Maximum Likelihood Inference of Time-scaled Cell Lineage Trees with Mixed-type Missing Data

Uyen Mai*, Gillian Chu*, and Benjamin J. Raphael†
Department of Computer Science, Princeton University, Princeton, NJ 08544, USA
*These authors contributed equally to this work.
† Correspondence: braphael@princeton.edu

Abstract

Recent dynamic lineage tracing technologies combine CRISPR-based genome editing with single-cell sequencing to track cell divisions during development. A key computational problem in dynamic lineage tracing is to infer a cell lineage tree from the measured CRISPR-induced mutations. Three features of dynamic lineage tracing data distinguish this problem from standard phylogenetic tree inference. First, the CRISPR-editing process modifies a genomic location exactly once. This *non-modifiable* property is not well described by the time-reversible models commonly used in phylogenetics. Second, as a consequence of non-modifiability, the number of mutations per time unit decreases over time. Third, CRISPR-based genome-editing and single-cell sequencing results in high rates of both heritable and non-heritable (dropout) missing data. To model these features, we introduce the Probabilistic Mixed-type Missing (PMM) model. We describe an algorithm, LAML (Lineage Analysis via Maximum Likelihood), to search for the maximum likelihood (ML) tree under the PMM model. LAML combines an Expectation Maximization (EM) algorithm with a heuristic tree search to jointly estimate tree topology, branch lengths and missing data parameters. We derive a closed-form solution for the M-step in the case of no heritable missing data, and a block coordinate ascent approach in the general case which is more efficient than the standard General Time Reversible (GTR) phylogenetic model. On simulated data, LAML infers more accurate tree topologies and branch lengths than existing methods, with greater advantages on datasets with higher ratios of heritable to non-heritable missing data. We show that LAML provides unbiased *time-scaled* estimates of branch lengths. In contrast, we demonstrate that maximum parsimony methods for lineage tracing data not only underestimate branch lengths, but also yield branch lengths which are not proportional to time, due to the nonlinear decay in the number of mutations on branches further from the root. On lineage tracing data from a mouse model of lung adenocarcinoma, we show that LAML infers phylogenetic distances that are more concordant with gene expression data compared to distances derived from maximum parsimony. The LAML tree topology is more plausible than existing published trees, with fewer total cell migrations between distant metastases and fewer reseeding events where cells migrate back to the primary tumor. Crucially, we identify three distinct time epochs of metastasis progression, which includes a burst of metastasis events to various anatomical sites during a single month.

Software: `https://github.com/raphael-group/LAML`

Data availability: `https://github.com/raphael-group/laml-experiments`

1 Introduction

Deriving the history of cell division and differentiation events that produce a multicellular organism from an embryo is of fundamental importance in developmental biology [25, 7, 27, 46, 43, 34] and tumor development [30, 6, 50, 42]. Since the cell division process is typically not directly observed, one may attempt to infer the history of cell divisions using heritable markers (e.g. somatic mutations) measured in cells at the present time. Such *retrospective lineage tracing* is challenged by the infrequency of naturally occurring somatic mutations which provide insufficient signal to construct a cell phylogeny. Recently, *dynamic lineage tracing* has overcome this challenge by using genome editing technologies, such as CRISPR/Cas9, to induce heritable mutations at pre-defined loci (i.e. *target sites* [28]) in the genomes of growing cells. These target sites are then sequenced via single-cell sequencing (sc-Seq) of a population of descendant cells to read out the induced mutations [27, 45]. Crucially, dynamic lineage tracing allows control over the strength of phylogenetic signal present in a cell culture by adjusting the number of target sites and genome editing parameters (e.g. amount of guide RNAs in CRISPR/Cas9). With suitable calibration, sufficient phylogenetic signal can be obtained for accurate reconstruction of cell lineage trees. While many dynamic lineage tracing technologies have been developed [27, 16, 34, 43, 2, 10, 5, 22, 50, 7, 9], we focus on more advanced technologies that are capable of recording medium to long experiment time, such as CARLIN [5], iTracer [22], Chan et. al [7], and Yang et. al [50].

Multiple computational methods have been developed to construct a cell lineage tree from dynamic lineage tracing data, including Cassiopeia [24], DCLEAR [20], AMbeRland [8], Startle [38] TiDeTree [39], and others. Several of the earlier methods were benchmarked in a DREAM challenge [19]. We classify existing methods into four categories: *distance-based methods* [47, 20], *maximum parsimony methods* [24, 38], *maximum likelihood methods*[15, 51], and *Bayesian methods*[39]. While it is well known in species phylogenetics that maximum likelihood and Bayesian methods often outperform maximum parsimony and distance-based methods [21, 18, 44], current methods in these categories for dynamic lineage tracing data [15, 51, 39] make assumptions that are only appropriate for older lineage tracing technologies [27, 10] but not for more recent CRISPR-based lineage tracing technologies [7, 50].

Three challenges hinder the application of existing probabilistic methods to newer dynamic lineage tracing data. First, the decay in the number of available target sites - a unique feature of the CRISPR-based editing system - is overlooked in many computational methods. In CRISPR-based lineage tracing systems, each target site accommodates at most one mutation, after which it can no longer be edited (i.e. *non-modifiable*). Consequently, the number of available (i.e. unmutated) target sites decreases over time, until mutations have saturated all available target sites, marking the end of *recordable time*. While innovations in experimental design have extended the recordable time of dynamic lineage tracing systems, on the computational front there has been no study of *how target site decay affects lineage tree reconstruction*. Second, lineage tracing data contains missing entries from two distinct mechanisms: *heritable missing data*, which results from silencing during the CRISPR-editing process and stochastic *dropout*, a technical error which occurs during single-cell sequencing. To the best of our knowledge, no existing computational method distinguishes these two types of missing data and leverages the phylogenetic signal from heritable missing entries. Third, current probabilistic methods do not analyze lineage tracing data with heterogeneous target sites (i.e. distinct per-site alphabet) as input.

We present the first probabilistic model and maximum likelihood method that can infer the cell phylogeny from data generated by recent dynamic lineage tracing technologies (such as [7] and [50]). Our proposed Probabilistic Mixed-type Missing (PMM) model, captures all key features of dynamic lineage tracing: non-modifiability of CRISPR, decay of the number of mutations over time, mixed-type missing data, and heterogeneous target sites,. Our maximum likelihood method, LAML (Lineage Analysis via Maximum Likelihood), jointly estimates tree topology, time-scaled branch lengths, and other parameters of the PMM model. In each iteration, LAML performs two steps: an EM algorithm for parameter estimation and tree rearrangements for topology search. Notably, the EM algorithm we derived under the PMM model is more efficient than that of existing phylogenetic models (such as the complex but widely used General Time Reversible (GTR) model). The complexity of the algorithm is presented in two cases: in the case of no heritable missing data, we present a closed form solution, and in the general case, we present an efficient block coordinate ascent approach.

We show that LAML outperforms existing methods on both topology and branch length inference on both simulated data and a mouse model of lung adenocarcinoma. In addition, we observe that maximum parsimony methods face a crucial problem in branch length estimation: maximum parsimony methods suffer from systematic bias where branches further from the root are more severely underestimated due to the decay in the number of unmutated target sites through time. Importantly, we demonstrate that LAML's ability to estimate branch lengths in time units enables a new application for inferred cell phylogenies: we can track metastasis progression over time, from which we discover

3 temporal epochs of metastasis progression in the lung adenocarcinoma model with distinct migration patterns.

2 The PMM model

Consider a dynamic lineage tracing experiment with K target sites[1] and N sequenced cells. We refer to the data obtained from that experiment as the *observed character matrix*, denoted by $\mathbf{D}$, which is an $N \times K$ matrix. Entries in column k of $\mathbf{D}$ take values in $\mathscr{A}^{(k)} = \{?, -1, 0, 1, ..., M^{(k)}\}$, where $\mathscr{A}^{(k)}$ is the *alphabet* of target site k, 0 is the *unmutated state*, -1 is the *silent state*, $?$ is the *missing state*, and $1, ..., M^{(k)}$ are *mutated states*.

We represent the cell lineages by a rooted tree topology T. Each leaf node of T represents a sampled cell which corresponds to one row of $\mathbf{D}$ and the internal nodes represent unobserved ancestral cells. Let $\mathscr{L}_T$ $\mathscr{V}_T$, and $\mathscr{E}_T$ be the *set of leaves*, *set of nodes*, and *set of edges* of T, respectively. Let r_T be the root of T. We assume that r_T has exactly one child (the progenitor cell needs time to divide) and all other internal nodes of T have exactly two children. Let (u, v) be the edge[2] in $\mathscr{E}_T$ from u to its child v (where $u, v \in \mathscr{V}_T$). Each branch (u, v) of T has an associated branch length that shows the distance between u and v, which can be measured in either time units or mutation units, as will be described in section 2.1.

Given the character matrix $\mathbf{D}$, our goal is to construct a cell lineage tree topology T and find all of its branch lengths that best describe $\mathbf{D}$. We will solve this problem using a probabilistic approach: first, we describe a model for the likelihood $\mathcal{P}(\mathbf{D}|T, \mathbf{\Theta})$ where $\mathbf{\Theta}$ includes the branch lengths and other parameters, and second, we develop an algorithm to find T that maximizes the likelihood. Our proposed Probabilistic Mixed-type Missing (PMM) model, consists of two layers corresponding to CRISPR/Cas9 editing of target sites and single-cell sequencing. Below we describe these two layers in more detail.

2.1 Layer 1: model of CRISPR/Cas9-induced mutations and heritable missing

Layer 1 describes the CRISPR/Cas9 editing process. At the beginning of the experiment, we assume there is exactly one *progenitor cell* that has all K target sites in the *unmutated state*. During the CRISPR/Cas9 process, each target site k either mutates into one of the *mutated states* in $\{1, ..., M^{(k)}\}$ or into the *silent state*. Recall that the cell phylogeny is denoted by T. We assume each site k mutates following a CTMC with the following transition rate matrix $\mathbf{Q}^{(k)}$:

$$\mathbf{Q}^{(k)} = \begin{array}{c} \\ 0 \\ 1 \\ \vdots \\ M^{(k)} \\ -1 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & \dots & M^{(k)} & -1 \end{array} \\ \begin{bmatrix} -(1+\nu) & q_1^{(k)} & \dots & q_{M^{(k)}}^{(k)} & \nu \\ 0 & -\nu & \dots & 0 & \nu \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & -\nu & \nu \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix} \end{array}, \tag{1}$$

where $q_1^{(k)}, q_2^{(k)}, ..., q_{M^{(k)}}^{(k)}$ are the rates of transitioning from 0 to the mutated states of site k (i.e. $\sum_{m=1}^{M^{(k)}} q_m^{(k)} = 1$) and ν is the heritable missing rate, which is shared across all sites.

Let λ be the editing rate (i.e. rate of changing from 0 to a mutated state) and t_e be the length of a branch $e \in \mathscr{E}_T$ in time units. We assume λ is shared across all sites and branches and t_e for each branch e is shared across all sites. On an edge $e \in \mathscr{E}_T$ at site k, the transition probabilities of every pair of states are determined by the *transition probability matrix*, $\mathbf{\Psi}_e^{(k)}$, computed from $\mathbf{Q}^{(k)}$ as $\mathbf{\Psi}_e^{(k)} = \mathrm{e}^{\mathbf{Q}^{(k)}\lambda t_e}$. Performing matrix exponentiation gives the following:

$$\mathbf{\Psi}_e^{(k)} = \mathrm{e}^{\mathbf{Q}^{(k)}\delta_e} = \begin{array}{c} \\ 0 \\ 1 \\ \vdots \\ M^{(k)} \\ -1 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & \dots & M^{(k)} & -1 \end{array} \\ \begin{bmatrix} e^{-\delta_e(1+\nu)} & q_1^{(k)} e^{-\delta_e \nu}(1-e^{-\delta_e}) & \dots & q_m^{(k)} e^{-\delta_e \nu}(1-e^{-\delta_e}) & 1-e^{-\delta_e \nu} \\ 0 & e^{-\delta_e \nu} & \dots & 0 & 1-e^{-\delta_e \nu} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & e^{-\delta_e \nu} & 1-e^{-\delta_e \nu} \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix} \end{array}, \tag{2}$$

where $\delta_e = \lambda t_e$, which is the length of e in mutation unit. Note that given λ, t_e and δ_e can be trivially computed from one another. In other words, let $\mathbb{X}$ be the *random* matrix of size $|\mathscr{V}_T| \times K$ whose entries take values in $\mathscr{A}^{(k)} \setminus \{?\}$.

[1] Following the convention in phylogenetics, we use *target site* and *site* interchangeably.

[2] Throughout this paper, we use *edge* and *branch* interchangeably.

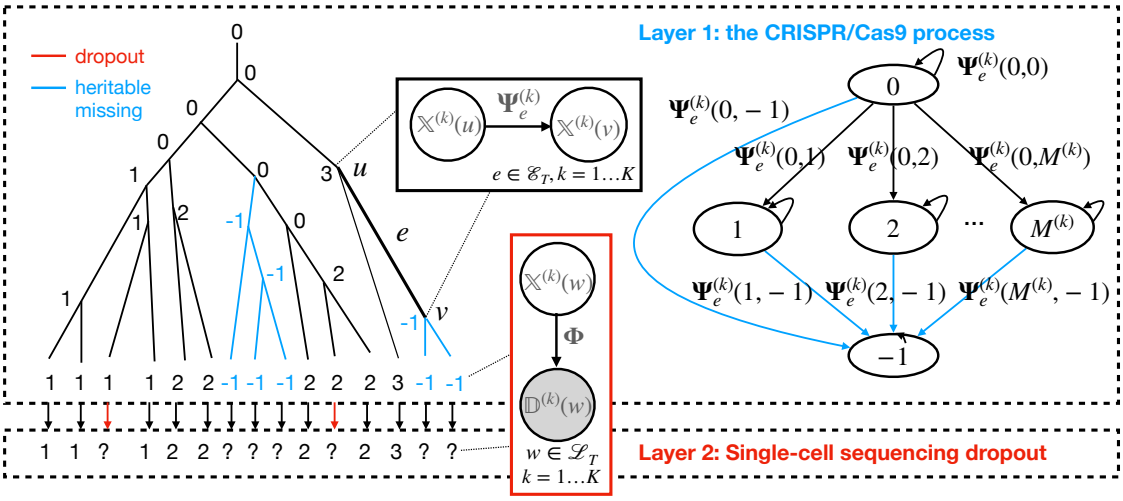

Figure 1: The PMM model with its two layers. Layer 1 (Top): A CTMC that models the process of CRISPR/Cas9 editing and heritable missing and (2) Stochastic dropout in single-cell sequencing. 0 denotes the unmutated state, -1 denotes the silent state, and other integers denote mutated states. State transitions are demonstrated by both a plate diagram and a transition graph. Transition probabilities are parameterized by $\mathbf{\Psi}$ (see Eq.2). Layer 2 (Bottom): A model for stochastic dropout in single-cell sequencing. The model is parameterized by $\mathbf{\Phi}$ as indicated by the plate diagram (see Eq. 4). Observed missing data is represented by "?" and is a mixture of heritable missing (part of Layer 1) and dropout (part of Layer 2). The two types of missing data are indistinguishable in the observed data.

Each entry in column k of $\mathbb{X}$ represents the state of the corresponding cell at target site k *before single-cell sequencing*. Then for all $\alpha_u, \alpha_v \in \mathscr{A}^{(k)} \setminus \{?\}$, we have:

$$\mathcal{P}(\mathbb{X}^{(k)}(v) = \alpha_v | \mathbb{X}^{(k)}(u) = \alpha_u) = \mathbf{\Psi}_e^{(k)}(\alpha_u, \alpha_v), \tag{3}$$

where $\mathbb{X}^{(k)}(u)$ and $\mathbb{X}^{(k)}(v)$ denote the entries of $\mathbb{X}$ corresponding to site k and cells u and v, respectively.

The model is illustrated in Figure 1. Note that we assume that t_e, δ_e, and ν are shared for all sites, but allow each site k to have a distinct alphabet $\mathscr{A}^{(k)}$ and a distinct set $\{q_1^{(k)}, q_2^{(k)}, ..., q_{M^{(k)}}^{(k)}\}$. To avoid overparameterization, we treat all $q_1^{(k)}, q_2^{(k)}, ..., q_{M^{(k)}}^{(k)}$ as hyperparameters (also referred to as *indel priors* in previous works [24, 38]). The user can specify these hyperparameters to be identical (as in [39]), or use external data to provide pre-estimated values (as in [24]). Notably, the transition matrix $\mathbf{\Psi}_e^{(k)}$ is sparse and has a special structure: non-zero entries are only in the first row, last column, and main diagonal.

2.2 Layer 2: model of dropout from single-cell sequencing

The second layer of the model describes the stochastic generation of dropout during single cell sequencing. Recall that after single cell sequencing, characters in the silent state (i.e. -1) will be observed in the missing state with probability 1, while the other states have some probability to transition to the missing state depending on the dropout rate of single cell sequencing. We use ϕ to represent the *dropout rate* and represent the transition probabilities during single cell sequencing by the following matrix:

$$\mathbf{\Phi} = \begin{array}{c} \\ 0 \\ 1 \\ \vdots \\ M^{(k)} \\ -1 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & \ldots & M^{(k)} & ? \end{array} \\ \begin{bmatrix} 1-\phi & 0 & \ldots & 0 & \phi \\ 0 & 1-\phi & 0 & \ldots & \phi \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 1-\phi & \phi \\ 0 & 0 & \ldots & 0 & 1 \end{bmatrix} \end{array} \tag{4}$$

Let $\mathbb{D}$ be the *random* matrix of size $|\mathscr{L}_T| \times K$ whose entries take values in $\mathscr{A}^{(k)} \setminus \{-1\}$. Each entry in column k of $\mathbb{D}$ represents the state of the corresponding cell at target site k *after single cell sequencing*. Note that $\mathbf{D}$, the character

matrix, is a realization (observed value) of $\mathbb{D}$. For all $\alpha \in \mathscr{A}^{(k)} \setminus \{?\}$, $\beta \in \mathscr{A}^{(k)} \setminus \{-1\}$ and $w \in \mathscr{L}_T$, we have:

$$\mathcal{P}(\mathbb{D}^{(k)}(w) = \beta | \mathbb{X}^{(k)}(w) = \alpha) = \mathbf{\Phi}(\alpha, \beta), \tag{5}$$

where $\mathbb{X}^{(k)}(w)$ and $\mathbb{D}^{(k)}(w)$ denote the entries of $\mathbb{X}$ and $\mathbb{D}$ corresponding to site k and cells w, respectively.

In summary, the PMM model has the following parameters: all branch lengths, the mutation rate λ, the silencing rate ν, and the dropout rate ϕ. In the rest of this paper, we use $\mathbf{\Theta} = (\{\delta_e\}, \lambda, \nu, \phi)$ to denote these parameters. The model is illustrated in Figure 1.

3 LAML: Maximum likelihood inference

We assume that sites mutate independently, and thus given tree topology T and $\mathbf{\Theta}$ describing branch lengths and missing data parameters, the log-likelihood $\log L(T, \mathbf{\Theta}; \mathbf{D})$ of the observed data $\mathbf{D}$ under the PMM model is the sum of the log-likelihoods of the individual sites:

$$\log L(T, \mathbf{\Theta}; \mathbf{D}) = \sum_{k=1}^{K} \log \mathcal{P}(\mathbf{D}^{(k)}; T, \mathbf{\Theta}). \tag{6}$$

The probability $\mathcal{P}(\mathbf{D}^{(k)}; T, \mathbf{\Theta})$ of one site k is computed by marginalizing over all possible realizations of $\mathbb{X}^{(k)}$:

$$\mathcal{P}(\mathbf{D}^{(k)}; T, \mathbf{\Theta}) = \sum_{\mathbf{x}} \mathcal{P}(\mathbb{D}^{(k)} = \mathbf{D}^{(k)}, \mathbb{X}^{(k)} = \mathbf{x}; T, \mathbf{\Theta}) = \sum_{\mathbf{x}} \prod_{e=(u,v) \in \mathscr{E}_T} \mathbf{\Psi}_e^{(k)}(\mathbf{x}(u), \mathbf{x}(v)) \prod_{w \in \mathscr{L}_{\mathscr{T}}} \mathbf{\Phi}(\mathbf{x}(w), \mathbf{D}^{(k)}(w)), \tag{7}$$

where $\mathbf{\Psi}$ and $\mathbf{\Phi}$ are defined in Eqs. 2 and 4. While the sum in the above equation is over an exponential number of possible realizations of $\mathbb{X}^{(k)}$, the log-likelihood can be computed in linear time, using Felsenstein's pruning algorithm [14]. See Supplementary Methods for a review of the pruning algorithm and the likelihood recurrences derived specifically for the PMM model.

3.1 Maximum likelihood inference

Our goal is to find the tree topology T and parameters $\mathbf{\Theta}$ that maximize the log-likelihood function:

$$\max_{T, \mathbf{\Theta}} \log L(T, \mathbf{\Theta}; \mathbf{D}), \tag{8}$$

such that

$$\sum_{e \in \mathrm{Path}(r_T, w)} \delta_e = \lambda \tau, \text{ for all } w \in \mathscr{L}_T, \tag{9}$$

where $\mathrm{Path}(r_T, w)$ denotes the path (a list of edges) from the root r_T to a leaf node w, and τ is length of the experiment (must be given by the user). Note that while $\mathbf{\Theta}$ does not include $\{t_e\}$, we can compute every t_e from λ and δ_e using $\delta_e = \lambda t_e$. As such, we infer *branch lengths in both mutation units and time units*.

In general, maximum likelihood inference of phylogenetic trees is NP-hard [35]. Therefore, we use the following algorithm to find the maximum likelihood tree and $\mathbf{\Theta}$, iterating over two subroutines: (i) for a fixed tree topology T, find $\mathbf{\Theta}$ to maximize $L(\mathbf{\Theta}; T, \mathbf{D})$, and (ii) search for the optimal tree topology T. To solve (i), we devise *an efficient EM algorithm* and to solve (ii) we use simulated annealing with Nearest-Neighbor-Interchange (NNI) operations. Details of both steps are provided below.

3.2 Estimation of Θ on a fixed tree topology T

Given a tree topology T the maximum likelihood estimate $\hat{\mathbf{\Theta}}$ is defined as

$$\hat{\mathbf{\Theta}} = \underset{\mathbf{\Theta} \text{ satisfies Eq. 9}}{\operatorname{argmax}} \log L(\mathbf{\Theta}; T, \mathbf{D}). \tag{10}$$

We compute $\hat{\mathbf{\Theta}}$ using the expectation maximization (EM) algorithm. Note that since T is fixed, we omit T from the equations below for brevity.

3.2.1 The EM algorithm

The expectation maximization (EM) algorithm has been used to compute maximum likelihood estimates of branch lengths given a fixed tree topology [17, 41], which iteratively updates $\{\hat{\delta}_e\}$ by alternating between an E-step and M-step. Building on these works, we derive an efficient EM algorithm to compute the maximum likelihood estimates for all parameters $\hat{\mathbf{\Theta}} = (\{\hat{\delta}_e\}, \hat{\lambda}, \hat{\nu}, \hat{\phi})$, which are the estimated branch lengths $\hat{\delta}_e$, mutation rate $\hat{\lambda}$, heritable missing rate $\hat{\nu}$, and dropout rate $\hat{\phi}$. This algorithm leverages the special structure of the transition matrix (equation (4)) to reduce the complexity of the E-step and simplify the optimization problem involved in the M-step.

We briefly describe the E-step and the M-step below with the full proof and pseudocode in Supplementary Methods.

E-step At iteration $t+1$, in the E-step we compute the expected values of the transitions $\mathbb{X}(u) \to \mathbb{X}(v)$ for every edge $e = (u, v)$ of T and expected values of the transitions $\mathbb{X}(w) \to \mathbb{D}(w)$ for every leaf w with respect to the estimate $\hat{\mathbf{\Theta}}^t$ of $\mathbf{\Theta}$ at iteration t. Because both $\mathbf{\Psi}_e$ and $\mathbf{\Phi}$ are sparse and have special structure, we group the transition types of $\mathbb{X}(u) \to \mathbb{X}(v)$ into the following 5 groups: (i) 0 to 0 ($z \to z$), (ii) 0 to any mutated state ($z \to a$), (iii) 0 to the silent state -1 ($z \to m$), (iv) any mutated state to another mutated state $a \to a$, and (v) any mutated state to silent state -1 ($a \to m$). Similarly, we group the transition types of $\mathbb{X}(w) \to \mathbb{D}(w)$ into the following two groups: no-dropout $\mathcal{B}$ and dropout ($\tilde{\mathcal{B}}$). The expected values of these transition types are computed as follows:

$$
\begin{aligned}
\mathcal{C}_e^{z\to z} &= \sum_{k=1}^{K} \mathcal{P}(\mathbb{X}^{(k)}(u) = 0, \mathbb{X}^{(k)}(v) = 0 | \mathbb{D}^{(k)} = \mathbf{D}^{(k)}; \hat{\mathbf{\Theta}}^t), \\
\mathcal{C}_e^{z\to a} &= \sum_{k=1}^{K} \sum_{\alpha \in \mathscr{A}^{(k)} \setminus \{0,-1,?\}} \mathcal{P}(\mathbb{X}^{(k)}(u) = 0, \mathbb{X}^{(k)}(v) = \alpha | \mathbb{D}^{(k)} = \mathbf{D}^{(k)}; \hat{\mathbf{\Theta}}^t), \\
\mathcal{C}_e^{z\to m} &= \sum_{k=1}^{K} \mathcal{P}(\mathbb{X}^{(k)}(u) = 0, \mathbb{X}^{(k)}(v) = -1 | \mathbb{D}^{(k)} = \mathbf{D}^{(k)}; \mathbf{\Theta}^t), \\
\mathcal{C}_e^{a\to a} &= \sum_{k=1}^{K} \sum_{\alpha \in \mathscr{A}^{(k)} \setminus \{0,-1,?\}} \mathcal{P}(\mathbb{X}^{(k)}(u) = \alpha, \mathbb{X}^{(k)}(v) = \alpha | \mathbb{D}^{(k)} = \mathbf{D}^{(k)}; \hat{\mathbf{\Theta}}^t), \\
\mathcal{C}_e^{a\to m} &= \sum_{k=1}^{K} \sum_{\alpha \in \mathscr{A}^{(k)} \setminus \{0,-1,?\}} \mathcal{P}(\mathbb{X}^{(k)}(u) = \alpha, \mathbb{X}^{(k)}(v) = -1 | \mathbb{D}^{(k)} = \mathbf{D}^{(k)}; \hat{\mathbf{\Theta}}^t), \\
\mathcal{B}_u &= |\{k : \mathbf{D}^{(k)}(u) \neq ?\}|, \quad \tilde{\mathcal{B}}_u = \sum_{k:\mathbf{D}^{(k)}(u)=?} (1 - \mathcal{P}(\mathbb{X}^{(k)}(u) = -1 | \mathbb{D}^{(k)} = \mathbf{D}^{(k)}; \hat{\mathbf{\Theta}}^t)).
\end{aligned}
\tag{11}
$$

It is known that all the posterior probabilities $\mathcal{P}(\mathbb{X}^{(k)}(u) = \alpha_u, \mathbb{X}^{(k)}(v) = \alpha_v | \mathbb{D}^{(k)} = \mathbf{D}^{(k)}; \hat{\mathbf{\Theta}}^{\boldsymbol{t}})$ can be computed in $\mathcal{O}(MNK)$ [17, 41] for all combinations of $e = (u, v), \alpha_u, \alpha_v$, where M is the maximum alphabet size, N is the number of sequenced cells, and K is the number of target sites. Therefore, the complexity of the E-step is also $\mathcal{O}(MNK)$. While this complexity is acceptable for most current lineage tracing data, we further reduce the complexity to $\mathcal{O}(NK)$ by leveraging special properties of our model to compute the expected values without knowing all the posterior probabilities (details in Supplementary methods).

