AMB ALGORITHMS FOR MOLECULAR BIOLOGY

RESEARCH Open Access

Recursive algorithms for phylogenetic tree counting

Alexandra Gavryushkina[1*], David Welch[1] and Alexei J Drummond[1,2]

Abstract

Background: In Bayesian phylogenetic inference we are interested in distributions over a space of trees. The number of trees in a tree space is an important characteristic of the space and is useful for specifying prior distributions. When all samples come from the same time point and no prior information available on divergence times, the tree counting problem is easy. However, when fossil evidence is used in the inference to constrain the tree or data are sampled serially, new tree spaces arise and counting the number of trees is more difficult.

Results: We describe an algorithm that is polynomial in the number of sampled individuals for counting of resolutions of a constraint tree assuming that the number of constraints is fixed. We generalise this algorithm to counting resolutions of a fully ranked constraint tree. We describe a quadratic algorithm for counting the number of possible fully ranked trees on n sampled individuals. We introduce a new type of tree, called a fully ranked tree with sampled ancestors, and describe a cubic time algorithm for counting the number of such trees on n sampled individuals.

Conclusions: These algorithms should be employed for Bayesian Markov chain Monte Carlo inference when fossil data are included or data are serially sampled.

Keywords: Ranked tree, Constraint tree, Resolution, Counting trees, Dynamic algorithms, Bayesian tree prior, Phylogenetics

Background

A phylogenetic tree is the common object of interest in many areas of biological science. The tree represents the ancestral relationships between a group of individuals. Given molecular sequence data sampled from a group of organisms it is possible to infer the historical relationships between these organisms using a statistical model of molecular evolution. At present, Bayesian Markov chain Monte Carlo (MCMC) methods are the dominant inferential tool for inferring molecular phylogenies [1].

It is a recent trend to include fossil evidence into the inference to obtain absolute estimates of divergence times [2,3]. Fossils may restrict the age of the most resent common ancestor of a subgroup of individuals. This imposes a constraint on the tree topology (the discrete component of a genealogy) and therefore reduces the space of allowable genealogies.

Another trend in phylogenetic analyses is serial (or heterochronous) sampling in which molecular data is obtained from significantly different time points and analysed together. This type of data arises most frequently with ancient DNA and rapidly evolving pathogens [4-6]. In this case tip dates become a part of the genealogy.

Including serially sampled or fossil data modifies or restricts the shape of a phylogenetic tree. Little has been done to describe and classify these modified trees. In this paper, we aim to explore the new spaces formed by these trees.

A genealogy consists of discrete and continuous components — the tree topology and the divergence times. The tree topologies form a finite tree space when the number of tips is bounded. An important characteristic of this space is the number of trees in it and we aim to find an efficient way to calculate this number.

In the case that fossil data restricts the tree topology, counting the number of trees that satisfy the imposed constraints reveals how much the constraints reduce the tree space.

*Correspondence: sasha.gavryushkina@auckland.ac.nz
[1]Department of Computer Science, The University of Auckland, Auckland, New Zealand
Full list of author information is available at the end of the article

The number of trees arises as a constant in tree prior distributions. Typically we model the distribution of tree topologies as independent of the distribution of divergence times. The density function of the distribution of genealogies is then a product of the density function for the divergence times and the distribution function for tree topologies. A common prior on tree topologies is uniform over all allowable topologies so the distribution function is a constant that is equal to one over the number of tree topologies. When inferring tree topologies using Bayesian MCMC methods, we do not usually need to know this constant but in some cases, as described below, the absolute value of the prior distribution is of interest and the constant has to be calculated.

When fossils are used to restrict the age of internal nodes, the tree prior should accurately account for this fact. Heled and Drummond [3] introduced a natural approach for tree prior specification when fossil evidence is employed in the inference. Their method requires counting of ranked phylogenetic trees that obey a number of constraints that arise from including the fossil evidence. The construction requires calculation of the marginal density for the time of the calibration node, the node representing the most recent common ancestor of a clade which may or may not be monophyletic. For a particular location of the calibration node, or particular constraints on the tree topology, the marginal density function is the marginal density function for the divergence times weighted by the number of trees satisfying the constraints. In this case, the weight constants do not cancel in the MCMC scheme and therefore have to be calculated.

Tree counting has a long history. For phylogenetic trees, the counting problem is to find the number of all possible trees on n leaves. For some types of phylogenetic tree, there are known closed form solutions to this problem. For other types, only recursive equations have been derived. In this paper, we consider only rooted trees.

A survey of results on counting different types of rooted trees is presented in [7] where trees with different combinations of the following properties are considered: trees are either labeled (only leaves are labeled) or unlabeled, ranked or non-ranked, and bifurcating or multifurcating. The results presented in the survey can also be found in [6,8,9].

In [10], Griffiths considered unlabeled, non-ranked rooted trees such that interior nodes can have one child or more and the root has at least two children. Using generating functions, he derived recursive equations for counting the number of all possible such trees on n leaves with s interior nodes. In [11], Felsenstein considered partially labeled trees, i.e., a tree in which all the leaves are labeled and some interior nodes also may be labeled. He derived the recursive equations for counting the number of rooted, non-ranked, partially labeled trees with n labeled nodes.

In this paper, we consider a number of counting problems for different classes of phylogenetic trees. First, we describe an effective way of counting the number of all possible fully ranked trees on n leaves, that is, trees on n leaves in which all internal and leaf nodes are ranked.

Second, we find the number of bifurcating trees that resolve a given multifurcating tree with n leaves. We give a solution to this problem for rooted, ranked, labeled trees and generalise the algorithm to count resolutions to fully ranked trees.

Finally, we introduce and formally describe a new type of phylogenetic tree and describe an algorithm for counting the number of all such trees on n leaves. This type of tree is important when we have a serial sample and sampled individuals can be direct ancestors of later sampled individuals. When the population size is small or the fraction of individuals sampled from the population is large, this type of tree should be included in the inference [12,13].

Serial sampling

We mainly follow the terminology from [9] for the definitions of phylogenetic trees. A tree is a finite connected undirected graph with no cycles. A rooted tree is a tree with a single node ρ designated as a root. Every rooted tree $T = (V, E, \rho)$ imposes a partial order on V that is defined as follows: $v_1 \leq_T v_2$ if a unique simple path from the root to v_2 passes through v_1. So the root is the smallest element. If $v_1 \leq_T v_2$ then we say that v_1 is an ancestor of v_2 and v_2 is a descendant of v_1. A node in a rooted tree is called interior if it has descendants and a leaf if it has no descendants. The root is considered interior. Denote $\mathring{V}$ the set of interior nodes of T. A node u is a parent of a node v and v is a child of u if $v <_T u$ and there is no $w \in V$ such that $v <_T w <_T u$. A rooted tree is called binary if every interior node has exactly two children. It is called weakly binary if every interior node has at most two children. We have chosen this terminology to fit with the usage of "binary" in the phylogenetics literature which may not agree with that in other literatures.

Let X be a finite non-empty set of labels. *A phylogenetic X-tree* is a pair $\mathcal{T} = (T, \phi)$, where T is a tree and ϕ is a bijection from X onto the set of leaves of T (we may omit X and say "tree" instead of "X-tree" if the set of labels is not specified). The tree T is called an underlying tree or a shape of the phylogenetic tree $\mathcal{T}$ and ϕ is a labeling function. If the underlying tree of $\mathcal{T}$ is rooted then $\mathcal{T}$ is called *a rooted phylogenetic tree*. In what follows, we consider only rooted trees unless explicitly stated otherwise. A phylogenetic tree is *binary (weak binary)* if its underlying tree is binary (weak binary). *A ranked phylogenetic tree* is a pair $(\mathcal{T}, h)$, where $\mathcal{T}$ is a rooted phylogenetic tree and h is an injective function (*ranking function*) from the

set $\mathring{V}$ into the set $\{1,\ldots,|\mathring{V}|\}$ such that $v_1 \leq_T v_2$ implies $h(v_1) \leq h(v_2)$ for every $v_1, v_2 \in \mathring{V}$. In other words, there is a linear order on the interior nodes of T that is consistent with the partial order of T.

Definition 1. ***A ranked X-tree*** *is a binary ranked phylogenetic X-tree.*

An example of a ranked tree is given in Figure 1.

