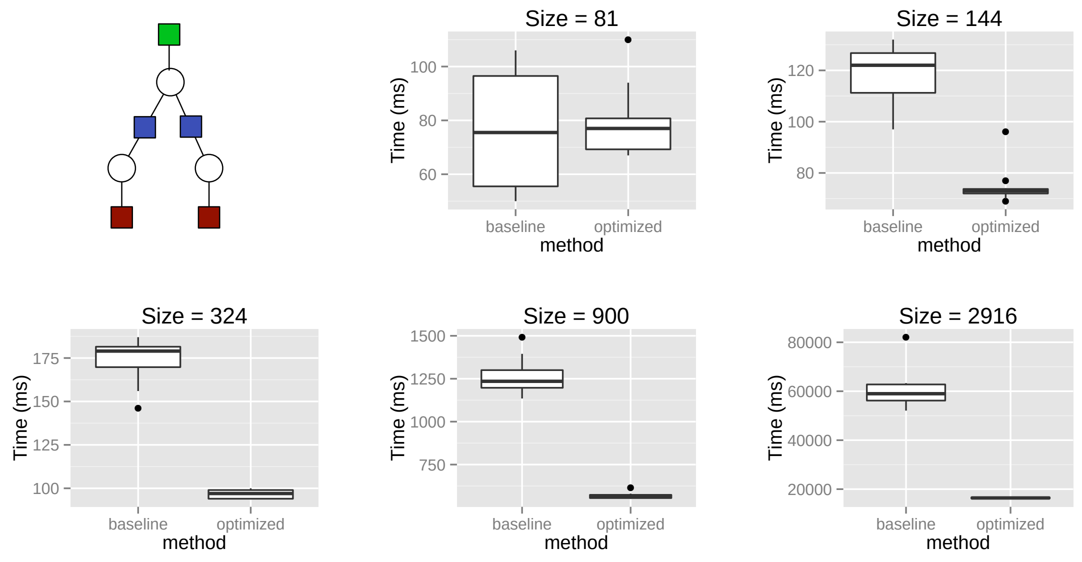


Figure 8: Wall clock times for computing the normalization of the SFG shown on the top left. By varying the size of the leaf factors, we obtain five normalization problems. Each box plot compares the running time of our method (denoted *optimized*, and built on top of the efficient matrix package BLAS/ATLAS [52]) to a baseline implementation of the classical transducer algorithms [38]. See Section 7 for details.

Equation (5) allows us to exploit sparsity patterns. By Lemma 11, the matrices $M_\epsilon^{(i)}$, $M_{\hat{\sigma}}^{(i)}$ are upper triangular sparse matrices (more precisely: cN by cN matrices with cN non zero entries), so the matrices A and B are upper triangular as well, with only $O((cN)^L)$ non-zero components. Furthermore, if we let $C = (A - B)^{-1}$, we can see that only its last column $\mathbf{x}$ is actually needed to compute Z. At the same time, we can write $(A - B)\mathbf{x} = [0, 0, \ldots, 0, 1]$, which can be solved using the back substitution algorithm, getting an overall running time of $O((cN)^L)$.

Note that in the above argument, the only assumption we have used on the factors at the leaves is triangularity. The same argument hence applies when these factors come from the elimination algorithm executed on a binary tree. The running time for the binary tree case can therefore be established by an inductive argument on the depth where the inductive step is the same as the argument above. □

7 Numerical simulations

In this section, we illustrate one practical advantage of our formalism: expressing SFG normalization purely in terms of matrix equations makes it simple to transfer advances in numerical linear algebra into the seemingly disconnected question of SFG normalization. These advances include platform-specific self-tuning, efficient memory access, and algorithmic improvements [52].

To investigate the gains achievable by this technology transfer, we generated a first set of normalization problems from a factor graph with five factors (Figure 8, top left), and a second set from a factor graph with eleven factors (Figure 9, left). We vary the size of the factors at the leaves to obtain two series of increasingly computationally intensive problems. Size is measured here by the length of the matrix M in Proposition 3. We used binary factors of the same size as those described in Section 2, generated artificial data at the leaves and computed the normalization using two methods.