M-step The M-step of iteration $t+1$ is the following optimization problem:

$$
\begin{aligned}
\hat{\mathbf{\Theta}}^{t+1} = \underset{(\{\delta_e\},\lambda,\nu,\phi) \text{ satisfies Eq. 9}}{\operatorname{argmax}} \sum_{e \in \mathscr{E}_T} \big(&-\mathcal{C}_e^{z\to z}(1+\nu)\delta_e + \mathcal{C}_e^{z\to a}(\log(1 - e^{-\delta_e}) - \nu\delta_e) + \\
&\mathcal{C}_e^{z\to m} \log(1 - e^{-\delta_e \nu}) - \mathcal{C}_e^{a\to a}\delta_e\nu + \mathcal{C}_e^{a\to m} \log(1 - e^{-\delta_e \nu}) \big) + \\
&\sum_{u \in \mathscr{L}_T} \Big(\mathcal{B}_u \log(1-\phi) + \tilde{\mathcal{B}}_u \log \phi \Big),
\end{aligned}
\tag{12}
$$

where the coefficients (of form $\mathcal{C}$ and $\mathcal{B}$) in Eq. 12 depend on $\hat{\mathbf{\Theta}}^t$ and were computed in the E-step. While this problem is not convex in general,in a special case where the heritable missing rate $\nu = 0$, the problem is convex and if we ignore the constraints, we can derive a closed-form solution by a simple change of variables. When $\nu \neq 0$, the problem is still solved efficiently using block coordinate ascent.

Case 1: heritable missing rate $\nu = 0$. If $\nu = 0$, then $\mathcal{C}_e^{z\to m} = \mathcal{C}_e^{a\to m} = 0$ for all edges e. In addition, for all $u \in \mathcal{L}_T$ we have $\mathcal{P}(\mathbb{X}^{(k)} = 0)$, so $\tilde{\mathcal{B}}_u = K - \mathcal{B}_u$. Thus, the M-step described by Eq. (12) reduces to:

$$\hat{\mathbf{\Theta}}^{t+1} = \underset{(\{\delta_e\},\lambda,\nu,\phi) \text{ satisfies Eq. 9}}{\operatorname{argmax}} \sum_{e\in\mathscr{E}_T} \left(-\mathcal{C}_e^{z\to z}(1+\nu)\delta_e + \mathcal{C}_e^{z\to a}\log(1-e^{-\delta_e})\right) + \sum_{u\in\mathscr{L}_T} \left(\mathcal{B}_u \log(1-\phi) + (K-\mathcal{B}_u)\log\phi\right),$$

This optimization problem is convex (the objective function is separable, its second partial derivatives are negative, and the constraints are linear).

The closed-form M-step when $\nu = 0$ and ignoring constraints *If we ignore the constraints given in Eq. 9*, we obtain a closed-form solution, as follows. Let $\rho_e = e^{-\delta_e}$. Then the problem reduces to

$$\max_{\{\rho_e\},\phi} \sum_{e\in E_T} \mathcal{C}_e^{z\to z}\log\rho_e + \mathcal{C}_e^{z\to a}\log(1-\rho_e) + \sum_{u\in\mathscr{L}_T} \mathcal{B}_u\log(1-\phi) + (K-\mathcal{B}_u)\log\phi.$$

We solve the above problem by simply finding the roots of the partial derivatives, obtaining the following *closed-form* solution:

$$\hat{\rho}_e = \frac{\mathcal{C}_e^{z\to z}}{\mathcal{C}_e^{z\to z} + \mathcal{C}_e^{z\to a}} \text{ and } \hat{\phi} = \frac{\sum_{u\in\mathscr{L}_T}(K-\mathcal{B}_u)}{\sum_{u\in\mathscr{L}_T} K} = \frac{N_{\text{missing}}}{NK}$$

where N_{missing} is the total number of missing entries "?" in $\mathbf{D}$. Then $\hat{\delta}_e = -\log\hat{\rho}_e$. Because $\hat{\phi}$ only depends on $\mathbf{D}$, we compute $\hat{\phi}$ outside the EM algorithm. We note that because the constraints given in Eq. 9 are ignored, this closed-form formula only gives solution for branch lengths in mutation units and does not give a closed form solution for λ.

Solving M-step in general case If $\nu \neq 0$, the M-step problem is not always convex. However, the problem is convex separately with respect to ϕ, all δ_e, and ν. In this case, we use block coordinate ascent to find a local optimum, where we sequentially optimize ϕ, δ, and ν (See Supplementary methods S1.6 for proof of convexity and the block coordinate ascent algorithm). In summary, our EM algorithm is an iterative algorithm where each iteration consists of two steps: (i) **E-step**: compute Eq. 11 and (ii) **M-Step**: solve for $\mathbf{\Theta}$ in Eq. 12. We present Pseudocode in Algorithm box 1.

Algorithm 1 Pseudocode for the EM algorithm. Inputs: $\mathbf{D}$: the character matrix, T: the tree topology, ϵ: the convergence threshold, and MaxIter (optional): the maximum number of EM iterations allowed. Output: the maximum likelihood estimate $\hat{\mathbf{\Theta}}$.

```
function EMALGORITHM(D, T, ϵ,MaxIter=1000)
    t ← 0
    Randomly set Θ̂^0                                          ▷ random initialization
    diff ← ϵ
    while diff ≥ ϵ do and t ≤ MaxIter                          ▷ While not converged
        for each edge e in topology T do                       ▷ E-Step begins
            Compute C_e^{z→z}, C_e^{z→a}, C_e^{z→m}, C_e^{a→a}, C_e^{a→m} using Eq. 11
        for each leaf u in topology T do
            Compute B(u), B̃(u) using Eq. 11                    ▷ E-Step ends
        Θ̂^new ← solution of Eq. 12                             ▷ M-Step
        diff ← L(T, Θ̂^new; D) − L(T, Θ̂^t; D)
        t ← t + 1
        Θ̂^t ← Θ̂^new                                           ▷ Update Θ̂^t
    return Θ̂^t
```

3.3 Topology search

We now describe Lineage Analysis via Maximum Likelihood (LAML), an algorithm that alternates between proposing a new tree topology and using the proposed EM algorithm to find the maximum likelihood estimate $\hat{\mathbf{\Theta}}$ on this topology. Topology search is performed using nearest neighbor interchange (NNI) with simulated annealing. We provide the following pseudocode for LAML (Algorithm Box 2). See further details about simulated annealing in Supplementary methods S1.7.

Algorithm 2 Pseudocode for the LAML algorithm. Inputs: D: character matrix, T_0: starting tree (either random or constructed using a fast tree construction method), $\epsilon_{\text{topology}}$ and ϵ_{em}: the thresholds determining convergence of topology search and EM algorithm, respectively. Output: a tree topology T and parameters $\hat{\mathbf{\Theta}}$.

```
function LAML(D, T_0, ε_topology, ε_em)
    T ← T_0
    Θ̂ ← Θ̂_0
    while T has unexamined NNI topology do
        T' ← a new NNI of (T)                    ▷ propose a new topology with nearest neighbor interchange
        Θ̂' ← EMALGORITHM(D, T', ε_em)
        if improvement or probabilistic accept then                          ▷ simulated annealing
            Δ = | log L(Θ̂'; T', D) − log L(Θ̂; T, D)|
            T ← T'
            Θ̂ ← Θ̂'
            if Δ < ε_topology then return T, Θ̂                               ▷ convergence reached
    return T, Θ̂
```

4 Results

We evaluate LAML on simulated data and two datasets from *in vivo* lineage tracing experiments: a mouse model of metastatic lung adenocarcinoma (KP-Tracer) [50], and a cellular developmental study of *Drosophila* with true recorded cell lineages (intMEMOIR) [10].

4.1 Simulated data

We compare LAML to four other algorithms for constructing trees from lineage tracing data: the Cassiopeia-Greedy method [24], distance-based Neighbor Joining (implemented in Cassiopeia [24]), DCLEAR [20] – another distance-based method, and Startle-NNI [38] – a maximum-parsimony method. We evaluate these methods on simulated trees with 1024 leaves (i.e. 10 cell generations) generated using the birth-only process in the Cassiopeia simulation engine [24]. Branch lengths in time units are drawn according to a lognormal distribution with mean 1 and standard deviation 0.1. We simulate $K = 30$ target sites under the PMM model, with the values of hyperparameters $q^{(k)}$ at each site k selected to match a recent lineage tracing dataset of trunk like structures (TLS) [4]. We also calibrate the simulated data to match two other statistics in the TLS dataset: the expected proportion (64%) of non-zero (mutated) entries in the character matrix and the expected proportion (25%) of missing data. To achieve 64% non-zero entries in the simulated data, we set the mutation rate $\lambda = 0.095$. To obtain a proportion of 25% overall missing data (with varying proportions of heritable and non-heritable missing data), we calibrate ϕ and ν for each of the five model conditions, as summarized in Table S2. To model the sampling of sequenced cells, each simulated character matrix is subsampled to 250 cells and the model tree is restricted to the same subset. We repeat the subsampling procedure five times for each character matrix. For each of five model conditions and ten sets of mutation alphabet hyperparameters, we created five replicates each, for a total of 250 replicates.

4.1.1 LAML infers more accurate tree topology and branch lengths than other methods

We compare the cell lineage tree topologies output by the five methods using Robinson-Foulds (RF) error (i.e. normalized RF distance between true and estimated trees, defined in Supplementary data S2.1). Our results show that the trees estimated by LAML have the lowest RF error in all model conditions (Figure 2A). For all methods, RF error decreases as the proportion of heritable missing increases. For instance, LAML's RF error decreases from 0.38 at h0d100 to 0.2 at h100d0. However, the improvement LAML achieves over other methods also increases with the proportion of heritable missing. For instance, compared to Startle – the second best method – the RF error of LAML is lower by 0.06 at h0d100 and lower by 0.14 at h100d0. This result suggests that LAML better leverages the phylogenetic signal in heritable missing data than the other methods.

As a positive control, we also evaluate each method by its ability to optimize its own objective (Table S3 and Fig. S2). As expected, LAML has the highest log-likelihood, while Startle-NNI (a maximum parsimony method) typically has the lowest weighted parsimony cost. However, in 16% of the cases the tree inferred by LAML has a lower weighted parsimony cost than the Startle-NNI tree. One possible reason is because Startle-NNI is a heuristic algorithm that does not always find the true optimum and since there is a high correlation between weighted parsimony

and log-likelihood, LAML will typically also produce trees achieving a good weighted parsimony score.

Next, we evaluate the accuracy of branch lengths estimated by LAML, where we give the method the true tree topology and use the EM algorithm to estimate branch lengths and missing data rates. Overall, the branch length estimates by LAML are accurate: the ratio of estimated to true branch length is around 1 regardless of that branch's distance to the root (Fig. 2B, purple line), and there is a high correlation ($R = 0.72$) between the true branch lengths and LAML's estimated branch lengths (Fig. S4). Furthermore, the estimated mutation rate $\hat{\lambda} = 0.094$ is very close to the true mutation rate $\lambda = 0.095$.

Although LAML is the only method that infers branch lengths in time units, we also made a best attempt comparison of LAML's branch lengths to those inferred by other methods. Note that in order to perform this evaluation, we need a given method to be able to infer branch lengths for the *true tree topology*. Unfortunately, none of the 4 other software packages (Cassiopeia-Greedy, Cassiopeia-NJ, Startle, and DCLEAR) perform branch length inference on a given (i.e. fixed) tree topology. Therefore, we only benchmark against the *MP branch lengths* – standard practice to produce branch lengths on trees inferred by maximum parsimony methods, previously used for downstream analyses on lineage tracing data [50]. To compute MP branch lengths, we use Sankoff's algorithm to infer the ancestral sequences and compute branch length as the number of mutations separating nodes from one another. We note that the MP branch lengths are not measured in time units; instead, each MP branch length is an estimate of the *number of mutations* that occurred on the corresponding branch (i.e. mutation count). As a best effort "conversion" of the MP branch lengths to time units, we divide them by the true mutation rate $\lambda = 0.095$ before evaluating them with respect to the true branch lengths. Overall, the MP branch lengths have much lower correlation to the true branch lengths ($R = 0.39$) than LAML branch lengths. More importantly, MP branch lengths also have a *systematic bias* where the ratios to the true branch lengths decrease with the distance from the root (i.e. branches further from the root are more seriously underestimated) (Fig. 2B, red and green lines).

We hypothesize that the observed systematic bias of MP branch lengths is the result of using mutation counts as a proxy to measure branch lengths. Because of the decay in the number of available target sites, *mutation counts are not proportional to branch lengths in time units* under the PMM model. Instead, the ratio of mutation counts to time-unit branch lengths actually degrades over time. To test this hypothesis, we compute both the theoretical and empirical mutation counts under the PMM model and added them as the two alternative means for branch length computation. Under the PMM model, the theoretical expected mutation count of each branch $e = (u, v)$ is $(1 - e^{-\delta_e(1+\nu)})e^{-d(r_T,u)(1+\nu)}$, where $d(r_T, u)$ denotes the distance from the root r_T to u (we omit a proof here, but the derivation is straight-forward using Eq. 2). The empirical mutation counts are simply collected from simulated data. We also divide both the theoretical and empirical mutation counts by λ to "convert" them to time units. Consistent with our hypothesis, we observe that both the theoretical and empirical mutation counts are systematically biased if used as branch lengths, and that branches further from the root are more seriously underestimated (Fig. 2B, red and green lines).

4.1.2 LAML accurately infers heritable missing and dropout rates

LAML produces accurate estimates for the missing data rates on all five model conditions, and furthermore these estimates are robust against topological error (Fig. S1A). However, more accurate tree topologies help reduce errors in these estimates. For instance, the values we estimate on the LAML tree topology (which achieved the lowest RF error) have the lowest RMSE. Note that LAML is the only method that estimates both heritable missing and dropout rates, so we cannot compare to other methods on this aspect.

We additionally evaluate LAML's ability to distinguish heritable missing data and dropout. LAML achieves nearly perfect classification accuracy when all missing entries (which account for 25% of the whole data) arise solely from dropout (heritable: 0%, dropout: 100%, denoted as: h0d100) or solely from heritable missing data (heritable: 100%, dropout: 0%, denoted as: h100d0). However, the accuracy drops when there are mixed-type missing data, with the lowest accuracy (88% on true tree, 87% on true topology with estimated branch lengths, and 85% on estimated topology) observed in model condition h50d50 (Fig. 2C). Interestingly, the classification accuracy is higher with lower variance on data where heritable missing events account for fewer of the missing entries (h25d75), compared to when heritable missing events account for more of the missing entries (h75d25), suggesting that LAML correctly classifies more dropout missing events than heritable missing events, possibly due to an overestimation of dropout. In all model conditions, LAML's missing data imputation is robust against errors in tree topology and branch length ($\leq 2\%$ difference when using the true versus estimated trees).

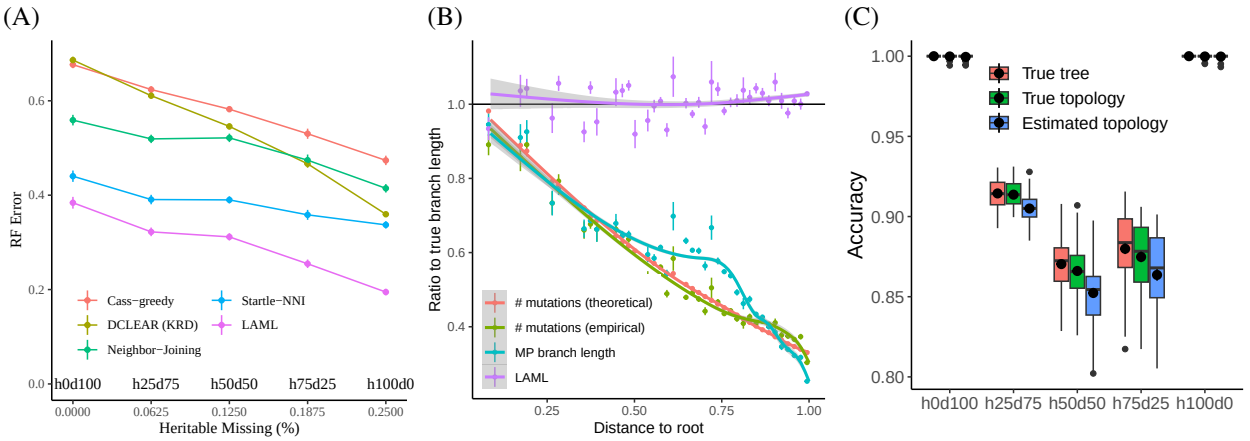

Figure 2: Comparison of LAML and other methods on simulated data. (A) Evaluation of tree topologies using normalized RF error under 5 different percentages of heritable ('h') and dropout ('d') of missing data. (B) The ratio of estimated to true branch lengths on true tree topologies Black line shows the perfect fit where estimate/true = 1. The x-axis is discretized into 50 bins and the estimated branch length by each method (or measure) is averaged for each bin, shown with error bar. (C) Classification accuracy for LAML in distinguishing the two types of missing data – heritable and dropout – shown for 3 scenarios: (red) True tree topology T_{true} with true parameters $\mathbf{\Theta}_{\text{true}}$ given; (green) The true topology T_{true} is given, the MLE $\hat{\mathbf{\Theta}}$ is estimated by LAML; (blue) both T and $\hat{\mathbf{\Theta}}$ are estimated by LAML.

4.2 Mouse lung adenocarcinoma (KP-Tracer)

We analyze a dataset of lung adenocarcinoma metastasis in a mouse model [50]. We show below that LAML not only infers a cell lineage tree that is *more plausible* than the published trees inferred by Cassiopeia [50] and Startle [38], but also our time-resolved branch lengths enable the study of *metastasis progression*, a novel application of the cell lineage tree.

4.2.1 Analysis of branch length estimation on six representative samples

We first use LAML to estimate branch lengths for the published tree topologies (inferred by Cassiopeia) for six representative samples (3432_NT_T1, 3435_NT_T4, 3520_NT_T1, 3703_NT_T1, 3703_NT_T2, and 3703_NT_T3) that have a high number of target sites (ranging from 29 to 73) and moderate numbers of cells (29 to 294). Note that in this experiment we fixed the tree topology and only estimate branch lengths. We compare the branch lengths estimated by LAML to the MP branch lengths (using Sankoff's algorithm to estimate ancestral states and computing mutation counts on each branch, which are treated as branch lengths on maximum parsimony trees). We observe that LAML's branch lengths are not proportional to the MP branch lengths (Fig. 3A), especially near the leaf nodes where many long LAML's branches have a low mutation count by maximum parsimony (red branches). The ratio of the maximum parsimony branch length to the branch inferred by LAML (MP/LAML) decreases with the distance to the root (Fig. 3B). Such a pattern is consistent with the results shown on simulated data (Fig. 2B) where the MP branch lengths decrease with distance to the root while LAML branch length estimates do not, and is observed in all of the six tested samples (Fig. S8A in Supplementary). Inspired by earlier studies [50, 51], we use mutual information between gene expression distances and the phylogenetic distances obtained from estimated branch lengths as an evaluation metric for branch length estimation. We use this metric to benchmark the branch lengths inferred by LAML against the MP branch lengths and "topology distances" - a baseline method that assigns a unit length to every branch. We find that LAML's phylogenetic distances yield substantially higher mutual information than the counterparts (Fig. 3C), and the same pattern is observed when testing on all six samples (Fig. S8B in Supplementary). Further details about data processing and evaluation metrics are in Supplementary data.

4.2.2 Analysis of lineage tree topology on the largest sample

We use LAML to simultaneously infer the lineage tree topology and time-resolved branch lengths of the largest sample of KP-Tracer. This sample (3724_NT_T1_All) has $21,108$ cells in total, but because many of the cells have identical CRISPR/Cas9 sequences, we only have 1,461 unique cells after deduplication. The tree inferred by LAML on the deduplicated data is shown in Fig. 4A. Interestingly, LAML produces a very different tree topology from the

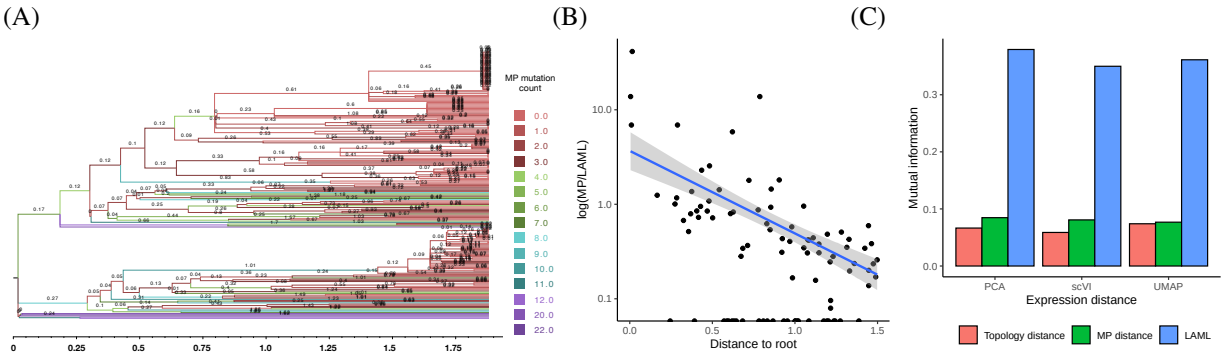

Figure 3: Evaluation of branch lengths on the cell lineage tree for sample 3432_NT_T1 from the KP-Tracer mouse lung adenocarcinoma lineage tracing study. (A) The tree topology published in [50] with branch lengths estimated by LAML. Numbers on branches are the lengths estimated by LAML and colors correspond to the number of mutations inferred by maximum parsimony. (B) The log-ratio between the length of a branch estimate by maximum parsimony and LAML versus the distance between the root node and the end of the branch. (C) Mutual information between gene expression and phylogenetic distances. Gene expression distance is computed in latent spaces computed by PCA, scVI, or UMAP. Phylogenetic distance computed by the sum of branch lengths between leaves using branches of length one (Topology distance), branch length estimates from LAML, or branch lengths from maximum parsimony (MP distance).

published trees inferred by Cassiopeia-Hybrid [50] (normalized RF 0.81) and Startle-NNI [38] (normalized RF 0.58). We discuss the biological implications of these different inferred topologies in this section and the next.

We benchmark the topology of LAML against Cassiopeia-Hybrid and Startle-NNI using the same procedure as in [38]: construct a cell *migration graph* [13] for each of these tree topologies and analyze these migration graphs. A migration graph is constructed as follows. Using the annotated anatomical location of the sampled cells and a tree topology, we first infer the anatomical locations of the ancestral nodes using Sankoff's algorithm (as described in [13, 38]). Next, we collapse all branches that have parent and child nodes labeled identically. Finally, we merge all branches that have identical labels on both ends and use the multiplicity as edge weight. Thus, the migration graph constructed on a tree topology is a directed graph whose *nodes are anatomical sites*, *edges indicate migration events*, and *edge weights show the number of times a migration event happened* on that tree (Fig. 4A). Migration events are classified into three types: primary tumor to a non-primary anatomical site (*metastasis*), non-primary anatomical site to the primary tumor (*reseeding*), and non-primary anatomical site to a different non-primary anatomical site (*metastasis to metastasis*). Similar to [38], we define the *migration cost* as the sum of edge weights in the migration graph and use it as an evaluation metric for lineage tree topology. We also define the *reseeding cost* as the total number of reseeding events and use it as a secondary evaluation metric. See Supplementary Section S2.3 for further information.

Compared to the two published trees, LAML achieves both lower migration cost and lower reseeding cost (Table S4). In addition, the migration graph of LAML is more plausible than the two published trees (Fig. 4A), as it shows a clearly lower *reseeding to seeding ratio* (4 to 11) than Startle-NNI (8 to 9) and Cassiopeia-Hybrid (11 to 7) for liver metastasis II and a clearly lower total number of migration events for liver metastasis III (22 versus 35 versus 36). Interestingly, all tree topologies agree on a high number of reseeding events for soft tissue metastasis. This result is not surprising, however, as these events could have resulted from local spread of tumors prior to metastasis through the blood or lymphatic system to distant anatomical sites [40]. Nevertheless, the LAML topology is still more plausible as it yields the lowest number of reseeding events (37), followed by Startle-NNI (48), and finally Cassiopeia-Hybrid (57). All methods detect a single migration event from soft tissue metastasis to liver metastasis III, and no other migration events between other pairs of non-primary sites.

4.2.3 Timing metastasis progression with time-resolved branch lengths

We use the time-scaled branch estimated by LAML to infer the time of cellular migrations spread between the primary tumor and the multiple anatomical sites. This analysis of the timing of metastasis was not performed in the original KP-tracer publication [50] because the lineage trees inferred by Cassiopeia do not have time-scaled branch lengths. Our results show three epochs of metastasis progression with distinct patterns of cell migration (Fig. 4B): Primary Growth (month 0-1), Metastasis (month 1-3), and Late Metastasis (month 3-6). In the Primary Growth epoch, cells grow and divide within the Primary tumor but do not migrate (i.e. no metastasis) (Fig. 4C). The Metastasis epoch

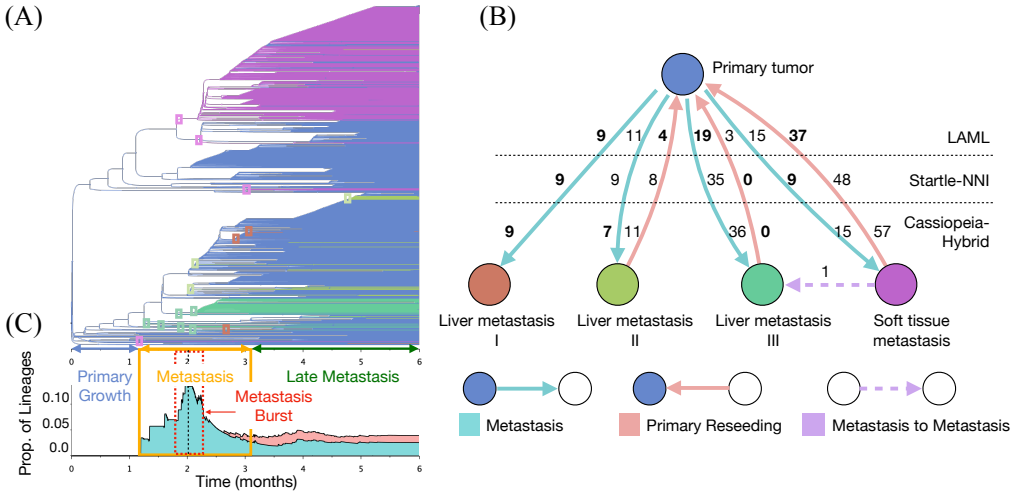

Figure 4: Migrations between anatomical sites in mouse lung adenocarcinoma sample 3724_NT_T1_All. (A) Migration graph inferred from the de-duplicated trees estimated using Cassiopeia-Hybrid, Startle-NNI, and LAML. Nodes represent the anatomical sites, arrows represent migration events, and numbers represent number of migration events inferred from each estimated tree. Bold indicates the lowest number for each migration event. Note that the Metastasis to Metastasis event from Soft tissue to Liver III appears in all three methods. (B) LAML's estimated phylogeny on 1,461 cells (de-duplicated data). Time-scaled branches are colored by the inferred anatomical location of incident child node. Small boxes indicate metastasis events. Time is divided into three epochs: Primary Growth (blue), Metastasis (yellow), and Late Metastasis (green). Red dashed lines mark the Metastasis Burst and the black dashed line marks the peak of metastasis burst. (C) Stacked area chart representing the proportion of lineages (branches) annotated as metastasis (aqua), reseeding (red) and metastasis to metastasis (purple) at each time point.

starts at around month 1, with a gradual increase in the number of metastasis events and lasts until around month 2, followed by a sudden surge (i.e. Metastasis Burst) and a later gradual reduction until month 3 (Fig. 4C, aqua area plot). The first metastasis event is a soft tissue metastasis (around the end of month 1), consistent with previous reports of local spread of tumors in soft tissue [40] (Fig. 4C, red area plot). Notably, in month 2, there is a surge of metastases (i.e. the Metastasis Burst window, see Fig. 4C) that lasts about half a month (from month 1.75 to 2.25). We observe that during Metastasis Burst, the rate of metastasis (i.e. the number of metastases per cell lineage per month) increases from a background rate of 0.00349 to a rate of 0.11328, a greater than 30-fold increase, and during this short time there are metastases to multiple anatomical sites. After the metastasis burst, there is a gradual decrease in the number of migrations, until the beginning of month 3 when the number reaches a stable state. We mark this time point as the start of the Late Metastasis epoch. In Late Metastasis, there is a stable proportion of lineages with migrations at every time point. In addition, we also observe reseeding events during this epoch, where cells are inferred as migrating back to the primary tumor from distant metastases [13]. Interestingly, the ratio of reseeding to all migration events is maintained at a constant value of about 0.4 throughout the Late Metastasis epoch. Because there are a low number of unmodified target sites from month 3 forward (≤ 3), the phylogenetic signal is weaker during this epoch, and thus we expect higher uncertainty in the inference of both tree branch lengths and metastasis events.