In biology, a phylogenetic tree represents the evolutionary history of a collection of sampled individuals. The collection of individuals is represented by the set X. The root of the tree is the most recent common ancestor of X and interior nodes are bifurcation events. The ranking function represents the time order of the bifurcation events. A general problem in evolutionary biology is how to reconstruct the phylogenetic tree from sequence data obtained from sampled individuals. Tackling this problem in a Bayesian framework may require counting the number of all possible histories on a sample of individuals.

When all individuals are sampled at the same time (as in Figure 1) counting tree problem has a simple solution.

Let X be a fixed label set such that $|X| = n$. The number of all ranked X-trees up to isomorphism is

$$R(n) = \frac{n!\,(n-1)!}{2^{n-1}}$$

This formula has been derived by many authors. Proofs can be found in [6,7], or [9]. The letter R in the equation comes from the word "ranked".

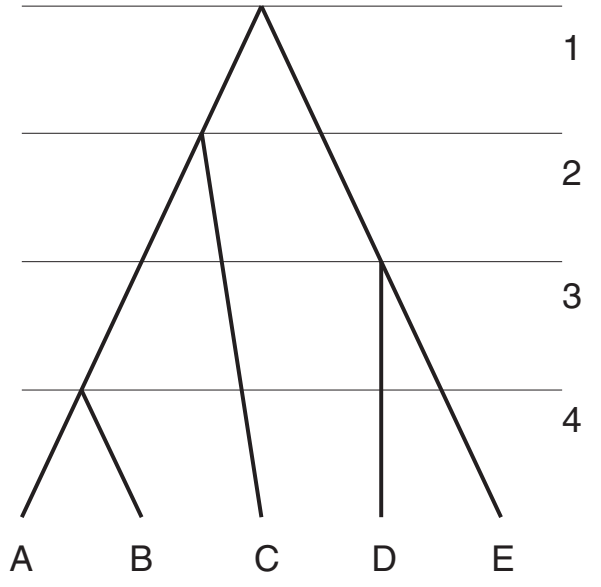

Figure 1 Ranked tree. Ranked X-tree, $X = \{A, B, C, D, E\}$. The numbers on the right are values of the ranking function.

The situation is different when individuals are sampled at different times (serially sampled). In this case, we need to define another kind of phylogenetic tree in which leaves are also ranked.

Definition 2. ***A fully ranked (FR) X-tree*** *is a pair $(\mathcal{T}, h)$, where $\mathcal{T}$ is a binary rooted phylogenetic X-tree and $h : V \to \{1,\ldots,l\}$ with $|\mathring{V}| < l \leq |V|$ is a surjective function such that*

- *$v_1 \leq_T v_2$ implies $h(v_1) \leq h(v_2)$ and*
- *$h(v_1) = h(v_2)$ implies $v_1 = v_2$ or $v_1, v_2 \in V \setminus \mathring{V}$.*

An example of a fully ranked X-tree is given in Figure 2.

Before the tree is reconstructed we observe only leaves (sampled individuals) of the tree that are grouped (pre-ranked) according to the times they were sampled. For the tree shown in Figure 2, we have two sampling times and hence two groups: A, B, and C form the first group; D and E form the second group.

Let $\mathfrak{T} = (\mathcal{T}, h)$ be a fully ranked X-tree with $h : V \to \{1,\ldots,l\}$. Let $m = |h(\phi(X))|$, that is, the number of sampling times. Define *a pre-ranking function* $\hat{h}$ from X onto $\{1,\ldots,m\}$ for tree $\mathfrak{T}$ such that for all $x_1, x_2 \in X$

- $h(\phi(x_1)) \leq h(\phi(x_2))$ implies $\hat{h}(x_1) \leq \hat{h}(x_2)$ and
- $h(\phi(x_1)) = h(\phi(x_2))$ iff $\hat{h}(x_1) = \hat{h}(x_2)$.

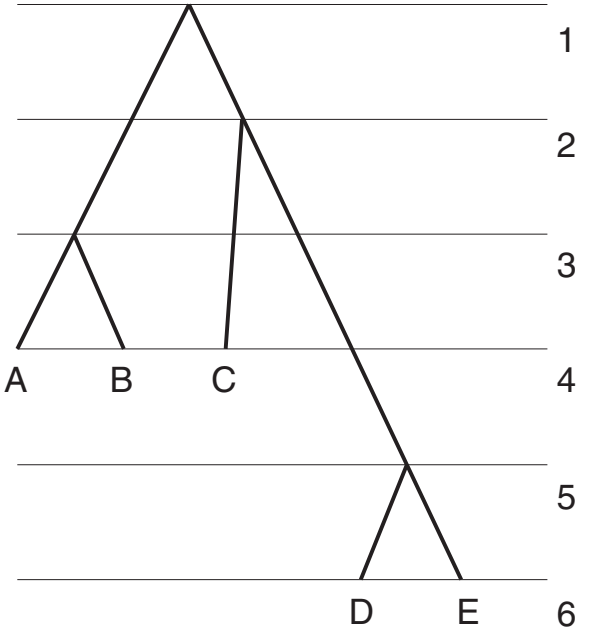

Figure 2 Fully ranked tree. Fully ranked X-tree. $X = \{A, B, C, D, E\}$. The numbers on the right are values of the ranking function.

For the tree given in Figure 1, $\hat{h}(A) = \hat{h}(B) = \hat{h}(C) = 1$ and $\hat{h}(D) = \hat{h}(E) = 2$.

Let X and $\hat{h} : X \to \{1, \ldots, m\}$ be fixed. We are interested in the number of all fully ranked X-trees that have $\hat{h}$ as a pre-ranking function. Note that this number depends only on the numbers $n_i = |\{x|\hat{h}(x) = i\}|$, the number of individuals sampled at the ith time point, not on X and $\hat{h}$ directly. We denote this quantity by $F(n_1, \ldots, n_m)$, where F stands for "fully ranked". Then

$$F(n_1, \ldots, n_m) = \sum_{i=1}^{n_m} \frac{R(n_m)}{R(i)} F(n_1, \ldots, n_{m-1} + i) \qquad (1)$$

and $F(n) = R(n)$.

Proof. Consider a continuous process of bifurcation in which lineages may bifurcate in time or be cut and labeled (sampled). The process finishes when all lineages are cut producing a tree. The discrete structure of the tree produced by this process is a fully ranked X-tree. It is easy to see that every fully ranked X-tree can be obtained as a result of this process. To count the required number we can count the number of different trees which can be produced by the process if we know that after it finishes there are n_i sampled individuals (i.e., cut and labeled lineages) at the ith time point, i.e., we have the sequence $(n_1, \ldots, n_m)$.

Suppose that at the $(m-1)$th time point there are i lineages that are ancestral to n_m individuals sampled at time m. When we look at this process backwards in time the bifurcation events become coalescence events. The number of different ways these n_m lineages coalesce to i lineages is $\frac{R(n_m)}{R(i)}$. This is the number of all possible ranked X-trees on n_m individuals but since we are not interested in the structure of the coalescent after we reach i lineages, it is divided by the number of ways in which the remaining i lineages can coalesce. Note that if coalescence patterns are different between the $(m-1)$th and mth time points then the trees are also different.

Further, for each of these coalescence patterns, we need to count the number of different ways these i lineages and other $n_1, \ldots, n_{m-1}$ lineages can coalesce. This is where we can apply the recursion. We can consider that we also cut these i lineages at time $m-1$ and label them with the ranked subtrees descendant from these lineages. Then, at time $m-1$, we have n_{m-1} sampled individuals and another i sampled individuals and it remains to count the number of trees on the sequence $(n_1, \ldots, n_{m-1} + i)$.

Note that two trees are different if they have different numbers of lineages at time $m-1$. The number of additional i lineages can be between 1 and n_m and we need to sum over all possible i to complete the recursion. □

We introduce a third type of tree in which sampled individuals may be direct ancestors of later sampled individuals. We call it a tree with sampled ancestors. This type of tree is not usually considered in phylogenetics since the probability of sampling a direct ancestor is often negligible. In small populations or when a large portion of the population is sampled, however, this can not be ignored.

Let $T = (V, E, \rho)$ be a weak binary tree. Define a set $\hat{V}$ as follows:

$$\hat{V} = \{v \in V | deg(v) = 1 \text{ or } [deg(v) = 2 \text{ and } v \text{ is not the root}]\}$$

A rooted S-phylogenetic X-tree is a pair $\mathcal{T} = (T, \phi)$, where T is a weak binary tree and $\phi : X \to \hat{V}$ is a bijection.

