Algorithm for Perfect Phylogeny and Gene Reconciliation

av

Basam Al Nashéa

2023 - M1

Algorithm for Perfect Phylogeny and Gene Reconciliation

Basam Al Nashéa

Självständigt arbete i matematik 30 högskolepoäng, Avancerad Nivå

Handledare: Lars Arvestad

2023

Abstract

A Perfect Phylogeny T_G for a set of genes G describes their common evolutionary lineage under the constraint that no two properties have evolved in separate places within that lineage. The problem of Reconciliation, under one particular model, is that of mapping the evolutionary tree T_G to a species tree S of the species that harbor the genes in G, with the goal of annotating the internal vertices with duplication or speciation. The biological significance is that this provides insight into which gene sequences are paralogous and orthologous. We develop a combined algorithm for excecuting both tasks based on merging two existing algorithm for each separate task of finding a perfect phylogeny, and then reconciling it.

Acknowledgements

I want to thank my supervisor Lars Arvestad for suggesting this topic and for guiding me through it.

I would also like to thank Lethius, wherever you are.

Finally, and most importantly, I would like to thank my parents for their encouragement through my entire studies. Your kindness and support is seemingly endless. *Thank you Mom & Dad*

Contents

1 Introduction

Phylogenetic trees are used in biology to model the evolutionary history and common ancestry of genes or species, and to study the relationship between them. By considering the elements of such a tree to be a sequence of characters (such as, for example, a DNA-string), the problem of constructing a phylogenetic tree is turned in to a mathematical task. A constrained version, the *perfect* phylogeny problem, considers lineages in which no property is evolved in two different places, and an algorithm for constructing such a phylogeny is given by R. Agarwala and D. Fernández-Baca [AFB94]. A different, but related problem, is that of reconciliation. By considering gene-trees, which are phylogenetic trees that consist of gene sequences as vertices, one can refine the lineage data by comparing the structure of the gene-tree with a known species tree of the species that harbor those genes. The method can be used to fill in informational gaps in terms of data loss, gene loss, gene duplications, and more. Completing the gene tree in terms of duplications is essential for inferring orthologous and paralogous sequences, which among other things, has implications for protein functions. A common approach for reconciliation is to minimize the number of duplications needed and a simple method for reconciliation is provided by C.M. Zmasek and S.R. Eddy [ZE01].

When reconciling a phylogenetic tree, the structure of that tree is vital to the outcome. Looking at the perfect phylogeny problem in isolation, a tree is constructed with no regard to any impact of the structure of the tree on any reconciliation. To our knowledge, there does not currently exist any algorithm that incorporates this impact when constructing a perfect phylogeny. In this thesis, we take a first step towards such an advancement by showing how one can merge the two algorithms previously mentioned.

Each chapter is self-contained and assumes only basic knowledge of graph theory and algorithms. In Chapter 2, we provide necessary and detailed theory to understand the algorithm of [AFB94]. Chapter 3 covers the topic of reconciliation and gives an explanation of the algorithm of [ZE01]. In Chapter 4, we discuss the incompatibilities between the two procedures and provide a method for dealing with these. Then, in Chapter 5 we state the main result and explain the modifications we have made. Finally in Chapter 6, we discuss different ways in which the main result can be improved. The Appendix A is reserved for lengthy examples.

2 Perfect Phylogeny

Our goal for this chapter is to describe in detail an algorithm that solves the perfect phylogeny problem. The algorithm we have chosen to describe and will ultimately modify in later chapters is the one given by R. Agarwala and D. Fernández-Baca [AFB94]. We will closely follow this article, providing the same framework and results which are needed in order to understand the correctness of the algorithm, while often deviating in the level of detail of the proofs, the discussions, and also deviating entirely on some of the topics covered; while the article provides notes on the complexity and running time of the procedures, we will instead omit these since they are not relevant to our goal. The curious reader is referred to the article itself.

2.1 Type I splits

To start, we will introduce the central concept of this chapter, namely a perfect phylogeny. The idea is that given a set of character sequences, such as for example a set of genes, we would like to model their evolutionary history while restricting the structure of that history in certain ways. Let us formalize what we mean.