Finally, we analyze the missing data rates inferred by LAML on this largest sample of the dataset. According to our estimates, this sample has mixed-type missing data: dropout rate $\phi = 0.043$ and heritable rate $\nu = 0.010$. Among the 753 missing entries in the data, LAML imputes 578 entries to be dropout and 175 entries to be heritable missing. The 175 heritable missing entries are attributed to 80 heritable missing events, with each event introducing 2 to 4 missing entries to the character matrix. Interestingly, the signal from heritable missing is lost when we estimate missing data rates and branch lengths using the Startle-NNI topology ($\phi = 0.057$, $\nu = 0$).

4.3 Drosophila cellular development (intMEMOIR)

We also apply LAML to a development study in *Drosophila* which uses the intMEMOIR dynamic lineage tracing technology [10, 19]. The dataset has ground truth lineage trees collected through microscopic observation of the cell division process. The Bayesian MCMC algorithm TiDeTree, which also uses a continuous-time Markov model and allows heritable missing data, can be used to infer trees from data obtained with this dynamic lineage tracing

technology. We compare LAML, TiDeTree, Startle-NNI and Cassiopeia on this dataset. LAML achieves the lowest average RF error compared to the ground truth tree of 0.53, followed by TiDeTree of 0.56, Startle-NNI of 0.58, and Cassiopeia of 0.63 (Figure S10A and Supplementary data S2.4). Interestingly, although the tree topologies inferred by the different methods have similar normalized RF distance from the true tree, they have large average RF distance from the LAML tree (Startle-NNI: 0.31, Cassiopeia: 0.46, TiDeTree (Pub): 0.46) (see Figure S10B). As the data has only ten heritable target sites and seems to support several very different tree topologies, there seems to be low overall phylogenetic signal, making lineage tree reconstruction difficult. Note that we were unable to run TiDeTree on either the simulated data or the KP-Tracer data above – both because the software does not scale to large numbers of cells and does not handle distinct per-site alphabets.

4.4 Runtime Evaluation

First, we explore the runtime of the LAML EM algorithm for branch length estimation. As supported by the theoretical complexity, in the special case where the heritable missing rate $\nu = 0$, both the E-step and the M-step run in linear time (Fig. S5 black and yellow lines). Although we could not analyze the theoretical complexity of the M-step block coordinate ascent algorithm when $\nu \neq 0$, our empirical runtime is also close to linear (Fig. S5 grey line). However, the overall runtime of the EM algorithm, depends on the number of EM iterations needed (i.e. the convergence rate of the EM algorithm) and seems to be worse than linear (Fig. S6).

Next, we compare the overall runtime (the combined runtime of topology search and branch length estimation) of LAML to other methods. On simulated data (250 cells), Cassiopeia-Greedy and Neighbor Joining are the two fastest methods (Fig. S7). The runtime of LAML is comparable to Startle-NNI. Since we use the Startle-NNI topology to initialize LAML, we added the Startle-NNI runtime to the reported runtime of LAML. The average runtime difference between LAML and Startle-NNI is ≈ 100 minutes. We also benchmarked the runtime of LAML against TiDeTree on the intMEMOIR data (Fig. S11), which has a much smaller number of cells (≤ 77) and target-sites (10). Due to the long runtime of TiDeTree, we only ran it for 10 of the largest samples that have from 51 to 77 cells. Overall, we observe that LAML is much faster than TiDeTree: LAML finished in just under a minute while TiDeTree took ≈ 9 hours to infer the lineage tree of each sample.

5 Discussion

Single-cell lineage tracing technologies continue to advance in scale and resolution, increasing the need for accurate methods to infer lineage trees from the resulting data. We introduce LAML, a maximum likelihood method that jointly estimates tree topology, time-scaled branch lengths, dropout rate, and heritable missing rate under the newly developed PMM model for dynamic lineage tracing. LAML can be run to (i) estimate time-scaled branch lengths and missing data rates on a given tree topology, or (ii) estimate the tree topology as well as all parameters. We show on simulated data that LAML is more accurate for topology estimation than many existing methods, including a greedy approach (Cassiopeia-greedy), parsimony approaches (Cassiopeia-ILP, and Startle-NNI), and Neighbor-Joining (with multiple distance matrices, including DCLEAR). Importantly, while the simulation used parameters derived from real lineage tracing datasets, the data was generated under our proposed generative model (PMM), and so it not surprising that LAML - a maximum likelihood inference under this model - performs well. Additional simulations with model misspecification, correlation of heritable missing between consecutive sites, additional sc-Seq errors, variation of mutation rates among sites, or changes in mutation rate across time would be helpful. Future work should explore the robustness of LAML on these violations of the model and its performance in comparison to other methods.

We show that LAML produces more plausible lineage trees on a mouse model of lung adenocarcinoma [50]. We demonstrate that branch lengths inferred under a maximum parsimony model are systematically biased, as branches further from the root are more severely underestimated. Using the PMM model that realistically reflects the decay of mutation counts through time, we enable the estimation of branch lengths in time unit, which are needed in studying the dynamic of cell growth and migration through time. On the largest sample of mouse lung adenocarcinoma [50], we demonstrate that LAML can be used to time metastasis events, a novel application of the cell lineage tree. Importantly, LAML discovers an interesting pattern of metastasis progression which consists of 3 time epochs . The Primary Growth epoch spans about 1 month during which there is no metastasis event happens. The Metastasis epoch spans from month 1 to 3, during which time there is a surge of metastasis events. The Late Metastasis epoch starts at around month 3 when there is a stable proportion of lineages undergoing metastasis and Primary reseeding at every time point.

There are multiple directions for future work. First, our biological findings will be strengthened by additional phylogenetic analyses such as bootstrapping [12] and relaxing the molecular clock assumption [23]. Second, the PMM model can be extended to model other aspects of cellular development, such as the presence of multiple progenitors,

clustering of cell types and gene expression, and the spatial structure of the tissue or tumor. Additionally, the current PMM model does not incorporate the empirical observation that dropouts occur in blocks (that is, that consecutive target sites are dropped). Future work could aim to relax the assumption of independence across sites. Third, the sc-MAIL algorithm could be improved by further incorporating subtree-pruning-and-regrafting (SPR) moves through tree topology space. Finally, although LAML's current runtime for estimation of the tree topology is acceptable scaling to $1,461$ cells on the largest sample from the KP-Tracer dataset, larger lineage-tracing datasets will likely be generated in the future. Both the runtime performance and scalability of LAML can be improved by parallelization [31] and algorithmically by a divide-and-conquer [48, 29] or hybrid framework, where one combines the accuracy of maximum likelihood with the speed of distance-based methods.

Our results show important implications for technological design and applications of dynamic lineage tracing. First, downstream analyses such as developmental tree inference, subclonal dynamics, and trajectory inference, benefit from the time-scaled branch lengths estimated by LAML. Second, experimentalists should aim for control over both the overall amount of missing data and the ratio of dropout to heritable missing types when developing new lineage tracing technology. Third, our probabilistic approach to tree inference could aid in the time-scaled reconstruction of *unobserved intermediate cell states* [32, 50]. Finally, more refined time calibration information, such as the fluorescence tagging in the lineage tracing vector [50, 43] or time points of multiple rounds of editing (present in technologies such as CARLIN, iTracer, scGESTALT), can be leveraged to better guide the conversion of the ML branch lengths to a time scale.

Acknowledgements

This research was supported by NIH/NCI grant U24CA248453 to B.J.R. U.M. was funded by the Presidential Postdoctoral Fellowship at Princeton University. G.C. was funded by the NSF GRFP DGE grant 2039656. We thank Palash Sashittal and Henri Schmidt for help with running the Startle code; Sophie Seidel for help with running the TiDeTree code; and Michelle Chan for helpful discussion.

References

[1] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[2] Anna Alemany, Maria Florescu, Chloé S Baron, Josi Peterson-Maduro, and Alexander Van Oudenaarden. Whole-organism clone tracing using single-cell sequencing. *Nature*, 556(7699):108–112, 2018.

[3] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019. URL: `http://docs.mosek.com/9.0/toolbox/index.html`.

[4] Adriano Bolondi, Benjamin K. Law, Helene Kretzmer, Seher Ipek Gassaloglu, Christina Riemenschneider, Dian Yang, René Buschow, Maria Walther, Jesse V. Veenvliet, Alexander Meissner, Zachary D. Smith, and Michelle M. Chan. Reconstructing transient progenitor field dynamics in stem cell-derived embryoids. In Review, 2023.

[5] Sarah Bowling, Duluxan Sritharan, Fernando G Osorio, Maximilian Nguyen, Priscilla Cheung, Alejo Rodriguez-Fraticelli, Sachin Patel, Wei-Chien Yuan, Yuko Fujiwara, Bin E Li, et al. An engineered crispr-cas9 mouse line for simultaneous readout of lineage histories and gene expression profiles in single cells. *Cell*, 181(6):1410–1422, 2020.

[6] Francisco Caiado, Diogo Maia-Silva, Carolina Jardim, Nina Schmolka, Tânia Carvalho, Cláudia Reforço, Rita Faria, Branka Kolundzija, André E Simões, Tuncay Baubec, et al. Lineage tracing of acute myeloid leukemia reveals the impact of hypomethylating agents on chemoresistance selection. *Nature communications*, 10(1):4986, 2019.

[7] Michelle M Chan, Zachary D Smith, Stefanie Grosswendt, Helene Kretzmer, Thomas M Norman, Britt Adamson, Marco Jost, Jeffrey J Quinn, Dian Yang, Matthew G Jones, et al. Molecular recording of mammalian embryogenesis. *Nature*, 570(7759):77–82, 2019.

[8] Cheng Chen, Yuanxin Liao, and Guangdun Peng. Connecting past and present: single-cell lineage tracing. *Protein & Cell*, 13(11):790–807, 2022.

[9] Junhong Choi, Wei Chen, Anna Minkina, Florence M Chardon, Chase C Suiter, Samuel G Regalado, Silvia Domcke, Nobuhiko Hamazaki, Choli Lee, Beth Martin, et al. A time-resolved, multi-symbol molecular recorder via sequential genome editing. *Nature*, 608(7921):98–107, 2022.

[10] Ke-Huan K Chow, Mark W Budde, Alejandro A Granados, Maria Cabrera, Shinae Yoon, Soomin Cho, Ting-Hao Huang, Noushin Koulena, Kirsten L Frieda, Long Cai, et al. Imaging cell lineage with a synthetic digital recording system. *Science*, 372(6538):eabb3099, 2021.

[11] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[12] Bradley Efron, Elizabeth Halloran, and Susan Holmes. Bootstrap confidence levels for phylogenetic trees. *Proceedings of the National Academy of Sciences*, 93(23):13429–13429, 1996.

[13] Mohammed El-Kebir, Gryte Satas, and Benjamin J Raphael. Inferring parsimonious migration histories for metastatic cancers. *Nature genetics*, 50(5):718–726, 2018.

[14] Joseph Felsenstein and Joseph Felenstein. *Inferring phylogenies*, volume 2. Sinauer associates Sunderland, MA, 2004.

[15] Jean Feng, William S DeWitt III, Aaron McKenna, Noah Simon, Amy D Willis, and Frederick A Matsen IV. Estimation of cell lineage trees by maximum-likelihood phylogenetics. *The Annals of Applied Statistics*, 15(1):343–362, 2021.

[16] Kirsten L Frieda, James M Linton, Sahand Hormoz, Joonhyuk Choi, Ke-Huan K Chow, Zakary S Singer, Mark W Budde, Michael B Elowitz, and Long Cai. Synthetic recording and in situ readout of lineage information in single cells. *Nature*, 541(7635):107–111, 2017.

[17] Nir Friedman, Matan Ninio, Itsik Pe'er, and Tal Pupko. A structural em algorithm for phylogenetic inference. In *Proceedings of the fifth annual international conference on Computational biology*, pages 132–140, 2001.

[18] Sudhindra R Gadagkar and Sudhir Kumar. Maximum likelihood outperforms maximum parsimony even when evolutionary rates are heterotachous. *Molecular biology and evolution*, 22(11):2139–2141, 2005.

[19] Wuming Gong, Alejandro A Granados, Jingyuan Hu, Matthew G Jones, Ofir Raz, Irepan Salvador-Martínez, Hanrui Zhang, Ke-Huan K Chow, Il-Youp Kwak, Renata Retkute, et al. Benchmarked approaches for reconstruction of in vitro cell lineages and in silico models of c. elegans and m. musculus developmental trees. *Cell systems*, 12(8):810–826, 2021.

[20] Wuming Gong, Hyunwoo J Kim, Daniel J Garry, and Il-Youp Kwak. Single cell lineage reconstruction using distance-based algorithms and the r package, dclear. *BMC bioinformatics*, 23(1):103, 2022.

[21] Masami Hasegawa and Miyako Fujiwara. Relative efficiencies of the maximum likelihood, maximum parsimony, and neighbor-joining methods for estimating protein phylogeny. *Molecular phylogenetics and evolution*, 2(1):1–5, 1993.

[22] Zhisong He, Ashley Maynard, Akanksha Jain, Tobias Gerber, Rebecca Petri, Hsiu-Chuan Lin, Malgorzata Santel, Kevin Ly, Jean-Samuel Dupré, Leila Sidow, et al. Lineage recording in human cerebral organoids. *Nature methods*, 19(1):90–99, 2022.

[23] Simon YW Ho and Sebastián Duchêne. Molecular-clock methods for estimating evolutionary rates and timescales. *Molecular ecology*, 23(24):5947–5965, 2014.

[24] Matthew G Jones, Alex Khodaverdian, Jeffrey J Quinn, Michelle M Chan, Jeffrey A Hussmann, Robert Wang, Chenling Xu, Jonathan S Weissman, and Nir Yosef. Inference of single-cell phylogenies from lineage tracing data using cassiopeia. *Genome biology*, 21(1):1–27, 2020.

[25] Reza Kalhor, Kian Kalhor, Leo Mejia, Kathleen Leeper, Amanda Graveline, Prashant Mali, and George M Church. Developmental barcoding of whole mouse via homing crispr. *Science*, 361(6405):eaat9804, 2018.

[26] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.

[27] Aaron McKenna, Gregory M Findlay, James A Gagnon, Marshall S Horwitz, Alexander F Schier, and Jay Shendure. Whole-organism lineage tracing by combinatorial and cumulative genome editing. *Science*, 353(6298):aaf7907, 2016.

[28] Aaron McKenna and James A Gagnon. Recording development with single cell dynamic lineage tracing. *Development*, 146(12):dev169730, 2019.

[29] Erin K Molloy and Tandy Warnow. Njmerge: a generic technique for scaling phylogeny estimation methods and its application to species trees. In *Comparative Genomics: 16th International Conference, RECOMB-CG 2018, Magog-Orford, QC, Canada, October 9-12, 2018, Proceedings 16*, pages 260–276. Springer, 2018.

[30] Nicholas E Navin and James Hicks. Tracing the tumor lineage. *Molecular oncology*, 4(3):267–283, 2010.

[31] Michael Ott, Jaroslaw Zola, Alexandros Stamatakis, and Srinivas Aluru. Large-scale maximum likelihood-based phylogenetic analysis on the ibm bluegene/l. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–11, 2007.

[32] Khalil Ouardini, Romain Lopez, Matthew G Jones, Sebastian Prillo, Richard Zhang, Michael I Jordan, and Nir Yosef. Reconstructing unobserved cellular states from paired single-cell lineage tracing and transcriptomics data. *bioRxiv*, pages 2021–05, 2021.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[34] Bushra Raj, Daniel E Wagner, Aaron McKenna, Shristi Pandey, Allon M Klein, Jay Shendure, James A Gagnon, and Alexander F Schier. Simultaneous single-cell profiling of lineages and cell types in the vertebrate brain. *Nature biotechnology*, 36(5):442–450, 2018.

[35] Sebastien Roch. A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(1):92–94, 2006.

[36] Brian C Ross. Mutual information between discrete and continuous data sets. *PloS one*, 9(2):e87357, 2014.

[37] Naruya Saitou and Masatoshi Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, 1987.

[38] Palash Sashittal, Henri Schmidt, Michelle M Chan, and Benjamin J Raphael. Startle: a star homoplasy approach for crispr-cas9 lineage tracing. *bioRxiv*, pages 2022–12, 2022.

[39] Sophie Seidel and Tanja Stadler. TiDeTree: A bayesian phylogenetic framework to estimate single-cell trees and population dynamic parameters from genetic lineage tracing data. *Proceedings of the Royal Society B*, 289(1986):20221844, 2022.

[40] Sohrab P Shah. Copy number signatures in ovarian cancer. *Nature Genetics*, 50(9):1208–1209, 2018.

[41] Adam Siepel. Expectation maximization for combined phylogenetic and hidden markov models. 2002.

[42] Elijah R Sommer, Giulia C Napoli, Cindy H Chau, Douglas K Price, and William D Figg. Targeting the metastatic niche: Single-cell lineage tracing in prime time. *Iscience*, 26(3), 2023.

[43] Bastiaan Spanjaard, Bo Hu, Nina Mitic, Pedro Olivares-Chauvet, Sharan Janjuha, Nikolay Ninov, and Jan Philipp Junker. Simultaneous lineage tracing and cell-type identification using crispr–cas9-induced genetic scars. *Nature biotechnology*, 36(5):469–473, 2018.

[44] David L Swofford, Peter J Waddell, John P Huelsenbeck, Peter G Foster, Paul O Lewis, and James S Rogers. Bias in phylogenetic estimation and its relevance to the choice between parsimony and likelihood methods. *Systematic biology*, 50(4):525–539, 2001.

[45] Daniel E Wagner and Allon M Klein. Lineage tracing meets single-cell omics: opportunities and challenges. *Nature Reviews Genetics*, 21(7):410–427, 2020.

[46] Daniel E Wagner, Caleb Weinreb, Zach M Collins, James A Briggs, Sean G Megason, and Allon M Klein. Single-cell mapping of gene expression landscapes and lineage in the zebrafish embryo. *Science*, 360(6392):981–987, 2018.

[47] Robert Wang, Richard Y Zhang, Alex Khodaverdian, and Nir Yosef. Theoretical guarantees for phylogeny inference from single-cell lineage tracing. *bioRxiv*, 2021.

[48] Tandy Warnow. Divide-and-conquer tree estimation: Opportunities and challenges. *Bioinformatics and Phylogenetics: Seminal Contributions of Bernard Moret*, pages 121–150, 2019.

[49] F Alexander Wolf, Philipp Angerer, and Fabian J Theis. Scanpy: large-scale single-cell gene expression data analysis. *Genome biology*, 19:1–5, 2018.

[50] Dian Yang, Matthew G Jones, Santiago Naranjo, William M Rideout III, Kyung Hoi Joseph Min, Raymond Ho, Wei Wu, Joseph M Replogle, Jennifer L Page, Jeffrey J Quinn, et al. Lineage tracing reveals the phylodynamics, plasticity, and paths of tumor evolution. *Cell*, 185(11):1905–1923, 2022.

[51] Hamim Zafar, Chieh Lin, and Ziv Bar-Joseph. Single-cell lineage tracing by integrating crispr-cas9 mutations with transcriptomic data. *Nature communications*, 11(1):1–14, 2020.

S1 Supplementary methods

Contents

S1.1 Extended notations . . . 1
S1.2 Computing the log-likelihood given T and Θ . . . 2
S1.3 The edge-inside likelihood, the outside likelihood, and the posterior . . . 3
S1.3.1 The edge-inside likelihood . . . 3
S1.3.2 The outside likelihood . . . 3
S1.3.3 The posterior . . . 3
S1.4 The EM algorithm: derivation and pseudocode . . . 4
S1.4.1 Derivation (Proof of Eq. 12 in the main text) . . . 4
S1.5 Reducing the complexity of the E-step . . . 5
S1.5.1 Special properties of α-clades and z-branches . . . 5
S1.5.2 The linear-time algorithm . . . 11
S1.6 Solving the M-step . . . 15
S1.6.1 Proof of convexity by variable blocks . . . 15
S1.6.2 The block coordinate ascent algorithm . . . 16
S1.7 Simulated annealing for topology search . . . 16

S1.1 Extended notations

Consider a dynamic lineage tracing procedure consisting of K target-sites. At the beginning of the experiment, there is exactly one *progenitor cell* that has all target-sites in the *unmutated state*, denoted by 0. During the CRISPR/Cas9 process, each target-site k of any cell can either mutate into one of the $M^{(k)}$ *mutated states* in $\{1, ..., M^{(k)}\}$ or go to the *silent state*, denoted by -1. During the single-cell sequencing (sc-Seq) process, cells in the silent state cannot be determined and will show up in the *missing state*; in addition, cells in the other states can also turn into the missing state because of dropout. We denote the missing state by "?" and refer to the set $\mathscr{A}^{(k)} = \{?, -1, 0, 1, ..., M^{(k)}\}$ as the *alphabet* of the target-site k. We assume that there are N cells sequenced by sc-Seq, so the dynamic lineage tracing data consists of $N \times K$ entries. Following the convention in phylogenetics, hereafter, we refer to *target-site* as *site* for brevity.

We represent the cell lineage of the lineage tracing data by a rooted tree T. Let $\mathscr{L}_T$, $\mathscr{V}_T$, and $\mathscr{E}_T$ be the *set of leaves*, *set of nodes*, and *set of edges* of T, respectively. Let (u, v) be the edge in $\mathscr{E}_T$ from u to its child v (where $u, v \in \mathscr{V}_T$), and r_T be the root of T. Unless otherwise specified, e is used to denote the edge connecting u to v. Throughout this paper, we use *edge* and *branch* interchangeably. We assume that the root of T has exactly one child and all of its other internal nodes has exactly two children. For each node $v \in \mathscr{V}_T$, we let T_v be the clade of T rooted at v, $\mathscr{C}_T(v)$ be the leafset of T_v, and $\tilde{\mathscr{C}}_T(v) := \mathscr{L}_T \setminus \mathscr{C}_T(v)$. When the context is clear, the subscript T can be omitted for brevity.

Throughout this paper, we will use the following two conventions for random/realization matrices/vectors: (1) blackboard-bold letters (e.g. $\mathbb{A}, \mathbb{a}, \mathbb{B}, \mathbb{b}, \mathbb{C}, \mathbb{c}$) will be used to indicate a *random* entity while normal bold letters (e.g. **A,a,B,b,C,c**) will be used to indicate a realization. (2) the uppercase letters (e.g. $\mathbb{A}, \mathbb{B}, \mathbb{C}$ and **A,B,C**) are reserved for matrices while lowercase letters (e.g. $\mathbb{a}, \mathbb{b}, \mathbb{c}$ and **a,b,c**) are reserved for vectors. For a set $\mathscr{S}$ of N cells with K sites, we define the CRISPR/Cas9 sequences of $\mathscr{S}$ by a $N \times K$ matrix and refer to it as the *cell-state representation matrix* of $\mathscr{S}$, which has rows indexed by $\mathscr{S}$ and columns indexed by the sites. We let $\mathbb{X}$ be the cell-state representation random matrix of $\mathscr{V}_T$ *before sc-Seq*, and **X** be its realization. Similarly, we let $\mathbb{D}$ be the cell-state representation random matrix of $\mathscr{L}_T$ *after sc-Seq*, and **D** be its realization. Note that entries of **X** take values in $\mathscr{A}^{(k)} \setminus \{?\}$ while entries of **D** take values in $\mathscr{A}^{(k)} \setminus \{-1\}$. In dynamic lineage tracing, $\mathbb{D}$ is observed exactly once and $\mathbb{X}$ is hidden. Therefore, throughout this paper, **D** represents the lineage tracing data, which is also the sole *observed* realization of $\mathbb{D}$. Borrowing the terminology from previous work [38, 50], we also refer to **D** as the *character matrix*.

We use the following conventions for any cell-state representation matrices, both random and realization. First, the superscript $\bullet^{(k)}$ is used to denote any entity associated with site k. For example, $\mathbb{X}^{(k)}$ denotes the kth column of $\mathbb{X}$. Second, we use $\bullet^{(k)}(v)$ to refer to the element of the cell-state representation matrix associated with the row index v and column index k. For example, $\mathbb{X}^{(k)}(v)$ denotes the random variable in $\mathbb{X}$ that is associated with node v and site k. Finally, if **M** is a cell-state representation matrix and $\mathscr{S}$ is a subset of the row indices of **M**, then we let $\mathbf{M}|_{\mathscr{S}}$ be the matrix constructed from the rows of **M** whose indices belong to $\mathscr{S}$. The same convention is applied to $\mathbb{X}, \mathbb{D}$, and their

realizations. However, for the sole realization $\mathbf{D}$ of $\mathbb{D}$, we let $\mathbf{D}_v$ be the shorthand for $\mathbf{D}|_{\mathscr{C}_T(v)}$ and $\tilde{\mathbf{D}}_v$ be the shorthand for $\mathbf{D}|_{\tilde{\mathscr{C}}_T(v)}$.

Table S1: List of Notations

Notations	Explanation
N	Number of cells
K	Number of sites
$M^{(k)}$	Number of mutated states associated with site k
$\mathscr{A}^{(k)}$	Alphabet of site k
T, r_T	Cell lineage tree and its root
$\mathscr{V}_T, \mathscr{E}_T, \mathscr{L}_T$	Set of nodes, edges, leaves of tree T
$\mathscr{C}_T(v)$	Set of leaf indices under node v of tree T
$\mathbb{X}, \mathbf{X}$	Random matrix and its realization
$\mathbb{x}, \mathbf{x}$	Random vector and its realization
$\mathbb{X}^{(k)}, \mathbf{X}^{(k)}$	Random vector associated with site k and its realization
$\mathbb{X}_i^{(k)}, \mathbf{X}_i^{(k)}$	Random entry with index i associated with site k and its realization
$\mathbb{X}^{(k)}(v), \mathbf{X}^{(k)}(v)$	Random entry associated with site k and node v, and its realization
X, D	Realized values of random matrices $\mathbb{X}, \mathbb{D}$
$\mathbf{X}^{(k)}, \mathbf{D}^{(k)}$	Realized Random vector associated with site k
$\mathbf{Q}^{(k)}$	Rate matrix at site k
$\mathbf{\Psi}^{(k)}$	Transition matrix at site k
$\mathbf{\Phi}$	Transition matrix for dropout
δ_e	Length of branch e
ν, ϕ	Heritable missing and dropout rates

S1.2 Computing the log-likelihood given T and Θ

The log-likelihood can be computed in linear-time using Felsenstein's pruning algorithm [14]. In this section we show that the pruning algorithm can be applied to our model by presenting recurrence equations.

To describe his algorithm, Felsenstein introduced the concept of *partial likelihood* at a node in the tree. Here we use the term **inside likelihood** (see Definition 1) to refer to the partial likelihood described by Felsenstein, for the purpose of distinguishing it from the **outside likelihood** that will be defined later. Consider a node $u \in \mathscr{V}_T$. The *node-inside likelihood of* u is defined as follows:

Definition 1. *The node-inside likelihood (a.k.a partial likelihood) of u at site k w.r.t a realization α_u of $\mathbb{X}^{(k)}(u)$, denoted by $\mathcal{L}^{(k)}_{\mathsf{node}}(u, \alpha; T, \mathbf{\Theta})$, is the likelihood of observing $\boldsymbol{D}^{(k)}_u$ given $\mathbb{X}^{(k)}(u) = \alpha_u$. In other words:*

$$\mathcal{L}^{(k)}_{\mathsf{node}}(u, \alpha_u; T, \mathbf{\Theta}) := \mathcal{P}(\boldsymbol{D}^{(k)}_u | \mathbb{X}^{(k)}(u) = \alpha_u; T, \mathbf{\Theta}) \tag{S1}$$

The node-inside likelihoods of all nodes u at all states $\alpha_u \in \mathscr{A}^{(k)}$ can be computed in one bottom-up tree traversal, using the following recurrence:

$$\mathcal{L}^{(k)}_{\mathsf{node}}(u, \alpha_u; T, \mathbf{\Theta}) = \begin{cases} \mathbf{\Phi}(\alpha_u, \mathbf{D}^{(k)}_u(u)), \text{ if } u \in \mathscr{L}_T \\ \prod\limits_{v \in \mathscr{C}_T(u)} \sum\limits_{\alpha_v \in \mathscr{A}^{(k)}} \mathbf{\Psi}^{(k)}_e(\alpha_u, \alpha_v) \mathcal{L}^{(k)}_{\mathsf{node}}(v, \alpha_v; T, \mathbf{\Theta}), \text{ otherwise} \end{cases} \tag{S2}$$

In the PMM model, $\mathbb{X}^{(k)}(u)$ can only take state 0. Therefore, the full likelihood $\mathcal{P}(\mathbf{D}^{(k)}; T, \mathbf{\Theta})$ is simply the node-inside likelihood of the root node w.r.t. state 0. In other words, for all site k we have:

$$\mathcal{P}(\mathbf{D}^{(k)}; T, \mathbf{\Theta}) = \mathcal{L}^{(k)}_{\mathsf{node}}(u, 0; T, \mathbf{\Theta})$$

Thus, the full likelihood can be computed in one bottom up traversal. The complexity is therefore $\mathcal{O}(NKM)$ where N is the number of cells, K is the number of target sites, and M is the maximum alphabet size.