Definition 3. ***A fully ranked X-tree with sampled ancestors (FRS X-tree)*** *is a pair $(\mathcal{T}, h)$, where $\mathcal{T}$ is a rooted S-phylogenetic X-tree and $h : V \to \{1, \ldots, l\}$ is a surjective function such that*

- $v_1 <_T v_2$ *implies* $h(v_1) < h(v_2)$ *and*
- $h(v_1) = h(v_2)$ *implies* $v_1 = v_2$ *or* $v_1, v_2 \in \hat{V}$;

(see Figure 3).

The definition of a pre-ranking function remains the same for FRS trees. Let $S(n_1, \ldots, n_m)$ (with S standing for 'sampled ancestors') denote the number of all FRS X-trees that have the same pre-ranking function $\hat{h}$, where $n_i = |\{x \mid \hat{h}(x) = i\}|$. Then

$$S(n_1, \ldots, n_m) = \sum_{i=1}^{n_m} \sum_{j=0}^{min\{i, n_{m-1}\}} \binom{i}{j} \binom{n_{m-1}}{j} \times \frac{R(n_m)}{R(i)} S(n_1, \ldots, n_{m-1} + i - j) \qquad (2)$$

and $S(n) = R(n)$.

Proof. Consider the same process as before with only change of sampling events. Now, at some points of time, some lineages are cut and labeled and others are only labeled but not cut.

Then this equation can be obtained as follows. We have n_m individuals that are sampled at the mth time point. At time $m-1$, there are between 1 and n_m ancestral lineages of those n_m individuals, depending on the number

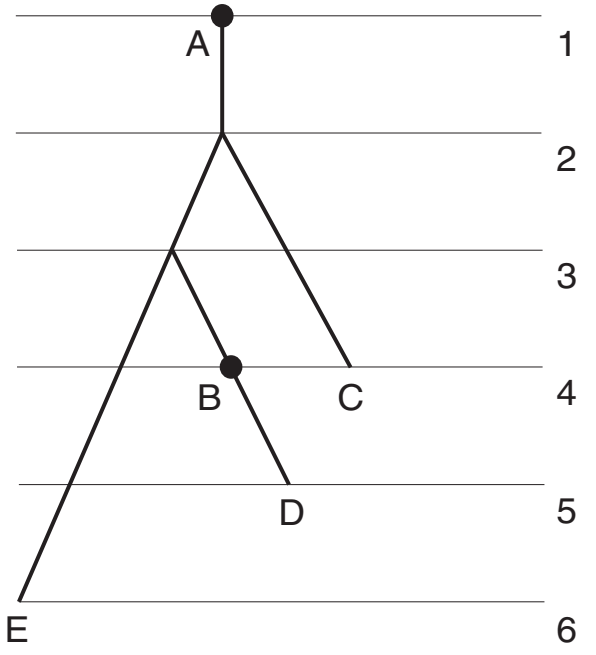

Figure 3 FRS tree. FRS X-tree with the labeled 1-degree root. $X = \{A, B, C, D, E\}$. The numbers on the right are values of the ranking function.

of coalescence between times m and $m-1$. Let i denote the number of these lineages. Then there are $\frac{R(n_m)}{R(i)}$ different possible coalescent patterns that can lead to this situation. Some of these i ancestral lineages may be among the individuals sampled at time $m-1$, i.e. lineages that are labeled but not cut at time $m-1$. Let j be the number of those ancestral lineages that are sampled at time $m-1$. There are $\binom{i}{j}$ ways to chose these j lineages out of i and there are $\binom{n_{m-1}}{j}$ possible ways to chose j sampled at time $m-1$ individuals that are not cut at time $m-1$.

Further, at time $m-1$, there are n_{m-1} sampled lineages and $i-j$ ancestral lineages that are not sampled and it remains to count the number of FRS trees on the sequence $(n_1, \ldots, n_{m-1} + i - j)$.

Finally, we sum over all possible i and j to complete the recursion. □

Dynamic counting

Calculating the recursions (1) and (2) directly is inefficient and impractical. Here we describe a more efficient algorithm for counting fully ranked trees using these recursions. Rewrite equation (1) as

$$F(n_1, \ldots, n_m) = R(n_m) \sum_{i=1}^{n_m} \frac{F(n_1, \ldots, n_{m-1} + i)}{R(i)}$$

Then instead of calculating $F(n_1, \ldots, n_j + \alpha)$ for $j \in \{1, \ldots, m-1\}$ and $\alpha \in \{0, \ldots, n_{j+1} + \ldots + n_m\}$ we can calculate

$$A^j(\alpha) = \frac{F(n_1, \ldots, n_j + \alpha)}{R(\alpha)} \tag{3}$$

using recurrence equations:

$$A^j(0) = R(n_j) \sum_{i=1}^{n_j} A^{j-1}(i) \text{ and} \tag{4}$$

$$A^j(\alpha+1) = \frac{(n_j+\alpha)(n_j+\alpha+1)}{\alpha(\alpha+1)} A^j(\alpha) + \frac{R(n_j+\alpha+1)}{R(\alpha+1)} A^{j-1}(n_j+\alpha+1). \tag{5}$$

This leads to Algorithm 1 to calculate $F(n_1, \ldots, n_m)$. Let n be the number of sampled individuals, i.e., $n = \sum_{i=1}^{m} n_i$. Calculation of all the $R(i)$ takes $O(n)$ steps and calculation of all the A^j takes at most $O(n)$ steps. In total, the algorithm takes $O(mn)$ steps.

Algorithm 1 Calculating the number of fully ranked trees

for $i = 1 \to n$ **do**
  calculate $R(i)$ using $R(i) = R(i-1)\frac{i(i-1)}{2}$
end for
for $i = 1 \to n_2 + \ldots + n_m$ **do**
  calculate $A^1(i)$ using equation (3) and equality $F(x) = R(x)$
end for
for $j = 2 \to m-1$ **do**
  for $\alpha = 0 \to n_{j+1} + \ldots + n_m$ **do**
    calculate $A^j(\alpha)$ using equations (4) and (5)
  end for
end for
compute $F(n_1, \ldots, n_m) = A^m(0)$

A similar approach leads to an $O(mn^2)$-time algorithm for counting FRS trees. The description of this algorithm is in Appendix 1.

Constraints

In phylogenetic analysis, it is common to have some limited information about the ancestors of sampled individuals. We consider two types of such information. First, we may know that a subgroup of sampled individuals is monophyletic. That means that the most recent common ancestor of the subgroup is not an ancestor of any other individual that does not belong to the subgroup. Second,

we may know the relative ages of the most recent common ancestors of monophyletic subgroups. This known information imposes constraints on the space of possible phylogenetic trees representing the evolutionary history of sampled individuals. The question is how many phylogenetic trees satisfy the constraints on a group of sampled individuals?

The number of resolutions of a constraint tree

We first describe a problem for contemporaneous sampling in terms of constraint trees. We call a rooted tree multifurcated if each interior node has at least 2 children. Note that in contrast to the common terminology we assume that a binary tree is also multifurcated. If we replace the word "binar" with "multifurcated" in the definition of a ranked tree we obtain a more general class of trees.

Definition 4. ***A constraint X-tree*** *is a multifurcated ranked phylogenetic X-tree.*

An example of a constraint tree is given in Figure 4. A constraint tree represents prior information about clades and ranking. Each interior node constrains a subgroup of individuals, leaves that are descendant from this node, to by monophyletic. The most recent common ancestor of the whole group of individuals, the root node, is also regarded as a constraint. The ranking function constrains the ages of the most recent common ancestors of the monophyletic subgroups to have a specified order.

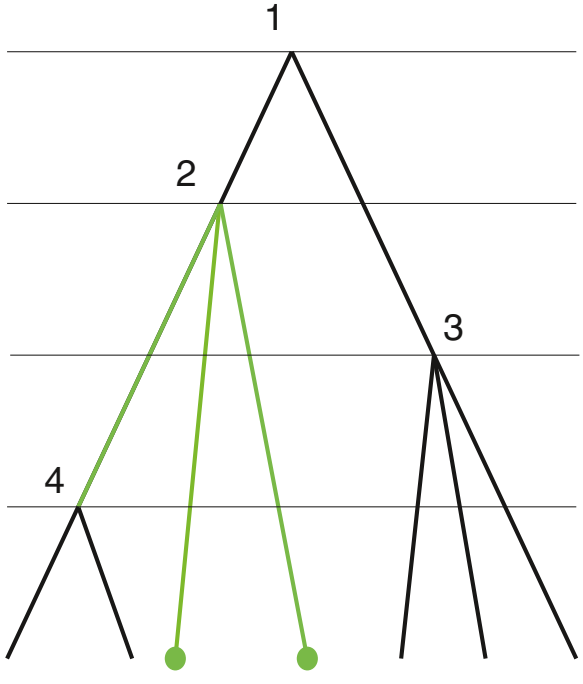

Figure 4 Constraint tree. Constraint tree, labels are omitted. Subtree 2 is coloured green. It has two child nodes that are leaves, therefore, $n_2 = 2$. The ancestor function for this tree is defined as $f(2) = f(3) = 1$ and $f(4) = 2$. A compact notation for this constraint tree is $(n_1, \ldots, n_k, f) = (0, 2, 3, 2, \{(2, 1), (3, 1), (4, 2)\})$.

