PHYLO2VEC: A VECTOR REPRESENTATION FOR BINARY TREES

Matthew Penn[1*], **Neil Scheidwasser**[2*], **Mark P. Khurana**[2], **Christl A. Donnelly**[2,3] **& Samir Bhatt** [1,3*]
University of Oxford[1], University of Copenhagen[2], Imperial College London[3], equal contribution[*]
{matthew.penn@st-annes, christl.donnelly@stats}.ox.ac.uk
{neil.clow, mark.khurana, samir.bhatt}@sund.ku.dk

ABSTRACT

Binary phylogenetic trees inferred from biological data are central to understanding the shared evolutionary history of organisms. Inferring the placement of latent nodes in a tree by any optimality criterion (e.g., maximum likelihood) is an NP-hard problem, propelling the development of myriad heuristic approaches. Yet, these heuristics often lack a systematic means of uniformly sampling random trees or effectively exploring a tree space that grows factorially, which are crucial to optimisation problems such as machine learning. Accordingly, we present Phylo2Vec, a new parsimonious representation of a phylogenetic tree. Phylo2Vec maps any binary tree with n leaves to an integer vector of length n. We prove that Phylo2Vec is both well-defined and bijective to the space of phylogenetic trees. The advantages of Phylo2Vec are twofold: i) easy uniform sampling of binary trees and ii) systematic ability to traverse tree space in very large or small jumps. As a proof of concept, we use Phylo2Vec for maximum likelihood inference on five real-world datasets and show that a simple hill climbing-based optimisation efficiently traverses the vastness of tree space from a random to an optimal tree.

1 INTRODUCTION

Phylogenetic trees are integral to multiple research domains in biology, including evolution (Morlon et al., 2010), conservation (Rolland et al., 2011), and epidemiology, where they allow us to better understand infectious disease transmission dynamics (Ypma et al., 2013; Faria et al., 2021).

Although basic data structures such as arrays or linked lists can be used to describe phylogenetic trees, the most common representation is the Newick format (Felsenstein, 2004), which characterises a tree through a string of nested parentheses. Comparing trees using this format is nontrivial, and sampling random Newick strings is difficult. Alternative, bijective representations do exist, such as pair matching (Diaconis & Holmes, 1998), polynomial representations (Liu et al., 2022), F-matrices (Kim et al., 2020) and the compact bijective ladderized vector (Voznica et al., 2022). In many of these representations, a key focus has been the polynomial-time computation of the distance between any two trees (to measure similarity). However, systematic methods to efficiently sample random trees or change tree topology with respect to an objective function have been somewhat neglected. In particular, creating sampling schemes around standard tree arrangements (e.g., subtree prune and regraft (SPR); cf. Yang (2014)) is cumbersome.

To overcome these limitations, we introduce Phylo2Vec, a new representation for any binary tree. In this framework, a binary tree can be completely described by a single integer vector $\boldsymbol{v}$ of dimension $k+1$, where $n = k+1$ is the number of leaves in the tree. The only constraint on $\boldsymbol{v}$ is $v_0 = 0$ and $v_i \leq 2(i-1)$ for all $i \in \{1, ..., k\}$. To demonstrate its utility, we apply Phylo2Vec to several phylogenetic learning problems, where the task is to find an optimal tree with respect to a tree likelihood-based loss function, given a set of genetic sequences. While state-of-the-art frameworks for phylogenetic inference typically rely on search heuristics based on deterministic tree arrangements, Phylo2Vec makes the first steps to a more systematic criterion for optimisation.

2 PHYLO2VEC

For a given binary tree with $n = k + 1$ leaves, we can build its Phylo2Vec representation as an integer vector $\boldsymbol{v}$ of length $k + 1$ where:

$$v_0 = 0 \text{ and } v_i \in 0, 1, ..., 2(i-1) \; \forall i \in \{1, ..., k\} \tag{1}$$

It is easy to see from Equation 1 that there are $2i - 1$ entries for any index i. Therefore, the number of possible vectors, and hence trees, is $\prod_{i=1}^{k}(2i-1)$, which is the number of possible rooted binary trees (Cavalli-Sforza & Edwards, 1967). To build a tree from $\boldsymbol{v}$, two assumptions are necessary:

- **Assumption 1.** The $k + 1$ leaves are initially labelled left to right as $0, 1, ..., k$.
- **Assumption 2.** Each node can only "see" nodes to its left. Thus, node i can see nodes $\{0, 1, 2..., i-1\}$ and initially labels them as $\{0, 1, 2..., i-1\}$.

From these assumptions, we can build a tree by parsing $\boldsymbol{v}$ from right to left, starting with v_k. Suppose we have reached v_i. If node i has been processed, or if it cannot see a node with label v_{i-1}, we move to entry v_{i-1}. Otherwise, if node i can see a node j with label v_i, then we merge this node with node j. Importantly, node j will be unprocessed, and so we consider this new node to also be node j. However, all nodes to the right of node i can "see" this merger and assign the new node the smallest label that they have not yet used. All other nodes cannot "see" this merger, thus the new node is labelled in the same way as node j. Node i is now considered processed, so we go back to the rightmost entry of $\boldsymbol{v}$ and begin a new iteration. The algorithm ends when we reach v_0.

We use a $(k+2) \times (k+2)$ "view matrix" L to keep track of each node's labellings. L_{ij} provides the label that node i assigns to node j for $i = 0, 1, ..., k$ and $j = 0, ..., k$. Row $k + 1$ (which is the final row as we are zero-indexing) provides the "correct" view of the process where all nodes are visible and is useful for labelling internal nodes.

Algorithm 1 Recovering an ultrametric tree from Phylo2Vec vector $\boldsymbol{v}$. The output matrix rows consist of two child nodes followed by their parent node (third column).

Input $v \in \mathbb{V}$ ▷ Phylo2Vec vector $\boldsymbol{v}$
Result $M \in \mathbb{Z}^{k \times 3}$ ▷ Matrix of child nodes and their parent
$\pi \in \{0, 1\}^{k+1}$ ▷ Which nodes have been processed
$M = [\quad]$ ▷ Empty matrix to add the parent and two children
$L \in \mathbb{Z}^{(k+2)\times(k+2)}$ with $L_{ij} = j\mathbb{I}_{\{i>j\}}$ ▷ Create *lower triangular* labels matrix
for $i \in \{0, \ldots, k\}$ **do**
  j = k ▷ We begin at the rightmost node
  while $\pi_j = 0$ or $|\{x : v_j = L_{jx}\}| = 0$ **do**
    $j = j - 1$ ▷ Move leftwards until processable node reached
  end while
  Find x such that $v_j = L_{jx}$ ▷ Find node which node j merges with
  Append row $[\max(L_{k+1,\cdot}) + 1, L_{k+1,j}, L_{k+1,x}]$ to M ▷ Add merger to M
  Set $L_{yx} = \max(L_{y,\cdot})$ for $y > j$ ▷ Update labels
  $\pi_j = 1$ ▷ Node j now processed
end for

The complete algorithm is described in Algorithm 1, with a detailed example in Figure S1. We show in Appendix C that Phylo2Vec is bijective and well-defined. The tree construction algorithm defined in Algorithm 1 runs in quadratic space and time in the most simple and unoptimised case. Savings can be made through streamed processing of the view matrix, ordering, and vectorisation. A basic `NumPy` version runs in milliseconds for $n = 1000$ taxa on a modern CPU.

3 EXPERIMENTS

To demonstrate the utility of Phylo2Vec, we apply our new representation to the common problem of phylogenetic inference. Among the possible approaches to effectively infer the optimal phylogenetic tree, we focus on maximum likelihood estimation (MLE) using molecular sequence data.