S1.3 The edge-inside likelihood, the outside likelihood, and the posterior

In this section, we define the **outside likelihood** and the **posterior** probabilities, which can be used later. We will show equations relating the inside likelihood, outside likelihood, and the posterior, and present a linear-time algorithm to compute all these entities for a tree (i.e. an extended version of the Felsenstein's prunning algorithm). We note that similar equations have been previously described in the literature (see [41] for example). Below we show the definitions and recurrence equations, but skip the proofs for brevity.

S1.3.1 The edge-inside likelihood

We first define the *inside likelihood of a branch*, as follows:

Definition 2. *The edge-inside likelihood of $e = (u, v)$ at site k w.r.t. a realization α_u of $\mathbb{X}^{(k)}(u)$, denoted by $\mathcal{L}^{(k)}_{\mathsf{edge}}(e, \alpha; T, \mathbf{\Theta})$, is the probability of observing $\boldsymbol{D}^{(k)}_v$ given $\mathbb{X}^{(k)}(u) = \alpha_u$. In other words:*

$$\mathcal{L}^{(k)}_{\mathsf{edge}}(e, \alpha_u; T, \mathbf{\Theta}) \coloneqq \mathcal{P}(\boldsymbol{D}^{(k)}_v | \mathbb{X}^{(k)}(u) = \alpha_u; T, \mathbf{\Theta}) \tag{S3}$$

The edge-inside likelihood of $e = (u, v)$ at any state α_u can be computed by summing over the inside likelihoods of v at all states α_v, as follows:

$$\begin{aligned}\mathcal{L}^{(k)}_{\mathsf{edge}}(e, \alpha_u; T, \mathbf{\Theta}) &= \sum_{\alpha_v \in \mathscr{A}(k)} \mathbf{\Psi}^{(k)}_e(\alpha_u, \alpha_v) \mathcal{P}(\mathbf{D}^{(k)}_v | \mathbb{X}^{(k)}(v) = \alpha_v; T, \mathbf{\Theta}) \\ &= \sum_{\alpha_v \in \mathscr{A}(k)} \mathbf{\Psi}^{(k)}_e(\alpha_u, \alpha_v) \mathcal{L}^{(k)}_{\mathsf{node}}(v, \alpha_v; T, \mathbf{\Theta})\end{aligned} \tag{S4}$$

Using Eq. S4, we can extend the bottom up traversal to also compute the inside likelihoods at all branches.

S1.3.2 The outside likelihood

Definition 3. *The outside likelihood of a node v at site k w.r.t. a realization α_v of $\mathbb{X}^{(k)}(v)$, denoted by $\mathcal{L}^{(k)}_{\mathsf{out}}(v, \alpha_u; T, \Theta)$, is the joint probability of $\mathbb{X}^{(k)}(v) = \alpha_v$ and $\tilde{\mathbb{D}}^{(k)}_v = \tilde{\boldsymbol{D}}^{(k)}_v$. In other words:*

$$\mathcal{L}^{(k)}_{\mathsf{out}}(v, \alpha_v; T, \Theta) \coloneqq \mathcal{P}(\tilde{\mathbb{D}}^{(k)}_v = \tilde{\boldsymbol{D}}^{(k)}_v, \mathbb{X}^{(k)}(v) = \alpha_v; T, \mathbf{\Theta}) \tag{S5}$$

The outside likelihood can be computed using the following topdown recurrence:

$$\mathcal{L}^{(k)}_{\mathsf{out}}(v, \alpha_v; T, \Theta) = \begin{cases} 1, \text{ if } v = r_T \text{ and } \alpha_v = 0 \\ 0, \text{ if } v = r_T \text{ and } \alpha_v \neq 0 \\ \mathbf{\Psi}^{(k)}_{(u,v)}(0, \alpha_v), \text{ if } u \text{ is the parent of } v \text{ and } u = r_T \\ \sum_{\alpha_u} \mathbf{\Psi}^{(k)}_{(u,v)}(\alpha_u, \alpha_v) \mathcal{L}^{(k)}_{\mathsf{edge}}((u, w), \alpha_u; T, \Theta) \mathcal{L}^{(k)}_{\mathsf{out}}(u, \alpha_u; T, \mathbf{\Theta}), \text{ otherwise} \end{cases} \tag{S6}$$

where in the last case, u and w are the parent and sister of v, respectively. In other words, one can compute the outside likelihood of any node from the outside likelihood of its parent and the inside likelihoods of the sister branch.

S1.3.3 The posterior

Next, we define the *posterior probability of a node* and *posterior probability of an edge*.

Definition 4. *The posterior probability of a node u at site k w.r.t. the realization α_u of $\mathbb{X}^{(k)}(u)$, denoted by $\mathcal{S}^{(k)}_{\mathsf{node}}(u, \alpha_u; T, \Theta)$, is the probability of $\mathbb{X}^{(k)}(u) = \alpha_u$ given $\mathbb{D}^{(k)} = \boldsymbol{D}^{(k)}$. In other words:*

$$\mathcal{S}^{(k)}_{\mathsf{node}}(u, \alpha_u; T, \mathbf{\Theta}) \coloneqq \mathcal{P}(\mathbb{X}^{(k)}(u) = \alpha_u | \mathbb{D}^{(k)} = \boldsymbol{D}^{(k)}; T, \mathbf{\Theta}) \tag{S7}$$

The posterior probability of a node can be computed using the inside and outside likelihoods, as follows:

$$\mathcal{S}^{(k)}_{\mathsf{node}}(u, \alpha_u; T, \mathbf{\Theta}) = \begin{cases} 1, \text{ if } u \text{ is the root and } \alpha_u = 0 \\ 0, \text{ if } u \text{ is the root and } \alpha_u \neq 0 \\ \frac{\mathcal{L}^{(k)}_{\mathsf{node}}(u, \alpha_u; T, \mathbf{\Theta}) \mathcal{L}^{(k)}_{\mathsf{out}}(u, \alpha_u; T, \mathbf{\Theta})}{\mathcal{P}(\mathbf{D}^{(k)}; T, \mathbf{\Theta})}, \text{ otherwise} \end{cases} \tag{S8}$$

Definition 5. *The posterior probability of an edge $e = (u, v)$ at site k w.r.t. the realization α_u of $\mathbb{X}^{(k)}(u)$ and α_v of $\mathbb{X}^{(k)}(v)$, denoted by $\mathcal{S}^{(k)}_{\mathsf{edge}}(u, v, \alpha_u, \alpha_v; T, \Theta)$, is the joint probability of $\mathbb{X}^{(k)}(u) = \alpha_u$ and $\mathbb{X}^{(k)}(v) = \alpha_v$ given $\mathbb{D}^{(k)} = \boldsymbol{D}^{(k)}$. In other words:*

$$\mathcal{S}^{(k)}_{\mathsf{edge}}(e, \alpha_u, \alpha_v; T, \boldsymbol{\Theta}) \coloneqq \mathcal{P}(\mathbb{X}^{(k)}(u) = \alpha_u, \mathbb{X}^{(k)}(v) = \alpha_v | \mathbb{D}^{(k)} = \boldsymbol{D}^{(k)}; T, \boldsymbol{\Theta}) \tag{S9}$$

The posterior probability of an edge $e = (u, v)$ can be computed from the other previously defined entities, as follows:

$$\mathcal{S}^{(k)}_{\mathsf{edge}}(e, \alpha_u, \alpha_v; T, \boldsymbol{\Theta}) = \frac{\mathcal{S}^{(k)}_{\mathsf{node}}(u, \alpha_u; T, \boldsymbol{\Theta}) \mathcal{L}^{(k)}_{\mathsf{node}}(v, \alpha_v; T, \boldsymbol{\Theta}) \boldsymbol{\Psi}^{(k)}_e(\alpha_u, \alpha_v)}{\mathcal{L}^{(k)}_{\mathsf{edge}}(e, \alpha_u; T, \boldsymbol{\Theta})} \tag{S10}$$

Thus, if all the inside likelihoods, outside likelihoods, and the full likelihood have been computed, we can use Eq. S10 and Eq. S8 to compute all the posterior assignments of states to nodes, in linear-time. With this observation, we can extend the Felsenstein's prunning algorithm to compute all the inside, outside likelihoods, and posterior assignments, as follows: step (1) compute and store the inside likelihoods at all nodes and branches using Eq. S2 and Eq. S4, in a bottom up tree traversal; step; step (2) compute and store the outside likelihoods at all nodes using Eq. S6, in a topdown tree traversal; step (3) compute the posterior assignments at all nodes using Eq. S10 and Eq. S8, in a tree traversal (either bottom-up or topdown). These values are needed for the E-step of our EM algorithm described later.

S1.4 The EM algorithm: derivation and pseudocode

S1.4.1 Derivation (Proof of Eq. 12 in the main text)

The basic setup of an EM algorithm to optimize $\mathcal{P}(\mathbb{D} = \mathbf{D}; \boldsymbol{\Theta})$ is as follows: in the E-step, compute for every site k the posterior probabilities $\mathcal{P}(\mathbb{X}^{(k)} = \mathbf{x} | \mathbb{D}^{(k)} = \mathbf{D}^{(k)}; \boldsymbol{\Theta}^t)$ for all possible realizations $\mathbf{x}$ of $\mathbb{X}^{(k)}$. In the M-step, compute $\boldsymbol{\Theta}^{t+1}$ as

$$\begin{aligned}
\boldsymbol{\Theta}^{t+1} &= \operatorname*{argmax}_{\boldsymbol{\Theta}} \sum_{k=1}^{K} \sum_{\mathbf{x}} \mathcal{P}(\mathbb{X}^{(k)} = \mathbf{x} | \mathbf{D}^{(k)}; \boldsymbol{\Theta}^t) \log \mathcal{P}(\mathbb{D}^{(k)} = \mathbf{D}^{(k)}, \mathbb{X}^{(k)} = \mathbf{x}; \boldsymbol{\Theta}) \\
&= \operatorname*{argmax}_{\boldsymbol{\Theta}} \sum_{k=1}^{K} \sum_{\mathbf{x}} \mathcal{P}(\mathbb{X}^{(k)} = \mathbf{x} | \mathbf{D}^{(k)}; \boldsymbol{\Theta}^t) \log \left[\prod_{e=(u,v) \in \mathscr{E}_T} \boldsymbol{\Psi}^{(k)}_e(\mathbf{x}(u), \mathbf{x}(v)) \prod_{w \in \mathscr{L}_{\mathscr{T}}} \boldsymbol{\Phi}(\mathbf{x}(w), \mathbf{D}^{(k)}(w)) \right] \\
&= \operatorname*{argmax}_{\boldsymbol{\Theta}} \sum_{k=1}^{K} \sum_{\mathbf{x}} \mathcal{P}(\mathbb{X}^{(k)} = \mathbf{x} | \mathbf{D}^{(k)}; \boldsymbol{\Theta}^t) \left[\sum_{e=(u,v) \in \mathscr{E}_T} \log \boldsymbol{\Psi}^{(k)}_e(\mathbf{x}(u), \mathbf{x}(v)) + \sum_{w \in \mathscr{L}_{\mathscr{T}}} \log \boldsymbol{\Phi}(\mathbf{x}(w), \mathbf{D}^{(k)}(w)) \right]
\end{aligned}$$

Note that every non-zero entries of $\boldsymbol{\Phi}$ can only be ϕ, $1 - \phi$, or 1. Therefore,

$$\begin{aligned}
&\sum_{k=1}^{K} \sum_{\mathbf{x}} \mathcal{P}(\mathbb{X}^{(k)} = \mathbf{x} | \mathbf{D}^{(k)}; \boldsymbol{\Theta}^t) \sum_{w \in \mathscr{L}_{\mathscr{T}}} \log \boldsymbol{\Phi}(\mathbf{x}(w), \mathbf{D}^{(k)}(w)) \\
&= \sum_{k=1}^{K} \sum_{\mathbf{x}} \sum_{w \in \mathscr{L}_{\mathscr{T}}} \mathcal{P}(\mathbb{X}^{(k)} = \mathbf{x} | \mathbf{D}^{(k)}; \boldsymbol{\Theta}^t) \log \boldsymbol{\Phi}(\mathbf{x}(w), \mathbf{D}^{(k)}(w)) \\
&= \sum_{w \in \mathscr{L}_T} \sum_{\mathbf{x}: \mathbf{x}(w) \neq -1} \mathcal{P}(\mathbb{X}^{(k)} = \mathbf{x} | \mathbf{D}^{(k)}; \boldsymbol{\Theta}^t) \sum_{k: \mathbf{D}^{(k)}(w) = ?} \log \phi + \sum_{w \in \mathscr{L}_T} \sum_{k: \mathbf{D}^{(k)}(w) \neq ?} \log(1 - \phi) \sum_{\mathbf{x}} \mathcal{P}(\mathbb{X}^{(k)} = \mathbf{x} | \mathbf{D}^{(k)}; \boldsymbol{\Theta}^t) \\
&= \sum_{w \in \mathscr{L}_T} \mathcal{P}(\mathbb{X}^{(k)}(w) \neq -1 | \mathbf{D}^{(k)}; \boldsymbol{\Theta}^t) \log \phi + \sum_{w \in \mathscr{L}_T} \sum_{k: \mathbf{D}^{(k)}(w) \neq ?} \log(1 - \phi) \\
&= \sum_{w \in \mathscr{L}_T} (1 - \mathcal{P}(\mathbb{X}^{(k)}(w) = -1 | \mathbf{D}^{(k)}; \boldsymbol{\Theta}^t)) \log \phi + \sum_{w \in \mathscr{L}_T} |\{k : \mathbf{D}^{(k)}(w) \neq ?\}| \log(1 - \phi) \\
&= \sum_{w \in \mathscr{L}_T} (\tilde{\mathcal{B}}_w \log \phi + \mathcal{B}_w \log(1 - \phi))
\end{aligned}$$

The derivation of the other term is similar: note that all Ψ_e share the same locations of non-zero entries, and $\log \mathbf{\Psi}_e^{(k)}(\mathbf{x}(u), \mathbf{x}(v))$ can only be $-\delta_e(1+\nu)$, $\log 1 - e^{-\delta_e}$, $-\nu\delta_e$, $-\delta\nu(1-e^{-\delta_e})$, or 0 (plus constants). Therefore, we have (skipping details, as the derivation is very similar to [41]):

$$\sum_{k=1}^{K} \sum_{\mathbf{x}} \mathcal{P}(\mathbb{X}^{(k)} = \mathbf{x} | \mathbf{D}^{(k)}; \mathbf{\Theta}^t) \sum_{e=(u,v)\in\mathscr{E}_T} \log \mathbf{\Psi}_e^{(k)}(\mathbf{x}(u), \mathbf{x}(v))$$
$$= \sum_{e\in\mathscr{E}_T} \left(-\mathcal{C}_e^{z\to z}(1+\nu)\delta_e + \mathcal{C}_e^{z\to a}(\log(1-e^{-\delta_e}) - \nu\delta_e) + \mathcal{C}_e^{z\to m}\log\left(1-e^{-\delta_e\nu}\right) - \mathcal{C}_e^{a\to a}\delta_e\nu + \mathcal{C}_e^{a\to m}\log\left(1-e^{-\delta_e\nu}\right)\right)$$

Adding the two terms, we get Eq. 12 in the main text.

S1.5 Reducing the complexity of the E-step

In the E-step, we need to compute $\mathcal{C}_e^{z\to z}, \mathcal{C}_e^{z\to a}, \mathcal{C}_e^{z\to m}, \mathcal{C}_e^{a\to a}$, and $\mathcal{C}_e^{a\to m}$ for all branches e. Below we introduce an algorithm to compute them in O(NK), where N and K are the number of cells and target sites, respectively. Importantly, the complexity of our algorithm is independent of the alphabet size of any target site. First, we introduce the concepts of **α-trees, α-clades**, and **z-branches** that will be useful in describing the algorithm.

Definition 6. *Consider a target site k of the lineage tracing data **D** whose alphabet is $\mathscr{A}^{(k)}$ and a character state $\alpha \in \mathscr{A}^{(k)} \setminus \{0, -1\}$. An **$\alpha$-tree** on site k at state α of **D** is a tree T with leafset $\mathscr{L}_T$ such that for all $w \in \mathscr{L}_T$, $\boldsymbol{D}^{(k)}(w) \in \{?, \alpha\}$.*

Definition 7. *An **α-clade** of a tree T with respect to a site k of a lineage tracing data **D** is a clade of T that forms an α-tree on site k at state α of **D**.*

Note that the definitions of α-tree and α-clade include the case where $\alpha =?$ (i.e. all leaves of T have state "?"). In such a case, we call the tree/clade a *masked tree/clade*.

Definition 8. *A **masked-clade** of a tree T with respect to a site k of lineage tracing data **D** is an α-clade of T at k where $\alpha =?$. Any branch belongs to a masked-clade is called a **masked branch**. Any branch that is not a masked branch is referred to as an **unmasked** branch.*

Note that an α-tree where $\alpha \neq ?$ may include some masked clades and masked branches inside it.

Definition 9. *A **z-branch** of a tree T with respect to a site k of a lineage tracing data **D** is a branch that does not belong to any α-clade of T on k, for all $\alpha \in \mathscr{A}^{(k)} \setminus \{0, -1\}$.*

S1.5.1 Special properties of α-clades and z-branches

Theorem 1. *[az-partition] Consider a target site k of the lineage tracing data **D** whose alphabet is $\mathscr{A}^{(k)}$. Any tree T on $\boldsymbol{D}^{(k)}$ can be partitioned into **edge-disjoint** α-clades (each can have a different $\alpha \in \mathscr{A}^{(k)} \setminus \{0, -1\}$) and z-branches. This partition is unique and is named the **az-partition** of T with respect to $\boldsymbol{D}^{(k)}$.*

Proof. The following is an algorithm to partition any tree T into α-clades and z-branches.

Algorithm 3 Linear-time algorithm to perform az-partition. Input: a tree topology T and a target-site $\mathbf{D}^{(k)}$. Output: the az-partition of T with respect to $\mathbf{D}^{(k)}$.

```
1:  function AZPARTITION(T, D^(k))
2:      alphaTrees, zBranches ← [], []
3:      for u in postorder traversal of T do
4:          if u is a leaf then                                  ▷ Set tag by the state of the leaf node.
5:              if D^(k)(u) == 0 then
6:                  u.tag ← "z"
7:                  Add the branch incident to u to zBranches
8:              else
9:                  u.tag ← D^(k)(u)
10:         else                                                 ▷ Set tag by the tags of the child nodes
11:             a, b ← children(u)
12:             if (a.tag == b.tag and a ≠ "z") or (one is "?") then
13:                 u.tag ← a.tag if a.tag != "?" else b.tag
14:             else
15:                 u.tag ←"z"                                   ▷ the branch above u is a z-branch.
16:             if u.tag =="z" then
17:                 Add the branch incident to u to zBranches
18:                 if a.tag is not "z" then                     ▷ u has tag "z" and a does not
19:                     Add the clade attached to branch (u, a) to alphaTrees
20:                 if b.tag is not "z" then                     ▷ u has tag "z" and b does not
21:                     Add the clade attached to branch (u, b) to alphaTrees
22:     return alphaTrees, zBranches
```

We will analyze this algorithm to demonstrate that it produces a unique az-partition of T with respect to $D^{(i)}$.

The algorithm iterates over every node u in the tree T, and considers the incident branch, so that the algorithm considers every branch e. A branch e is either tagged and output as a zBranch (lines 6-7, 15-17), or tagged with the leaf state (lines 12-13). If the branch is tagged with a leaf state α, then it must necessarily be output as part of the corresponding α-tree at a later iteration. Since a zBranch marks the first branch where not all leaves share the same state, the branches above a zBranch cannot lie in an alphaTree, so alphaTrees must be pairwise disjoint. Therefore, each branch must be output either as part of an alphaTree or as a zBranch, and will only appear once in the output. Thus, we have that the algorithm produces a valid partition.

We should next show that alphaTrees are indeed α-clades of T for some α. For any alphaTree T_α returned by the algorithm, we know that by construction all branches in that alphaTree are tagged with either α or "?", except for the root branch, which is tagged with a z. For a branch to be tagged with a leaf state α, the incident node u's children have tags in the set $\{\alpha, ?\}$, recursively down to the leaves (line 11-13) such that all the leaf nodes under this branch must have state in the set $\{\alpha, ?\}$. Lines 19 and 21, where the entire set of branches below a z-branch is returned, determine that the alphaTrees returned must be a clade. Therefore, by construction we know that the returned trees must be clades that each form an alphaTree with corresponding leafset $\in \{\alpha, ?\}$, as stipulated by Def. 6 and 7.

We should next show that returned zBranches fit Def. 9. zBranches are returned in a few cases: on line 5-6, a zBranch could be returned if the leaf state is 0, which is consistent with Def 9. The other case when a zBranch is returned happens in lines 16-17, where if a node u is tagged with a z, then the incident branch b will be returned as a zBranch. A few scenarios force a node u to be tagged with z: (1) its two children do not share the same tag, (2) two children are both tagged with z. In Case (1), the leaves under branch b do not share states $\in \{\alpha, ?\}$, and so this branch cannot belong to any α-clade and must necessarily be a zBranch. In Case (2), if both children are tagged with z, then the leaves under each child node respectively do not share states $\in \{\alpha, ?\} \forall \alpha \in \Sigma_i \setminus \{0, -1\}$, so that this branch cannot belong to any α-clade and must also necessarily be a zBranch. Therefore, all elements of the returned zBranches must be a zBranch, and since all branches are represented at least once, and all alphaTree branches are accounted for, the algorithm must have returned all zBranches.

Putting the above statements together, we want to show that if the algorithm always returns an answer, then the partition must exist. Suppose towards a contradiction the algorithm returns a set of alphaTrees and zBranches, but

that no partition exists. However, we saw above that the algorithm necessarily outputs all branches, and each branch appears once as either a zBranch, or as a branch in an alphaTree. Therefore, we must have a valid partition, but we have assumed that no partition exists, and we arrive at a contradiction. We can see that the algorithm must always return an answer given a valid input, since the algorithm output is an assignment for each branch. Consequently, it must be true that if the algorithm returns an answer, then the partition must exist.

Finally, it remains to be shown that the partition is unique. Suppose towards a contradiction that we have two partitions P_1 and P_2 which are both valid partitions for an input tree T and data $\mathbf{D}^{(k)}$, but which are different without loss of generality by one branch b. Since all branches must show up in both valid partitions, this different branch b could be different in tow ways: Case (1): $\in$ the set of zBranches in P_1 and $\in$ an alphaTree in P_2, or Case (2): $\in$ topology alphaTree T_α in P_1 and $\notin$ topology alphaTree T_β in P_2.

In Case (1), suppose towards a contradiction that it is true a branch b could be a zBranch in one valid partition, and in an alphaTree in another valid partition. Then, in valid partition P_1, the leaf states below b must not lie in the set $\{\alpha, ?\}$, and in valid partition P_2, the leaf states do lie in the set $\{\alpha ?\}$. However, both cannot be true, since we have the same tree topology T and data $\mathbf{D}^{(k)}$. Therefore, we have arrived at a contradiction, and we know that any branch must either lie in a zBranch or in an alphaTree exclusively, in all valid partitions.

In the other possible Case (2), suppose towards a contradiction that b has been assigned to two different alphatrees; that is, b is assigned to $T_\alpha \in P_1$ and not assigned to $T_\beta \in P_2$, where T_α and T_β are topologically the same except for branch b. Then, all leaves under branch $b \in P_1$ must necessarily lie in the allowed T_α leaf states. In partition P_2, for branch b to belong to a different alphaTree, it must be tagged with a different α. However, the input data is the same, and so all the leaves below branch b must have the same leaf state in both partitions. We have arrived at a contradiction, and thus we have that the partition returned by the algorithm must be unique.

Since missing clades are special cases of alphaTrees, we will not discuss them specially here, except to note that all branches in an alphaTree (masked branches included) will only be output as part of an alphaTree when the ancestral zBranch is processed. We should note that the branch incident to the case where one child has the missing state, and the other child has a nonmissing, nonzero state is not a zBranch.

Thus, we can see that the algorithm must necessarily provide a valid partition of our input tree and data into alphaTrees and a set of zBranches fitting the proposed definitions. □

Corollary 1. *Any branch $e = (u, v)$ of T is either a z-branch, a masked-branch, or an unmasked branch of some α_0-clade of T where $\alpha_0 \in \mathscr{A}^{(k)} \setminus \{?, 0, -1\}$, with respect to $\boldsymbol{D}^{(k)}$. In addition, if e is not a z-branch or masked-branch, it has a unique associated α_0.*

Theorem 2. *[Properties of a z-branch] Given a tree topology T parameterized by Θ, a target site k, and a z-branch $e = (u, v)$ of T on site k. We have the following statements:*

$$(1)\mathcal{L}^{(k)}_{\mathsf{node}}(v, \alpha; T, \boldsymbol{\Theta}) = \mathcal{L}^{(k)}_{\mathsf{edge}}(e, \alpha; T, \boldsymbol{\Theta}) = 0,\ \forall \alpha \neq 0$$
$$(2)\mathcal{S}^{(k)}_{\mathsf{edge}}(e, \alpha, \beta; T, \boldsymbol{\Theta}) = 0,\ \forall (\alpha, \beta) \neq (0, 0)$$
$$(3)\mathcal{S}^{(k)}_{\mathsf{edge}}(e, 0, 0; T, \boldsymbol{\Theta}) = 1$$
$$(4)\mathcal{S}^{(k)}_{\mathsf{node}}(u, 0; T, \boldsymbol{\Theta}) = 1$$
$$(5)\mathcal{S}^{(k)}_{\mathsf{node}}(u, \alpha; T, \boldsymbol{\Theta}) = 0,\ \forall \alpha \neq 0$$

Proof. **Proof of Statement (1)**. To prove (1), first we prove that if e is a z-branch of site k, then leaves in $\mathbf{D}^{(k)}_v$ belong to two distinct unmasked states. Suppose towards a contradiction that all leaves in $\mathbf{D}^{(k)}_v$ are either in masked state "?" or a same state α_0; by definition of α-clade, $\mathbf{D}^{(k)}_v$ defines an α-clade on T where $\alpha = \alpha_0$ or $\alpha = ?$. By definition, e belongs to the clade defined by $\mathbf{D}^{(k)}_v$, so it is part of an α-clade, contradicting the assumption that e is a z-branch.

Hence, there exists two leaves l_1 and l_2 under v such that $\mathbf{D}^{(k)}(l_1) = \beta_1 \neq \mathbf{D}^{(k)}(l_2) = \beta_2$, where $\beta_1, \beta_2 \in \mathscr{A}^{(k)} \setminus \{?\}$. Let w be the LCA of l_1 and l_2, so $\mathbb{D}^{(k)}(l_1)$ and $\mathbb{D}^{(k)}(l_2)$ are conditionally independent given $\mathbb{X}^{(k)}(w)$).