We have a character set $C = \{1, \dots, m\}$ in which the elements are called characters. For any character $c \in C$ we have the allowable states $A_c = \{\alpha_0, \dots, \alpha_{r_c}\}$, the elements of which will usually be referred to as *character states of c*. A sequence $s = (s_1, \dots, s_m)$ is an element of the multiset of sequences $S \subseteq A_1 \times \cdots \times A_m$. That is to say, we allow multiples of each sequence s. The element s_c will be called the *state of s on c*.

As an example consider DNA-sequences. These are represented by strings of characters in which each character has the allowable states {A,C,G,T} which symbolize the adenine, cytosine, guanine, and thymine bases found in DNA sequences. When referring to sequences we will sometimes omit the parenthesis and commas, for example a species of (A, B, A) would be written ABA.

We can now define the central concept of this chapter.

Definition 2.1 (Perfect Phylogeny). *Given a set of species S, if there exists a tree T_S such that*

(i) $S \subseteq V(T_S) \subseteq A_1 \times \cdots \times A_m$,

(ii) each leaf in T_S is an element of S,

(iii) for each $c \in C$ and each $k \in A_c$, the set of all $v \in V(T_S)$ such that $v_c = k$ induces a subtree of T_S,

*then we say that T_S is a **Perfect Phylogeny** of S.*

The *perfect phylogeny problem* is that of determining whether a perfect phylogeny exists and if so, produce one. The tree T_S, if it exists, models a possible evolutionary history of the species (sequences) in S and how they have been derived from common ancestry. The first two conditions of 2.1 are quite straightforward, (i) tells us the species we consider should be contained in the tree, and that the tree is made up of species, while (ii) tells us that we only infer ancestors of the elements of S. The first two conditions define a phylogeny, and a phylogeny is said to be perfect if it also satisfies (iii). The third condition imposes the evolutionary relation that character states can not be evolved by two different species. Hence if two different species has a character state in common then they must both have a common ancestor from which the state is inherited. This can be considered too restrictive for real data sets, the authors in [FBL03] for example consider *"near-perfect"* phylogenies in which sets of species are allowed to deviate from (iii) by a fixed parameter measure.

The perfect phylogeny problem can be stated in a multitude of different ways, such as in terms of $n \times m$ matrices in which each row is a species, and each column corresponds to a character. One can state the problem such that T_S needs to be rooted, with or without specifying an explicit starting vertex. In this case the perfect phylogenies are technically unrooted, however we will sometimes think of them as being rooted as this will be required in later chapters. We will explain how we can do this once we have introduced splits of type I and II. The authors in [AFB94] are the first to have shown that the perfect phylogeny problem is solvable in polynomial time for any fixed maximum number of character states. The algorithm constructed takes a dynamical programming approach to construct perfect phylogenies from the bottom up.

We would like to note here that the species set S and vertex set $V(T_S)$ can be thought of in two ways, as a set or as a multiset. In practice a species may occur many times so it is convenient for us to allow this. However, for the sake of simplifying some proofs we may need to restrict S and V to contain distinct elements. Any vertex $v \in V(T_S)$ can be safely replaced with multiple copies $v_1, \dots, v_k$ with an arbitrary tree between these, as this will not affect perfect phylogeny. Similarly if there are multiples then these must be directly connected to each other due to condition (*iii*) of Definition 2.1. So we may safely remove all $v_1, \dots, v_k$ and replace them with a single copy v that has all the connections and neighbours of all the previous multiples.

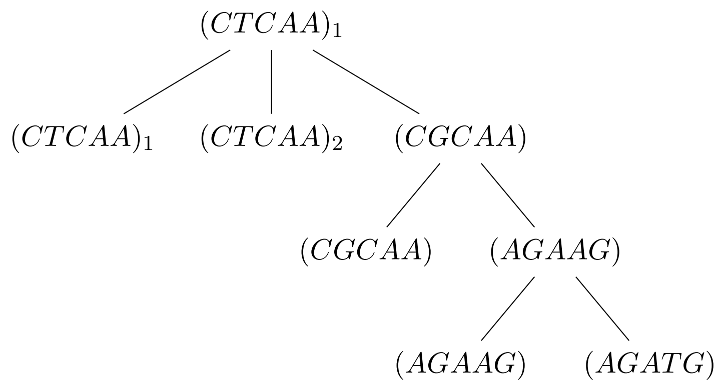
Although the converse of condition (*ii*) does not need to hold, we may consider it to be true if needed. That is, all the species of S are leaves in T_S, if any $s \in S$ is not a leaf in T_S then we could add another copy of s as a vertex in T_S and add an edge between the original and the copy, making the newly added vertex a leaf.