We say that a ranked X-tree $\mathfrak{T}_1 = (\mathcal{T}_1, h_1)$ *resolves* a constraint X-tree $\mathfrak{T}_2 = (\mathcal{T}_2, h_2)$ if there is an isomorphic embedding of $\mathfrak{T}_2$ into $\mathfrak{T}_1$, i.e., there is an injective mapping $f : V_2 \to V_1$ such that

- $\phi_1(x) = f(\phi_2(x))$ for each $x \in X$;
- $v \leq_{T_2} u$ iff $f(v) \leq_{T_1} f(u)$ for each $u, v \in V_2$; and
- $h_2(v) \leq h_2(u)$ implies $h_1(f(v)) \leq h_1(f(u))$ for each $u, v \in \mathring{V}_2$.

We wish to calculate the number of ranked trees that resolve a given constraint tree $\mathfrak{T} = (T, \phi, h)$. It is easy to see that this number depends only on the underlying tree T and ranking function h, but does not depend on the labeling function ϕ or the label set X.

We now introduce some notation in order to define recursive equations. We label interior nodes according to their ranks such that node i is a node v such that $h(v) = i$. A subtree induced by node i and its children is called subtree i. Child nodes in a subtree may be leaves in the initial tree. Let $n_i \geq 0$ denote the number of such child nodes in subtree i. Let $f : \{2, \ldots, k\} \to \{1, \ldots k-1\}$ be the parent function on interior nodes, i.e., $f(i) = j$ whenever j is a parent to i. When $k = 1$, i.e., there is only one interior node, $f = \emptyset$. See Figure 4 for an example of introduced notation. Note that a tuple $(n_1, \ldots, n_k, f)$ completely defines a pair (T, h).

Let $R^r(n_1, \ldots, n_k, f)$ be the number of ranked trees resolving a constraint tree defined by the tuple $(n_1, \ldots, n_k, f)$. The superscript r stands for "resolution". Then the following equations hold.

$$R^r(2, \emptyset) = 1, \tag{6}$$

$$R^r(n_1, \ldots, n_{k-1}, 2, f) = \sum_{i \in C} \binom{n_i}{2} R^r(n_1, \ldots, n_i - 1, \ldots, n_{k-1}, 2, f) + R^r(n_1, \ldots, n_{f(k)} + 1, \ldots, n_{k-1}, f|_{\{2,\ldots,k-1\}}) \text{ and} \tag{7}$$

$$R^r(n_1, \ldots, n_k, f) = \sum_{i \in C} \binom{n_i}{2} R^r(n_2, \ldots, n_i - 1, \ldots, n_k, f), \quad \text{if } n_k > 2, \tag{8}$$

where C is the collection of nodes that have more than 2 children and at least 2 of them are leaves, i.e., $C = \{i <$

$k \mid n_i \geq 2$ and $n_i + \alpha_i > 2\}$ and $\alpha_i = |\{a | f(a) = i\}|$. Note that $n_i + \alpha_i$ is the number of children of node i.

Proof. When a constraint tree has 2 leaves, it is unique and is resolution of itself. So Equation (6) is trivial. To explain the main sum in Equation (7) and (8) we consider the constraint tree which is defined by $(3, 3, \{(2, 1)\})$ and shown in Figure 5, left. The last interior node of a resolving tree (that is, the interior node with the highest rank or the furthest node from the root) is either a parent to leaves in subtree 1 or leaves in subtree 2. Suppose it is the first case (see Figure 5, centre). Since leaves have distinct labels from X, there are $\binom{3}{2}$ ways to chose two leaves that are children of that last node. We can partition all the resolving trees for which the last node is in subtree 1 in $\binom{3}{2}$ groups. The number of trees in each group is the number of trees that resolve a constraint tree defined by $(2, 3, \{(2, 1)\})$ and shown on the right of Figure 5. A similar argument holds if the last node in a resolving tree is a parent to nodes from subtree 2.

So in the general case, there are k subtrees and if the last node is in subtree i we have $\binom{n_i}{2}$ ways to choose two lineages that coalesce and then we should count the number of trees resolving the tree defined by $(n_1, \ldots, n_i - 1, \ldots, n_k, f)$. Note that in the example above, we consider the tree with more than 2 leaves in each subtree. However, the last interior node of a resolving tree can not be in subtree i for $i < k$ if there is not enough leaves in this subtree. This can happen either if there are less than 2 leaves in subtree i or if there are 2 leaves in subtree i and node i has only these two leaves as its children. Both cases imply that any parent to leaves of subtree i in a resolving tree has a lesser rank than the rank of node k. This explains why we sum only over the elements of the set C.

Finally, we should consider one more case which explains why there is one more summand in equation (7). If the last node in a constraint tree has only two children, i.e., there are 2 leaves in subtree k, then there is one more group of resolving trees, the group that consists of resolving trees that have this node as the last node. □

Dynamic counting

We will calculate $R^r(n_1, \ldots, n_k, f)$ for the corresponding constraint tree. In order to find $R^r(n_1, \ldots, n_k, f)$, at each step s, we will calculate numbers $R^r(x_1, \ldots, x_t, f|_{\mathbf{t}_{-1}})$ with $\sum_{i \leq t} x_i = s$, $\mathbf{t}_{-1} = \{2, \ldots, t\}$, and $t \leq k$. Note that we do not have to calculate all such numbers. To determine which numbers are required we define two upper triangular matrices $\mathtt{m}$ and $\mathtt{M}$ of size $k \times k$.

Suppose we draw a horizontal line which is strictly below the line that passes through node j and strictly above the line that passes through node $j + 1$ (or all the leaves if $j = k$). Then $\mathtt{m}_{i,j}$ is the minimal possible number of intersections of this line with branches of subtree i in a resolving tree and $\mathtt{M}_{i,j}$ is the maximal possible number. An example is given in Figure 6.

Let $a_{i,j} = |\{x \leq j \mid f(x) = i\}|$ for $i \leq j$. So $a_{i,j}$ is the number of children of node i with ranks at most j. Then

$$\mathtt{M}_{i,j} = n_i + \alpha_i - a_{i,j} \text{ and}$$

$$\mathtt{m}_{i,j} = \begin{cases} 2 \text{ if } a_{i,j} = 0, \\ 1 \text{ if } a_{i,j} > 0 \text{ and } \mathtt{M}_{i,j} > 0, \\ 0 \text{ otherwise.} \end{cases}$$

Let $t \leq k$ and $x_1, \ldots, x_t \in \mathbb{N}$. We call a tuple $(x_1, \ldots, x_t, f|_{\mathbf{t}_{-1}})$ eligible if $\mathtt{m}_{i,t} \leq x_i \leq \mathtt{M}_{i,t}$ for $1 \leq i \leq t$.

We now turn to Algorithm 2 to count resolutions. At each step $s \leq n$, we construct a set S_s. A unique element of S_n is $R^r(n_1, \ldots, n_k, f)$ and calculating elements of S_s only requires elements of S_{s-1}.