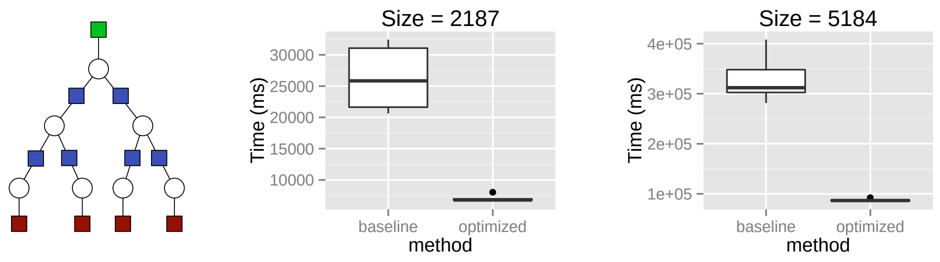


Figure 9: Wall clock times for computing the normalization of the SFG shown on the left. Refer to Figure 8 and the text.

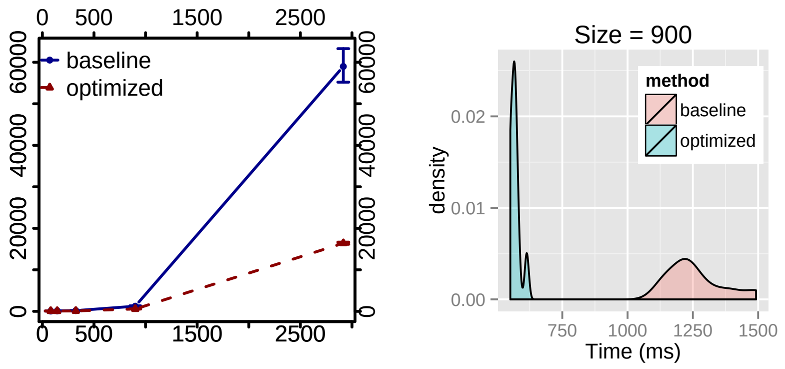


Figure 10: Left: Wall clock time as a function of the instance size for the graph with five factors. This shows that the empirical gains of our optimized method increase with the problem size. Right: Histogram of the running times providing a more detailed picture of the running times for the fourth datapoint on this plot.

The first method, denoted *optimized*, is based on our matrix formulation and is implemented on top of an optimized matrix library, BLAS/ATLAS [52]. We compare this to a baseline implementation of the classical transducer algorithms [38]. The goal of this experiment is to show that advances in numerical linear algebra can be translated into performance gains in SFG normalization. Note that neither methods exploit triangular properties in this experiment.

In each experiment, the two methods provide the same final result (up to numerical precision), but with widely different execution times. For each problem instance, we measured the wall-clock time in milliseconds needed for computing the normalization operation under both methods, and we replicated all experiments ten times. We show Figure 8 that for all but the smallest problem instance size, the optimized method outperforms the baseline method. The same trend is visible in the larger tree, Figure 9. We also show in Figure 10 that the empirical gains of the optimized method increase with the problem size.

8 Discussion

By exploiting closed-form matrix expressions rather than specialized automaton and transducer algorithms, our method gains several advantages. Our algorithms are arguably easier to understand and implement since their building blocks are well-known linear algebra operations. At the same time, our algorithms have asymptotic running times better or equal to previous transducer algorithms, so there is no loss of performance incurred, and even potential gains. In particular, by building implementations on top of existing matrix packages, our method automatically gets access to optimized libraries [52], or to graphical processing unit (GPU) parallelization from off-the-shelf libraries such as gpumatrix [36]. This last point is particularly attractive as GPUs are increasingly used outside graphics as inexpensive massively parallel processing units. Last but not least, our method also has applications as a proving tool. For example, despite a large body of theoretical work on the complexity of inference in the TKF91 model [17, 46, 18, 31, 19], basic questions such as computing the expectation of additive metrics under the TKF91 model remain open [8]. Our concise closed-form expression for exact inference could be useful to attack this type of problems.

In phylogenies that are too large to be handled by exact inference algorithms, our results can still be used as building blocks for approximate inference algorithms. For example, the acceptance ratio of most existing MCMC samplers for the TKF91 model can be expressed in terms of normalization problems on a small subtree of the original phylogeny [25]. The simplest way to obtain this subtree is to fix the value of the strings at all internal nodes except for two adjacent internal nodes. Resampling a new topology for these two nodes amounts to computing the normalization of two SFGs over a subtree of four leaves (their ratio appears in the Metropolis-Hastings ratio) [40]. Our approach can also serve as the foundation of Sequential Monte Carlo (SMC) approximations [47, 15, 5]. In this case, the ratio of normalizations appears as a particle weight update. The matrix formulation also opens the possibility of using low-rank matrices as an approximation scheme.