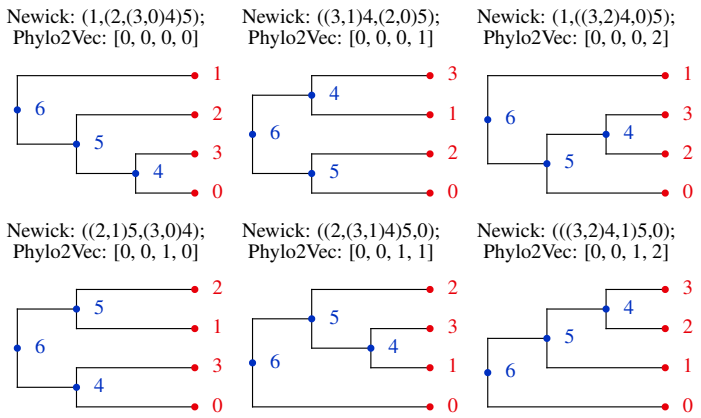


Figure 1: A subset of possible trees with $n = 4$ leaves/taxa along with both Newick and Phylo2Vec formats specifying the topology. Leaf and internal nodes are coloured in red and blue, respectively.

This parametric approach builds on a Markov model of DNA evolution (cf. Appendix B) and typically relies on Felsenstein's tree likelihood (Felsenstein, 1983) as an objective function. We apply Phylo2Vec to a corpus of five standard datasets (Table S1), which spans across different biological entities, taxa, and genetic sequence sizes.

For each dataset, we optimise tree topology using Phylo2Vec via a hill climbing-based procedure (Algorithm 3). At each step, branch lengths and evolution parameters are optimised using `RAxML-NG` (Kozlov et al., 2019). As the distance between two trees is not symmetric with respect to the labelling of the trees by Phylo2Vec (cf. Appendix C), optimisation can be inefficient and easily fall into local minima. A simple solution to mitigating this asymmetry is reordering taxon/leaf labels using level-order traversal, where $v_i < i$ for all $i > 0$. The advantage of carrying out our hill climbing scheme on these ordered trees is that it removes the secondary effects of changing an element of $\boldsymbol{v}$ which can occur by the processing order dramatically shifting. That is, if only the value of v_i is changed, the nodes in $\{1, ..., k\}/\{i\}$ will still be processed in reverse order. This prevents our algorithm from getting stuck in local minima, as more parts of the tree can be easily edited.

Figure 2a shows the optimisation results for four of the datasets described in Table S1. Starting from 10 random trees, we generally achieved the same minimal loss without being trapped in local optima. This is comparable to state-of-the-art software that also searches topological space (Stamatakis, 2014; Minh et al., 2020). For each dataset, a vanishingly small number of optimisations were needed to reach a minimum (e.g., 10^5 steps in the M501 dataset with 29 taxa = $\sim 8e^{36}$ possible trees). In the Zika dataset, two runs converged at a loss slightly (0.07%) greater than the minimum from the other eight runs. The resultant trees from these minima show that we get trapped in these suboptimal minima due to rooting issues, preventing single changes in v from finding a better optimum. This highlights that our algorithm is attempting to solve a more difficult problem by searching the space of rooted trees rather than unrooted trees. Due to the pulley principle (Felsenstein, 2004) (cf. Appendix B for details), all rootings of an unrooted tree have the same negative log-likelihood and therefore no paths between rooted trees exist to aid our optimisation algorithm.

Subsequently, we describe the negative log-likelihood paths for a small dataset (Yeast; 8 taxa) in Figure 2b using the above-mentioned optimisation pipeline. We observe a large plateau of many trees with similar likelihoods, along with a much smaller set of trees with increasing likelihood. Across several runs, whether starting from a random tree or the worst possible tree, our algorithm rapidly converges to the true tree presented in (Rokas et al., 2003). Approximately 100 likelihood evaluations were needed – a very small fraction of the total number of trees.

Finally, we show that our formulation of $\boldsymbol{v}$ naturally creates a measure of distance between trees. For any two Phylo2Vec trees $\boldsymbol{v}$ and $\boldsymbol{w}$, a Hamming distance can be defined as $\mu = \sum_{i=0}^{k} \mathbb{I}_{v_i \neq w_i}$. We observe a nonlinear correspondence between the Phylo2Vec distance and other commonly used phylogenetic tree distances (Figure S2). Modifying several indexes in v results in significant jumps across tree space, leading to new trees that are very dissimilar. As a result, common tree distances saturate as we increase the number of changes in v (St. John, 2017). However, we note that small changes in v_i can also readily correspond to very minor topological changes.

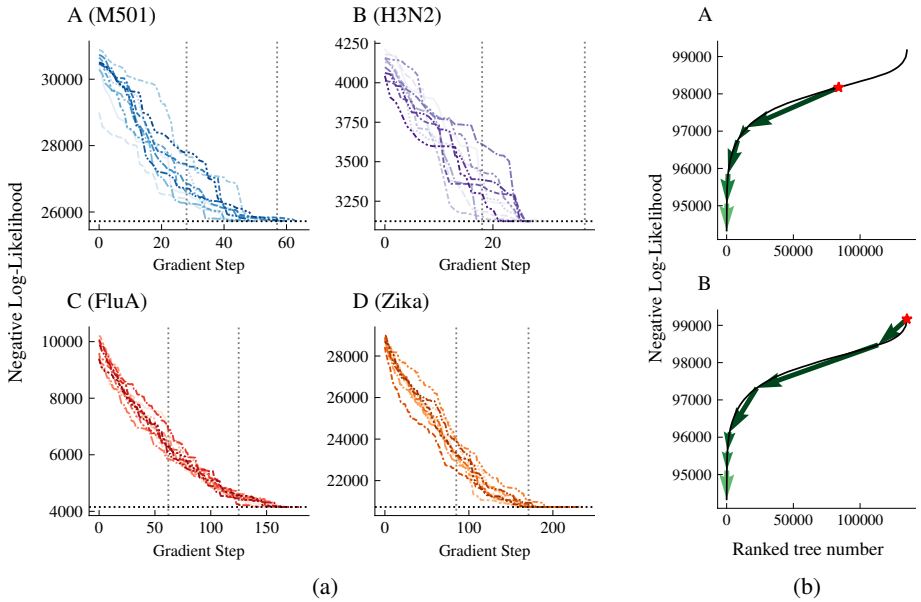


Figure 2: Tree topology optimisation using Phylo2Vec. (a) Phylo2Vec-based likelihood optimisation results for four datasets described in Table S1. The horizontal and vertical lines indicate local minima and epochs (i.e., one pass through every single v_i), respectively. (b) Negative log-likelihood path drawn from all possible trees of the Yeast dataset. A and B respectively show the path to the minimum from a random tree and the worst possible tree.

4 Discussion

We present Phylo2Vec, a parsimonious representation for phylogenetic trees whose validity extends to any binary tree. This representation facilitates tree sampling, distance calculation, and allows the formation of a simple algorithm for phylogenetic optimisation. Following from trends in phylogenetics, Phylo2Vec should be integrated within other phylogenetic libraries (e.g., `Beagle` (Ayres et al., 2012)) to facilitate its use. We have not yet considered Bayesian inference, but this is likely a useful application of Phylo2Vec, as random walks can be trivially implemented (cf. Figure S2).

Other more efficient and complex optimisation schemes can also be developed. Our optimisation algorithm only makes changes at a single index at a time, but far larger changes could be considered to more radically explore tree space (St. John, 2017). By construction, we have ensured that Phylo2Vec can be differentiable through transforming v into a matrix $W \in \mathbb{R}^{0,1}$. Via this transform, a gradient descent-based optimisation framework is theoretically possible, but its particulars remain to be developed. Similarly, we expect Phlyo2Vec representations to be applied in Monte Carlo tree search (MCTS) frameworks which may explore tree space more efficiently. The simplicity of Phylo2Vec's formulation means that it can be used in well-established machine learning paradigms such as self-supervised learning from large existing tree libraries (e.g., Piel et al. (2000)).

The primary limitations of Phylo2Vec are its quadratic complexity in time and memory and its focus on rooted trees, as most inference is conducted on unrooted trees. Whereas the reordering scheme allows Algorithm 1 to be processed in linear time, the rooting aspect can make topological changes difficult. Thus, Phylo2Vec-based optimisation schemes might not be able to evade local optima that could be escaped by unrooted SPR move.