Algorithm 2 Calculating the number of resolutions of a constraint tree

$S_2 = \{R^r(2, \emptyset)\}$
for $s = 3 \to n - 1$ **do**
  while there is a new element $R^r(x_1, \ldots, x_t, f|_{\mathbf{t}_{-1}})$ in the set S_{s-1} **do**
    if $t < k$ and *eligible* $(x_1, \ldots, x_{f(t+1)} - 1, \ldots, x_t, 2, f|_{\mathbf{t}_{-1}})$ **then**
      calculate $R^r(x_1, \ldots, x_{f(t+1)} - 1, \ldots, x_t, 2, f|_{\mathbf{t}_{-1}})$ and add it to S_s
    end if
    for $i = 1 \to t$ **do**
      if *eligible*$(x_1, \ldots, x_i + 1, \ldots, x_t, f|_{\mathbf{t}_{-1}})$ **then**
        $R^r(x_1, \ldots, x_i + 1, \ldots x_t, f|_{\mathbf{t}_{-1}})$ and add it to S_s
      end if
    end for
  end while
end for

Proposition 1. *When k is fixed Algorithm 2 does at most* $O(n^k)$ *steps.*

Proof. The algorithm does $O(k)$ steps for each eligible tuple and, since we assume k is a constant, it is $O(1)$. For given j there are

$$\prod_{i=1}^{j} (\mathtt{M}_{i,j} - \mathtt{m}_{i,j} + 1)$$

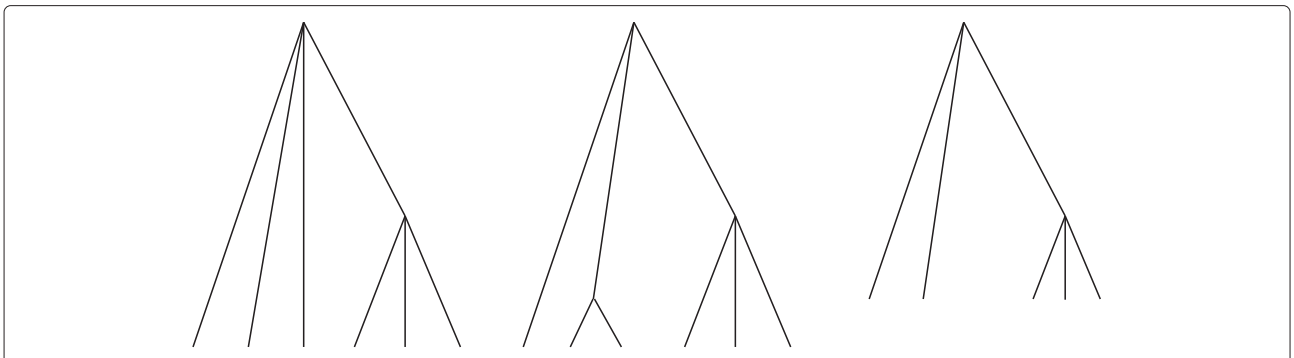

Figure 5 Recursive approach. The last interior node in a resolving tree locates in subtree 2.

eligible tuples of size j. Since $\mathtt{M}_{i,j} - \mathtt{m}_{i,j} + 1 \leq n_i + \alpha_i - 1$, for the total number of eligible tuples we have

$$\sum_{j=1}^{k}\prod_{i=1}^{j}(\mathtt{M}_{i,j}-\mathtt{m}_{i,j}+1) \leq \sum_{i=1}^{k}(n_1+\alpha_1-1)\cdot\ldots\cdot(n_i+\alpha_i-1) < k(n_1+k-1)\cdot\ldots\cdot(n_k+k-1) \leq k(\frac{n}{k}+k-1)^k = O(n^k)$$

□

The number of resolutions of a fully ranked constraint tree

We can generalise the results of the previous section to fully ranked trees. Now we replace the word "binary" with "multifurcated" in the definition of a fully ranked tree to get

Definition 5. ***A fully ranked constraint X-tree*** *is a pair* $(\mathcal{T}, h)$*, where* $\mathcal{T}$ *is a multifurcated rooted phylogenetic X-tree and* h *is a function such that* $h : V \to \{1, \ldots, l\}$*, where* $|\mathring{V}| < l \leq |V|$*, and*

- $v_1 \preceq_T v_2$ *implies* $h(v_1) \leq h(v_2)$*;*
- $h(v_1) = h(v_2)$ *implies* $v_1 = v_2$ *or* $v_1, v_2 \in V \setminus \mathring{V}$*.*

We say that a fully ranked X-tree $\mathfrak{T}_1 = (\mathcal{T}_1, h_1)$ *resolves* a fully ranked constraint X-tree $\mathfrak{T}_2 = (\mathcal{T}_2, h_2)$ if there is an isomorphic embedding of $\mathfrak{T}_2$ into $\mathfrak{T}_1$, i.e., there is an injective mapping $f : V_2 \to V_1$ such that

- $\phi_1(x) = f(\phi_2(x))$ for each $x \in X$;
- $v \preceq_{T_2} u$ iff $f(v) \preceq_{T_1} f(u)$ for each $u, v \in V_2$;
- $h_2(v) \leq h_2(u)$ implies $h_1(f(v)) \leq h_1(f(u))$ for each $u, v \in V_2$; and
- $h_2(v) = h_2(u)$ iff $h_1(f(v)) = h_1(f(u))$ for each $u, v \in V_2$.

The problem is to count the number of fully ranked resolutions of a fully ranked constraint tree.

Let $\mathfrak{T} = (\mathcal{T}, h)$ be a fully ranked constraint tree with $h : V \to \{1, \ldots, l\}$ and $|\mathring{V}| = k$. Again we label interior nodes with numbers from $\{1, \ldots, k\}$. However, labels and ranks of interior nodes do not necessary coincide now. Let $r : \{1, \ldots, k\} \to h(\mathring{V})$ be an injective increasing function which maps the kth interior node to its rank. Note that $r(1)$ is always equal to 1 because the root node always has rank 1 and that $r(k) < l$ because only leaves can have rank l. Now, node i is the node that has rank $r(i)$ and sub-

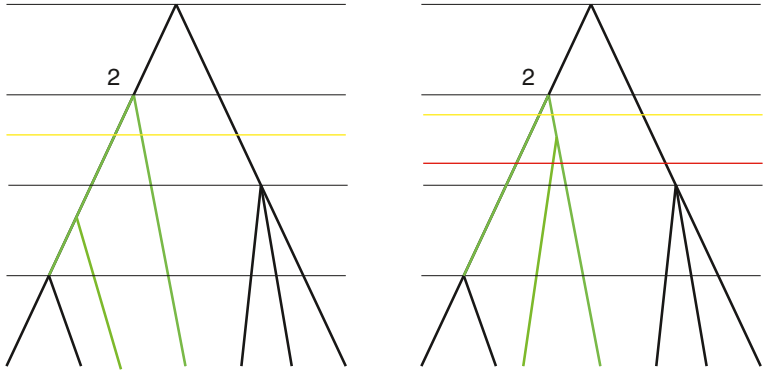

Figure 6 Defining matrices $\mathtt{m}$ and $\mathtt{M}$. Two trees that resolve a constraint tree from Figure 4 (only resolutions of subtree 2 are shown) and three ways to draw a horizontal line. The yellow lines correspond to the minimal number of intersections and the red line, to maximal. Thus, $\mathtt{m}_{2,2} = 2$ and $\mathtt{M}_{2,2} = 3$. We can also note that $\mathtt{m}_{1,2} = \mathtt{M}_{1,2} = 1$.

tree i is the subtree which is induced by node i and its children. Since leaves in subtrees are now ranked, we need more parameters to encode the tree. Let $\mathbf{n} = [n_{i,j}]$ be a matrix of size $k \times l$, where $n_{i,j}$ is the number of leaves of rank j in subtree i or the number of children of node i with rank j. Note that if node i is a parent of node j then $n_{i,r(j)} = 1$ and $n_{i,x} = 0$ for $x \neq r(j)$ and this means that the parent function is uniquely defined by the matrix $\mathbf{n}$ and function r and the number of resolutions of $\mathfrak{T}$ depends only on $(\mathbf{n}, r)$.

If a constraint tree has only 2 leaves then there is only 1 tree resolving it:

$$F^r((2), r_0) = F^r((1,1), r_0) = 1$$

with r_0 mapping 1 to 1.

For the main recursion we need to determine the location of the last interior node in a resolving tree. As before, this node can be a parent to leaves in different subtrees. Also, since leaves now may have different ranks, the last interior node in a resolving tree may be ranked in different ways with respect to ranks of the leaves. Let $N_{c,x}$ denote the number of children of node c with ranks at least x, that is, $N_{c,x} = \sum_{j=x}^{l} n_{c,j}$. For each $p \in \{r(k)+1, \ldots, l\}$, we define a set C^p of candidate subtrees in which the last node can be placed at level p, i.e. between the $(p-1)$th and pth time points.

$$C^p = \{c \mid N_{c,p} \geq 2 \text{ and } N_{c,r(c)+1} > 2\}$$

See Figure 7 for an example of subtree where the last node can be placed between the $(p-1)$th and pth time points.