References

[1] E.M. Airoldi. Getting started in probabilistic graphical models. *PLoS Computational Biology*, 3(12), 2007.

[2] C.M. Bishop. *Pattern Recognition and Machine Learning*, chapter 8, pages 359–422. Springer, 2006.

[3] A Bouchard-Côté and M. I. Jordan. Evolutionary inference via the Poisson indel process. *Proceedings of the National Academy of Sciences of the USA*, 10.1073/pnas.1220450110, 2012.

[4] A. Bouchard-Côté, M. I. Jordan, and D. Klein. Efficient inference in phylogenetic InDel trees. In *Advances in Neural Information Processing Systems 21*, 2009.

[5] A Bouchard-Côté, S. Sankararaman, and M. I. Jordan. Phylogenetic Inference via Sequential Monte Carlo. *Systematic Biology*, 61:579–593, 2012.

[6] Alexandre Bouchard-Côté and Michael I. Jordan. Evolutionary inference via the Poisson indel process. *Proceedings of the National Academy of Sciences*, 10.1073/pnas.1220450110, 2012.

[7] RK Bradley and I. Holmes. Transducers: an emerging probabilistic framework for modeling indels on trees. *Bioinformatics*, 23(23):3258–62, 2007.

[8] C. Daskalakis and S. Roch. Alignment-free phylogenetic reconstruction: Sample complexity via a branching process analysis. *Annals of Applied Probability*, 2012.

[9] M. Dreyer, J. R. Smith, and J. Eisner. Latent-variable modeling of string transductions with finite-state methods. In *Proceedings of EMNLP 2008*, 2008.

[10] M. Droste and W. Kuich. *Handbook of Weighted Automata*, chapter 1. Monographs in Theoretical Computer Science. Springer, 2009.

[11] S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, 1974.

[12] J. Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, 17:368–376, 1981.

[13] J. Felsenstein. *Inferring Phylogenies.* Sinauer Associates, 2003.

[14] P. Fernandez, B. Plateau, and W.J. Stewart. Optimizing tensor product computations in stochastic automata networks. *Revue française d'automatique, d'information et de recherche opérationnelle*, 32 (3):325–351, 1998.

[15] D. Görür and Y. W. Teh. An efficient sequential Monte-Carlo algorithm for coalescent clustering. In *Advances in Neural Information Processing*, pages 521 – 528, Red Hook, NY, 2008. Curran Associates.

[16] J. Hein. A unified approach to phylogenies and alignments. *Methods in Enzymology*, 183:625–944, 1990.

[17] J. Hein. A generalisation of the thorne-kishino-felsenstein model of statistical alignment to k sequences related by a binary tree. *Pac.Symp.Biocompu.*, pages 179–190, 2000.

[18] J. Hein. An algorithm for statistical alignment of sequences related by a binary tree. *Pac. Symp. Biocomp.*, page 179190, 2001.

[19] J. Hein, J. Jensen, and C. Pedersen. Recursions for statistical multiple alignment. *Proceedings of the National Academy of Sciences of the USA*, 100(25):14960–5, 2003.

[20] David M. Higdon. Auxiliary variable methods for Markov Chain Monte Carlo with applications. *Journal of the American Statistical Association*, 93(442):585–595, 1998.

[21] I. Holmes. Using guide trees to construct multiple-sequence evolutionary hmms. *Bioinformatics*, 19(1):147–57, 2003.

[22] I. Holmes. Phylocomposer and phylodirector: analysis and visualization of transducer indel models. *Bioinformatics*, 23(23):3263–4, 2007.

[23] I. Holmes and W. J. Bruno. Evolutionary HMM: a Bayesian approach to multiple alignment. *Bioinformatics*, 17:803–820, 2001.

[24] I. Holmes and G. M. Rubin. An expectation maximization algorithm for training hidden substitution models. *J. Mol. Biol.*, 2002.

[25] J. Jensen and J. Hein. Gibbs sampler for statistical multiple alignment. Technical report, Dept of Theor Stat, U Aarhus, 2002.

[26] M. I. Jordan. Graphical models. *Statistical Science*, 19:140–155, 2004.

[27] A. Kawakita, T. Sota, J. S. Ascher, M. Ito, H. Tanaka, and M. Kato. Evolution and phylogenetic utility of alignment gaps within intron sequences of three nuclear genes in bumble bees (Bombus). *Mol Biol Evol.*, 20(1):87–92, 2003.

[28] B. Knudsen and M. Miyamoto. Sequence alignments and pair hidden Markov models using evolutionary history. *J Mol Biol.*, 333:453–460, 2003.