Recall that $\mathbf{D}_v^{(k)}$ includes $\mathbf{D}^{(k)}(l_1)$ and $\mathbf{D}^{(k)}(l_2)$. For any $\alpha \neq 0$, we have:

$$\begin{aligned}
0 &\leq \mathcal{L}_{\mathsf{node}}^{(k)}(v, \alpha; T, \mathbf{\Theta}) = \mathcal{P}(\mathbb{D}_v^{(k)} = \mathbf{D}_v^{(k)} | \mathbb{X}^{(k)}(v) = \alpha) \\
&\leq \mathcal{P}(\mathbb{D}^{(k)}(l_1) = \mathbf{D}^{(k)}(l_1), \mathbb{D}^{(k)}(l_2) = \mathbf{D}^{(k)}(l_2) | \mathbb{X}^{(k)}(v) = \alpha) \\
&= \mathcal{P}(\mathbb{D}^{(k)}(l_1) = \beta_1, \mathbb{D}^{(k)}(l_2) = \beta_2 | \mathbb{X}^{(k)}(v) = \alpha) \\
&= \sum_{\beta \in \mathscr{A}^{(k)}} \mathcal{P}(\mathbb{D}^{(k)}(l_1) = \beta_1, \mathbb{D}^{(k)}(l_2) = \beta_2 | \mathbb{X}^{(k)}(w) = \beta) \mathcal{P}(\mathbb{X}^{(k)}(w) = \beta | \mathbb{X}^{(k)}(v) = \alpha) \\
&= \sum_{\beta \in \mathscr{A}^{(k)}} \mathcal{P}(\mathbb{D}^{(k)}(l_1) = \beta_1 | \mathbb{X}^{(k)}(w) = \beta) \mathcal{P}(\mathbb{D}^{(k)}(l_2) = \beta_2 | \mathbb{X}^{(k)}(w) = \beta) \mathcal{P}(\mathbb{X}^{(k)}(w) = \beta | \mathbb{X}^{(k)}(v) = \alpha) \\
&= 0,
\end{aligned}$$

where the last equation holds because all terms inside the last summation are 0: if $\beta = 0$ or $\beta \notin \{\alpha, -1\}$, then $\mathcal{P}(\mathbb{X}^{(k)}(w) = \beta | \mathbb{X}^{(k)}(v) = \alpha) = 0$; else if $\beta = -1$ then $\mathcal{P}(\mathbf{D}^{(k)}(l_1) = \beta_1 | \mathbb{X}^{(k)}(w) = \beta) = \mathcal{P}(\mathbf{D}^{(k)}(l_2) = \beta_2 | \mathbb{X}^{(k)}(w) = \beta) = 0$ because $\beta_1 \neq ?$ and $\beta_2 \neq ?$; otherwise, $\beta = \alpha$, so either $\mathcal{P}(\mathbf{D}^{(k)}(l_1) = \beta_1 | \mathbb{X}^{(k)}(w) = \beta) = 0$ or $\mathcal{P}(\mathbf{D}^{(k)}(l_2) = \beta_2 | \mathbb{X}^{(k)}(w) = \beta) = 0$ because $\beta_1 \neq \beta_2$ and $\alpha \neq 0$. Thus, $\mathcal{P}(\mathbf{D}_v^{(k)} | \mathbb{X}^{(k)}(v) = \alpha) = 0$ for all $\alpha \neq 0$. Now we prove the second equality of (1). For any $\alpha \neq 0$, we have:

$$\mathcal{P}(\mathbf{D}_v^{(k)} | \mathbb{X}^{(k)}(u) = \alpha; T, \Theta) = \sum_{\beta \in \mathscr{A}^{(k)}} \mathcal{P}(\mathbf{D}_v^{(k)} | \mathbb{X}^{(k)}(v) = \beta; T, \Theta) \mathcal{P}(\mathbb{X}^{(k)}(v) = \beta | \mathbb{X}^{(k)}(u) = \alpha; T, \Theta) = 0,$$

where the second equality holds because $\mathcal{P}(\mathbb{X}^{(k)}(v) = 0 | \mathbb{X}^{(k)}(u) = \alpha; T, \Theta) = 0$ (recall that $\alpha \neq 0$) and $\mathcal{P}(\mathbf{D}_v^{(k)} | \mathbb{X}^{(k)}(v) = \beta; T, \Theta) = 0$ for all $\beta \neq 0$.

Proof of Statement (2). Using Eq. S10, we have:

$$\mathcal{S}_{\mathsf{edge}}^{(k)}(e, \alpha, \beta; T, \mathbf{\Theta}) = \frac{\mathcal{S}_{\mathsf{node}}^{(k)}(u, \alpha; T, \mathbf{\Theta}) \mathcal{L}_{\mathsf{node}}^{(k)}(v, \beta; T, \mathbf{\Theta}) \mathbf{\Psi}_e^{(k)}(\alpha, \beta)}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e, \alpha; T, \mathbf{\Theta})}$$

Recall that $(\alpha, \beta) \neq (0, 0)$, so either $\beta \neq 0$ or ($\beta = 0$ and $\alpha \neq 0$).

- Case 1: $\beta \neq 0$. In this case, $\mathcal{L}_{\mathsf{node}}^{(k)}(v, \beta; T, \mathbf{\Theta}) = 0$ (Statement (1)), so $\mathcal{S}_{\mathsf{edge}}^{(k)}(e, \alpha, \beta; T, \Theta) = 0$.
- Case 2: $\beta = 0$ and $\alpha \neq 0$. In this case, $\mathbf{\Psi}_e^{(k)}(\alpha, \beta) = 0$ (see Eq. 2 in the main text), so $\mathcal{S}_{\mathsf{edge}}^{(k)}(e, \alpha, \beta; T, \Theta) = 0$.

Proof of Statement (3). It is straightforward to derive Statement (3) from (2):

$$\begin{aligned}
\mathcal{S}_{\mathsf{edge}}^{(k)}(e, 0, 0; T, \mathbf{\Theta}) &= \mathcal{P}(\mathbb{X}^{(k)}(u) = 0, \mathbb{X}^{(k)}(v) = 0 | \mathbf{D}^{(k)}; T, \Theta) \\
&= 1 - \sum_{(\alpha,\beta) \neq (0,0)} \mathcal{P}(\mathbb{X}^{(k)}(u) = \alpha, \mathbb{X}^{(k)}(v) = \beta | \mathbf{D}^{(k)}; T, \Theta) \\
&= 1 - \sum_{(\alpha,\beta) \neq (0,0)} \mathcal{S}_{\mathsf{edge}}^{(k)}(e, \alpha, \beta; T, \mathbf{\Theta}) \\
&= 1
\end{aligned}$$

Proof of Statement (4). We prove Statement (4) using Statements (2) and (3):

$$\begin{aligned}
\mathcal{S}_{\mathsf{node}}^{(k)}(u, 0; T, \mathbf{\Theta}) &= \mathcal{P}(\mathbb{X}^{(k)}(u) = 0 | \mathbf{D}^{(k)}; T, \Theta) \\
&= \textstyle\sum_\beta \mathcal{P}(\mathbb{X}^{(k)}(u) = 0, \mathbb{X}^{(k)}(v) = \beta | \mathbf{D}^{(k)}; T, \Theta) \\
&= \textstyle\sum_{\beta \neq 0} \mathcal{P}(\mathbb{X}^{(k)}(u) = 0, \mathbb{X}^{(k)}(v) = \beta | \mathbf{D}^{(k)}; T, \Theta) + \mathcal{P}(\mathbb{X}^{(k)}(u) = 0, \mathbb{X}^{(k)}(v) = 0 | \mathbf{D}^{(k)}; T, \Theta) \\
&= \textstyle\sum_{\beta \neq 0} \mathcal{S}_{\mathsf{edge}}^{(k)}(e, 0, \beta; T, \mathbf{\Theta}) + \mathcal{S}_{\mathsf{edge}}^{(k)}(e, 0, 0; T, \mathbf{\Theta}) = 1
\end{aligned}$$

The last equality is justified by Statements (2) and (3).

Proof of Statement (5). For all $\alpha \neq 0$, we have:

$$\begin{aligned}
0 \leq \mathcal{S}^{(k)}_{\mathsf{node}}(u, \alpha; T, \mathbf{\Theta}) &\leq \sum_{\beta \neq 0} \mathcal{S}^{(k)}_{\mathsf{node}}(u, \beta; T, \mathbf{\Theta}) \\
&= \sum_{\beta \neq 0} \mathcal{P}(\mathbb{X}^{(k)}(u) = \beta | \mathbf{D}^{(k)}; T, \Theta) \\
&= 1 - \mathcal{P}(\mathbb{X}^{(k)}(u) = 0 | \mathbf{D}^{(k)}; T, \Theta) \\
&= 1 - \mathcal{S}^{(k)}_{\mathsf{node}}(u, 0; T, \mathbf{\Theta}) = 0
\end{aligned}$$

The last equality is justified by Statement (4). Thus, $\mathcal{S}^{(k)}_{\mathsf{node}}(u, \alpha; T, \mathbf{\Theta}) = 0$ for all $\alpha \neq 0$. □

Theorem 3. *[Properties of a masked-branch] Consider a masked clade of a tree **T** at a site k of **D** and $e = (u, v)$ is an arbitrary branch of this masked clade. Then we have*

$$\begin{aligned}
&(1)\ \mathcal{L}^{(k)}_{\mathsf{node}}(v, -1; T, \mathbf{\Theta}) = \mathcal{L}^{(k)}_{\mathsf{edge}}(e, -1; T, \mathbf{\Theta}) = 1 \\
&(2)\ \mathcal{L}^{(k)}_{\mathsf{edge}}(e, \alpha; T, \mathbf{\Theta}) = \mathcal{L}^{(k)}_{\mathsf{edge}}(e, \beta; T, \mathbf{\Theta}), \forall \alpha, \beta \in \mathscr{A}^{(k)} \setminus \{0, -1, ?\}, \alpha \neq \beta \\
&(3)\ \mathcal{L}^{(k)}_{\mathsf{node}}(v, \alpha; T, \mathbf{\Theta}) = \mathcal{L}^{(k)}_{\mathsf{node}}(v, \beta; T, \mathbf{\Theta}), \forall \alpha, \beta \in \mathscr{A}^{(k)} \setminus \{0, -1, ?\}, \alpha \neq \beta
\end{aligned}$$

Proof. **Proof of (1)**. Let Clade(v) denote the leaf set of the clade of T below v. We have:

$$\begin{aligned}
&1 \geq \mathcal{P}(\mathbf{D}^{(k)}_v | \mathbb{X}^{(k)}(v) = -1; T, \Theta) \geq \prod_{l \in \mathrm{Clade}(v)} \mathcal{P}(\mathbf{D}^{(k)}(l) | \mathbb{X}^{(k)}(v) = -1; T, \Theta) = 1 \\
&\implies \mathcal{P}(\mathbf{D}^{(k)}_v | \mathbb{X}^{(k)}(v) = -1; T, \Theta) = 1 \\
&\implies \mathcal{L}^{(k)}_{\mathsf{node}}(v, -1; T, \mathbf{\Theta}) = 1
\end{aligned}$$

Now we prove the second equality of Statement (1). Indeed:

$$\begin{aligned}
\mathcal{L}^{(k)}_{\mathsf{edge}}(e, -1; T, \mathbf{\Theta}) &= \mathcal{P}(\mathbf{D}^{(k)}_v | \mathbb{X}^{(k)}(u) = -1; T, \Theta) \\
&= \textstyle\sum_{\alpha \in \mathscr{A}^{(k)}} \mathcal{P}(\mathbf{D}^{(k)}_v | \mathbb{X}^{(k)}(v) = \alpha; T, \mathbf{\Theta}) \mathcal{P}(\mathbb{X}^{(k)}(v) = \alpha | \mathbb{X}^{(k)}(u) = -1; T, \Theta) \\
&= \textstyle\sum_{\alpha \in \mathscr{A}^{(k)}} \mathcal{L}^{(k)}_{\mathsf{node}}(v, \alpha; T, \mathbf{\Theta}) \mathbf{\Psi}_e(-1, \alpha) \\
&= \textstyle\sum_{\alpha \neq 0} \mathcal{L}^{(k)}_{\mathsf{node}}(v, \alpha; T, \mathbf{\Theta}) \mathbf{\Psi}_e(-1, \alpha) + \mathcal{L}^{(k)}_{\mathsf{node}}(v, -1; T, \mathbf{\Theta}) \mathbf{\Psi}_e(-1, -1) \\
&= \mathcal{L}^{(k)}_{\mathsf{node}}(v, -1; T, \mathbf{\Theta}) \mathbf{\Psi}_e(-1, -1) = 1
\end{aligned}$$

Proof of (2). We will prove by induction. Base case: e is a terminal branch (i.e. v is a leaf node), so that $\mathbf{D}^{(k)}_v = \mathbf{D}^{(k)}(v)$. In addition, because e is masked, $\mathbf{D}^{(k)}(v) = ?$. So, we have:

$$\begin{aligned}
\mathcal{L}^{(k)}_{\mathsf{edge}}(e, \alpha; T, \mathbf{\Theta}) &= \mathcal{P}(\mathbb{D}^{(k)}_v = \mathbf{D}^{(k)}_v | \mathbb{X}^{(k)}(u) = \alpha) \\
&= \mathcal{P}(\mathbb{D}^{(k)}(v) = ? | \mathbb{X}^{(k)}(u) = \alpha) \\
&= \textstyle\sum_{\gamma \in \mathscr{A}^{(k)}} \mathcal{P}(\mathbb{D}^{(k)}(v) = ? | \mathbb{X}^{(k)}(v) = \gamma) \mathcal{P}(\mathbb{X}^{(k)}(v) = \gamma | \mathbb{X}^{(k)}(u) = \alpha) \\
&= \textstyle\sum_{\gamma \in \mathscr{A}^{(k)}} \mathbf{\Phi}(\gamma, ?) \mathbf{\Psi}^{(k)}(\alpha, \gamma) \\
&= \phi p^{\nu}_e + (1 - p^{\nu}_e)
\end{aligned}$$

Similarly, we can prove that $\mathcal{L}^{(k)}_{\mathsf{edge}}(e, \beta; T, \mathbf{\Theta}) = \phi p^{\nu}_e + (1 - p^{\nu}_e)$, so $\mathcal{L}^{(k)}_{\mathsf{edge}}(e, \alpha; T, \mathbf{\Theta}) = \mathcal{L}^{(k)}_{\mathsf{edge}}(e, \beta; T, \mathbf{\Theta})$.

Now we prove the induction case. If e is not a terminal branch, then v has two children v_1, v_2. Suppose that the statement is correct for the two branches $e_1 = (v, v_1)$ and $e_2 = (v, v_2)$, we will prove that it is also correct for branch $e = (u, v)$. For any state $\alpha \in \mathscr{A}^{(k)} \setminus \{0, -1, ?\}$, we have:

$$\begin{aligned}
\mathcal{L}^{(k)}_{\mathsf{edge}}(e, \alpha; T, \mathbf{\Theta}) &= \mathcal{P}(\mathbf{D}^{(k)}_v | \mathbb{X}^{(k)}(u) = \alpha) \\
&= \mathcal{P}(\mathbf{D}^{(k)}_{v_1}, \mathbf{D}^{(k)}_{v_2} | \mathbb{X}^{(k)}(u) = \alpha) \\
&= \textstyle\sum_{\gamma \in \mathscr{A}^{(k)}} \mathcal{P}(\mathbf{D}^{(k)}_{v_1}, \mathbf{D}^{(k)}_{v_2}, \mathbb{X}^{(k)}(v) = \gamma | \mathbb{X}^{(k)}(u) = \alpha) \\
&= \textstyle\sum_{\gamma \in \mathscr{A}^{(k)}} \mathcal{P}(\mathbf{D}^{(k)}_{v_1} | \mathbb{X}^{(k)}(v) = \gamma) \mathcal{P}(\mathbf{D}^{(k)}_{v_2} | \mathbb{X}^{(k)}(v) = \gamma) \mathcal{P}(\mathbb{X}^{(k)}(v) = \gamma | \mathbb{X}^{(k)}(u) = \alpha) \\
&= \textstyle\sum_{\gamma \in \mathscr{A}^{(k)}} \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \gamma; T, \mathbf{\Theta}) \mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, \gamma; T, \mathbf{\Theta}) \mathbf{\Psi}_e(\alpha, \gamma)
\end{aligned}$$

Recall that $\mathbf{\Psi}_e$ is a sparse matrix. Because $\alpha \neq 0$, $\mathbf{\Psi}_e(\alpha, \gamma) = 0$ for all $\gamma \neq \{-1, \alpha\}$. Therefore, we have:

$$\begin{aligned}\mathcal{L}^{(k)}_{\mathsf{edge}}(e, \alpha; T, \mathbf{\Theta}) &= \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, -1; T, \mathbf{\Theta})\mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, -1; T, \mathbf{\Theta})\mathbf{\Psi}_e(\alpha, -1) + \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \alpha; T, \mathbf{\Theta})\mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, \alpha; T, \mathbf{\Theta})\mathbf{\Psi}_e(\alpha, \alpha) \\ &= \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, -1; T, \mathbf{\Theta})\mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, -1; T, \mathbf{\Theta})(1 - p^{\nu}_e) + \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \alpha; T, \mathbf{\Theta})\mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, \alpha; T, \mathbf{\Theta})p^{\nu}_e\end{aligned}$$

By induction assumptions, for any state $\beta \in \mathscr{A}^{(k)} \setminus \{0, -1, ?\}$ that is distinct from α, we have $\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \alpha; T, \mathbf{\Theta}) = \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \beta; T, \mathbf{\Theta})$ and $\mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, \alpha; T, \mathbf{\Theta}) = \mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, \beta; T, \mathbf{\Theta})$. Thus:

$$\begin{aligned}\mathcal{L}^{(k)}_{\mathsf{edge}}(e, \alpha; T, \mathbf{\Theta}) &= \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, -1; T, \mathbf{\Theta})\mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, -1; T, \mathbf{\Theta})(1 - p^{\nu}_e) + \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \beta; T, \mathbf{\Theta})\mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, \beta; T, \mathbf{\Theta})p^{\nu}_e \\ &= \mathcal{L}^{(k)}_{\mathsf{edge}}(e, \beta; T, \mathbf{\Theta})\end{aligned}$$

Proof of (3). Similar to (2), we can use induction to prove this statement. □

Theorem 4. *[Properties of unmasked branch of an α-tree] Consider an α-clade of a tree T at a site k of $\mathbf{D}$ where $\alpha \neq ?$ and $e = (u, v)$ is an arbitrary unmasked branch of this clade. Then:*

$$\begin{aligned}&(1)\ \mathcal{L}^{(k)}_{\mathsf{node}}(v, \beta; T, \mathbf{\Theta}) = 0,\ \forall \beta \in \mathscr{A}^{(k)} \setminus \{0, \alpha\} \\ &(2)\ \mathcal{L}^{(k)}_{\mathsf{edge}}(e, \beta; T, \mathbf{\Theta}) = 0,\ \forall \beta \in \mathscr{A}^{(k)} \setminus \{0, \alpha\} \\ &(3)\ \mathcal{S}^{(k)}_{\mathsf{edge}}(\beta, \gamma; T, \mathbf{\Theta}) = 0,\ \forall (\beta, \gamma) \in (\mathscr{A}^{(k)})^2 \setminus \{(0, 0), (0, \alpha), (\alpha, \alpha)\}\end{aligned}$$

Proof. **Proof of (1)** Because $e = (u, v)$ is an unmasked branch of an α-clade of T on $\mathbf{D}^{(k)}$, by definition there exists at least one leaf node l of T under v that has $\mathbf{D}^{(k)}(l) = \alpha$ and $\mathbf{D}^{(k)}(l) \in \mathbf{D}^{(k)}_v$. Let w be the parent of l, and $e_1 = (w, l)$. It is easy to see that $\mathcal{P}(\mathbb{X}^{(k)}(l) = \gamma | \mathbb{X}^{(k)}(v) = \beta; T, \mathbf{\Theta}) \leq \mathcal{P}(\mathbb{X}^{(k)}(l) = \gamma | \mathbb{X}^{(k)}(w) = \beta; T, \mathbf{\Theta}) = \mathbf{\Psi}_{e_1}(\beta, \gamma)$ for all $\gamma \in \mathscr{A}^{(k)}$. So we have:

$$\begin{aligned}0 &\leq \mathcal{L}^{(k)}_{\mathsf{node}}(v, \beta; T, \mathbf{\Theta}) \\ &= \mathcal{P}(\mathbb{D}^{(k)}_v = \mathbf{D}^{(k)}_v | \mathbb{X}^{(k)}(v) = \beta; T, \mathbf{\Theta}) \\ &\leq \mathcal{P}(\mathbb{D}^{(k)}(l) = \mathbf{D}^{(k)}(l) | \mathbb{X}^{(k)}(v) = \beta; T, \mathbf{\Theta}) \\ &= \mathcal{P}(\mathbb{D}^{(k)}(l) = \alpha | \mathbb{X}^{(k)}(v) = \beta; T, \mathbf{\Theta}) \\ &= \sum_{\gamma} \mathcal{P}(\mathbb{D}^{(k)}(l) = \alpha, \mathbb{X}^{(k)}(l) = \gamma | \mathbb{X}^{(k)}(v) = \beta; T, \mathbf{\Theta}) \\ &= \sum_{\gamma} \mathcal{P}(\mathbb{D}^{(k)}(l) = \alpha | \mathbb{X}^{(k)}(l) = \gamma; T, \mathbf{\Theta})\mathcal{P}(\mathbb{X}^{(k)}(l) = \gamma | \mathbb{X}^{(k)}(v) = \beta; T, \mathbf{\Theta}) \\ &\leq \sum_{\gamma} \mathbf{\Phi}(\gamma, \alpha)\mathbf{\Psi}_{e_1}(\beta, \gamma) = 0,\end{aligned}$$

where the last equality holds because of the sparsity of $\mathbf{\Phi}$ and $\mathbf{\Psi}_{e_1}$ (note that $\beta \neq 0$ and $\beta \neq \alpha$).

Proof of (2) Similar to the proof for (1), observe that $\mathcal{P}(\mathbb{X}^{(k)}(l) = \gamma | \mathbb{X}^{(k)}(u) = \beta; T, \mathbf{\Theta}) \leq \mathcal{P}(\mathbb{X}^{(k)}(l) = \gamma | \mathbb{X}^{(k)}(w) = \beta; T, \mathbf{\Theta}) = \mathbf{\Psi}_{e_1}(\beta, \gamma)$ for all $\gamma \in \mathscr{A}^{(k)}$. So we have:

$$\begin{aligned}0 &\leq \mathcal{L}^{(k)}_{\mathsf{edge}}(e, \beta; T, \mathbf{\Theta}) \\ &= \mathcal{P}(\mathbb{D}^{(k)}_v = \mathbf{D}^{(k)}_v | \mathbb{X}^{(k)}(u) = \beta; T, \mathbf{\Theta}) \\ &\leq \mathcal{P}(\mathbb{D}^{(k)}(l) = \mathbf{D}^{(k)}(l) | \mathbb{X}^{(k)}(u) = \beta; T, \mathbf{\Theta}) \\ &= \mathcal{P}(\mathbb{D}^{(k)}(l) = \alpha | \mathbb{X}^{(k)}(u) = \beta; T, \mathbf{\Theta}) \\ &= \sum_{\gamma} \mathcal{P}(\mathbb{D}^{(k)}(l) = \alpha, \mathbb{X}^{(k)}(l) = \gamma | \mathbb{X}^{(k)}(u) = \beta; T, \mathbf{\Theta}) \\ &= \sum_{\gamma} \mathcal{P}(\mathbb{D}^{(k)}(l) = \alpha | \mathbb{X}^{(k)}(l) = \gamma; T, \mathbf{\Theta})\mathcal{P}(\mathbb{X}^{(k)}(l) = \gamma | \mathbb{X}^{(k)}(u) = \beta; T, \mathbf{\Theta}) \\ &\leq \sum_{\gamma} \mathbf{\Phi}(\gamma, \alpha)\mathbf{\Psi}_{e_1}(\beta, \gamma) = 0\end{aligned}$$

Proof of (3) From Eq. S10, for all $(\beta, \gamma) \in (\mathscr{A}^{(k)})^2 \setminus \{(0,0), (0,\alpha), (\alpha,\alpha)\}$, we have:

$$\mathcal{S}^{(k)}_{\mathsf{edge}}(e, \beta, \gamma; T, \mathbf{\Theta}) = \frac{\mathcal{S}^{(k)}_{\mathsf{node}}(u, \beta; T, \mathbf{\Theta})\mathcal{L}^{(k)}_{\mathsf{node}}(v, \gamma; T, \mathbf{\Theta})\mathbf{\Psi}^{(k)}_e(\beta, \gamma)}{\mathcal{L}^{(k)}_{\mathsf{edge}}(e, \beta; T, \mathbf{\Theta})}$$

Consider the following exclusive cases:

- Case 1: $\gamma = 0$, then $\beta \neq 0$ (because $(\beta, \gamma) \neq (0, 0)$). It is followed that $\mathbf{\Psi}^{(k)}_e(\beta, \gamma) = \mathbf{\Psi}^{(k)}_e(\beta, 0) = 0$, so $\mathcal{S}^{(k)}_{\mathsf{edge}}(e, \beta, \gamma; T, \mathbf{\Theta})$.
- Case 2: $\gamma = \alpha$, then $\beta \neq \alpha$ (because $(\beta, \gamma) \neq (\alpha, \alpha)$). It is also followed that $\mathbf{\Psi}^{(k)}_e(\beta, \gamma) = 0$, so $\mathcal{S}^{(k)}_{\mathsf{edge}} = 0$
- Case 3: $\gamma \notin \{0, \alpha\}$. Using Statement (1), we have $\mathcal{L}^{(k)}_{\mathsf{node}}(v, \gamma; T, \mathbf{\Theta}) = 0$, so $\mathcal{S}^{(k)}_{\mathsf{edge}}(e, \beta, \gamma; T, \mathbf{\Theta}) = 0$.