In conclusion, Phylo2Vec represents an effort to introduce new representations of binary trees that are more parsimonious and easy to work with than existing representations. We believe this framework is among the first where questions such as how different two trees are, how we can sample uniformly from tree space, or how we can infer the best tree according to some objective function are all possible. An implementation is available at: `https://github.com/Neclow/phylo2vec`.

REFERENCES

Daniel L Ayres, Aaron Darling, Derrick J Zwickl, Peter Beerli, Mark T Holder, Paul O Lewis, John P Huelsenbeck, Fredrik Ronquist, David L Swofford, Michael P Cummings, et al. BEAGLE: an application programming interface and high-performance computing library for statistical phylogenetics. *Syst. Biol.*, 61(1):170–173, 2012.

L L Cavalli-Sforza and A W Edwards. Phylogenetic analysis. models and estimation procedures. *Am. J. Hum. Genet.*, 19(3 Pt 1):233–257, 1967.

Persi W Diaconis and Susan P Holmes. Matchings and phylogenetic trees. *PNAS*, 95(25):14600–14602, 1998.

Nuno R Faria, Thomas A Mellan, Charles Whittaker, Ingra M Claro, Darlan da S Candido, Swapnil Mishra, Myuki AE Crispim, Flavia CS Sales, Iwona Hawryluk, John T McCrone, et al. Genomics and epidemiology of the P. 1 SARS-CoV-2 lineage in Manaus, Brazil. *Science*, 372(6544):815–821, 2021.

Joseph Felsenstein. Statistical inference of phylogenies. *J R Stat Soc Ser A*, 146(3):246–262, 1983.

Joseph Felsenstein. *Inferring phylogenies*, volume 2. Sinauer associates Sunderland, MA, 2004.

Mathieu Fourment and Aaron E Darling. Evaluating probabilistic programming and fast variational bayesian inference in phylogenetics. *PeerJ*, 7:e8272, 2019.

J R Garey, T J Near, M R Nonnemacher, and S A Nadler. Molecular evidence for acanthocephala as a subtaxon of rotifera. *J. Mol. Evol.*, 43(3):287–292, 1996.

Jaime Huerta-Cepas, François Serra, and Peer Bork. ETE 3: reconstruction, analysis, and visualization of phylogenomic data. *Mol. Biol. Evol.*, 33(6):1635–1638, 2016.

Jaehee Kim, Noah A Rosenberg, and Julia A Palacios. Distance metrics for ranked evolutionary trees. *PNAS*, 117(46):28876–28886, 2020.

Alexey M Kozlov, Diego Darriba, Tomáš Flouri, Benoit Morel, and Alexandros Stamatakis. Raxml-ng: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics*, 35(21):4453–4455, 2019.

Pengyu Liu, Priscila Biller, Matthew Gould, and Caroline Colijn. Analyzing phylogenetic trees with a tree lattice coordinate system and a graph polynomial. *Syst. Biol.*, 71(6):1378–1390, 2022.

Bui Quang Minh, Heiko A Schmidt, Olga Chernomor, Dominik Schrempf, Michael D Woodhams, Arndt von Haeseler, and Robert Lanfear. IQ-TREE 2: New models and efficient methods for phylogenetic inference in the genomic era. *Mol. Biol. Evol.*, 37(5):1530–1534, 2020.

Hélène Morlon, Matthew D. Potts, and Joshua B. Plotkin. Inferring the dynamics of diversification: A coalescent approach. *PLoS Biol.*, 8(9), 2010.

Lam-Tung Nguyen, Heiko A Schmidt, Arndt Von Haeseler, and Bui Quang Minh. Iq-tree: a fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Mol. Biol. Evol.*, 32(1):268–274, 2015.

Emmanuel Paradis, Julien Claude, and Korbinian Strimmer. APE: analyses of phylogenetics and evolution in R language. *Bioinformatics*, 20(2):289–290, 2004.

William H Piel, Michael Donoghue, Mike Sanderson, and L Netherlands. Treebase: a database of phylogenetic information. In *Proceedings of the 2nd International Workshop of Species*, volume 2000, 2000.

Sebastien Roch. A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 3(1):92–94, 2006.

F James Rohlf. Numbering binary trees with labeled terminal vertices. *Bull. Math. Biol.*, 45(1): 33–40, 1983.

Antonis Rokas, Barry L Williams, Nicole King, and Sean B Carroll. Genome-scale approaches to resolving incongruence in molecular phylogenies. *Nature*, 425(6960):798–804, 2003.

Jonathan Rolland, Marc W. Cadotte, Jonathan Davies, Vincent Devictor, Sebastien Lavergne, Nicolas Mouquet, Sandrine Pavoine, Ana Rodrigues, Wilfried Thuiller, Laure Turcati, and et al. Using phylogenies in conservation: New perspectives. *Biol. Lett.*, 8(5):692–694, 2011.

Pavel Sagulenko, Vadim Puller, and Richard A Neher. TreeTime: Maximum-likelihood phylodynamic analysis. *Virus Evol.*, 4(1):vex042, 2018.

Michael J Sanderson, Michelle M McMahon, and Mike Steel. Terraces in phylogenetic tree space. *Science*, 333(6041):448–450, 2011.

Klaus Peter Schliep. phangorn: phylogenetic analysis in r. *Bioinformatics*, 27(4):592–593, 2011.

Katherine St. John. Review paper: The shape of phylogenetic treespace. *Syst. Biol.*, 66(1):e83–e94, January 2017.

Alexandros Stamatakis. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014.

Jeremy G Sumner, Peter D Jarvis, Jesús Fernández-Sánchez, Bodie T Kaine, Michael D Woodhams, and Barbara R Holland. Is the general time-reversible model bad for molecular phylogenetics? *Syst. Biol.*, 61(6):1069–1074, 2012.

Jakub Voznica, Anna Zhukova, Veronika Boskova, E Saulnier, F Lemoine, Mathieu Moslonka-Lefebvre, and Olivier Gascuel. Deep learning from phylogenies to uncover the epidemiological dynamics of outbreaks. *Nat. Commun.*, 13(1):1–14, 2022.

Ziheng Yang. *Molecular evolution: a statistical approach*. Oxford University Press, 2014.

Rolf J Ypma, W Marijn van Ballegooijen, and Jacco Wallinga. Relating phylogenetic trees to transmission trees of infectious disease outbreaks. *Genetics*, 195(3):1055–1062, 2013.

A APPENDIX

In this appendix, we present a rigorous mathematical framework summarising the results in this paper. We also detail further algorithms that may be useful in the implementation of Phylo2Vec.

A BACKGROUND

BASIC TERMINOLOGY

Formally, a phylogenetic tree is typically a binary tree where terminal nodes (or leaves) correspond to observable taxa (e.g., present-day organisms for which data exists). Conversely, internal (or parental) nodes constitute common ancestors of their children. They are latent variables which must be inferred. To measure dissimilarity between a parent and its children, each parent-children edge is weighted by a *branch length*.

The number of rooted labelled binary trees with $(k+1)$ leaves has been shown to be $(2k-1)(2k-3)...3 \times 1$ (Felsenstein, 1983; Cavalli-Sforza & Edwards, 1967). The total number of nodes in an unrooted tree is $2k$, and is $2k+1$ in a rooted tree.

RELATED WORK

Phylogenetic inference A fundamental problem in phylogenetics is phylogenetic inference, the task of reconstructing an evolutionary tree given observations from different taxa. A plethora of data sources can be used as input, including genetic sequences (e.g., DNA), sensory data (e.g., animal vocalisations), behaviour (e.g., kinematics), and morphology (e.g., skeletal or body shapes).

To effectively infer phylogenetic reconstruction, various optimisation-based approaches have been proposed. We focus on maximum likelihood estimation (MLE). This parametric approach builds on a discrete Markov model of evolution and typically relies on Felsenstein's tree likelihood (Felsenstein, 1983) as an objective function. The optimal tree is searched using a custom search heuristic based on different tree arrangements, including *nearest neighbour interchange* (swapping neighbouring branches), *subtree pruning and regrafting* (cutting and pasting a subtree at a different location), and *tree bisection and reconnection* (cutting and rejoining two parts of a tree at another location). Popular software packages for MLE include `IQ-TREE` (Nguyen et al., 2015) and `RAxML-NG` (Kozlov et al., 2019).