For each p and c such that $c \in C^p$, there is a distinct group of resolving trees. The quantity of trees in this group is equal to the number of resolutions of a constraint tree which is defined by matrix $\mathbf{n}^{c,p} = [n^{c,p}_{i,j}]$ of size $k \times p$ such that

- $n^{c,p}_{i,j} = n_{i,j}$ for $1 \leq i \leq k$ and $j < p$;
- $n^{c,p}_{c,p} = N_{c,p} - 1$;
- $n^{c,p}_{i,p} = N_{i,p}$ for $i \leq k$ and $i \neq c$.

If node k has only 2 children then, as before, there is one more group of resolving trees. This group consists of the resolving trees in which the last interior node coincides with node k. Let $\mathbf{n}' = [n'_{i,j}]$ be a matrix of size $(k-1) \times r(k)$ such that

- $n'_{i,j} = n_{i,j}$ for $1 \leq i \leq k$ and $j < r(k)$,
- $n'_{i,r(k)} = N_{i,r(k)}$ for $1 \leq i \leq k-1$.

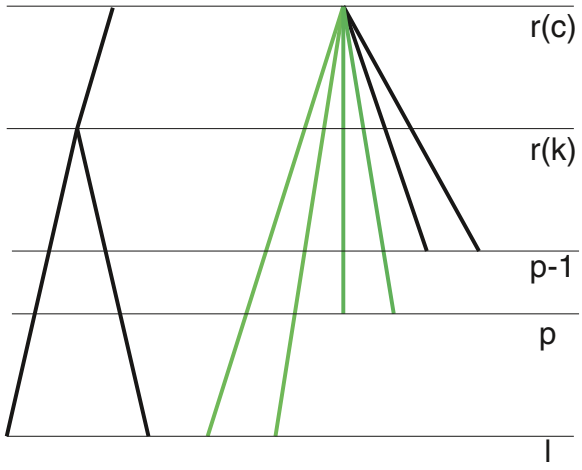

Figure 7 Location of a new node between the $(p-1)$-th and p-th time points. A new node can be placed in subtree c between the $(p-1)$-th and p-th time points if the number of green branches is greater or equal to 2 and the number of all branches in subtree c is greater than 2.

This matrix defines a constraint tree for which the number of resolutions is equal to the number of trees in the last group.

Then the main recursion is as follows:

$$F^r(\mathbf{n}, r) = \sum_{p=r(k)+1}^{l} \sum_{c \in C^p} \binom{N_{c,p}}{2} F^r(\mathbf{n}^{c,p}, r) + F^r(\mathbf{n}', r|_{\{1,\ldots,k-1\}}) \text{ if } N_{k,r(k)+1} = 2; \quad (9)$$

$$F^r(\mathbf{n}, r) = \sum_{p=r(k)+1}^{l} \sum_{c \in C^p} \binom{N_{c,p}}{2} F^r(\mathbf{n}^{c,p}, r) \quad \text{otherwise;} \quad (10)$$

Using these equations, we can calculate $F^r(\mathbf{n}, r)$ for $O(m^2 n^k)$ steps, where m is the number of sampling points, i.e., $m = l - k$. The calculation is described in Appendix 2.

Conclusions

In Table 1, we summarised the complexity (upper bounds) of the counting algorithms. We can see that counting fully ranked and FRS trees is reasonably fast, particularly when the number of sampling points is small. Counting of resolutions of a constraint tree is an expensive procedure but for small k it remains possible. In practice, k is typically small so this algorithm will be of practical use. Counting resolutions of an FRS constraint tree, using the same methods, has a greater cost and is less elegant, but its practicality and bounds on its complexity remain to be assessed.

Table 1 The complexity of algorithms

	No constraints ($k = 1$)	k constraints
Contemporaneous sampling ($m = 1$)	$O(n)$	$O(n^k)$
Serial sampling with no sampled ancestors	$O(mn)$	$O(m^2n^k)$
Serial sampling with sampled ancestors	$O(mn^2)$	-

The table summaries the complexity of the counting algorithms, where n is the sample size, k is the number of constraints, and m is the number of sampling time-points. We assume that k is fixed.

These algorithms can be implemented in software for phylogenetic analysis that involves serial sampling scheme or limited prior knowledge about ancestors of particular clades for calculating tree prior distributions.

Appendix 1: Algorithm for counting FRS trees

Rewrite equation (2) as follows:

$$S(n_1,\dots,n_m)=\sum_{i=0}^{n_m}A(i,n_{m-1},n_m)S(n_1,\dots,n_{m-1}+i)$$

where $A(i,a,b)=\sum_{x=0}^{min\{a,b-i\}}\frac{R(b)}{R(i+x)}\binom{i+x}{x}\binom{a}{x}$. Then we can calculate $S(n_1,\dots,n_m)$ recursively using Algorithm 3. We use the notation $N_j=\sum_{i=j}^{m}n_i$.

Algorithm 3 Calculating the number of FRS trees

for $i=1\to N_1$ **do**
  calculate $R(i)$ using $R(i)=R(i-1)\frac{i(i-1)}{2}$
end for
for $j=2\to m$ **do**
  calculate $A(0,n_{j-1},n_j),\dots,A(n_j,n_{j-1},n_j)$
  calculate $S(n_1,\dots,n_j)$
  for $\alpha=1\to N_{j+1}$ **do**
    calculate $A(0,n_{j-1},n_j+\alpha),\dots,A(n_j+\alpha,n_{j-1},n_j+\alpha)$
    calculate $S(n_1,\dots,n_j+\alpha)$
  end for
end for

At each step $j>1$ of the algorithm, we calculate $S(n_1,\dots,n_j+\alpha)$ for $0\le\alpha\le N_{j+1}$. We skip step $j=1$ and do not calculate $S(n_1+\alpha)$ for $0\le\alpha\le N_2$ because $S(x)=R(x)$ and we have already calculated all the necessary $R(x)$. Further, for calculating each $S(n_1,\dots,n_j+\alpha)$, we need to calculate the coefficients A and this is the most expensive part of the algorithm. So, at each step $j>1$, we need to calculate $A(i,n_{j-1},n_j+\alpha)$ for $0\le i\le n_j+\alpha$ and $0\le\alpha\le N_{j+1}$.

First, we calculate these values for $\alpha=0$. Denote

$$B(i,a,b,x)=\frac{R(b)}{R(i+x)}\binom{i+x}{x}\binom{a}{x},\text{ for } x\le a \text{ and } i+x\le b$$

then

$$A(i,a,b)=\sum_{x=0}^{min\{a,b-i\}}B(i,a,b,x)$$

where $0\le i\le b$, $a=n_{j-1}$, and $b=n_j$. When i is fixed, calculation of $A(i,n_{j-1},n_j)$ requires values of $B(i,n_{j-1},n_j,x)$ for $0\le x\le min\{n_{j-1},n_j-i\}$. Rewrite this as $B(n_j-\beta,n_{j-1},n_j,x)$ for $0\le x\le min\{n_{j-1},\beta\}$ with $\beta=n_j-i$. Now we can calculate these values for $0\le\beta\le n_j$ and, hence, for $0\le i\le n_j$ using recursive equations:

$$B(n_j,n_{j-1},n_j,0)=1$$

$$B(n_j-(\beta+1),n_{j-1},n_j,x)=\frac{(n_j-(\beta+1)+x)(n_j-\beta)B(n_j-\beta,n_{j-1},n_j,x)}{2} \tag{11}$$

$$B(n_j-(\beta+1),n_{j-1},n_j,(\beta+1))=\frac{(n_j-\beta)(n_{j-1}-\beta)B(n_j-\beta,n_{j-1},n_j,\beta)}{(\beta+1)^2} \tag{12}$$

We use equation (11) for $0\le x\le min\{n_{j-1},\beta\}$. If $min\{n_{j-1},\beta+1\}\ne min\{n_{j-1},\beta\}$ and therefore $min\{n_{j-1},\beta+1\}=min\{n_{j-1},\beta\}+1=\beta+1$ then we also use equation (12). The cost of this calculation is dominated by the number of the summands $B(n_j-\beta,n_{j-1},n_j,x)$ for $0\le x\le min\{n_{j-1},\beta\}$ and $0\le\beta\le n_j$. This number is $O(n_j^2)$.