Thus, $\mathcal{S}^{(k)}_{\mathsf{edge}}(e, \beta, \gamma; T, \mathbf{\Theta}) = 0$ for all $(\beta, \gamma) \in (\mathscr{A}^{(k)})^2 \setminus \{(0,0), (0,\alpha), (\alpha,\alpha)\}$. □

S1.5.2 The linear-time algorithm

Recall that at iteration $t+1$ of the EM algorithm, we need to compute the posterior probabilities that are parameterized by $\mathbf{\Theta}^t$, where $\mathbf{\Theta}^t$ is the estimate at iteration t. Below we describe a new algorithm for the E-step that has lower complexity than that presented in the main text, using the special properties of α-clades and z-branches. Because the tree topology T and parameter $\mathbf{\Theta}^t$ are shared for all entities described below, in the rest of this section, we will drop these two parameters in all notations. Let

$$\mathcal{C}^{z\to z}_{e,k} = \mathcal{S}^{(k)}_{\mathsf{edge}}(e, 0, 0)\ ; \mathcal{C}^{z\to a}_{e,k} = \sum_{\alpha \in \mathscr{A}^{(k)} \setminus \{0,-1,?\}} \mathcal{S}^{(k)}_{\mathsf{edge}}(e, 0, \alpha)\ ; \mathcal{C}^{z\to m}_{e,k} = \mathcal{S}^{(k)}_{\mathsf{edge}}(e, 0, -1)$$
$$\mathcal{C}^{a\to a}_{e,k} = \sum_{\alpha \in \mathscr{A}^{(k)} \setminus \{0,-1,?\}} \mathcal{S}^{(k)}_{\mathsf{edge}}(e, \alpha, \alpha)\ ; \mathcal{C}^{a\to m}_{e,k} = \sum_{\alpha \in \mathscr{A}^{(k)} \setminus \{0,-1,?\}} \mathcal{S}^{(k)}_{\mathsf{edge}}(e, \alpha, -1)$$

From Eq. 11 (see the main text) and the above equations, it is easy to see that:

$$\mathcal{C}^{z\to z}_{e} = \sum_{k=1}^{K} \mathcal{C}^{z\to z}_{e,k}$$
$$\mathcal{C}^{z\to a}_{e} = \sum_{k=1}^{K} \mathcal{C}^{z\to a}_{e,k}$$
$$\mathcal{C}^{a\to a}_{e} = \sum_{k=1}^{K} \mathcal{C}^{a\to a}_{e,k}$$
$$\mathcal{C}^{a\to m}_{e} = \sum_{k=1}^{K} \mathcal{C}^{a\to m}_{e,k}$$

Let $p_e = e^{-\delta_e}$. Using theorems 2, 3, and 4, we will prove the followings:

Casework for $\mathcal{C}^{z\to z}_{e,k}$

$$\mathcal{C}^{z\to z}_{e,k} = \begin{cases} 1, \text{ if } e \text{ is a z-branch at site } k \\ \mathcal{S}^{(k)}_{\mathsf{node}}(u, 0)\frac{\mathcal{L}^{(k)}_{\mathsf{node}}(v,0)p_e^{1+\nu}}{\mathcal{L}^{(k)}_{\mathsf{edge}}(e,0)}, \text{ otherwise} \end{cases}$$

Proof. If e is a z-branch, Eq. S1.5.2 is a direct corollary of Statement (3) of Theorem 2. Otherwise, from Eq. S10 we have:

$$\mathcal{C}^{z\to z}_{e,k} = \mathcal{S}^{(k)}_{\mathsf{edge}}(e, 0, 0) = \frac{\mathcal{S}^{(k)}_{\mathsf{node}}(u, 0)\mathcal{L}^{(k)}_{\mathsf{node}}(v, 0)\mathbf{\Psi}^{(k)}_e(0, 0)}{\mathcal{L}^{(k)}_{\mathsf{edge}}(e, 0)} = \mathcal{S}^{(k)}_{\mathsf{node}}(u, 0)\frac{\mathcal{L}^{(k)}_{\mathsf{node}}(v, 0)p_e^{1+\nu}}{\mathcal{L}^{(k)}_{\mathsf{edge}}(e, 0)}$$

□

Casework for $\mathcal{C}_{e,k}^{z\to a}$

$$\mathcal{C}_{e,k}^{z\to a} = \begin{cases} 0, \text{ if } e \text{ is a z-branch at site } k \\ \mathcal{S}_{\mathsf{node}}^{(k)}(u,0) - \frac{\mathcal{S}_{\mathsf{node}}^{(k)}(u,0)\mathcal{L}_{\mathsf{node}}^{(k)}(v,0)}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,0)}, \text{ if } e \text{ is unmasked at } i \\ \mathcal{S}_{\mathsf{node}}^{(k)}(u,0) - \frac{\mathcal{S}_{\mathsf{node}}^{(k)}(u,0)}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,0)}(\mathcal{L}_{\mathsf{node}}^{(k)}(v,0)p_e^{1+\nu} + (1-p_e^{\nu})), \text{ otherwise} \end{cases}$$

Proof. From the definition of $\mathcal{C}_{e,k}^{z\to a}$ and Eq. S10, we have:

$$\begin{aligned} \mathcal{C}_{e,k}^{z\to a} &= \textstyle\sum_{\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?\}} \mathcal{S}_{\mathsf{edge}}^{(k)}(e,0,\alpha) \\ &= \frac{\mathcal{S}_{\mathsf{node}}^{(k)}(u,0)}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,0)} \textstyle\sum_{\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?\}} \mathcal{L}_{\mathsf{node}}^{(k)}(v,\alpha)\mathbf{\Psi}_e^{(k)}(0,\alpha) \end{aligned}$$

If e is a z-branch, then according to Theorem 2, $\mathcal{L}_{\mathsf{node}}^{(k)}(v,\alpha) = 0$ for all $\alpha \in \mathscr{A}^{(k)} \setminus \{0,-1,?\}$, so $\mathcal{C}_{e,k}^{z\to a} = 0$. Otherwise, we have:

$$\begin{aligned} &\sum_{\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?\}} \mathcal{L}_{\mathsf{node}}^{(k)}(v,\alpha)\mathbf{\Psi}_e^{(k)}(0,\alpha) \\ &= \sum_{\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?\}} \mathcal{P}(\mathbf{D}_v^{(k)}|\mathbb{X}^{(k)}(v)=\alpha)\mathcal{P}(\mathbb{X}^{(k)}(v)=0|\mathbb{X}^{(k)}(u)=0) \\ &= \mathcal{P}(\mathbf{D}_v^{(k)}|\mathbb{X}^{(k)}(u)=0) - \mathcal{P}(\mathbf{D}_v^{(k)}|\mathbb{X}^{(k)}(v)=0)\mathcal{P}(\mathbb{X}^{(k)}(v)=0|\mathbb{X}^{(k)}(u)=0) \\ &- \mathcal{P}(\mathbf{D}_v^{(k)}|\mathbb{X}^{(k)}(v)=-1)\mathcal{P}(\mathbb{X}^{(k)}(v)=-1|\mathbb{X}^{(k)}(u)=0) \\ &= \mathcal{L}_{\mathsf{edge}}^{(k)}(e,0) - \mathcal{L}_{\mathsf{node}}^{(k)}(v,0)\mathbf{\Psi}_e(0,0) - \mathcal{L}_{\mathsf{node}}^{(k)}(v,-1)\mathbf{\Psi}_e(0,-1) \end{aligned}$$

If e is a masked branch, then $\mathcal{L}_{\mathsf{node}}^{(k)}(v,-1) = 1$; otherwise, e is unmasked and is not a z-branch, so $\mathcal{L}_{\mathsf{node}}^{(k)}(v,-1) = 0$ according to Theorem 4. Also, $\mathbf{\Psi}_e(0,-1) = 1 - p_e^{\nu}$ by definition. Thus, from Eq. S1.5.2 and S1.5.2, we get the proposed equation to compute $\mathcal{C}_{e,k}^{z\to a}$. □

Casework for $\mathcal{C}_{e,k}^{z\to m}$

$$\mathcal{C}_{e,k}^{z\to m} = \begin{cases} \mathcal{S}_{\mathsf{node}}^{(k)}(u,0)\frac{1-p_e^{\nu}}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,0)}, \text{ if } e \text{ is a masked-branch} \\ 0, \text{ otherwise} \end{cases}$$

Proof. From the definition of $\mathcal{C}_{e,k}^{z\to m}$ and Eq. S10, we have:

$$\begin{aligned} \mathcal{C}_{e,k}^{z\to m} &= \mathcal{S}_{\mathsf{edge}}^{(k)}(e,0,-1) \\ &= \frac{\mathcal{S}_{\mathsf{node}}^{(k)}(u,0)\mathcal{L}_{\mathsf{node}}^{(k)}(v,-1)\mathbf{\Psi}_e^{(k)}(0,-1)}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,0)} \end{aligned}$$

If e is not a masked branch, then according to Theorem 2 and Theorem 4, $\mathcal{L}_{\mathsf{node}}^{(k)}(v,-1) = 0$, so $\mathcal{C}_{e,k}^{z\to m} = 0$. Otherwise, if e is a masked branch, then $\mathcal{L}_{\mathsf{node}}^{(k)}(v,-1) = 1$ (Theorem 3), and $\mathbf{\Psi}_e^{(k)}(0,-1) = 1-p_e^{\nu}$. From here we get the proposed equation. □

Casework for $\mathcal{C}_{e,k}^{a\to a}$

$$\mathcal{C}_{e,k}^{a\to a} = \begin{cases} 0, \text{ if e is a z-branch at site } k \\ \frac{\mathcal{L}_{\mathsf{node}}^{(k)}(v,\alpha_0)p_e^{\nu}}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,\alpha_0)}\mathcal{S}_{\mathsf{node}}^{(k)}(u,\alpha_0), \text{ if e is an unmasked branch of } \alpha_0\text{-clade at } k \\ \dfrac{\mathcal{L}_{\mathsf{node}}^{(k)}(v,\alpha_0)p_e^{\nu}}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,\alpha_0)}(1 - \mathcal{S}_{\mathsf{node}}^{(k)}(u,0) - \mathcal{S}_{\mathsf{node}}^{(k)}(u,-1)), \text{ if } e \text{ is a masked branch, } \alpha_0 \text{ is arbitrarily chosen in } \mathscr{A}^{(k)} \setminus \{0,-1,?\}) \end{cases}$$

Proof.

$$\begin{aligned}\mathcal{C}_{e,k}^{a\to a} &= \sum_{\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?\}} \mathcal{S}_{\mathsf{edge}}^{(k)}(e,\alpha,\alpha)\\ &= \sum_{\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?\}} \frac{\mathcal{S}_{\mathsf{node}}^{(k)}(u,\alpha)\mathcal{L}_{\mathsf{node}}^{(k)}(v,\alpha)\mathbf{\Psi}_e^{(k)}(\alpha,\alpha)}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,\alpha)}\\ &= \sum_{\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?\}} \frac{\mathcal{S}_{\mathsf{node}}^{(k)}(u,\alpha)\mathcal{L}_{\mathsf{node}}^{(k)}(v,\alpha)p_e^{\nu}}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,\alpha)}\end{aligned}$$

Case 1: If e is a z-branch, then $\mathcal{S}_{\mathsf{node}}^{(k)}(u,\alpha)=0$ for all $\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?\}$ (Theorem 2), so $\mathcal{C}_{e,k}^{a\to a}=0$.
Case 2: If e is an unmasked branch of some α_0-clade, then $\mathcal{P}(\mathbf{D}_v^{(k)}|\mathbb{X}^{(k)}(v)=\alpha)=0$ for all $\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?,\alpha_0\}$ (Theorem 4), so the summation is reduced to the only term containing α_0. Thus:

$$\mathcal{C}_{e,k}^{a\to a} = \frac{\mathcal{L}_{\mathsf{node}}^{(k)}(v,\alpha_0)p_e^{\nu}}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,\alpha_0)}\mathcal{S}_{\mathsf{node}}^{(k)}(u,\alpha_0)$$

Case 3: If e is a masked branch, then from Statements (2) and (3) of Theorem 3, we have:

$$\begin{aligned}\mathcal{C}_{e,k}^{a\to a} &= \tfrac{\mathcal{L}_{\mathsf{node}}^{(k)}(v,\alpha_0)p_e^{\nu}}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,\alpha_0)}\textstyle\sum_{\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?\}}\mathcal{S}_{\mathsf{node}}^{(k)}(u,\alpha)\\ &= \tfrac{\mathcal{L}_{\mathsf{node}}^{(k)}(v,\alpha_0)p_e^{\nu}}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,\alpha_0)}(1-\mathcal{S}_{\mathsf{node}}^{(k)}(u,0)-\mathcal{S}_{\mathsf{node}}^{(k)}(u,-1)),\end{aligned}$$

where α_0 is arbitrarily chosen in $\mathscr{A}^{(k)}\setminus\{0,-1,?\}$ □

Casework for $\mathcal{C}_{e,k}^{a\to m}$

$$\mathcal{C}_{e,k}^{a\to m} = \begin{cases}\dfrac{1-p_e^{\nu}}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,\alpha_0)}(1-\mathcal{S}_{\mathsf{node}}^{(k)}(u,0)-\mathcal{S}_{\mathsf{node}}^{(k)}(u,-1)), & \text{if } e \text{ is a masked branch, } \alpha_0 \text{ is arbitrarily chosen in } \mathscr{A}^{(k)}\setminus\{0,-1,?\})\\ 0, & \text{otherwise}\end{cases}$$

Proof. From the definition of $\mathcal{C}_{e,k}^{a\to m}$ and Eq. S10, we have:

$$\begin{aligned}\mathcal{C}_{e,k}^{a\to m} &= \textstyle\sum_{\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?\}}\mathcal{S}_{\mathsf{edge}}^{(k)}(e,\alpha,-1)\\ &= \textstyle\sum_{\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?\}}\tfrac{\mathcal{S}_{\mathsf{node}}^{(k)}(u,\alpha)\mathcal{L}_{\mathsf{node}}^{(k)}(v,-1)\mathbf{\Psi}_e^{(k)}(\alpha,-1)}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,\alpha)}\end{aligned}$$

If e is not a masked branch, then according to Theorem 2 and Theorem 4, $\mathcal{L}_{\mathsf{node}}^{(k)}(v,-1)=0$, so $\mathcal{C}_{e,k}^{a\to m}=0$. Otherwise, if e is a masked branch, then $\mathcal{L}_{\mathsf{node}}^{(k)}(v,-1)=1$ (Theorem 3), $\mathbf{\Psi}_e^{(k)}(\alpha,-1)=1-p_e^{\nu}$ for all $\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?\}$. Combining with Statements (2) and (3) of Theorem 3, we have:

$$\begin{aligned}\mathcal{C}_{e,k}^{a\to m} &= \tfrac{1-p_e^{\nu}}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,\alpha_0)}\textstyle\sum_{\alpha\in\mathscr{A}^{(k)}\setminus\{0,-1,?\}}\mathcal{S}_{\mathsf{node}}^{(k)}(u,\alpha)\\ &= \tfrac{1-p_e^{\nu}}{\mathcal{L}_{\mathsf{edge}}^{(k)}(e,\alpha_0)}(1-\mathcal{S}_{\mathsf{node}}^{(k)}(u,0)-\mathcal{S}_{\mathsf{node}}^{(k)}(u,-1)),\end{aligned}$$

where α_0 is arbitrarily chosen in $\mathscr{A}^{(k)}\setminus\{0,-1,?\}$ □

Thus, for any branch $e=(u,v)$, if e is a z-branch of site k, then each of the 5 entities $\mathcal{C}_{e,k}^{z\to z}$, $\mathcal{C}_{e,k}^{z\to a}$, $\mathcal{C}_{e,k}^{z\to m}$, $\mathcal{C}_{e,k}^{a\to a}$, $\mathcal{C}_{e,k}^{a\to m}$ is either 0 or 1 (Theorem 2). Otherwise, to compute them we need the following 6 entities:

- $\mathcal{L}_{\mathsf{edge}}^{(k)}(e,0)$
- $\mathcal{L}_{\mathsf{edge}}^{(k)}(e,\alpha_0)$

- $\mathcal{L}^{(k)}_{\mathsf{node}}(v, 0)$
- $\mathcal{L}^{(k)}_{\mathsf{node}}(v, \alpha_0)$
- $\mathcal{S}^{(k)}_{\mathsf{node}}(u, 0)$
- $\mathcal{S}^{(k)}_{\mathsf{node}}(u, -1)$,

where α_0 is arbitrarily chosen in $\mathscr{A}^{(k)} \setminus \{0, -1\}$ if e is a masked-branch; otherwise, e belongs to an α-clade where α is uniquely defined and we let $\alpha_0 = \alpha$.

Computing $\mathcal{L}^{(k)}_{\mathsf{edge}}(e, 0)$ and $\mathcal{L}^{(k)}_{\mathsf{edge}}(e, \alpha_0)$ Base case: if v is a leaf node, then

$$\mathcal{L}^{(k)}_{\mathsf{edge}}(e, 0) = \begin{cases} \phi p_e^{\nu} - p_e^{\nu} + 1, \text{ if } e \text{ is a masked branch} \\ q_{\alpha_0} p_e^{\nu}(1 - p_e)(1 - \phi), \text{ otherwise} \end{cases}$$

$$\mathcal{L}^{(k)}_{\mathsf{edge}}(e, \alpha_0) = \begin{cases} \phi p_e^{\nu} - p_e^{\nu} + 1, \text{ if } e \text{ is a masked branch} \\ p_e^{\nu}(1 - \phi), \text{ otherwise} \end{cases}$$

Recursion: if v has two children, v_1 and v_2, then let $e_1 = (v, v_1), e_2 = (v, v_2)$, we have:

$$\mathcal{L}^{(k)}_{\mathsf{edge}}(e, 0) = \begin{cases} \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, 0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, 0)p_e^{\nu+1} + \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \alpha_0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, \alpha_0)(1 - p_e)p_e^{\nu} + (1 - p_e^{\nu}), \text{ if } e \text{ is a masked branch} \\ \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, 0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, 0)p_e^{\nu+1} + \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \alpha_0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, \alpha_0)q_{\alpha_0}(1 - p_e)p_e^{\nu}, \text{ otherwise} \end{cases}$$

Computing $\mathcal{L}^{(k)}_{\mathsf{node}}(v, 0)$ and $\mathcal{L}^{(k)}_{\mathsf{node}}(v, \alpha_0)$

$$\mathcal{L}^{(k)}_{\mathsf{node}}(v, 0) = \begin{cases} \phi, \text{ if v is a leaf and } \mathbf{D}^{(k)}(v) = ? \\ 1 - \phi, \text{ if v is a leaf and } \mathbf{D}^{(k)}(v) = 0 \\ 0, \text{ if v is a leaf and } \mathbf{D}^{(k)}(v) \notin \{0, ?\} \\ \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, 0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, 0), \text{ if v has two children, } v_1 \text{ and } v_2;\ e_1 = (v, v_1), e_2 = (v, v_2) \end{cases}$$

$$\mathcal{L}^{(k)}_{\mathsf{node}}(v, \alpha_0) = \begin{cases} \phi, \text{ if v is a leaf and } \mathbf{D}^{(k)}(v) = ? \\ 1 - \phi, \text{ if v is a leaf and } \mathbf{D}^{(k)}(v) = \alpha_0 \\ 0, \text{ if v is a leaf and } \mathbf{D}^{(k)}(v) \notin \{\alpha_0, ?\} \\ \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \alpha_0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_2, \alpha_0), \text{ if v has two children, } v_1 \text{ and } v_2;\ e_1 = (v, v_1), e_2 = (v, v_2) \end{cases}$$

Computing $\mathcal{S}^{(k)}_{\mathsf{node}}(u, 0)$ and $\mathcal{S}^{(k)}_{\mathsf{node}}(u, -1)$. Consider a branch (u, v). From Eq. (S8), we have: if u is the root, then $\mathcal{S}^{(k)}_{\mathsf{node}}(u, 0) = 1$ and $\mathcal{S}^{(k)}_{\mathsf{node}}(u, -1) = 0$. Otherwise, to compute these two entities we need 5 other entities: (1) $\mathcal{L}^{(k)}_{\mathsf{node}}(u, 0)$, (2) $\mathcal{L}^{(k)}_{\mathsf{node}}(u, -1)$, (3) $\mathcal{P}(\mathbf{D}^{(k)})$, (4) $\mathcal{L}^{(k)}_{\mathsf{out}}(u, 0)$, and (5) $\mathcal{L}^{(k)}_{\mathsf{out}}(u, -1)$. Among them, (1) and (2) are already computed (see the above section). Entity (3) is simply the full likelihood of site k, so it is equal to $\mathcal{L}^{(k)}_{\mathsf{edge}}(e_0, 0)$ where e_0 is the root branch. To compute (4) and (5) (i.e. the outside likelihoods), we can use Eq. (S6), which gives a top-down recursive formula. Let u_0 be the parent of u, $e = (u_0, u)$ be the branch connecting u_0 and u, and u_1 be the sister of u (if there exists one) and $e_1 = (u_0, u_1)$. From Eq. (S6), it is straightforward to derive the following:

$$\mathcal{L}^{(k)}_{\mathsf{out}}(u, 0) = \begin{cases} p_e^{1+\nu}, \text{ if } e \text{ is the root branch} \\ p_e^{1+\nu}\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, 0)\mathcal{L}^{(k)}_{\mathsf{out}}(u_0, 0), \text{ otherwise} \end{cases}$$

Thus, we have a top-down recursion to compute $\mathcal{L}^{(k)}_{\mathsf{out}}(u, 0)$. The computation of $\mathcal{L}^{(k)}_{\mathsf{out}}(u, -1)$, however, is more complicated. A naive application of Eq. (S6) requires the outside likelihoods with respect to all states β at the parent of u. Therefore, it takes $\mathcal{O}(M)$ time to compute $\mathcal{L}^{(k)}_{\mathsf{out}}(u, -1)$ for each node u, so the overall complexity is $\mathcal{O}(NKM)$ where M is the maximum alphabet size. Nevertheless, we can still employ the special properties of the model to reduce the algorithm to $\mathcal{O}(NK)$, as shown below.

Let v be the parent of u, $e = (v, u) \in \mathscr{E}_T$ be the branch connecting v and u, w be the sister of u, and $e_1 = (v, w)$. Using Eq. S6, it is straight forward to prove the followings:

- If e_1 is a z-branch, then $\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \beta) = 0$ for all $\beta \neq 0$. Therefore, $\mathcal{L}^{(k)}_{\mathsf{out}}(u, -1) = (1-p^\nu_e)\mathcal{L}^{(k)}_{\mathsf{out}}(v, 0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, -1)$
- If e_1 is an unmasked branch of an α_0-clade, then $\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \beta) = 0$ for all $\beta \notin \{0, \alpha_0\}$. Therefore, $\mathcal{L}^{(k)}_{\mathsf{out}}(u, -1) = (1 - p^\nu_e)\mathcal{L}^{(k)}_{\mathsf{out}}(v, 0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, 0) + (1 - p^\nu_e)\mathcal{L}^{(k)}_{\mathsf{out}}(v, \alpha_0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \alpha_0)$
- If e_1 is a masked branch, then $\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \alpha) = \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \beta)$ for all $\alpha \neq \beta$, $\alpha, \beta \in \mathscr{A}^{(k)} \setminus \{0, -1, ?\}$ (Theoreom 3). Let α_0 be an arbitrarily chosen character state in $\mathscr{A}^{(k)} \setminus \{0, -1, ?\}$. Then $\mathcal{L}^{(k)}_{\mathsf{out}}(u, -1) = (1 - p^\nu_e)\mathcal{L}^{(k)}_{\mathsf{out}}(v, 0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, 0)+\mathcal{L}^{(k)}_{\mathsf{out}}(v, -1)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, -1)+(1-p^\nu_e)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \alpha_0)M_v$, where $M_v = \sum_{\beta \notin \{0,-1\}} \mathcal{L}^{(k)}_{\mathsf{out}}(v, \beta)$.

From the above equations, we see that in all cases $\mathcal{L}^{(k)}_{\mathsf{out}}(u, -1)$ can be computed from $\mathcal{L}^{(k)}_{\mathsf{out}}(v, \alpha_0)$, M_v, and other known entities. Therefore, what left to be done is deriving top-down recursive formulas for $\mathcal{L}^{(k)}_{\mathsf{out}}(v, \alpha_0)$ and M_v. Below we show the formulas and leave the derivation to the reader.

Recursive formula for M_v. Consider a branch $e = (u, v)$, let w be the sister of v and $e_1 = (u, w)$. Recall that by definition, $M_v = \sum_{\beta \notin \{0,-1\}} \mathcal{L}^{(k)}_{\mathsf{out}}(v, \beta)$. Below is the topdown recursive formula to compute M_v from M_u and other known entities:

$$M_v = \begin{cases} \mathcal{L}^{(k)}_{\mathsf{out}}(u, 0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, 0)p^\nu_e(1 - p_e), \text{ if } e_1 \text{ is a z-branch} \\ \mathcal{L}^{(k)}_{\mathsf{out}}(u, 0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, 0)p^\nu_e(1 - p_e) + p^\nu_e M_u \mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \alpha_0), \text{ if } e_1 \text{ is a masked branch} \\ \mathcal{L}^{(k)}_{\mathsf{out}}(u, 0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, 0)p^\nu_e(1 - p_e) + \mathcal{L}^{(k)}_{\mathsf{out}}(u, \alpha_0)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \alpha_0)p^\nu_e, \text{ otherwise} \end{cases}$$

Recursive formula for $\mathcal{L}^{(k)}_{\mathsf{out}}(v, \alpha_0)$. Consider a branch $e = (u, v)$, let w be the sister of v and $e_1 = (u, w)$. For any $\alpha_0 \notin \{0, -1\}$, we have the following topdown recursive formula to compute $\mathcal{L}^{(k)}_{\mathsf{out}}(v, \alpha_0)$ from $\mathcal{L}^{(k)}_{\mathsf{out}}(u, 0)$, $\mathcal{L}^{(k)}_{\mathsf{out}}(u, \alpha_0)$, and other known entities:

$$\mathcal{L}^{(k)}_{\mathsf{out}}(v, \alpha_0) = \mathcal{L}^{(k)}_{\mathsf{out}}(u, 0)q_{\alpha_0}p^\nu_e(1 - p_e)\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, 0) + \mathcal{L}^{(k)}_{\mathsf{out}}(u, \alpha_0)p^\nu_e\mathcal{L}^{(k)}_{\mathsf{edge}}(e_1, \alpha_0)$$

S1.6 Solving the M-step

S1.6.1 Proof of convexity by variable blocks

Recall that at iteration t of the proposed EM algorithm, in the M-step we need to solve the following optimization problem:

$$\begin{aligned} \max_{\boldsymbol{\Theta}} f(\boldsymbol{\Theta}; \boldsymbol{\Theta}^{t-1}) = \sum_{e=(u,v)\in\mathscr{E}_T} & \big(-\mathcal{C}^{z\to z}_e(1 + \nu)\delta_e + \mathcal{C}^{z\to a}_e(\log(1 - e^{-\delta_e}) - \nu\delta_e) + \\ & \mathcal{C}^{z\to m}_e \log(1 - e^{-\delta_e\nu}) - \mathcal{C}^{a\to a}_e\delta_e\nu + \mathcal{C}^{a\to m}_e \log(1 - e^{-\delta_e\nu})\big) + \\ & \sum_{u\in\mathscr{L}_T} \Big(\mathcal{B}_u \log(1 - \phi) + \tilde{\mathcal{B}}_u \log\phi\Big) \end{aligned}$$

Below we prove that $f(\boldsymbol{\Theta}; \boldsymbol{\Theta}^{t-1})$ is concave with respect to $\{\delta_e\}$, ν, and ϕ separately. Note that all variables are separable, so to prove concavity, we need to show that the second-order partial derivatives are negative. Indeed, we have:

$$\frac{\partial^2 f}{\partial\phi^2} = \sum_{u\in\mathscr{L}_T} \left(-\frac{\mathcal{B}_u}{(1 - \phi)^2} - \frac{\tilde{\mathcal{B}}_u}{\phi^2}\right) < 0 \text{ for all } \phi \notin \{0, 1\}$$

$$\frac{\partial^2 f}{\partial\nu^2} = \sum_{e\in\mathscr{E}_T} -(\mathcal{C}^{z\to m}_e + \mathcal{C}^{a\to m}_e)\frac{e^{-\delta_e\nu}}{(1 - e^{-\delta_e\nu})^2}\delta^2_e < 0 \text{ for all } \nu \geq 0$$

$$\frac{\partial^2 f}{\partial\delta^2_e} = \sum_{e\in\mathscr{E}_T} \left(-\mathcal{C}^{z\to a}_e\frac{e^{-\delta_e}}{(1 - e^{-\delta_e})^2} - (\mathcal{C}^{z\to m}_e + \mathcal{C}^{a\to m}_e)\frac{e^{-\delta_e\nu}}{(1 - e^{-\delta_e\nu})^2}\nu^2\right) < 0 \text{ for all } \delta_e \geq 0$$

S1.6.2 The block coordinate ascent algorithm

We initialize $\mathbf{\Theta}^t \leftarrow \mathbf{\Theta}^{t-1}$, then successively minimize f along the coordinate of ϕ, ν, and the block of $\{\delta_e\}$ while fixing the others, and iterate until convergence. In other words, let $\phi^{(t,1)} = \phi^{(t)}$, $\nu^{(t,1)} = \nu^{(t)}$, and $\{\delta\}^{(t,1)} = \{\delta\}^{(t)}$; at each iteration p of coordinate ascent, we find $\phi^{(t,p+1)}$, $\nu^{(t,p+1)}$, and $\{\delta\}^{(t,p+1)}$ such that:

$$\phi^{(t,p+1)} = \underset{\phi}{\operatorname{argmax}} \sum_{u \in \mathscr{L}_T} \Big(\mathcal{B}_u \log(1-\phi) + \tilde{\mathcal{B}}_u \log \phi \Big) = \frac{\sum_{u \in \mathscr{L}_T} \tilde{\mathcal{B}}_u}{\sum_{u \in \mathscr{L}_T} (\mathcal{B}_u + \tilde{\mathcal{B}}_u)}$$

$$\begin{aligned}\nu^{(t,p+1)} = \underset{\nu}{\operatorname{argmax}} \sum_{e \in \mathscr{E}_T} \Big(& -\mathcal{C}_e^{z\to z}(1+\nu)\delta_e^{(t,p)} + \mathcal{C}_e^{z\to a}(\log(1-e^{-\delta_e^{(t,p)}}) - \nu\delta_e^{(t,p)}) + \\ & \mathcal{C}_e^{z\to m}\log(1-e^{-\delta_e^{(t,p)}\nu}) - \mathcal{C}_e^{a\to a}\delta_e^{(t,p)}\nu + \mathcal{C}_e^{a\to m}\log(1-e^{-\delta_e^{(t,p)}\nu})\Big)\end{aligned}$$

$$\begin{aligned}\{\delta_e\}^{(t,p+1)} = \underset{\{\delta_e\}}{\operatorname{argmax}} \sum_{e \in \mathscr{E}_T} \Big(& -\mathcal{C}_e^{z\to z}(1+\nu^{(t,p+1)})\delta_e + \mathcal{C}_e^{z\to a}(\log(1-e^{-\delta_e}) - \nu^{(t,p+1)}\delta_e) + \\ & \mathcal{C}_e^{z\to m}\log(1-e^{-\delta_e\nu^{(t,p+1)}}) - \mathcal{C}_e^{a\to a}\delta_e\nu^{(t,p+1)} + \mathcal{C}_e^{a\to m}\log(1-e^{-\delta_e\nu^{(t,p+1)}})\Big)\end{aligned}$$

While $\phi^{(t,p+1)}$ has a closed-form, $\nu^{(t,p+1)}$ and $\{\delta_e\}^{(t,p+1)}$ require solving convex optimization problems. We use the CVXPY package [11, 1] with the MOSEK solver [3] to solve them.