Although widely used, tree search using any optimality criteria (including MLE) is NP-hard (Roch, 2006). Another critical challenge is the size of the tree space: for a tree with n leaves, there are $(2n-3)(2n-5)...5 \times 3 \times 1$ possible binary trees. Lastly, optimisation-based approaches often face a jagged "loss" landscape containing many trees with the same criterion score (Sanderson et al., 2011).

Phylogenetic tree representation The most common format for representing phylogenetic trees is the Newick format, where tree topology is encoded as a string of nested parentheses. Despite its popularity, a string format is not suitable as input for optimisation or other inferential methods based on random sampling. Alternative representations have been developed, the earliest to our knowledge being integer assignment of labelled terminal vertices (Rohlf, 1983) where every possible tree of k leaves is assigned a unique integer that identifies that tree. Extending this concept, Diaconis and Holmes (Diaconis & Holmes, 1998) created a pair matching representation of integers that bijects to the space of trees. More recently, Kim *et al.* (Kim et al. (2020)) introduced a matrix representation used to compute distance metrics. Liu *et al.* (Liu et al. (2022)) proposed to describe phylogenies as a graph polynomial and also utilise their framework for distance-based inference. Lastly, Voznica *et al.* (Voznica et al. (2022)) has introduced the Compact Bijective Ladderized Vector (CBLV) to encode simulated trees as real-valued vectors. For a tree with n leaves and m internal nodes, CBLV provides a representation of size $n+m$.

B LIKELIHOOD

BINARY TREES WITH GENETIC DATA

Throughout this paper, we assume that each leaf of the tree is associated with some sequence data, allowing for phylogenetic inference on the tree topology. We label the leaves $\{0, ..., k\}$ and represent the data assigned to leaf i as D_i. We will use $\mathcal{D}$ to refer to the ordered set of data $(D_0, D_1, ..., D_k)$.

As we consider phylogenetic examples throughout this paper, each D_i is a vector of bases $D_i \in \{A, C, G, T\}^d$, where d is the number of bases in our set. We shall assume that d is independent of i. Without loss of generality uracil and thymine are interchangeable.

MODELLING EVOLUTION (ROOTED TREES)

Consider first the case of rooted trees, which provide a more natural link to evolutionary models. We suppose that the root represents a species that is the defined as the most recent common ancestor (MRCA) of the species represented by the leaf nodes.

We suppose that each non-root node i evolved from its parent node after some time b_i. In this process of evolution, it is possible for the genotype to change according to some parametric probabilistic model that must be specified. Almost all parametric models of evolution are continuous time Markov chains. In the examples of this paper, two models are considered – the Jukes-Cantor (JC) model, which assumes that the four bases an equal equilibrium base frequency, and the General Time Reversible (GTR) model, though other models are possible, including non-reversible models (Sumner et al., 2012).

Both of these models assume that genetic mutation can be modelled as a continuous time Markov process on the four possible DNA states. They also assume that each base mutates independently, that is, independence between sites.

The parameters in both models can be fit to the data using the package `RaxML-NG` (Kozlov et al., 2019) in the likelihood optimisation process. Note that JC is a special case of GTR with the π_i all equal to 0.25 and the rate parameters all equal to 4μ.

Given the Q matrix, the genotypes of node i and its parent j, and the branch length b_i, it is possible to calculate the probability of node i evolving from node j after time b_i. This can be done from the transition matrix $P(b_i)$ of the relevant Markov Chain at time b_i.

Hence, for the remainder of this appendix, we define a function $q(\tilde{D}, D, b)$ which is (for the relevant gene transition model) the probability that a species with genotype D evolved from a species with genotype $\tilde{D}$ after time b.

TREE LIKELIHOOD (ROOTED TREES)

In general, the species corresponding to internal nodes are no longer available for study, and so only the genotypes of the species corresponding to leaf nodes are known.

However, one can still calculate the probability that a given evolutionary tree $\mathcal{T}$ with branch lengths $\boldsymbol{b}$ would have given rise to the leaf genotypes through the methods of Felsenstein (Felsenstein, 1983).

For each tree $\mathcal{T}$, define a $(2k) \times 2$ matrix P where each row contains two nodes that are joined by an edge of $\mathcal{T}$. We suppose the first column contains the parent node and the second column contains the child node.

Define $\mathcal{G}$ to be the space of possible values of the unknown genotypes $G = (D_{k+1}, ..., D_{2k})$, the likelihood L is

$$L(\mathcal{T}, \boldsymbol{b}) = \sum_{G \in \mathcal{G}} \left(\prod_{i=1}^{2k} q(D_{P_{i1}}, D_{P_{i2}}, b_{P_{i2}}) \right) \tag{2}$$

This expression appears intractable but, following (Felsenstein, 1983), it is possible to calculate it efficiently via the pruning algorithm (Felsenstein, 2004) which involves a post order tree traversal.

TREE LIKELIHOOD (UNROOTED TREES)

Because the Markov processes used by both the JC and GTR models are symmetric, the evolution order induced by the root of a rooted tree is not necessary to calculate a sensible likelihood function. Indeed, note that by the time-reversibility of the Markov Chain

$$q(\tilde{D}, D, b) = q(D, \tilde{D}, b) \tag{3}$$

for all $\tilde{D}$, D, b. Thus, if P' contains the list of edge endpoints in any order and if the edge corresponding to row i of P' has length b'_i,

$$L(\mathcal{T}, \boldsymbol{b}') = \left(\sum_{G' \in \mathcal{G}'} \prod_{i=1}^{2k-1} q(D_{P'_{i1}}, D_{P'_{i2}}, b'_i) \right) \tag{4}$$

where $\mathcal{G}'$ is the space of genotypes $G' = (D_{k+1}, ..., D_{2k})$. Note the $2k$ becomes a $2k-1$ as the tree now has one fewer edge.

CONSISTENCY OF ROOTED AND UNROOTED TREE LIKELIHOODS

A perhaps surprising property of the gene mutation processes used in this paper is that removing the root from a rooted tree does not change the likelihood.

To show this, it is first important to define the way in which the root is removed from a rooted tree. Suppose that, in a rooted tree, the two children of the root are x and y. To "unroot" the tree, we remove the edges connecting x and y to the root and add in a new edge connecting x and y. This edge is given length $b_x + b_y$. We shall use P' here to refer to the matrix containing the edge pairs of the unrooted tree and b' to refer to their lengths. Moreover, we shall use $\mathcal{T}'$ to refer to the unrooted tree.

We use $S = \{A, C, G, T\}^d$ to be the space of possible values of D_{2k}. Moreover, we suppose without loss of generality that the $(2k-1)^{\text{th}}$ and $(2k)^{\text{th}}$ rows of P correspond to the edges connecting the root, and that the $(2k-1)^{\text{th}}$ row of P' corresponds to the edge joining x and y (so that $b'_{2k-1} = b_x + b_y$). We can write the rooted likelihood as

$$\begin{aligned}
L(\mathcal{T}, \boldsymbol{b}) &= \sum_{G \in \mathcal{G}} \left(\prod_{i=1}^{2k} q(D_{P_{i1}}, D_{P_{i2}}, b_{P_{i2}}) \right) && (5)\\
&= \sum_{G' \in \mathcal{G}'} \sum_{D_{2k} \in S} \left(\prod_{i=1}^{2k} q(D_{P_{i1}}, D_{P_{i2}}, b_{P_{i2}}) \right) && (6)\\
&= \sum_{G' \in \mathcal{G}'} \Bigg[\left(\prod_{i=1}^{2k-2} q(D_{P_{i1}}, D_{P_{i2}}, b_{P_{i2}}) \right) \times && (7)\\
&\quad \sum_{D_{2k} \in S} q(D_{2k}, D_x, b_x) q(D_{2k}, D_y, b_y) \Bigg]
\end{aligned}$$

Now, using reversibility and the Markov property, we know that

$$\begin{aligned}
&\sum_{D_{2k} \in S} q(D_{2k}, D_x, b_x) q(D_{2k}, D_y, b_y)\\
&= \sum_{D_{2k} \in S} q(D_{2k}, D_x, b_x) q(D_y, D_{2k}, b_y) && (8)\\
&= q(D_y, D_x, b_x + b_y) && (9)
\end{aligned}$$

as, if one considers D_x to be the initial value of the chain, then the second line is simply conditioning over all possible values at an intermediate time b_x. Thus,

$$L(\mathcal{T}, \boldsymbol{b}) = \sum_{G' \in \mathcal{G}'} \left(\prod_{i=1}^{2k-2} q(D_{P_{i1}}, D_{P_{i2}}, b_{P_{i2}}) \times \right.$$

$$\left. q(D_y, D_x, b_x + b_y) \right) \tag{10}$$

$$= \sum_{G' \in \mathcal{G}'} \left(\prod_{i=1}^{2k-1} q(D_{P'_{i1}}, D_{P'_{i2}}, b'_i) \right) \tag{11}$$

$$= L(\mathcal{T}', \boldsymbol{b}') \tag{12}$$

as required.