Now let us illustrate Definition 2.1 with an example.

Example 2.2. Let $C = \{1, 2, 3, 4\}$ and $A_i = \{A, C, G, T\}$ for each $i \in C$. Consider the species set

$$S = \left\{ \begin{array}{l} \texttt{(AGATG)}, \\ \texttt{(CGCAA)}, \\ \texttt{(AGAAG)}, \\ \texttt{(CTCAA)}_1, \\ \texttt{(CTCAA)}_2 \end{array} \right\}$$

Then as one perfect phylogeny we get $T_S =$



We can easily verify that each character state induces a subtree. In this example we did not need to infer the existence of any species outside our set, later on we will see examples where that is not the case. The species $(CTCAA)$ has a multiplicity of 2, and in T_S many of the species appear in multiple vertices. Removing the duplicates would also give a perfect phylogeny.

It is not always the case that a perfect phylogeny exists for a set of species, as shown in the following example.

Example 2.3. Consider the species set $S := \{11, 01, 10, 00\}$. The species 01 has to have an edge to 11 since they share a character state 1 on the second character, and no other species does. Similarly 10 shares a character state with 11 on the first character. But for analogous reasons the species 01 and 10 must also both have edges to 00, forcing the existence of a loop. Hence there can be no perfect phylogeny for S.

One important consequence to Definition 2.1 is that, if we would like a vertex v to belong to a perfect phylogeny and if v is on the path between two species with equal character state on c then the v itself must also have that character state on c. So in other words if v is a vertex of some perfect phylogeny T_S which lies on the path between two species a, b with $a_c = b_c = \alpha$, then $v_c = \alpha$. We note an important detail. Whenever we have a perfect phylogeny T_S then we may safely assume that for any $v \in V(T_S)$ and any $c \in C$ we have that v_c is the character state of some $s \in S$.

We now introduce some of the key concepts used in [AFB94] that are vital to the understanding of their algorithm. These are the ideas of *distinguishing characters*, and *splits*. We would like some characteristics of a set of species which separate them from the rest of the set in terms of their own branch in any perfect phylogeny. If we look back to condition (*iii*) from Definition 2.1, this condition gives us a way to look at a tree as subtrees distinguished by the character states that are unique to them. See Figure 1. In general if some subset of species has a *distinguishing* character, one which differs in all states from the rest of the species, then we are inclined to try to view these as their own respective branches in a perfect phylogeny. For this to work we need to put limitations on the characters which are not distinguishing so we can connect the two separate sets back together. With all this in mind we make the following definition.

Definition 2.4. *Let $G \subset S$ and let $G' = S \setminus G$ in S. The set $D(G)$ is the set of all characters $c \in C$ such that for all $a \in G$ and $b \in G'$ we have $a_c \neq b_c$. The set $D(G)$ is called the **distinguishing characters of** G. The **common characters of** G is the set $M(G) = C \setminus D(G)$. For any $c \in C$ define*

$$M_c(G) := \{\alpha \in A_c \mid \exists s \in G, t \in G' : s_c = t_c = \alpha\}$$

$$D_c(G) := \{\alpha \in A_c \mid \exists s \in G : s_c = \alpha, \forall t \in G' : t_c \neq \alpha\}.$$