Now we need to calculate $A(0,n_{j-1},n_j+\alpha),\dots,A(n_j+\alpha,n_{j-1},n_j+\alpha)$ for $1\le\alpha\le N_{j+1}$. Having $A(0,n_{j-1},n_j+\alpha),\dots,A(n_j+\alpha,n_{j-1},n_j+\alpha)$ calculated (note that we have already calculated these values for $\alpha=0$), we can calculate $A(0,n_{j-1},n_j+(\alpha+1)),\dots,A(n_j+(\alpha+1),n_{j-1},n_j+(\alpha+1))$ using equations

$$A(i,a,b+1)=\begin{cases}\frac{(b+1)b}{2}A(i,a,b)+\binom{b+1}{i}\binom{a}{b+1-i} & \text{if } b-i<a,\\ \frac{(b+1)b}{2}A(i,a,b) & \text{if } a\le b-i \text{ and}\\ & A(b+1,a,b+1)=1\end{cases} \tag{13}$$

where $0 \le i \le b$, $a = n_{j-1}$, and $b = n_j + \alpha$.

Moreover, for each α, we can optimize calculation of the second summands in the first case of equation (13) using recursion

$$\binom{n_j+\alpha}{i+1}\binom{n_{j-1}}{n_j+\alpha-(i+1)} = \frac{(n_j+\alpha-i)^2}{(i+1)(n_{j-1}-n_j-\alpha+i+1)}\binom{n_j+\alpha}{i}\binom{n_{j-1}}{n_j+\alpha-i} \tag{14}$$

To apply this recursion we need an initial value. This value depends on n_{j-1}, n_j, and α and, since only the first case of equation (13) contains the second summand, it is not necessary the value of this summand for $i = 0$. There are 3 cases for calculation of all the initial values that are necessary at step j.

Case 1: $N_j < n_{j-1}$. That means that we use the first case of equation (13) for all α. In this case, for all α, the initial value for recursion (14) is the value of the second summand in (13) when $i = 0$. So we need to calculate $\binom{n_j+\alpha}{0}\binom{n_{j-1}}{n_j+\alpha+0}$ for $1 \le \alpha \le N_{j+1}$ and those are $\binom{n_{j-1}}{n_j+1}, \ldots, \binom{n_{j-1}}{n_j+N_{j+1}}$. This takes $O(N_j)$ steps.

Case 2: $n_j + \alpha_0 = n_{j-1}$ for some $\alpha_0 \in \{0, \ldots, N_{j+1}\}$. If $\alpha_0 > 0$ then, for all $\alpha \le \alpha_0$, we use the first case of (13) and therefore we need to calculate $\binom{n_{j-1}}{n_j+1}, \ldots, \binom{n_{j-1}}{n_j+\alpha_0}$, this takes $O(\alpha_0)$ steps. For all $\alpha > \alpha_0$, we use the second case first and, starting from $i = \alpha - \alpha_0$, we use only the first case. So we need to calculate $\binom{n_j+\alpha}{\alpha-\alpha_0}\binom{n_{j-1}}{n_j+\alpha-(\alpha-\alpha_0)}$ for $\alpha \in \{\alpha_0 + 1, \ldots, N_{j+1}\}$, which are $\binom{n_{j-1}+1}{1}, \ldots, \binom{n_{j-1}+(N_{j+1}-\alpha_0)}{(N_{j+1}-\alpha_0)}$. This takes $O(N_{j+1} - \alpha_0)$ steps. In total, we have $O(N_{j+1})$ steps.

Case 3: $n_{j-1} < n_j$. That means that, for all α, we use the second case first and, starting from $i = n_j + \alpha - n_{j-1}$, we use only the first case. Calculate $\binom{n_j+1}{n_{j-1}}, \ldots, \binom{n_j+N_{j+1}}{n_{j-1}}$ for $O(n_{j-1} + N_{j+1})$ steps.

Each case costs at most $O(N_j)$. Provided that the initial values are calculated, the cost of the rest calculation at step j is dominated by the number of the coefficients $A(i, n_{j-1}, n_j + \alpha)$ for $0 \le i \le n_j + \alpha$ and $1 \le \alpha \le N_{j+1}$ and it is $O(N_j^2)$. Summing up $O(n_j^2)$, $O(N_{j-1})$, and $O(N_j^2)$ gives us the cost of each step j, which is $O(N_j^2)$. Since $1 < j \le m$ and calculation of $R(i)$ for $0 \le i \le N_1 = n$ takes $O(n)$, the algorithm does $O(mn^2)$ steps in total.

Appendix 2: Algorithm for counting fully ranked resolutions of a fully ranked constraint tree

The algorithm for counting resolutions of a constraint tree requires a few changes to count resolutions of a fully ranked constraint tree. Recall that, at each step s, we calculated the set S_s which consists of the numbers of resolutions of intermediate trees with s leaves. To construct S_s, for each element of S_{s-1}, we proposed a collection of tuples that may define intermediate trees with s leaves. To accept eligible tuples we defined two matrices $\mathtt{m}$ and $\mathtt{M}$. The general scheme for counting resolutions of a fully ranked constraint tree is the same. However, we need to define matrices $\mathtt{m}$ and $\mathtt{M}$ for a fully ranked constraint tree and describe the procedure of proposing new tuples. The procedure becomes more technical because of additional ranking.

We will calculate $S^r(\mathbf{n}, r)$ for the corresponding fully ranked constraint tree $\mathfrak{T}$. Let

$$a_{i,j} = |\{x \mid \text{ node } i \text{ is a parent of node } x \text{ and } r(x) \le j\}|$$

for $1 \le i \le k$ and $r(i) \le j \le l - 1$, i.e., $a_{i,j}$ is the number of interior nodes that are children of node i and have rank at most j. Define two matrices $\mathtt{m}$ and $\mathtt{M}$ of size $k \times (l-1)$ such that

$$\mathtt{M}_{i,j} = N_{i,j+1} \quad \text{and} \quad \mathtt{m}_{i,j} = \begin{cases} 2 \text{ if } a_{i,j} = 0;\\ 1 \text{ if } a_{i,j} > 0 \text{ and } \mathtt{M}_{i,j} > 0;\\ 0 \text{ otherwise.}\end{cases}$$

As before, if we consider a horizontal line that is strictly between the horizontal line that passes through the nodes of rank j and the horizontal line that passes through the nodes of rank $(j + 1)$ then these matrices determine the minimal and maximal possible numbers of intersections of this line with branches of subtree i.

Let $\mathbf{x} = [x_{i,j}]$ be a matrix of size $t \times q$, where $t \le k$ and $q \le l$, and $\mathbf{t} = \{1, \ldots, t\}$. A tuple $(\mathbf{x}, r|_{\mathbf{t}})$ is eligible if

- $x_{i,j} = n_{i,j}$ for $1 \le i \le t$ and $1 \le j < q$, and
- $\mathtt{m}_{i,q-1} \le x_q^i \le \mathtt{M}_{i,q-1}$ for $1 \le i \le t$.

Having an intermediate tree with $s-1$ leaves, we need to consider all the possible ways to transform this tree to a tree with s leaves. First we need to add a new leaf to some subtree. We give the highest rank to this leaf. After that we may add new time points below the last time point. If we add new time points, we should rerank the leaves with the highest rank such that the new tree has enough leaves at each time point. Let $\mathbf{x} = [x_{i,j}]$ be a matrix of size $t \times q$ for $t \leq k$ and $q \leq l$. This matrix represents an intermediate tree with $s-1$ leaves. Suppose we add a new leaf to subtree i_0, where $1 \leq i_0 \leq t$. Let q_0 be the number of time points in a new tree and, therefore, $q \leq q_0 \leq r(t+1)$ (or $q \leq q_0 \leq l$ if $t = k$). Define a function $addone((\mathbf{x}, r|_{\mathbf{t}}), i_0, q_0)$ which returns a tuple $(\mathbf{x}', r|_{\mathbf{t}})$, where $\mathbf{x}' = [x'_{i,j}]$ is a matrix of size $t \times q_0$ such that

- $x'_{i_0,q_0} = x_{i_0,q} - \sum_{j=q}^{q_0-1} n_{i_0,j} + 1$,
- $x'_{i,q_0} = x_{i,q} - \sum_{j=q}^{q_0-1} n_{i,j}$ for $i \neq i_0$,
- $x'_{i,j} = x_{i,j}$ for $1 \leq i \leq t$ and $j < q$, and
- $x'_{i,j} = n_{i,j}$ for $1 \leq i \leq t$ and $q \leq j < q_0$.

If this tuple is eligible then it represents a new intermediate tree with s leaves.