C PHYLO2VEC

VECTOR REPRESENTATION

As described in the main text, Phylo2Vec is a way to represent binary trees with $n = k + 1$ leaves using a single vector, $\boldsymbol{v}$ of dimension $n = k + 1$ that is simply constrained by

$$v_0 = 0 \quad \text{and} \quad v_i \in 0, 1, ..., 2(i-1) \quad \forall i \in \{1, ..., k\} \tag{13}$$

The construction of the tree from this representation and the bijectivity between $\boldsymbol{v}$ and the space of trees are covered in the main text.

IS ALGORITHM 1 WELL DEFINED?

It is important to show that Algorithm 1 constructing a tree from $\boldsymbol{v}$ is well-defined - that is, that each node (except node 0) is eventually processed so that a tree is formed.

For the proofs presented in this and the following section, note that the term "node i" will be used to refer to the node with original label i, where it is assumed that when two nodes merge, the node with the smaller label persists (so that, for example, if nodes 1 and 2 merge, then the parent node will still be called node 1).

Lemma 1. *Suppose that node i can see a node which it has labelled v_i. Then, no nodes $j < i$ will be processed until node i merges with node v_i*

Proof: Suppose that the node which node i has labelled as v_i had an original label of m, meaning $L_{im} = v_i$. The only way for L_{im} to change is if two nodes to the left of node i merge. However, this means that node i would be skipped in this step of the algorithm, which is impossible as node i is unprocessed and can see a node with label v_i. As all the nodes $\{i + 1, ..., k\}$ can only be processed once, we will eventually consider the entry v_i and at this stage, node i will merge with node v_i. This must happen before any entries v_j for $j < i$ are considered, as required.

Lemma 2. *Suppose that an unprocessed node i cannot see a node with label v_i. Then, $v_i > \max_j L_{ij}$*

Proof: Suppose that node i cannot see node v_i. Note that the set of node i's labels, $\{L_{ij} : j \in \{0, 1..., i-1\}\}$, has a maximum that increases by 1 every time two nodes to the left of node i merge, and remains constant otherwise. Thus, if $v_i \leq \max_j(L_{ij})$, then there must have been a point at which $v_i = L_{im}$ for some m. However, then node i could have seen a node with label v_i, and hence by Lemma 1, no nodes $q < i$ have been processed since this point. Thus, it must still be the case that $L_{im} = v_i$, which is a contradiction as we assumed that node i could not see node v_i. Hence, we have shown that $v_i > \max_j\{L_{ij}\}$.

Lemma 3. *Every node except for node 0 will be processed, i.e., the map from the set of $\boldsymbol{v}$ to the set of trees is well-defined*

Proof: Suppose that this lemma does not hold. We can only reach v_0 and terminate the algorithm if none of the other nodes can be processed. Define $\mathcal{U}$ to be the set of unprocessed nodes, and suppose that $j = \min(\mathcal{U})$, then, all entries $1, ..., j-1$ have been processed and hence $\max\{L_{jp} : p \in \{0, ..., j-1\}\} = 2(j-1)$ Thus, from Lemma 2, it is necessary that $v_j > 2(j-1)$ which gives the required contradiction.

BIJECTIVITY OF $\boldsymbol{v}$ TO THE SPACE OF ALL POSSIBLE TREES

We can show that our mapping from $\mathbb{V}$ to the space of trees is a bijection.

Lemma 4. *The mapping between the set of vectors $\boldsymbol{v} \in \mathbb{V}$ and the set of (topologically equivalent, labelled) trees is a bijection*

Proof: As, by construction, the number of possible vectors $\boldsymbol{v}$ and the number of trees are the same $(2(n-3)!!)$, it is simply necessary to show that this mapping is injective.

Suppose that two vectors $\boldsymbol{v}$ and $\boldsymbol{v}'$ result in the same tree, but $\boldsymbol{v} \neq \boldsymbol{v}'$. Define $\mathcal{I} := \{i : v_i \neq v'_i\}$, and suppose that the tree construction algorithm is run up to the first point where a node $i \in \mathcal{I}$ is processed in one of the two cases (so that, up to this point, the two trees are identical). Without loss of generality, suppose that it is processed in the case of $\boldsymbol{v}$, but not necessarily $\boldsymbol{v}'$.

Suppose that $v_i = j$, so that node i is joined to the node m satisfying $L_{im} = j$ (which exists as we have assumed that node i is indeed processed by $\boldsymbol{v}$). In order for $\boldsymbol{v}'$ to create a topologically equivalent tree, it is necessary for node i and node m to be joined in this tree before either of these nodes is joined with another node.

Because the two processes have been identical up to this point, the view matrix L' in the $\boldsymbol{v}'$ case satisfies $L = L'$. Thus, in particular, $L_{im} = L'_{im}$.

We consider two cases depending on whether or not node i can see a node with label v'_i. Firstly, if this is true, then node i will merge with this node in this step. As $L'_{im} \neq v'_i$, this node is not node m and hence the two trees will not be topologically equivalent. Conversely, if node i cannot see a node with label v'_i, then by Lemma 2, $v'_i > \max_j\{L_{ij}\}$ and hence, it can only merge with a node q after this node q has merged with some other node r, where $r, q < i$ (since the label of this node will need to be updated by node i). Hence, it cannot merge with node m before node m merges with another node, and so the tree cannot be topologically equivalent.

Hence, topological non-equivalence holds in both cases, and the map from $\boldsymbol{v}$ to the set of trees is therefore injective and therefore bijective.

INVERTING THE CONSTRUCTION

Often, binary trees are represented by a $k \times 3$ matrix M where each row gives a parent and its two children. It is therefore convenient to have an algorithm which goes from a more common tree representation to the $\boldsymbol{v}$ formulation. This is possible because, as proved in the previous section, our formulation gives a bijective mapping. Note that the matrix P used in the previous sections of this appendix can be easily constructed from M (and M can be easily constructed from P).

This method comes from attempting to construct the given tree according to the vector $\boldsymbol{v}$. We iteratively add nodes to the tree (initialising with the leaf nodes) and the edges that connect these nodes to their children (note: this is identical to the way in which a tree is constructed from $\boldsymbol{v}$ except that we use the language of "adding" a parent node rather than merging two child nodes for clarity).

As in the main text, we assume that the leaves are labelled from left to right. Here, we make the further assumption that a parent node is placed directly above its leftmost child. That is, if the nodes are in an (x, y) plane, a node with children at (x_1, y_1) and (x_2, y_2) will be placed at $(\min(x_1, x_2), y_3)$ for some $y_3 > \max(y_1, y_2)$. This will help to keep track of the progeny of the internal nodes.

When constructing the tree, it is important that these nodes are added in the correct order – while there exists a unique $\boldsymbol{v}$ for each tree, this relies on the tree being constructed according to the algorithm in the main text. To find this order, we require two lemmas.