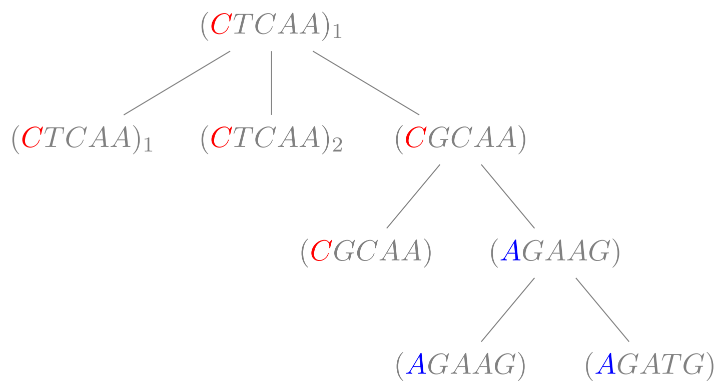


Figure 1: The character state C on the first character, marked in red, is distinct to the set on the upper left. The character state A, marked in blue, is distinct to the lower right part of the tree. If we look at the species without having the complete perfect phylogeny first, we could easily suggest that the red and blue should be their own branches since they completely differ on the first character.

*$M_c(G)$ and $D_c(G)$ are called the set of **common character states** and **distinguishing character states** of G on c respectively.*

While M and D denote the common and distinguishing characters, the sets M_c and D_c denote the set of states on those characters. Note that for any $G \subset S$ and $G' = S \setminus G$ we have $D(G) = D(G')$, $M(G) = M(G')$, and $M_c(G) = M_c(G')$ for all c. However we always have $D_c(G) \cap D_c(G') = \emptyset$. The concepts of $M_c(G)$ and $D_c(G)$ are not explicitly defined in [AFB94], we have added them for convenience. Now we introduce the concept of splits.

Definition 2.5. *Let $G \subset S$ and $G' = S \setminus G$. Then we call (G, G') a **split** if $\forall c \in M(G)$ we have that*

$$|M_c(G)| \leq 1.$$

*If (G, G') is a split then we call G and G' **partitions** and m_c denotes the singular element of $M_c(G)$ for each c. If also $D(G) \neq \emptyset$ then we call (G, G') a **c-split** and we call G and G' **c-partitions**.*

Remark. This implies that if $|G| = 1$ or $|S \setminus G| = 1$ then (G, G') is trivially a split.

We ask in 2.5 that for each character there will not be more than one character state in common between G and G'. If there were two or more then that particular choice of (G, G') would not be appropriate for a split, since in that case we would need to connect G and G' in multiple places when building T_S, which is not necessarily a problem but it is not what we want. The following Lemma is the first result presented in [AFB94] and suggests that if G or G' does not have a perfect phylogeny then neither does S.

Lemma 2.6. *Let S be a set of species. Then S has a perfect phylogeny if and only if every subset of S has one.*

Proof. If every subset of S has a perfect phylogeny then S as a subset of itself has one. Conversely assume that S has a perfect phylogeny T_S. Let $G \subset S$ and let T_G denote the tree T_S where we have removed any leaf in T_S which is not in G. Then T_G satisfies $G \subset V(T_G) \subset A_1 \times \cdots \times A_m$ hence (i) of Definition 2.1 holds. But also (iii) holds because we have not removed any internal node of T_S to obtain T_G. Since we removed the leaves which are not in G we know that also (ii) holds. $\square$

This lemma tells us that if we are ever in a situation where a set $G \subset S$ does not have a a perfect phylogeny then we can immediately stop the procedure because there will not be one for S either. If the set G does have one however, we would like a way to use this to build a perfect phylogeny for S. So we would want to find a split (G, G') with a perfect phylogeny for each partition and glue the two trees together. If the two partitions do have a perfect phylogeny then in order to glue them together we would need a species which has the appropriate character state in the common characters of G and G' so as to preserve condition (iii) of Definition 2.1. This leads us to the following definition.

Definition 2.7. *A vector $s \in A_1 \times \cdots \times A_m$ is said to be* ***compatible*** *with a subset $G \subset S$ if for all $c \in M(G)$, $s_c = m_c$. Likewise we say in that case that G is compatible with s.*

Now we are ready to define the different kinds of splits which the article describes.