S1.7 Simulated annealing for topology search

We repeat the following procedure, starting with iteration $t = 1$, until convergence or $t > t_{\mathsf{max}}$:

1. At iteration t, do the following steps:
   (a) Propose a new topology $T^{(t)}$ using an NNI move on $T^{(t-1)}$.
   (b) Optimize for $\mathbf{\Theta}^{(t)}$ using EM algorithm
   (c) Compute the log-likelihood $l^{(t)} = \log L(T^{(t)}, \mathbf{\Theta}^{(i)}; \mathbf{D})$
   (d) Go to (2)
2. If $l^{(t)} > l^{(t-1)}$, then accept $T^{(t)}$ and $\mathbf{\Theta}^{(t)}$ and go to (3). Otherwise, compute the temperature $\mathsf{Temp(t)}$ and acceptance probability $p(t)$, as follows:

$$\mathsf{Temp}(t) = \max\left(\epsilon, \frac{\alpha^t - \alpha^c}{1-\alpha^c}\right)$$
$$p(t) = \min(1, \exp(l(t) - l(t-1) - \epsilon)/\mathsf{Temp}(t)),$$

   where c, α and ϵ are hyperparameters controlling the simulated annealing procedure. By default, we set $\epsilon = 1e-12, c = 20$, and $\alpha = 0.9$. Accept $T^{(t)}$ and $\mathbf{\Theta}^{(t)}$ with probability $p(t)$ and go to (3).
3. If $T^{(t)}$ and $\mathbf{\Theta}^{(t)}$ are accepted in (2), then increase t and iterate back to step (1). Otherwise, reject $T^{(t)}$ and $\mathbf{\Theta}^{(t)}$; without increasing i, turn back to step (1) and try another NNI move of $T^{(t-1)}$. If there is no NNI move left or the likelihood improvement is below a predefined threshold η, stop the procedure and return $T^{(t-1)}$ and $\mathbf{\Theta}^{(t-1)}$.

S2 Supplementary data

Contents

S2.1 Evaluation metrics . . . 1
S2.1.1 The Robinson-Foulds (RF) error . . . 1
S2.1.2 Migration Cost . . . 1
S2.1.3 Variations on the Migration Cost . . . 2
S2.1.4 Mutual information with gene expression . . . 2
S2.2 Simulation procedure . . . 2
S2.2.1 Benchmarking LAML . . . 3
S2.2.2 Leveraging the Statistical Model in Distance-Based Approaches . . . 4
S2.3 The KP-tracer data . . . 4
S2.4 The intMEMOIR data . . . 4
S2.5 Supplementary Results for Simulated Data . . . 5
S2.6 Benchmarking LAML . . . 6
S2.6.1 Branch Length Results . . . 8
S2.6.2 Runtime Results . . . 9
S2.7 Supplementary Results for KP-Tracer . . . 10
S2.7.1 Migration Metrics Table . . . 10
S2.7.2 Metastasis Analysis . . . 10
S2.8 Supplementary Results for intMEMOIR . . . 11
S2.9 Distance-Based Exploration . . . 14
S2.10 Review of Lineage Tracing Technologies . . . 15
S2.11 Review of Other Computational Approaches . . . 15

S2.1 Evaluation metrics

S2.1.1 The Robinson-Foulds (RF) error

For simulated data and the intMemoir dataset where the ground-truth topology is known, we measure topological error by the *RF error*.

The RF distance between two trees T_1 and T_2 on a same leafset is the sum of false positives (FP - bipartitions in T_2 not present in T_1) and false negatives (FN - bipartitions in T_1 not present in T_2). We normalize the RF distance by the total number of internal edges in both trees. Thus:

$$\Delta_{RF}(T_1, T_2) = \frac{FN + FP}{n_{internal}(T_1) + n_{internal}(T_2)}, \tag{S11}$$

where $n_{internal}$ denotes the number of internal edges of a tree. The RF error of an estimated topology is simply its normalized RF distance to the true topology. We use *root-mean-square error (RMSE)* to compute the error in estimating ϕ and ν. Specifically, for each model condition where $\theta^* = (\phi^*, \nu^*)$ is the true set of parameters, the error of an estimate $\hat{\theta} = (\hat{\phi}, \hat{\nu})$ is $d(\hat{\theta}, \theta^*) = (\hat{\phi} - \phi^*)^2 + (\hat{\nu} - \nu^*)^2$. The RMSE of the estimates across m replicates is

$$\text{RMSE}(\{\hat{\theta}_i\}, \theta^*) = \sqrt{\sum_{i=1}^{m} \frac{d(\hat{\theta}_i, \theta^*)}{m}}, \tag{S12}$$

where $m = 50$ in all model conditions.

S2.1.2 Migration Cost

For the sample 3724_NT_T1_All of KP-tracer data where topology is re-estimated using LAML, we compare the trees in terms of *migration cost*, which is the phylogeny's parsimony cost on the annotated anatomical labels of the cells. This metric is the same as that used in [38], and was introduced and implemented in MACHINA [13]. Below we briefly describe the construction of a *migration graph*; see [38] and [13] for more details.

1. First, infer the ancestral labeling using MP (i.e. migration annotations on the internal nodes).

2. Collapse all branches that share identical labels between the parent node and child node.

3. Group all unique (parent state, child state) edges. Each state tuple should be annotated with the number of times edges labeled with (parent state, child state) were observed in the tree. These correspond to migration events.

The *migration cost* is the sum of the edge weights.

S2.1.3 Variations on the Migration Cost

The KP-Tracer sample (`3724_NT_ALL`) includes two useful meta annotations on the observed cells. First, we have the anatomical site labeling (e.g. primary tumor, lung metastasis 1, lung metastasis 2, lung metastasis 3, soft tissue metastasis). This indicates where in the body each cell was sampled from. Second, we have a "refinement" of this anatomical site labeling for certain tumors, that indicates the approximate spatial location of the sample in that tumor (e.g. primary tumor can be broken into 15 components).

As defined above, a migration graph is defined by the tree and the clustering. Thus, given a tree topology T and a clustering SC for spatial clustering and AC for anatomical clustering, we can get migration graph $T + SC \rightarrow MG_{SC}$ and $T + AC \rightarrow MG_{AC}$. We presented results on the anatomical clustering in the main body of the paper. We include additional results on MG_{SC}, and compute different costs on this migration graph (below) in the Supplementary Results section for KP-Tracer S2.7.

1. Total Cost: The migration cost of $MG_{SC}(T_{LAML})$ and $MG_{SC}(T_{CassH})$.

2. Induced Anatomical Cost: Using the parsimonious ancestral labeling inferred using the spatial clustering SC, infer the number of transitions between anatomical sites.

3. Spatially-aware Migration Cost: Within the **primary tumor only**, each transition is weighted as follows: migration cost $*$ length of path between spatial locations.

4. Constrained Parsimony: Given a tree topology T, we first compute a parsimonious ancestral labeling solution given the anatomical clustering AC. Then, for clades within each anatomical site, we compute a parsimonious ancestral labeling solution given the spatial clustering SC.

S2.1.4 Mutual information with gene expression

For the 6 samples of KP-tracer where branch lengths are evaluated, we use gene expression data to benchmark the branch lengths estimated by MP and ML. We construct two pairwise *phylogenetic distance matrices*, one for MP and one for ML, based on the corresponding estimated branch lengths. In addition, we add another distance matrix computed solely from tree topology (i.e. setting all branch lengths to 1), used as control. We preprocess the gene expression data by performing dimensionality reduction, using PCA (10 PCs), scVI (10 PCs), and UMAP (2 PCs). We use the published vectors provided in KP-tracer for scVI and UMAP, and perform PCA using SCANPY [49]. We constructed 3 pairwise *gene expression distance matrices* corresponding to each of those 3 dimensionality reduction techniques, using Euclidean distances. Thus, we have 6 distance matrices for each KP-tracer sample: 3 constructed from phylogenetic distances and 3 constructed from gene expression distances. We pair each phylogenetic with each gene expression distance matrices, producing 9 phylo-expression pairs. For each of these 9 pairs, we compute the non-parametric mutual information [26, 36] using scikit-learn [33]. These mutual information scores are used to benchmark MP, ML, and topological distances.

S2.2 Simulation procedure

We simulate a model tree of 1024 leaves (i.e. 10 cell generations) using the birth-only model (implemented in Cassiopeia). Branch lengths in time unit are drawn according to a lognormal distribution with mean 1 and standard deviation 0.1. We simulate sequences of length 30 under our generative model, with hyperparameters $q^{(k)}$ at each site k drawn from that of the trunk-like single-cell lineage tracing mouse (TLSCL) dataset (unpublished data). We also match two other statistics of the TLSCL: the expected proportion of non-zero entries (64%) and missing data (25%). To achieve an expected proportion of 64% non-zeroes, we use a molecular clock that has *constant mutation rate* $\lambda = 0.095$. To obtain a 25% overall missing proportion, we calibrated ϕ and ν for each of the 5 model conditions, as summarized in Table S2. Each simulated character matrix is subsampled to 250 to reflect sparse sampling, and the model tree is pruned to the same subset. We repeat the subsampling procedure 5 times for each character matrix, creating 250 replicates in total.

Model condition	Heritable percent	Dropout percent	Heritable rate (ν)	Dropout rate (ϕ)
h0d100	0%	100%	0	0.25
h25d75	25%	75%	0.065	0.2
h50d50	50%	50%	0.134	0.143
h75d25	75%	25%	0.208	0.077
h100d0	100%	0%	0.288	0

Table S2: Simulated dataset under five model conditions. All model conditions was simulated with 25% missing data; the attribution of the missing entries to heritable missing versus dropout depending on the model condition.

S2.2.1 Benchmarking LAML

Using this simulated dataset, we benchmark LAML against other methods: Greedy method (based on the principle of perfect phylogeny, used as the baseline), Startle [38], Neighbor Joining [37] (implemented in Cassiopeia, with weighted Hamming distances [24]), and DCLEAR. For the Greedy method, we use the implementation of Cassiopeia [24]). We run Startle in NNI mode with 250 iterations (Startle-NNI) and use the topology estimated by Greedy method as the starting tree. LAML was run using Startle-NNI as the starting tree; simulated annealing NNI with EM optimization was performed until convergence of likelihood.

We used the following commands to run each method. For all Cassiopeia-implemented methods, we used the wrapper script: `cassiopeia_solvers_with_pickle.py`. Following the default tutorial pipeline, we allow Cassiopeia to collapse all mutationless edges, producing trees with polytomies.

1. Startle-NNI: `python startle.py <seed_tree> <character_matrix> -e <mutation_priors> --iterations 250 --output tree.nwk`
2. Greedy method (Cassiopeia-Greedy): `cas.solver.VanillaGreedySolver()`
3. Neighbor Joining (Cassiopeia): `cas.solver.NeighborJoiningSolver( dissimilarity_function=cas.solver.dissimilarity.weighted_hamming_distance, add_root=True)`
4. LAML: `python run_problin.py -t <starting_tree> -c <character_matrix> -p <mutation_prior> -o <output> -v --delimiter comma --nInitials 1 --topology_search --ultrametric > <logfile> 2>&1`

To run the Cassiopeia-Hybrid (CH) and Cassiopeia-ILP (CILP) versions, we tried Cassiopeia Release (tag 2.0.0) `https://github.com/YosefLab/Cassiopeia/releases/tag/2.0.0`, as well as the development version (last updated Aug 8 2023, commit hash: `f89530145adad929fc47a2e80f70a93c6af2cf3f`). Unfortunately, we were unable to run either of these methods on our simulated dataset. CH failed on all of the simulated data inputs by throwing an index error within the Cassiopeia code. CILP failed on approximately half the simulated data inputs (some with Gurobi errors and some with index errors from within the Cassiopeia code).

There are two ways of running DCLEAR. The first, DCLEAR using a k-mer based approach, is referred to as `DCLEAR (KRD)`, and the second, DCLEAR using a training-based approach, is referred to as `DCLEAR (WHD)`. Here we report results for `DCLEAR (KRD)` only, because `DCLEAR (WHD)` produced segmentation fault errors. We followed the tutorial available here: `https://ikwak2.github.io/tmphtml/Example_subchallenge2`.

To run `DCLEAR (KRD)`, we note that `DCLEAR` assumes characters across different sites share an alphabet. However, when we converted our simulated data to follow this convention, `DCLEAR` could not scale to handle alphabet sizes of approximately 250. Instead, we report results from running `DCLEAR` directly on the input character matrices, which were also directly provided to the other benchmarked methods. We followed this `DCLEAR (KRD)` vignette: `https://colab.research.google.com/gist/gongx030/653a76bffc4ee6ff41499e0026b6d39a/krd.ipynb`.

Given a fixed topology, we used LAML to estimate numeric parameters (branch lengths, missing data rates). We used the following command to do so:

```
python run_problin.py -t <fixed_topology> -c <character_matrix>
 -p <mutation_priors> -o <output> -v --delimiter comma --nInitials 1 --ultrametric
```

S2.2.2 Leveraging the Statistical Model in Distance-Based Approaches

One of our methodological contributions in this paper is a new probabilistic model of CRISPR/Cas9 lineage tracing. In this section we elaborate on the applications of this new statistical model.

Using the statistical model, we can estimate maximum likelihood distances between any two sequences. To explore how maximum likelihood distances under our statistical model compare to other ways of computing distances, we compute a distance matrix using several different approaches and run the widely-used distance-based method Neighbor Joining (NJ). This comparison is run on simulated data under all model conditions (varying missing data type composition).

We explain each pipeline in greater detail below:

1. wHD (weighted Hamming Distance): Standard Hamming distance is modified by provided weights. These weights are negative log transformed from the prior transition probabilities for each site's alphabet. wHD is recommended within the Cassiopeia implementation of NJ [**?**, 24].
   `cas.solver.NeighborJoiningSolver(dissimilarity_function=`
   `cas.solver.dissimilarity.weighted_hamming_distance, add_root=True)`

2. modified-AC (modified Allelic Coupling distance): Standard allelic coupling distance [50] is modified here by consideration of the missing state. This distance matrix is passed to PHYLIP-NJ to construct a tree for each sample. We use the `get_wAC.py` script to calculate the distance matrix.

3. MLpair: Given an input character matrix, we use LAML to estimate maximum likelihood distances for each rooted pair of leaves. After building this distance matrix, we run PHYLIP-NJ to construct a tree for each sample. We use the `split_pairwise.py` script to split the input distance matrix into pairs, and compute the maximum likelihood distances for each rooted pair.

4. MLall: Running LAML in the default way, we estimate maximum likelihood distances over all pairs of leaves at once, exploring tree topology space to find a maximum likelihood tree.

S2.3 The KP-tracer data

We benchmark LAML on KP-tracer, a recently published dynamic lineage tracing experiment in mouse models of metastatic lung adenocarcinoma [50]. The data come with character matrices for multiple samples, each has its own per-site alphabet hyperparameters $\mathbf{q}^{(k)}$.

To evaluate tree topology, we use the largest tumor dataset, `3724_NT_T1_All`, which consists of $21,108$ cells and 9 target sites. Originally, Yang et al. published a phylogeny using the Cassiopeia-Hybrid (Cass-hybrid) method [50]. Later, Sashittal et al. [38] published another phylogeny using Startle-NNI that yielded a more parsimonious explanation of the metastatic process compared to the published Cass-hybrid phylogeny. We use the migration cost metric, which we discuss below Supplementary Section S2.1.2. To run LAML, we first remove cells with identical sequences from the original character matrix, resulting in $1,461$ unique sequences. Then we prune the published Startle-NNI tree to this set of $1,461$ sequences and use it as starting tree. We search for the tree that optimizes the log-likelihood of the sequence data using the provided hyperparameters. We run topology search in LAML for 450 iterations to optimize for log-likelihood, which is improved from -12288 in the Startle-NNI tree to -12208 in the LAML tree. Finally, we add the duplicated sequences to the tree as polytomies.

To evaluate branch length estimation, we use 6 other samples of KP-tracer that have a medium-high number of target sites (29 to 73) and moderate number cells (29 to 294); these are: 3432_NT_T1, 3435_NT_T4, 3520_NT_T1, 3703_NT_T1, 3703_NT_T2, and 3703_NT_T3. For these 6 selected samples, we use the published tree topologies (estimated by Cassiopeia) and estimate branch lengths using LAML (here after referred to as dML), and compare our estimate to that of maximum parsimony (here after referred to as dMP).

S2.4 The intMEMOIR data

The MEMOIR [16] technology encodes DNA barcodes, and performs lineage tracing by accumulating variable edits in the encoded synthetic target array, and which allows readout by imaging. The intMEMOIR technology [10] permits extended recording and germline transmission, so that the true evolutionary history of sampled cells is recorded. In the intMEMOIR technology, only two edits are possible (either deleting or inverting the target region), so that only three states are possible at any of the sites. We should note this is in sharp contrast to the technology used in [7], where each site can have a unique, variably-sized alphabet of possible edits. In this dataset, we have 106 samples,

each with 10 sites, with a maximum of 40 cells and a minimum of 4 cells. The intMEMOIR dataset was used in the original DREAM challenge [19], making it a widely used benchmarking dataset. TiDeTree [39], which takes a Bayesian approach to inferring phylogenies, and was intended for use as a module in the BEAST package, was tested on the intMEMOIR dataset.

The TiDeTree [39] paper details that the topologies for all 106 intMEMOIR samples were jointly estimated, with shared scarring rates and population dynamic parameters across all trees, and assuming a molecular clock rate. They benchmark against AMbeRland, Guna lab, Cassiopeia, Jasper06, pRennert and RnlLabs. The author kindly made their resulting maximum clade credibility (MCC) trees available to us. We post-processed these trees as TiDeTree's supplement specified, by running BEAST's TreeAnnotator tool (with the "maxium clade credibility" option to create a summary tree.

We benchmarked LAML against Startle-NNI, Cassiopeia and TiDeTree using the following commands:

1. Startle-NNI:
```
python startle.py $seed_tree $character_matrix -e $mut_priors
--iterations ${iters} --output ${output_dir}/startlenni_tree_collapsed.newick
```

2. Cassiopeia: We ran Cassiopeia-Hybrid with a cut-off of 100 cells, so that we effectively report results for Cassiopeia-ILP on all intMEMOIR samples.
```
cas.solver.HybridSolver(cas.solver.VanillaGreedySolver(), cas.solver.
ILPSolver(convergence_time_limit=100, convergence_iteration_limit=1,
maximum_potential_graph_layer_size=1000), cell_cutoff=100)}
```

3. LAML:
```
python /n/fs/ragr-research/projects/problin/run_problin.py -t $tree
-c $msa -p $prior -o $prefix.txt -v --delimiter comma --nInitials 1
--topology_search --ultrametric --maxIters 2500 --parallel >
$prefix.log 2>&1
```

4. TiDeTree:
We adapted the provided example scripts `https://github.com/seidels/tidetree/blob/main/examples/`, using the default parameters and provided the sequence information. We installed BEAST 2.7 and used the following recommended compiled jar file on the adapted example XML file:
```
java -jar bin/tidetree.jar examples/adapted_example.xml
```

S2.5 Supplementary Results for Simulated Data

	Greedy method	**Startle-NNI**	**LAML**
RF distance to the true tree	0%	0%	**100%**
Weighted parsimony cost	0%	**84%**	16%
Log-likelihood	0%	0%	**100%**

Table S3: Robinson-Foulds error (RF), weighted maximum parsimony (WMP), and log likelihood (LLH) of the methods on simulated data, summarized over all five model conditions (250 replicates). For each pair of method and metric, the percentage of times the method scored the best according to that metric is shown.

S2.6 Benchmarking LAML

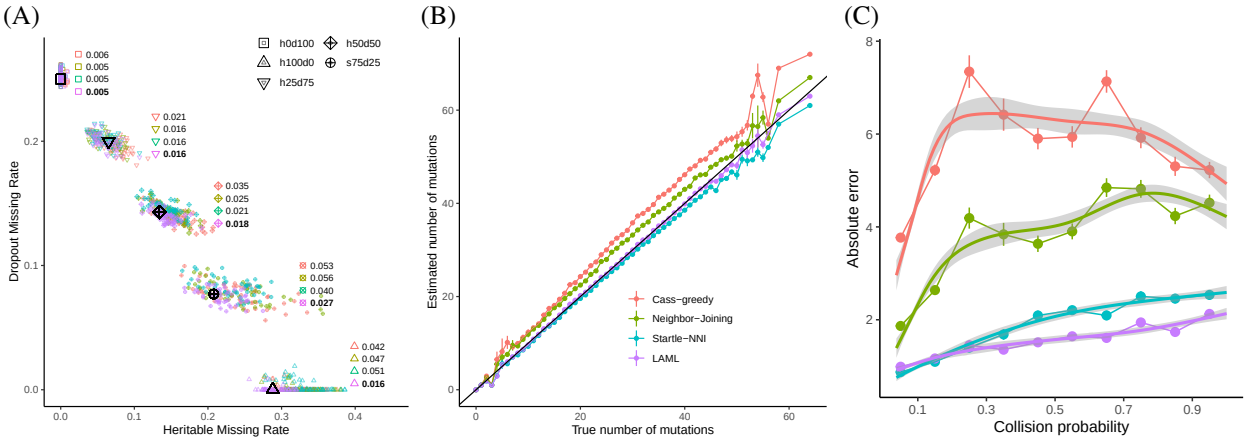

Figure S1: Supplementary results on the simulated data. (A) Estimation of ϕ and ν. True values of each model condition are shown in black, while the estimates using different tree topologies are shown in different colored, together with the root-mean-square-error (RMSE) of the estimates on each model condition. (B) Estimated versus true number of mutations. Each dot is the average estimated value by each method around one true value, shown with error bar. (C) Absolute error in estimating number of mutations versus collision probability of each target site. Collision probability (x-axis) is computed for each target site across all model condition. The x-axis is discretized into 10 bins and absolute error is averaged for each bin, shown with error bar. For (B) and (C), results are combined for all model conditions.

In Figure S2, we have simulated data results illustrating that LAML produces the most likely tree, as well as relatively parsimonious trees. However, we should note that the most parsimonious tree (Startle-NNI) shows very different RF error (2). Thus, likelihood and RF are more correlated than parsimony and RF are, suggesting that likelihood may be a better metric for scoring trees.

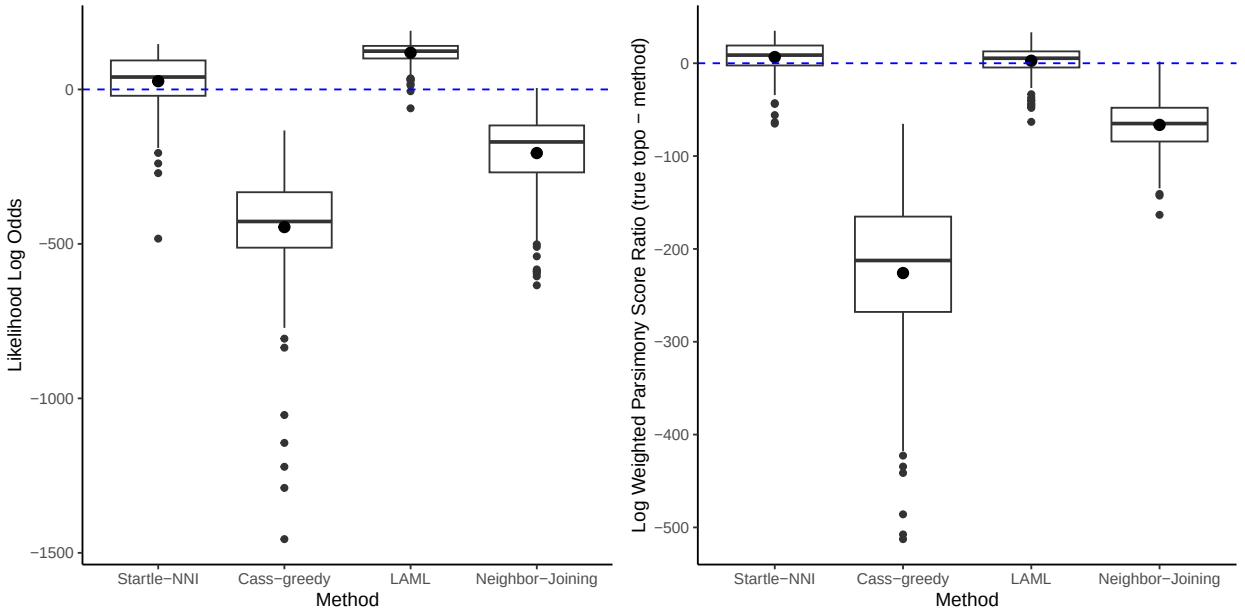

Figure S2: Comparison of the benchmarked methods based on two metrics: estimated log likelihood (calculated by LAML) to the true tree likelihood on simulated data (left); and weighted parsimony score (right).

In Figure S3, we show an additional comparison to Cassiopeia-ILP. Unfortunately, Cassiopeia-ILP was unable to

run on all of the simulated data, finishing on $44/250$ samples (h0d100: 13, h100d0: 8, h25d75: 15, h50d50: 4, h75d25: 4). Of those that finished with an error, 100 of the jobs did not finish running given 168 hours of runtime (7 days) and, 48 jobs ran out of memory (given 4G per job), 26 finished with a GurobiError, and 32 finished with an "IndexError" from within the Cassiopeia code.

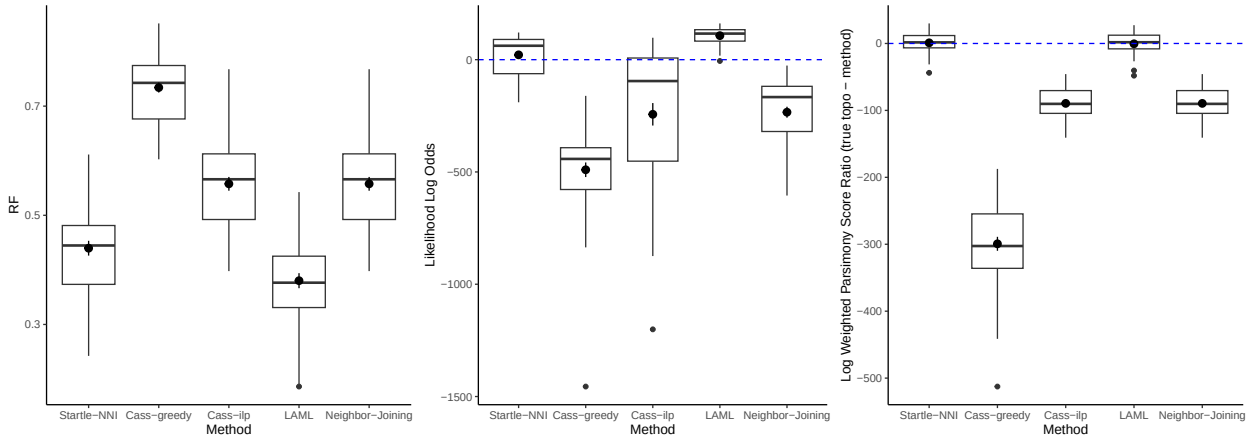

Figure S3: Comparison of the benchmarked methods based on two metrics: estimated log likelihood (calculated by LAML) to the true tree likelihood on simulated data (left); and weighted parsimony score (right).