Lemma 5. *Consider constructing a tree from a vector $\boldsymbol{v}$. Then, after m nodes have been processed, the graph is a forest with trees $T_0, ..., T_{k-m}$ (as at each merger, two trees are connected). For each tree T_i, define l_i to be the leaf node that the root of this tree is above. Then, the set of unprocessed nodes is $\{l_0, ..., l_{k-m}\}$*

Proof: This follows simply from noting that a node is only processed when it merges with a node to the left of it – that is, when it has a parent that is to the left of it. Thus, the set of $k - m + 1$ unprocessed nodes must be part of a tree where the root is above them.

Lemma 6. *Consider constructing a tree from $\boldsymbol{v}$. Suppose that S is the set of internal nodes (and the root) which will be added in the course of the algorithm. At the start of some iteration in the algorithm, define $\mathcal{N} \subseteq S$ to be the set of nodes that have not yet been added to the tree. Define a subset $\tilde{\mathcal{N}}$ to be the members of $\mathcal{N}$ such that the nodes that will be their children have already been added to a tree. Further, for each node in $n \in \tilde{\mathcal{N}}$, we define $r_n \in \{0, 1, 2..., k\}$ to be the leaf node which their rightmost child has been placed above (note that though these nodes have not been added, their children have). Then, the next node added is the node i satisfying*

$$i = \text{argmax}_{n \in \tilde{\mathcal{N}}}\{r_n\} \tag{14}$$

Proof: Clearly, $i \in \tilde{\mathcal{N}}$ as otherwise it cannot correctly be appended to the tree. Define

$$j = \text{argmax}_{n \in \tilde{\mathcal{N}}}\{r_n\} \tag{15}$$

Suppose for a contradiction that $i \neq j$. As two nodes cannot share a child, $r_i \neq r_j$ and so $r_i < r_j$. By Lemma 5, leaf r_j has not yet been processed. Moreover, node i will be placed by processing leaf r_i, as this is the only unprocessed leaf that is in the relevant tree.

Thus, it is necessary that node r_j is skipped in this iteration of the algorithm. Thus, node r_j cannot currently see a node with label v_{r_j}. Suppose that the node which r_j should eventually merge with (i.e. the leftmost child of node j) has label x. The only way that node r_j can change its view of the label of this node is if node x merges with another node. However, this is impossible, as then node x will not have the same parent as node r_j. Thus, node r_j must label x as v_{r_j} which is the required contradiction and proves Lemma 6.

Lemma 6 is powerful as it means that we can construct a tree from the matrix M in the same order as $\boldsymbol{v}$ would have constructed the tree. When we add in a node that merges some node above leaf i with some node above leaf $j > i$, we can ensure that the construction according to $\boldsymbol{v}$ would have completed the same step by imposing that the value of v_j is correct, and ensuring that any unprocessed nodes x such that $x > j$ would have been skipped. These conditions must be consistent and lead to a unique $\boldsymbol{v}$ by the bijectivity of the $\boldsymbol{v}$ construction to the space of trees. To meet the first of these conditions, it is helpful to note the following lemma:

Lemma 7. *Consider constructing a tree from $\boldsymbol{v}$. Suppose that exactly $m > 0$ nodes to the left of node j are processed before node j is processed. Then $v_j = j + m - 1$.*

Proof: Suppose that $v_j > j + m - 1$. When m nodes to the left of v_j have been processed, it has had to assign m new labels, meaning that it now sees nodes with labels in the set $\{0, ..., j - 1 + m\}$. Thus, it cannot see a node with label v_j, giving a contradiction (as it will therefore not be processed)

Conversely, suppose that $v_j < j + m - 1$ so that $v_j = j + m - 1 - s$ for some $s > 0$. If $s > m - 1$, then node j can initially see a node with label v_j, and hence will be processed before any nodes to the left of it, contradicting $m > 0$. Otherwise, after s nodes to the left of j have been processed, node i will be able to see a node with label $j + m - 1 - s$, as this will be the next available label at the point that the sth node to the left of j is processed. Thus, v_i will be processed before the $(s+1)^{th}$ node to the left of it is processed, which is a contradiction as $s + 1 \leq m$. This completes the proof of Lemma 7.

This means that we can iteratively construct $\boldsymbol{v}$ by starting with a vector of all zeros. Then, every time a node j is skipped, we change the value of v_j to $\max(j, v_j + 1)$ so that, when this node is finally processed, the value of v_j will be correct by Lemma 7. If it is never skipped, then we simply set $v_j = i$ if the node it needs to merge with is above leaf node i.

The details of this algorithm are summarised in Algorithm 2. Note that we assume column 0 of M contains the parents while columns 1 and 2 contain the children. To aid the calculations, we keep

Algorithm 2 Inverting Tree Construction

Input M ▷ $k \times 3$ matrix describing tree edges
Result $\boldsymbol{v}$ ▷ vector describing tree
$\boldsymbol{v} = (0, 0, ..., 0)$ ▷ Initialise $\boldsymbol{v}$ (length $k+1$)
$\boldsymbol{a} = (0, 1, 2, ..., k)$ ▷ Initialise labels of nodes with no parent
$\boldsymbol{l} = (0, 1, 2, ..., k, -1, ..., -1)$ ▷ Initialise node positions (length $2k$)
while $\max(\boldsymbol{a}) \geq 0$ **do** ▷ while nodes with no parent
  $i = \text{argmax}\{\max(l_{M_{i1}}, l_{M_{i2}}) : a_{M_{i1}}, a_{M_{i2}} \geq 0\}$ ▷ Find edge to process by Lemma 6
  $j = \text{argmax}\{l_{M_{ij}} : j \in \{1, 2\}\}$ ▷ Finding the right hand node
  $v_x = \left(\mathbb{I}_{\{a_x \geq 0\}} \max(x, v_x + 1)\right) + \left(\mathbb{I}_{\{a_x < 0\}} v_x\right)$ for $x \in \{l_{M_{ij}} + 1, ..., k\}$ ▷ Updating skipped nodes
  $v_{l_{M_{ij}}} = \max(l_{M_{i,3-j}}, v_{l_{M_{ij}}})$ ▷ Update merged node
  $a_{l_{M_{ij}}} = -1$ ▷ Node above $l_{M_{ij}}$ now has a parent
  $a_{l_{M_{i,3-j}}} = M_{i0}$ ▷ Node above $l_{M_{i,3-j}}$ has a new label
  $l_{M_{i0}} = l_{M_{i,3-j}}$ ▷ Node M_{i0} now has a position
  Delete row i from M ▷ This row has now been processed
end while

track of a list $\boldsymbol{a}$ of the label (assigned by M) of any node without a parent currently above each leaf node (with $a_i = -1$ if the node is currently not placed on the tree). We also keep track of $\boldsymbol{l}$, which gives the positions of each node (with $l_i = -1$ if the node has not yet been placed). Then, we simply add nodes in turn, using Lemma 6 to order this process, and update $\boldsymbol{v}, \boldsymbol{a}$ and $\boldsymbol{l}$ at each step.

LABEL-ASYMMETRY OF $\boldsymbol{v}$-INDUCED DISTANCE

As discussed in the main text, $\boldsymbol{v}$ induces a natural distance function between trees – namely, that the distance between $\boldsymbol{v}$ and $\boldsymbol{w}$ is equal to $\sum_{i=0}^{k} \mathbb{I}_{v_i \neq w_i}$.

However, this distance function is dependent on the labels assigned to each leaf (and is hence label-asymmetric). A simple example of this can be found in the case of four leaves.

Consider the tree given by $\boldsymbol{v}_1 = (0, 1, 2)$. This is a ladder tree (that is, each internal node and the root is parent to at least one leaf node) with nodes in order $0, 1, \{2, 3\}$ (where the $\{2, 3\}$ is used to denote the fact that nodes 2 and 3 are from the same generation and so could be read in either order). This is distance 1 away from $\boldsymbol{v}_2 = (0, 1, 4)$, which is again a ladder tree with nodes now ordered as $3, 0, \{1, 2\}$. Thus, if distance were symmetric, then any ladder tree with ordered nodes $a, b, \{c, d\}$ would be distance 1 away from the ladder trees with ordered nodes $d, a, \{b, c\}$ and $c, a, \{b, d\}$. However, the ladder tree with ordered nodes $2, 0, \{1, 3\}$ is given by $\boldsymbol{v}_3 = (0, 2, 1)$, which is distance 2 away from $\boldsymbol{v}_1$. Thus, the distance function is not label-symmetric.