[29] A. N. Langville and W. J. Stewart. The Kronecker product and stochastic automata networks. *Journal of Computational and Applied Mathematics*, 167(2):429–447, 2004.

[30] G. Lunter, I. Miklós, A. Drummond, J. Jensen, and J. Hein. Bayesian coestimation of phylogeny and sequence alignment. *BMC Bioinformatics*, 6(1):83, 2005.

[31] Miklós I. Song Y. S. Lunter, G. A and J. Hein. An efficient algorithm for statistical multiple alignment on arbitrary phylogenetic trees. *J. Comput. Biol.*, 10:869–889, 2003.

[32] D Metzler, R Fleissner, A Wakolbinger, and von Haeseler A. Assessing variability by joint sampling of alignments and mutation rates. *J Mol Evol*, 2001.

[33] I. Miklós, A. Drummond, G. Lunter, and J. Hein. *Algorithms in Bioinformatics*, chapter Bayesian Phylogenetic Inference under a Statistical Insertion-Deletion Model. Springer, 2003.

[34] I. Miklós, G. A. Lunter, and I. Holmes. A long indel model for evolutionary sequence alignment. *Mol Biol Evol.*, 21(3):529–40, 2004.

[35] I. Miklós and Z. Toroczkai. An improved model for statistical alignment. In *First Workshop on Algorithms in Bioinformatics*, Berlin, Heidelberg, 2001. Springer-Verlag.

[36] S. Mingming. Gpumatrix library, April 2012.

[37] M. Mohri. Generic epsilon-removal and input epsilon-normalization algorithms for weighted transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143, 2002.

[38] M. Mohri. *Handbook of Weighted Automata*, chapter 6. Monographs in Theoretical Computer Science. Springer, 2009.

[39] Á. Novák, I. Miklós, R. Lyngsoe, and J. Hein. Statalign: An extendable software package for joint bayesian estimation of alignments and evolutionary trees. *Bioinformatics*, 24:2403–2404, 2008.

[40] B. D. Redelings and M. A. Suchard. Joint bayesian estimation of alignment and phylogeny. *Syst. Biol*, 54(3):401–18, 2005.

[41] B. D. Redelings and M. A. Suchard. Incorporating indel information into phylogeny estimation for rapidly emerging pathogens. *BMC Evolutionary Biology*, 7(40), 2007.

[42] E. Rivas. Evolutionary models for insertions and deletions in a probabilistic modeling framework. *BMC Bioinformatics*, 6(1):63, 2005.

[43] R. Satija, L. Pachter, and J. Hein. Combining statistical alignment and phylogenetic footprinting to detect regulatory elements. *Bioinformatics*, 24:1236–1242, 2008.

[44] M. P. Schützenberger. On the definition of a family of automata. *Inf. Control*, 4:245–270, 1961.

[45] Y. S. Song. A sufficient condition for reducing recursions in hidden Markov models. *Bulletin of Mathematical Biology*, 68:361–384, 2006.

[46] M. Steel and J. Hein. Applying the Thorne-Kishino-Felsenstein model to sequence evolution on a star-shaped tree. *Appl. Math. Let.*, 14:679–684, 2001.

[47] Y. W. Teh, H. Daume III, and D. M. Roy. Bayesian agglomerative clustering with coalescents. In *Advances in Neural Information Processing*, pages 1473–1480, Cambridge, MA, 2008. MIT Press.

[48] J. L. Thorne, H. Kishino, and J. Felsenstein. An evolutionary model for maximum likelihood alignment of DNA sequences. *Journal of Molecular Evolution*, 33:114–124, 1991.

[49] J. L. Thorne, H. Kishino, and J. Felsenstein. Inching toward reality: an improved likelihood model of sequence evolution. *J. of Mol. Evol.*, 34:3–16, 1992.

[50] O. Westesson, G. Lunter, B. Paten, and I. Holmes. Phylogenetic automata, pruning, and multiple alignment. *pre-print*, 2011. arXiv:1103.4347.

[51] O. Westesson, G. Lunter, B. Paten, and I. Holmes. Accurate reconstruction of insertion-deletion histories by statistical phylogenetics. *PLoS ONE*, 7(4):e34572, 2012.

[52] R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001.

[53] V. V. Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, 2012.

[54] K. M. Wong, M. A. Suchard, and J. P. Huelsenbeck. Alignment uncertainty and genomic analysis. *Science*, 319(5862):473–6, 2008.