At each step s of the algorithm, for each tuple $(\mathbf{x}, r|_{\mathbf{t}})$ such that $F^r(\mathbf{x}, r|_{\mathbf{t}}) \in S_{s-1}$, we need to propose a collection of new tuples. The first part of this procedure is Algorithm 4. To stop the procedure, when we recognise that a new tree can not have x time points, we use a predicate $ext(x)$ which is true if we can extend the number of time points in a new tree to x. The number of time points in a new tree can not always be extended to any arbitrary number because there may not be enough leaves of the highest rank to rerank them in such a way that there will be $n_{i,j}$ leaves of each new rank j. So $ext(q)$ is always true because the tree to which we add a new leaf already has q time points.

Algorithm 4 Working with a tuple $(\mathbf{x}, r|_{\mathbf{t}})$ such that $F^r(\mathbf{x}, r|_{\mathbf{t}}) \in S_{s-1}$ (part 1)

```
q_0 = q
while q_0 ≤ r(t+1) (or q_0 ≤ l if t = k) and ext(q_0) do
  for i = 1 → t do
    (x', r|_t) = addone((x, r|_t), i, q_0)
    if eligible((x', r|_t)) then
      calculate F^r(x', r|_t) and add it to S_s
    end if
    if all the proposed tuples were not eligible then
      ext(q_0 + 1) = false
    end if
    q_0 = q_0 + 1
  end for
end while
```

If $t < k$ then the proposing procedure contains the second part. In this case, we also can increase the number of leaves in an intermediate tree by adding a new subtree. That means that one of the leaves of the tree becomes a parent to two new leaves. Again we may add new time points. Let q_0 be the number of time points in a new tree and, therefore, $r(t+1) < q_0 \leq r(t+2)$ (or $r(t+1) < q_0 \leq l$ when $t+1 = k$). We define another function $addconstraint((\mathbf{x}, r|_{\mathbf{t}}), q_0)$ which returns a tuple $(\mathbf{x}', r|_{\mathbf{t+1}})$, where $\mathbf{x}' = [x'_{i,j}]$ is a matrix of size $(t+1) \times q_0$ such that

- $x'_{t+1,q_0} = 2 - \sum_{j=r(t+1)+1}^{q_0-1} n_{t+1,j}$,
- $x'_{t+1,j} = n'_{t+1,j}$ for $1 \leq j < q_0$,
- $x'_{i,q_0} = x_{i,q} - \sum_{j=q}^{q_0-1} n_{i,j}$ for $1 \leq i \leq t$,
- $x'_{i,j} = x'_{i,j}$ for $1 \leq i \leq t$ and $j < q$, and
- $x'_{i,j} = n'_{i,j}$ for $1 \leq i \leq t$ and $q \leq j < q_0$.

This tuple defines a new tree if it is eligible. The second part of the procedure is Algorithm 5. The values of q_0 and $ext(q_0)$ are as after running Algorithm 4.

Algorithm 5 Working with a tuple $(\mathbf{x}, r|_{\mathbf{t}})$ such that $F^r(\mathbf{x}, r|_{\mathbf{t}}) \in S_{s-1}$ (part 2)

```
while q_0 ≤ r(t+2) (or q_0 ≤ l when t+1 = k) and
ext(q_0) do
  (x', r|_{t+1}) = addconstraint((x, r|_t), q_0)
  if eligible(x', r|_{t+1}) then
    calculate F^r(x', r|_{t+1}) and add it to S_s
  else
    ext(q_0 + 1) = false
  end if
  q_0 = q_0 + 1
end while
```

Finally, as before, the main algorithm calculates sets S_s recursively for $0 \leq s \leq n$, where n is the number of leaves in a constraint tree $\mathfrak{T}$, and a unique element of S_n is $S^r(\mathbf{n}, r)$.

Let m be a number of sampling points, i.e., $m = l - k$.

Proposition 2. *If k is fixed then the algorithm does at most $O(m)$ steps for each eligible tuple.*

Proof. Let $(\mathbf{x}, r|_{\mathbf{t}})$ be an eligible tuple. First, having all the necessary summands for equations (9) and (10), we need to calculate $F^r(\mathbf{x}, r|_{\mathbf{t}})$. This takes at most $O(t \times [\, r(t+1) - r(t)])$ steps. Since we assume k is a constant and since $t \leq k$ and $r(t+1) - r(t) \leq m$, it is $O(m)$. Second, we need to perform the procedure of proposing new tuples for this tuple. The most expensive part here is calculation of the last columns of matrices $\mathbf{x}'$ because it involves calculation of the sums $\sum_{j=q}^{q_0-1} n_{i,j}$. Note that, storing the values of

$x_{i,q} - \sum_{j=q}^{q_0-1} n_{i,j}$ for $1 \leq i \leq t$ at each step q_0, we can optimise these calculations. Then it takes $O(t \times (q - r(t+2))$ steps and it is $O(m)$ again. □

Proposition 3. *If k is fixed then the algorithm has $O(m^2 n^k)$ time complexity.*

Proof. Let i_x be such a number that $r(i_x) \leq x < r(i_x+1)$. Then the number of eligible tuples is

$$\sum_{j=1}^{l-1} \prod_{i=1}^{i_j} (\mathtt{M}_{i,j} - \mathtt{m}_{i,j} + 1)$$

Note that

$$\mathtt{M}_{i,j} - \mathtt{m}_{i,j} + 1 \leq N_{i,j} + 1 \leq N_{i,r(i)+1} + 1 = b_i + 1$$

where $b_1 + \ldots + b_k = n + k - 1$. Then

$$\sum_{j=1}^{l-1} \prod_{i=1}^{i_j} (\mathtt{M}_{i,j} - \mathtt{m}_{i,j} + 1) < (l-1)(b_1+1)\ldots(b_k+1)$$
$$< (l-1)(\frac{n}{k})^k = O(mn^k)$$

From this and Proposition 2, it follows that the algorithm does at most $O(m^2 n^k)$ steps. □

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
The authors equally contributed to conceive the work. AG drafted the manuscript and DW and AJD revise it. All authors read and approved the final manuscript.

Acknowledgements
AJD was partially funded by a Rutherford Discovery Fellowship from the Royal Society of New Zealand.

Author details
[1]Department of Computer Science, The University of Auckland, Auckland, New Zealand. [2]Allan Wilson Centre for Molecular Ecology and Evolution, University of Auckland, Auckland, New Zealand.

Received: 15 February 2013 Accepted: 8 October 2013
Published: 28 October 2013

References

1. Yang Z, Rannala B: **Bayesian phylogenetic inference using DNA sequences: a Markov chain Monte Carlo method.** *Mol Biol Evol* 1997, **14**(7):717–724.
2. Yang Z, Rannala B: **Bayesian estimation of species divergence times under a molecular clock using multiple fossil calibrations with soft bounds.** *Mol Biol Evol* 2005, **23:**212–226.
3. Heled J, Drummond A: **Tree priors for relaxed phylogenetics and divergence time estimation.** *Syst Biol* 2012, **61:**138–149.
4. Rodrigo AG, Felsenstein J: *The Evolution of HIV*. Baltimore: Johns Hopkins Univ Press; 1999.
5. Drummond A, Pybus O, Rambaut A, Forsberg R, Rodrigo A: **Measurably evolving populations.** *Trends Ecol Evol* 2003, **18:**481–488.
6. Felsenstein J: *Inferring Phylogenies*. Sunderland: Sinauer Associates; 2004.
7. Murtagh F: **Counting dendograms: a survey.** *Discrete Appl Math* 1984, **7:**191–199.
8. Gordon AD: **A review of hierarchical classification.** *J R Stat Soc A* 1987, **150**(2):119–137.
9. Semple C, Steel M: *Phylogenetics*. New York: Oxford University Press; 2003.
10. Griffiths RC: **Counting genealogical trees.** *J Math Biol* 1987, **25:**423–431.
11. Felsenstein J: **The number of evolutionary trees.** *Syst Zool* 1978, **23:**27–33.
12. Stadler T: **Sampling-through-time in birth-death trees.** *J Theor Biol* 2010, **267**(3):396–404.
13. Stadler T, Kouyos RD, von Wyl, V, Yerly S, Böni J, Bürgisser P, Klimkait T, Joos B, Rieder P, Xie D, Günthard HF, Drummond A, Bonhoeffer S, the Swiss HIV Cohort Study: **Estimating the basic reproductive number from viral sequence data.** *Mol Biol Evol* 2012, **29:**347–357.

doi:10.1186/1748-7188-8-26
Cite this article as: Gavryushkina *et al.*: Recursive algorithms for phylogenetic tree counting. *Algorithms for Molecular Biology* 2013 **8**:26.