UNROOTED TREE EQUIVALENCE CLASSES

Noting from the main text that there are $(2k-3)!! = \prod_{i=0}^{k-2}(2i+1)$ unrooted trees with $k+1$ leaves, we can partition the space of trees with $k+1$ leaves into $(2k-3)!!$ equivalence classes $\{\mathcal{E}_i : i = 1, 2, ..., (2k-3)!!\}$ such that the removing the root from each of the trees in a given class $\mathcal{E}_i$ (according to the procedure Appendix B) results in the same unrooted tree. These equivalence classes all contain $(2k-1)$ trees (as, reversing the unrooting procedure, a root can be added onto each of the $(2k-1)$ edges of an unrooted tree).

From the previous work, Felsenstein's likelihood is constant for each equivalence class. This has two consequences when attempting to maximise the likelihood. Firstly, a global minimum in the space of rooted trees will not be unique, as any of the other trees in the same equivalence class will also give a maximal likelihood. Secondly, if $\boldsymbol{v}$ is a local maximum (that is, all $\boldsymbol{w}$ with one entry different to $\boldsymbol{v}$ give a lower likelihood value), equivalence classes provide a way to change to a new vector $\boldsymbol{u}$ without having to decrease the likelihood.

Similarly to the case of reordering, the set of equivalence classes of vectors that are distance 1 from $\boldsymbol{v}$ will not in general be the same as the set of vectors that are distance 1 from $\boldsymbol{u}$. One can either change equivalence class randomly or (unlike in the case of reordering), as there are only $2k-1$ members of each class, systematically iterate through them. Finding the vectors $\boldsymbol{w}$ corresponding to the trees in an equivalence class can again be done by constructing the tree matrix M, making the appropriate changes and then using the inversion algorithm to create a new vector representation

Our experiments have shown that changing $\boldsymbol{v}$ is effective in increasing algorithmic performance. However, we found that reordering provided somewhat better increase in performance, and therefore did not present this other switching method in the algorithm in the main text.

SAMPLING FROM ROOTED TREE EQUIVALENCE CLASSES

As referenced in the main text, we can use $\boldsymbol{v}$ to uniformly sample unrooted trees. This uses the fact that the set of rooted trees corresponding to the vectors

$$\boldsymbol{v}' = (0, v_1, ..., v_{k-1}, 2(k-1)) \quad \text{with} \tag{16}$$

$$v_i \in \{0, ..., 2(i-1)\} \quad \forall i \leq k-1 \tag{17}$$

contains exactly one tree from each equivalence class.

To prove this, consider any unrooted tree and introduce a root on the (unique) edge connected to the leaf node k. This creates a rooted tree that can be represented by some $\boldsymbol{v}$. In this rooted tree, there must be an edge connecting node k with the root, which can only happen if node k is the final node to be processed. We claim that $v_k = 2(k-1)$.

Suppose for a contradiction that $v_k \neq 2(k-1)$ so, necessarily, $v_k < 2(k-1)$. If $v_k < k$, then it will be processed first, and therefore will not be processed last (unless $k = 1$, in which case $v_k = 0 = 2(k-1)$ is guaranteed), giving a contradiction as required. Otherwise, we can write $v_k = k - 1 + m$ for $1 \leq m < k-1$. We know that at least one node is processed before node k in this case, and hence, by Lemma 7, we know that exactly $m < k-1$ nodes are processed before node k, so node k is the mth node to be processed, giving the required contradiction.

Note that it may appear that one could fix the final value of $\boldsymbol{v}$ to be any fixed integer. However, this is not the case – for example, $v_k = 0$ means that node k and 0 share a parent, which is not the case in all unrooted trees.

REORDERING INDICES

As discussed in the main text, the asymmetry that is intrinsic to this formulation can lead to difficulties when optimising, as certain nodes may get "stuck" in the wrong position in a tree.

To remedy this, we can "reorder" the labels of the tree using some perturbation σ on $\{0, 1, ..., k\}$. This is explained conceptually in the main text, though we provide a more rigorous framework here.

Consider some $\boldsymbol{v}$ that gives a tree $\mathcal{T}$. As before, we can create a $(2k) \times 2$ matrix P from $\boldsymbol{v}$ where each row of P gives respectively the parent and child of an edge in $\mathcal{T}$.

A reordered version of the tree (which we shall here denote as $\sigma(\mathcal{T})$, slightly abusing notation) should therefore give a matrix $\sigma(P)$ such that

$$\sigma(P)_{ij} = \begin{cases} P_{ij} & \text{if } P_{ij} > k \\ \sigma(P_{ij}) & \text{if } P_{ij} \leq k \end{cases} \tag{18}$$

where here $\sigma(P_{ij})$ is the new value of label P_{ij} under the perturbation σ (that is, it is actually the output of σ when given P_{ij} as an input!)

Note that the labels assigned to internal nodes are irrelevant as they have no data associated with them. Thus, extending the arguments of the likelihood function to include the dataset $\mathcal{D} = (D_0, ..., D_k)$ and defining $\sigma(\mathcal{D}) = (D_{\sigma^{-1}(0)}, ..., D_{\sigma^{-1}(k)})$ as in the main text, one can see that, for $M_{ij} \leq k$

$$\sigma(D)_{\sigma(P)_{ij}} = D_{\sigma^{-1}(\sigma(P_{ij}))} = D_{P_{ij}} \tag{19}$$

as required. Thus,

$$L(\mathcal{T}, \boldsymbol{b}, \mathcal{D}) = L(\mathcal{T}, \boldsymbol{b}, \mathcal{D}) \tag{20}$$

follows as no terms in the likelihood are changed.

Note that one can simply create the reordered $\boldsymbol{v}$ by using the previously defined inversion algorithm on the reordered $\sigma(M)$ constructed from $\sigma(P)$.

ORDERED TREES

As discussed in the main text, we say that a tree is *ordered* if

$$v_i < i \quad \forall i > 0 \tag{21}$$

We claim that any tree constructed from a Phylo2Vec vector $\boldsymbol{w}$ such that

$$\sum_{i=0}^{n} \mathbb{I}_{\{v_i \neq w_i\}} = 1 \tag{22}$$

can also be constructed by performing a single SPR move on the tree given by $\boldsymbol{v}$.

Define the tree created from $\boldsymbol{v}$ to be $\mathcal{T}$ and the tree created from $\boldsymbol{w}$ to be $\tilde{\mathcal{T}}$.

Suppose as before that the nodes are labelled from left to right and that a parent is placed above its leftmost child. Suppose further that it is the ith entry which differs between $\boldsymbol{v}$ and $\boldsymbol{w}$. The construction algorithms therefore run identically up to the point where node i is processed.

From Lemma 5, we know that at this point, the graph is a forest containing a tree T_i with root above node i. Moreover, as the set of unprocessed nodes is $\{0, 1, ..., i\}$, we can label the trees in the forest as $T_0, ..., T_i$ where the root of T_j is above leaf node j.

As nodes $i+1, ..., k$ have already been processed, T_i will only merge with another tree when node i is processed. As none of the active nodes will see this merger, this will not change the construction of the rest of the tree.

Hence, define $\hat{\mathcal{T}}$ to be the tree created if subtree rooted at the root of T_i is removed from $\mathcal{T}$. One can construct $\hat{\mathcal{T}}$ from the original $\mathcal{T}$ by simply removing the subtree rooted at the node of the lowest generation that is above node i. This is exactly a "prune" in SPR.

The only difference between $\hat{\mathcal{T}}$ and the final tree $\tilde{\mathcal{T}}$ is that the tree T_i will be joined to some edge in $\hat{\mathcal{T}}$ (by the addition of an extra node on this edge). This is exactly a "regraft" procedure in SPR.

Thus, we see that $\boldsymbol{w}$ is exactly one SPR move away from $\boldsymbol{v}$ as claimed.