S2.6.1 Branch Length Results

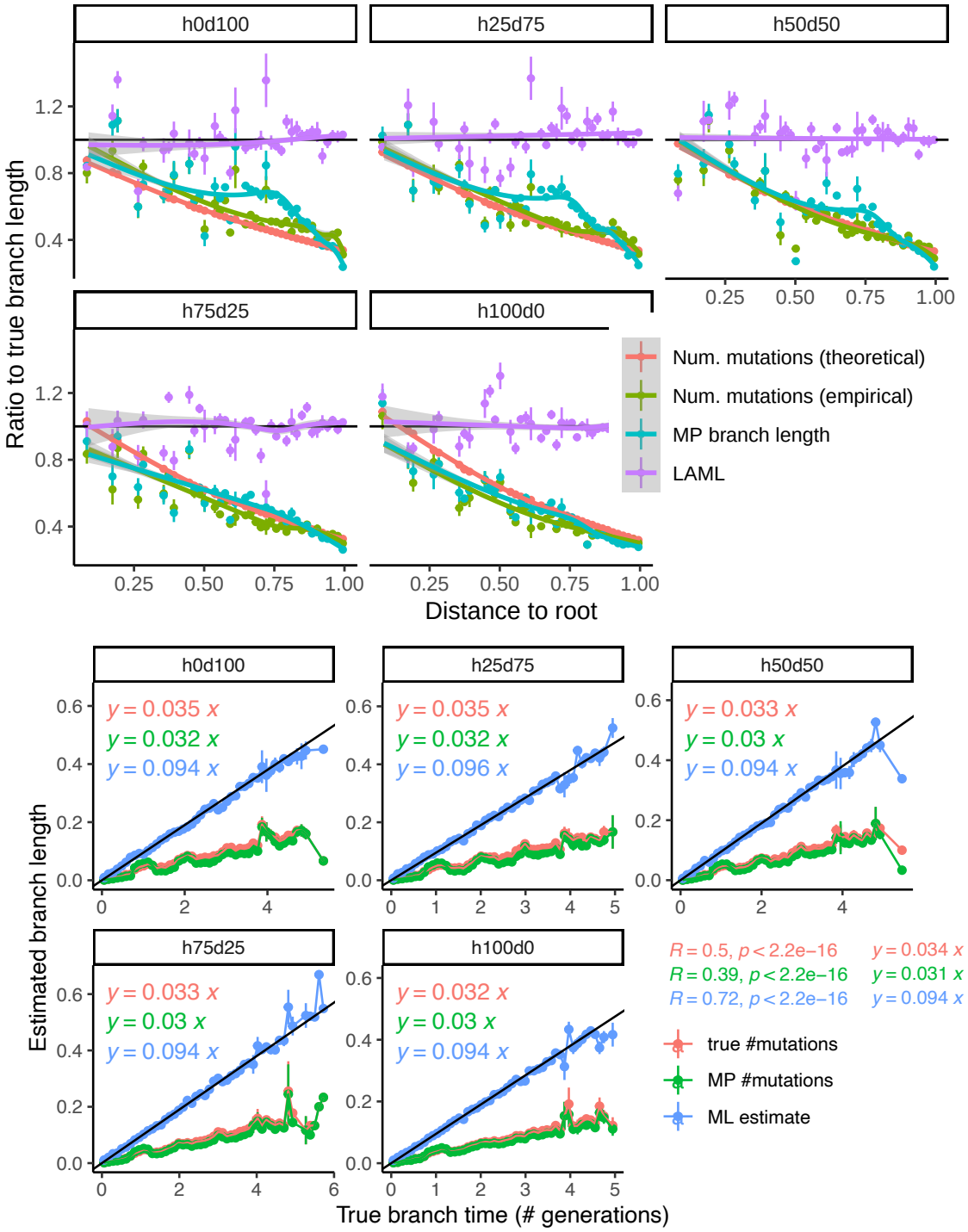

Figure S4: Branch length estimation analysis on true topologies. Top: The the ratio of estimated to true branch length versus distance to root. LAML's estimate (i.e. ML estimate, shown in blue) is compared to two alternatives: true # mutations and MP # mutations, which are the true number of mutations and the estimate by maximum parsimony respectively. The x-axis is discretized into 50 bins and the estimated branch length by each method is averaged for each bin, shown with error bar. The black line shows the perfect estimated/true = 1. Bottom: regression of the estimated versus true branch length. The x-axis is discretized into 50 bins and the estimated branch length by each method is averaged for each bin, shown with error bar. The black line shows the ideal fit using the true known mutation rate $\lambda = 0.095$. Correlation coefficient (R), p-value (p), and the equation of the linear regression are shown for each method.

S2.6.2 Runtime Results

Fig S5 illustrates the breakdown of the EM algorithm into the E-step and two cases of the M-step. This is on the model condition with zero heritable and 100 dropout (h0d100). We ran on varying number of cells (up to 5000 subsampled cells). The M-step non-convex case does not have a value for the largest case as those jobs did not complete in the 7 days of runtime provided.

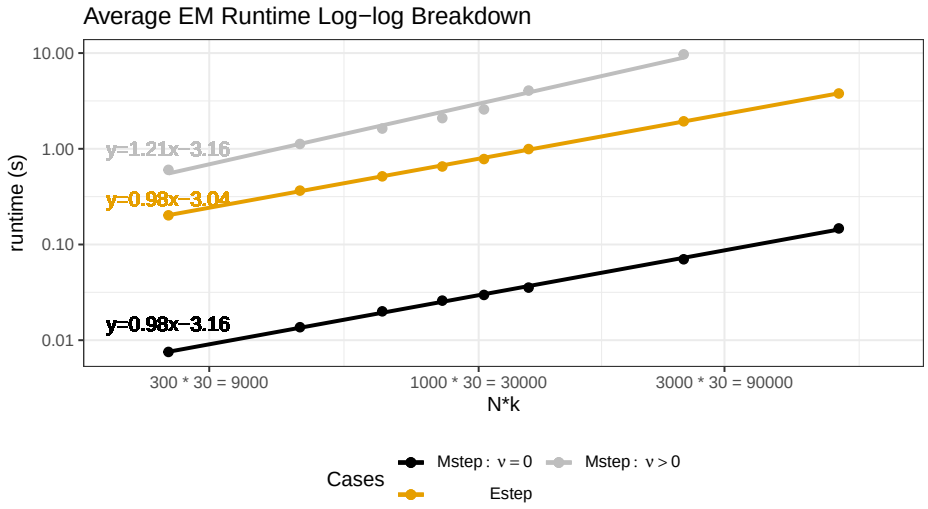

Figure S5: Runtime of EM algorithm, shown for E-step and M-step with the non-convex heritable and convex non-heritable cases (log-log plot). The slope of the fitted lines are all around 1, indicating that empirical runtime of these steps are all close linear.

We analyze the runtime efficiency of the EM algorithm first, then provide an exploration of the total topology search runtime. Fig. S6 illustrates how runtime and EM iterations changes as the number of cells increase.

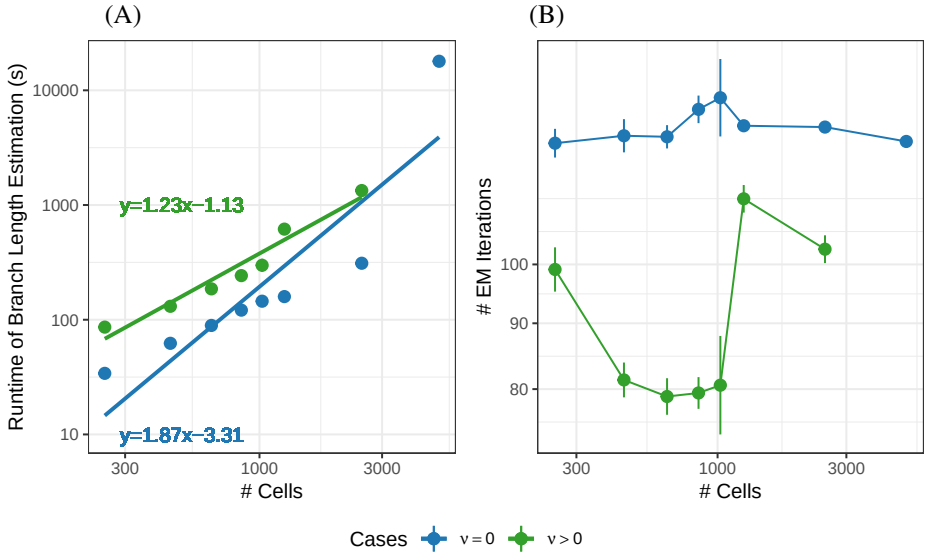

Figure S6: Empirical runtime (A) and number of EM iterations (B) of LAML on simulated data with varying number of cells (up to 5000 cells).

We provide a runtime comparison of topology search algorithms (Cassiopeia-Greedy, LAML, Startle-NNI and Cassiopeia-ILP). We run on simulated data sub-sampled to 250 cells, across all model conditions but only on replicates (≈ 40) for which Cass-ILP successfully completed.

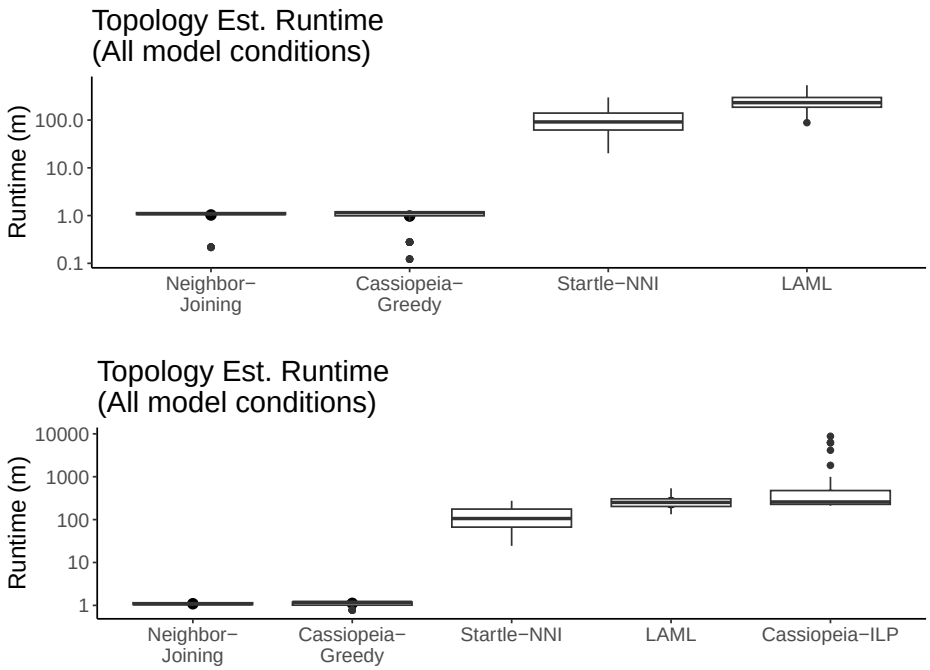

Figure S7: Average runtime comparison for topology estimation on simulated data. Runtime is shown in minutes in log scale.

S2.7 Supplementary Results for KP-Tracer

	Migration Types	LAML	Cass-hybrid	Startle-NNI
De-duplicated (1,461 cells)	All migrations	**99**	136	119
	Reseeding	**42**	68	56
Full Tree (21,108 cells)	All migrations	**1568**	1600	1783
	Reseeding	**191**	219	203

Table S4: Migration cost of the trees inferred by Cass-hybrid, Startle-NNI, and LAML on KP-tracer sample 3724_NT_T1_All.

S2.7.1 Migration Metrics Table

Metrics	LAML (T_{LAML})	Cassiopeia-Hybrid (T_{CassH})
Total Cost	836	959
Induced Anatomical Cost	125	159
Spatially-aware Migration Cost (Primary Tumor)	730	828
Constrained Parsimony	935	1095

Table S5: Metrics computed on the MG_{SC} migration graph with spatial clustering of all observed cells.

From Table S5, we can observe that the LAML tree topology always achieves a lower migration cost than the Cassiopeia-Hybrid tree does.

S2.7.2 Metastasis Analysis

With the knowledge that this experiment ran for 6 months, we scale our ultrametric tree, which had root-to-tip distance of 2.47 to be 6 with a scale factor of 2.425. To produce the bottom half of Figure 4(B), we begin with our tree with inferred anatomical labels on the internal nodes as described in the main text [13] and in Supplementary Section S2.1.2.

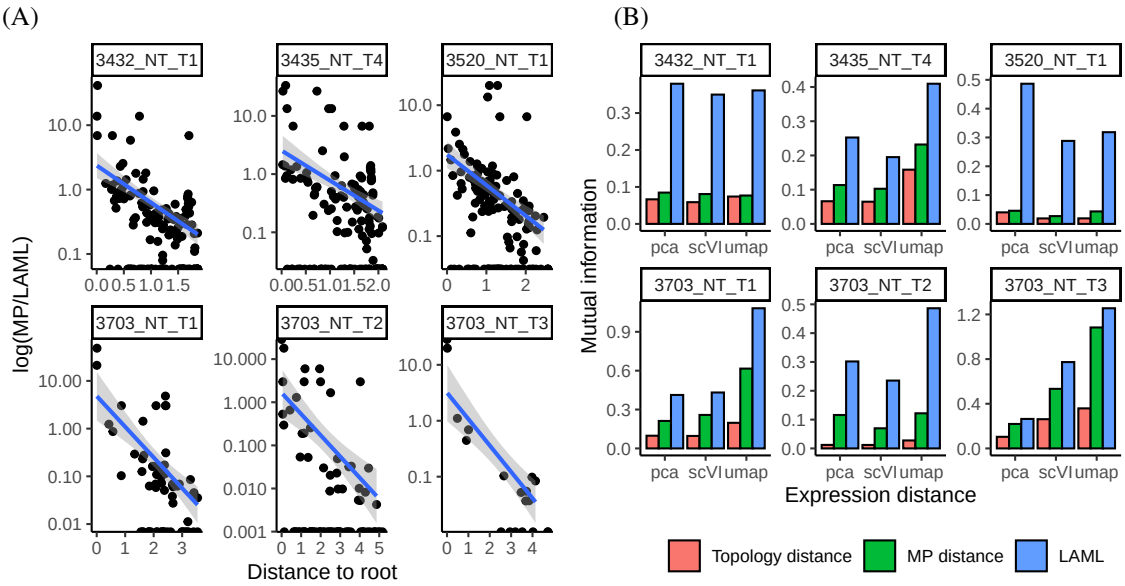

Figure S8: Branch length estimation by LAML (dML) and Maximum Parsimony (dMP) on the 6 selected samples of KP-tracer. (A) The ratio of each cell's branch length (i.e. the branch above the node) estimated by maximum parsimony (dMP) and maximum likelihood (dML) versus its distance to the root. The y-axis is shown in log-scale. (B) Mutual information of pairwise gene expression and phylogenetic distances. Gene expression is represented either by PCA, scVI, or UMAP. For these 6 samples, we use the published tree topology (estimated by Cassiopeia) and estimate branch lengths by LAML (ML distance), maximum parsimony (MP distance), or solely use the tree topology (Topology distance).

We annotate each edge $e = (u, v)$ based on the anatomical labels of the parent node u and child node v. We use edge, branch and lineage interchangeably in this section.

Notably, we are interested in transition edges, where one of the nodes is labeled with the primary tumor anatomical site, and the other node is labeled with one of the other 4 anatomical sites. Thus, we define four edge types: non-transition edge, primary to non-primary (metastasis), non-primary to primary (reseeding), and non-primary i to non-primary j (where $i \neq j$).

Next, we establish $2,470$ intervals ranging from 0.0 to 2.47 with a step size of 0.001. In each interval, we find the branches which overlap with this interval. Thus, we can compute the number of each branch type which overlap with this interval. Since the tree grows to include more cells in each cell generation, we normalize the counts of each edge type by the total number of branches overlapping this interval.

We can do an additional analysis to compute the expected number of unedited sites left at any given time point. We use the following equation: $\exp(-d/c) * 9$, where d is the time-scaled distance to root, and c is our scaling factor. It is useful to consider the expected number of unedited sites left at various time points (i.e. at the 3 month mark in the main paper), to give a sense of whether there is still phylogenetic signal left.

We also analyze the length of any branches which lie entirely within a single epoch in Figure S9. Note that the branches in the Late Metastasis epoch are much longer as expected due to the loss of phylogenetic signal and subsequent low tree resolution. The branch lengths observed in the the Primary Growth epoch are shorter, suggesting the phylogenetic signal is capturing cell divisions. Interestingly, the Metastasis epoch has the shortest branches < 1.0 month, perhaps suggesting a general increase in cell division rates. The Metastasis Burst window within the Metastasis epoch has even fewer outlier branch lengths.

S2.8 Supplementary Results for intMEMOIR

Comparison with TiDeTree on all the intMEMOIR samples. Note that the TiDeTree topologies shown here are the maximum credibility clade trees obtained from the TiDeTree authors, which show trees jointly estimated (sharing parameters across all samples). We ran Cassiopeia-Hybrid with a cut-off of 100 cells, so that we effectively report results for Cassiopeia-ILP on all intMEMOIR samples.

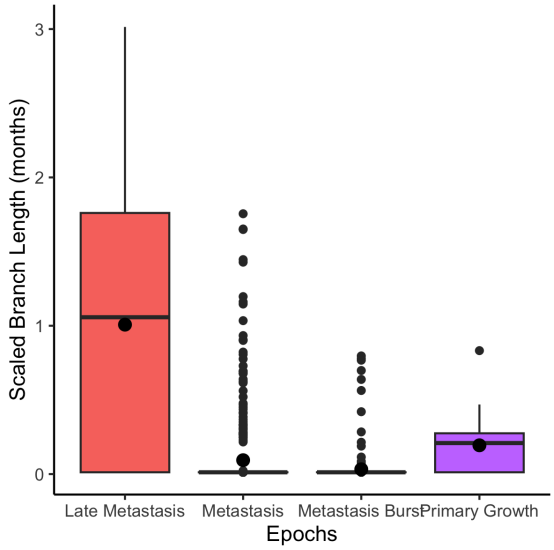

Figure S9: Length of branches lying entirely within each epoch.

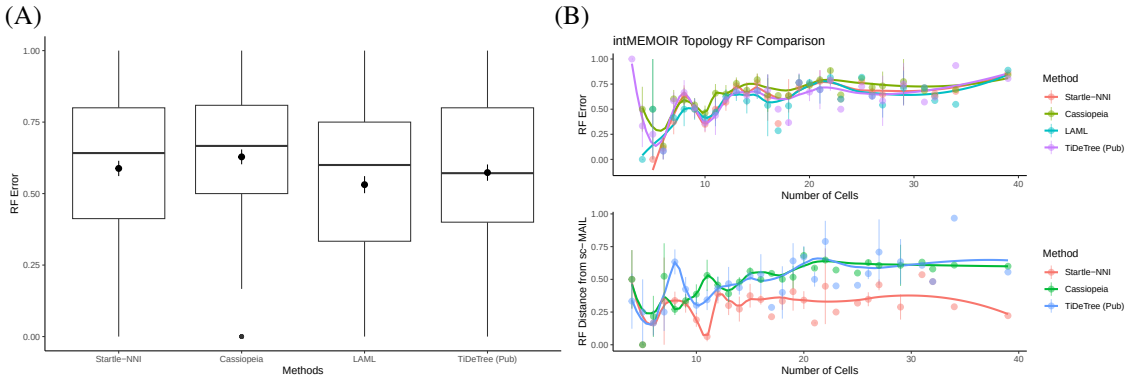

Figure S10: (A) Topology accuracy (RF) comparison between the true tree and trees estimated using LAML and the benchmarked methods, on all intMEMOIR samples. (B) RF distance plotted against the varying number of cells. The top plot shows RF distance from the true tree and the bottom plot shows RF distance from the LAML tree.

Runtime comparison on the intMEMOIR data (see Fig S11).

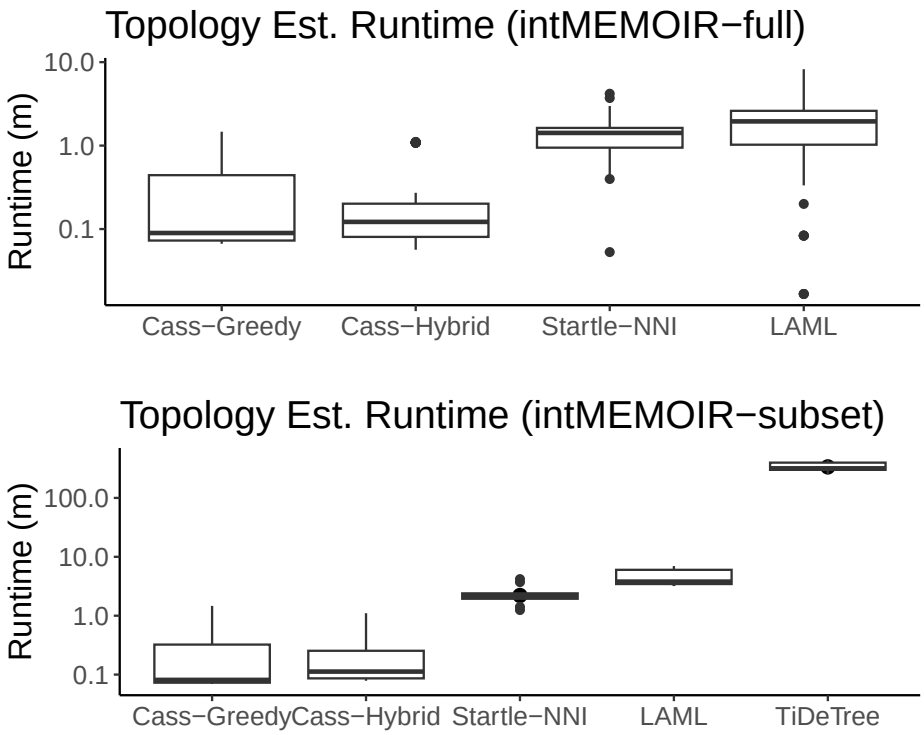

Figure S11: Topology estimation runtime comparison between LAML and TiDeTree on intMEMOIR. Note that the runtime of each methods' starting tree has also been added to that method's runtime.

S2.9 Distance-Based Exploration

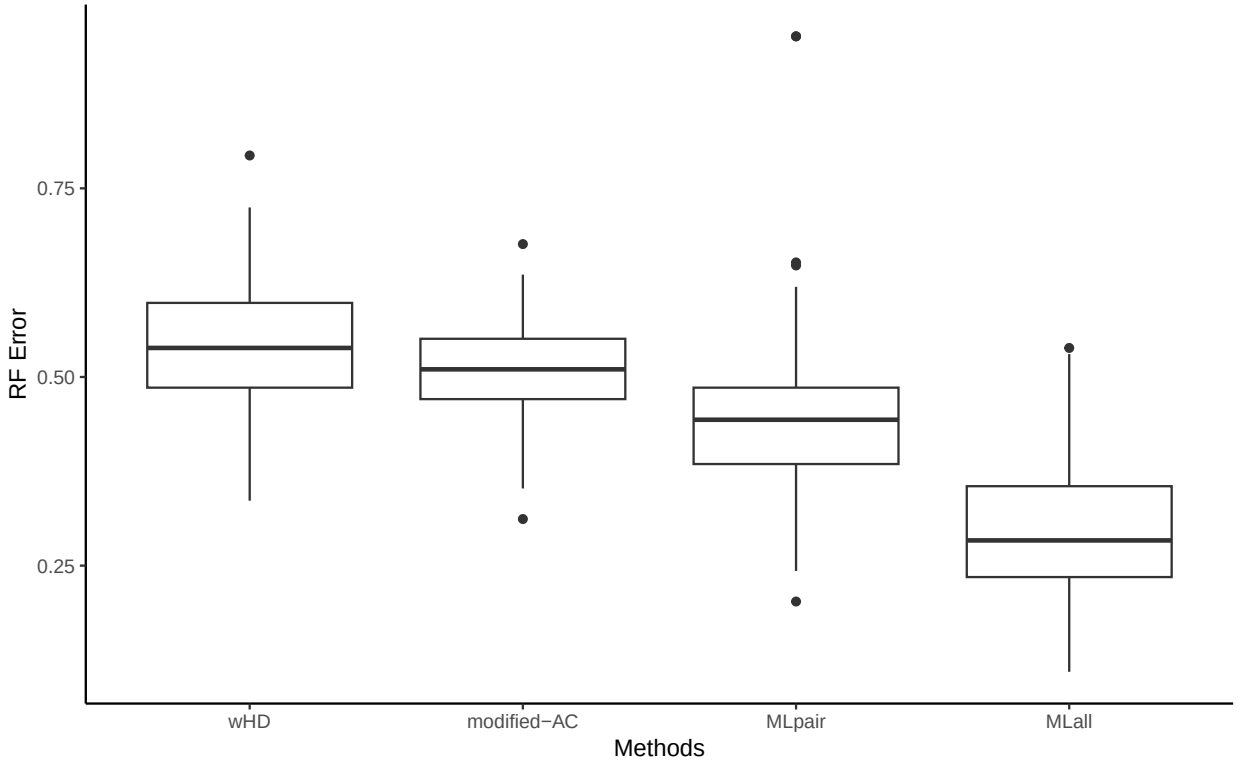

Figure S12: Comparison of different approaches to distance estimation. Two pipelines: (1) From the left, the first three pipelines first estimate rooted pairwise distances, before tree estimation using NJ. (2) MLall (LAML) finds the maximum likelihood distance matrix and tree topology over all pairs of leaves at once.

Figure S12 illustrates how distances estimated under this statistical model improve tree estimation. wHD, which does not take the missing state into consideration, produces trees with the highest RF error across all methods. This can be contrasted with the mAC, which does consider the missing state. The MLpair approach, which explicitly models the two types of missing data for each pair of cells, further improves the RF error. As expected, the MLall approach, which is able to optimize distances between sequences over all pairs of sequences, achieves the lowest RF error. This indicates the importance of estimating distances under a statistical model explicitly modeling the two types of missing data.

S2.10 Review of Lineage Tracing Technologies

Dynamic lineage tracing technologies begin by engineering one or more progenitor cells with artificial recording sites, then accumulating heritable insertions or deletions ("indels") [45] over cell generations. We classify dynamic lineage tracing technologies into 3 generations of technology. In the first generation of dynamic lineage tracing (GESTALT[27] and MEMOIR [16]), the recordable time is low because of limited number of target sites (no more than 10) and edit states (limited to 2: inversion and deletion). In the second generation (scGESTALT [34], LINNAEUS [43], ScarTrace [2], and intMEMOIR[10]), other information such as gene expression and spatial information, are recorded in addition to CRIPSR-Cas9 edit, enhancing tree inference [34]. However, the recordable tree depth remains low (only 6 cell generations) [27, 34], and there is a high error rate as edits are often removed or overwritten [43, 45]. In the third generation, novel dynamic lineage tracing technologies, such as CARLIN [5], iTracer [22], Chan et al. [7], Yang et al. [50] and TypeWriter [9], substantially improve recordable tree depth. We can further classify these technologies into two groups. The first group comprises technologies that induce editing at distinct time points. The second group comprises TypeWriter [9] and Chan et al. [7], the technologies focusing on increasing the number of cell generations by optimizing the number of editable sites and states. According to TypeWriter, the recordable tree depth can scale to 20 generations. The technology developed by Chan et al. is later demonstrated in [50], where number of target sites and edit states scale up to hundreds, and the recordable tree depth scales to 15 generations. Notably, the largest sample has 21,108 cells spanning 5-6 months of cell growth.

We should note that although the number of editable target sites affects the number of cell generations one can record over, that the quality of target sites affect how many of them turn out to be informative. Note also the distinction between continuous editing and inducible editing approaches: the depth of a given reconstructed cell lineage tree may span different numbers of cell generations and developmental time. **PMM is applicable to *all* of these lineage tracing technologies.**

Lineage Tracing Technology	Lineage Tree Depth	Time Recorded Over	# Target Sites	Approach	Size of Edit Library	Edits Control
ScarTrace	2	10 hours	8	barcode	uncontrolled	Cas9 injection
LINNAEUS	6	10 hours	16-32	CFP region	uncontrolled	Cas9 injection
CARLIN	3	/	10	barcode	uncontrolled	dox inducible
iTracer	5	100 hours	4 (Fig 6., Ext. Fig 10.)	barcode	uncontrolled	dox inducible
scGESTALT	5	/	9	barcode	uncontrolled	heat shock inducible
Chan et al.	12	6 days	9-45	barcode	uncontrolled	continuous
Yang et al.	15	6 months	10-30	barcode	uncontrolled	continuous
intMEMOIR	4	36 hours	no target sites 10 transgenes	barcode not CRISPR edits	uncontrolled	heat shock inducible
Typewriter	20	25 days	48 "tapes"	prime editing	by design: prototyped up to 16	dox inducible

Table S6: Summarizing key properties of the different lineage tracing technologies.

S2.11 Review of Other Computational Approaches

Computational methods for lineage tracing can be broadly categorized into three groups: maximum parsimony (MP), distance-based, and probabilistic modeling, with some methods exhibiting hybrid characteristics. Maximum parsimony methods [24, 38] rely on the heuristic criterion of maximum parsimony (MP). Cassiopeia [24], a highly regarded method in the DREAM challenge [19], infers the MP tree using a modified Camin-Sokal (C-S) model to leverage the non-modifiability property of CRISPR/Cas9 edits. Sashittal et al. [38] later refer to this evolutionary model as the "star homoplasy model" and develop Startle [38] to find the MP tree. However, due to the use of the heuristic MP criterion, neither Cassiopeia or Startle can adequately capture the stochastic nature of the CRISPR/Cas9 process. The second category encompasses distance-based methods, such as neighbor joining and triplet-based approaches. A critical challenge in these methods lies in computing pairwise distances, which are essential for generating the distance matrix (for neighbor joining) or triplet topologies (for triplet-based methods). Recent research [**?**] has shown that accurate computation of distances can be achieved with theoretical guarantees, but only when a probabilistic model is assumed. The last category comprises methods that formulate lineage tracing as a statistical inference problem. Bayesian [39] and maximum likelihood [15, 27, 51] methods fall within this category. Many of these methods model a specific set of lineage tracing technologies and do not generalize well to other technologies. For example, GAPML [15] introduced a probabilistic model tailored for the scGESTALT technology, focusing on single-barcode evolution. LiNTIMaT [51] can only be used for the second generation of lineage tracing technologies, where there are only two edit states, and requires expression data as input. TiDeTree [39] is designed more generally for barcode-based technologies, but their

model assumes sites are independent and *identically distributed*, limited its application to the handful of technologies having all target sites shared a same set of edit states.