We note that ordering also makes processing of v considerably simpler and changes the complexity of Algorithm 1 from quadratic to linear.

UNDERSTANDING $\boldsymbol{v}$ MOVES ON ORDERED TREES

Changing v_i means that we prune and regraft the subtree rooted at the oldest node that is above leaf node i (which may be leaf node i itself). This can be seen by noting that all other nodes in this subtree $i+1, ..., k$ have been processed by the point at which we process node i.

At this point in the algorithm, we have $i-1$ other trees with roots above the leaf nodes $0, ..., i-1$. Choosing $v_i = j$ for some $j \in \{0, ..., i-1\}$ will therefore regraft our subtree on the edge connecting the root of T_j to its parent.

Choosing $v_j = j - 1 + m$ means that we will regraft the subtree on the edge joining the node added when we process v_{j-m} to its parent. This can be seen from Lemma 7.

One can consider this more intuitively by supposing that the tree was constructed by iteratively adding pairs of children to a tree (starting with the root), rather than by merging leaf nodes (ending with the root). We add these nodes in an order such that this "additive" construction process is the reverse of our Phylo2Vec process. In this case, we can define the *age* of a node or an edge to be the "time" at which it first appears on the graph, where we increase our time by 1 at each addition.

Thus, the implication of our previous discussion is that our possible prune nodes are the oldest nodes above each leaf node (again, which may include some leaf nodes). Given such a node, the set of possible regraft edges is the set of edges which are older than that node.

Thus, our optimisation procedure works by propagating subtrees "up" the tree (that is, moving them further back in time). This directional nature of the movement means that for some optimisation problems, it may be necessary to re-root the tree. By choosing the deepest node as the outgroup in this re-rooted tree, we can reverse the propagation order and make further progress to towards finding the optimal tree.

D DATA

Table S1: Evaluation datasets, sorted by number of taxa.

Name	Type	# taxa	# bases
Yeast (Rokas et al., 2003) (Paradis et al., 2004)	Fungi	8	127,018
H3N2 (Sagulenko et al., 2018)	Virus	19	1,407
M501 (Garey et al., 1996)	Animal	29	2,520
FluA (Fourment & Darling, 2019)	Virus	69	987
Zika (Sagulenko et al., 2018)	Virus	86	10,807

E IMPLEMENTATION

Phylo2Vec definition, transforms and gradient-based optimisation methods were implemented in Python using `NumPy`. Tree manipulation scripts were written using `ete3` (Huerta-Cepas et al., 2016). Dataset construction was based on `phangorn` (Schliep, 2011) in R and `TreeTime` (Sagulenko et al., 2018) in Python. Maximum likelihood estimation was performed using `RAxML-NG` (Kozlov et al., 2019). An implementation is available at: `https://github.com/Neclow/phylo2vec`.

F TREE TOPOLOGY OPTIMISATION USING HILL-CLIMBING

Algorithm 3 Hill-climbing optimisation

Input $\boldsymbol{v} \in \mathcal{T}_n$ ▷ initialise with a random tree
Result $\boldsymbol{v}^* \in \mathcal{T}_n$ ▷ (local) optimal tree
$\ell_{\text{best}} = \ell(\boldsymbol{v})$ ▷ Initial best likelihood value
while $\max(G_{i=0,\dots k}) > 0$ **do** ▷ Continue iterating until local minimum
  Sample $i \in \{0, \dots k-1\}$
  Reorder(v) ▷ Reorder the labels
  $G_i = \nabla_{\text{discrete}}(\boldsymbol{v})_{i,j=\{0,\dots 2i\}}$ ▷ Search possible changes for i
  Define $\boldsymbol{j} = \text{argmax}(G_i)$ ▷ Find the best change
  change $\boldsymbol{v}_i \rightarrow \boldsymbol{j}$ ▷ Change $\boldsymbol{v}$
  $\ell_{\text{best}} = \ell(\boldsymbol{v})$
end while

G SUPPLEMENTARY FIGURES

Rules: Next pair: L[-1, **m**], L[-1, **n**]
Increment: column **m**, rows $\geq$ **n**

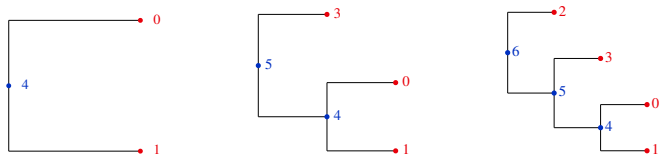
Step:	0	1	2	3
P:	[T, T, T, T]	[T, F, T, T]	[T, F, T, F]	
L:	$\begin{bmatrix} \mathbf{0} & 0 & 0 & 0 & 0 \\ \mathbf{0} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 1 & \mathbf{2} & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \end{bmatrix}$	$\begin{bmatrix} \mathbf{0} & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 \\ \mathbf{2} & 1 & 0 & 0 & 0 \\ \mathbf{3} & 1 & 2 & 0 & 0 \\ 4 & 1 & 2 & 3 & 0 \end{bmatrix}$	$\begin{bmatrix} \mathbf{0} & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 \\ \mathbf{2} & 1 & 0 & 0 & 0 \\ \mathbf{4} & 1 & 2 & 0 & 0 \\ 5 & 1 & 2 & 3 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 0 \\ 5 & 1 & 2 & 0 & 0 \\ 6 & 1 & 2 & 3 & 0 \end{bmatrix}$
	$\begin{array}{rcccc} & 0 & 0 & 2 & 3 \\ \leq & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{2} \\ \hline & T & T & F & F \\ \& & T & T & T & T \\ \hline & T & \mathbf{T} & F & F \end{array}$	$\begin{array}{rcccc} & 0 & 0 & 2 & 3 \\ \leq & \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} \\ \hline & T & T & T & T \\ \& & T & F & T & T \\ \hline & T & F & T & \mathbf{T} \end{array}$	$\begin{array}{rcccc} & 0 & 0 & 2 & 3 \\ \leq & \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{4} \\ \hline & T & T & T & T \\ \& & T & F & T & F \\ \hline & T & F & \mathbf{T} & F \end{array}$	
	n = **1** $\rightarrow$ v[n] = 0 L[n, 0] = 0 $\rightarrow$ m = **0**	n = **3** $\rightarrow$ v[n] = 3 L[n, 0] = 3 $\rightarrow$ m = **0**	n = **2** $\rightarrow$ v[n] = 2 L[n, 0] = 2 $\rightarrow$ m = **0**	
New pair:	**(0, 1)**	**(4, 3)**	**(5, 2)**	
Tree:				

Figure S1: Example of tree construction from Phylo2Vec with $\boldsymbol{v}$ = [0, 0, 2, 3]. P: boolean vector indicating whether a node has been processed or not. L: label matrix (cf. Algorithm 1).

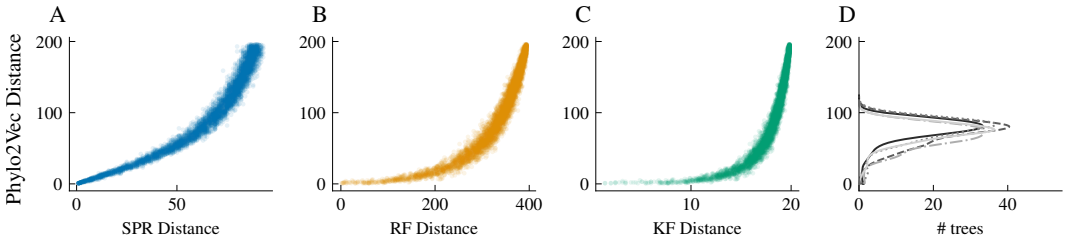


Figure S2: (A): comparison of Phylo2Vec moves with three popular tree distances: number of subtree prune and regraft moves (SPR); (B): Robinson-Foulds (RF); (C): Kuhner-Felsenstein (KF). To generate the distances, a random walk of 5000 steps was performed from a random initial v with 200 taxa. At each step, each v_i can increment, decrement or remain unchanged. (D): Density plots of μ for an equivalence set of rooted trees with a fixed v. 5 random v are shown.