Definition 2.8. *For any split (G, G'), if there exists a species $s \in G$ that is compatible with G, and if also $|G \setminus \{s\}| \geq 1$ and $|G'| \geq 1$, then we call this a split of type I and we call s a* ***connecting species****. Otherwise we say it is of type II.*

In other words we ask that if G and G' have common character states, which are at most 1 each in the case where (G, G') are a split, then s must have that common character state for each character. This makes s fitting for connecting the respective trees together. In the recursive algorithm which [AFB94] builds up to, one of the first steps is to search for type I splits (G, G'), then with a connecting species s call the algorithm again with $G \cup S$ and $G' \cup s$ as inputs separately. If both calls succeed then the perfect phylogenies T_G and $T_{G'}$ can be connected to one for S through s. We try to do this as far as possible, however we will sometimes run in to the case where all the remaining splits are of type II. We will see in the following section how to deal with these. Note that we may think of s as a root of T_G and of $T_{G'}$

Lastly for this section we would like to mention a key detail which is not explicitly worked out in the article. When dealing with type I splits, the algorithm that is constructed in [AFB94] works exclusively with c-splits and the reason for this is that the complexity of the algorithm reduces significantly in that case. We want to comment on the justification of doing it this way. Lemma 2.6 tells us that if there are c-splits then we could restrict ourselves to looking at these. However, it might be that S contains only splits with no distinguishing characters. In that case we would argue that all the species of S are the same. We state the following theorem and provide a proof for it by induction.

Theorem 2.9. *If S has no c-splits then S contains only one element with multiplicities.*

Proof. If $|S| = 1$ then the claim is trivial. Take S to be such that $|S| = n$ and such that S has no c-splits. Let S' be the species set S with any one species removed, and assume for the sake of reasoning by induction that the claim holds for any species set of size $n - 1$. To apply the inductive hypothesis to S', we only need to show that S' has no c-splits. Assume that S' has a c-split (G, G'), then both $(G \cup \{s\}, G')$ and $(G, G' \cup \{s\})$ are not c-splits since we assumed that S has none. We must have for every $c \in D(G)$ that $s_c \in D_c(G)$ and $s_c \in D_c(G')$. This can not happen however since $D_c(G) \cap D_c(G') = \emptyset$. So S' can not have any c-splits.

Since $|S'| = n - 1$ and S' has no c-splits, by the induction hypothesis, S' contains nothing other than one element t with multiplicities. Then because $S = S' \cup \{s\}$ has no c-splits, s can not be different from t since otherwise $(\{s\}, S')$ would be a c-split in S. Hence $s = t$ and that concludes the proof by induction. □

As we have mentioned before, the case where S contains multiples, or in this case only one element with multiples, causes us no problems due to how perfect phylogenies are defined. We can easily move between both cases at our convenience.

This concludes the discussion on type I splits and we can now begin to cover the type II case.

2.2 Type II splits

In the last section we introduced the concept of a Perfect Phylogeny and some important definitions related to these. The algorithmic approach which the article takes is dependent on splits of various types, which are partitions in the species set signifying a possible branching in the final perfect phylogeny. These splits are subdivided into two categories, type I which was covered in the last section, and type II which is the lengthier case to navigate. To start we have the following Lemma which is presented in [AFB94]. We will reconstruct the proof here in more detail.

Lemma 2.10. *If all c-splits are of type II, then in every perfect phylogeny T of S, each species $s \in S$ is a leaf in T.*

The idea of the proof is that if there are any $s \in S$ that are internal in a perfect phylogeny, then we could construct a type I c-split with s as connecting species.

Proof. Assume that all c-splits of S are of type II. Let T be any perfect phylogeny of S and assume that there exists some $s \in S$ which is not a leaf. Then let T' be any connected component of $T \setminus \{s\}$ and let G' be the set of species of T', with $G := S \setminus G'$.

We start by showing that the pair (G, G') is a split. Take any $c \in M(G)$, if $|M_c(G)| = n > 1$ then there are n pairs (g, g') with $g \in G$, $g' \in G'$ where $g_c = g'_c$ and with all the pairs having different character states on c. Since s is a vertex separating G and G' in T, we must have that s lies on the path between each (g, g') in T. So s_c is forced in $n > 1$ ways which is impossible. Hence for each $c \in M(G)$ we must have $|M_c(G)| \leq 1$, and so (G, G') is a split.

Now we show that the split (G, G') is a c-split. Since T' is a single component it must contain one and only one t which is a neighbour of s. This t must differ in at least one character c from s. If any children of t were to have any common character state on c with any species in G, then s would be forced to have this character state, and hence also t. But this contradicts that $s_c \neq t_c$. So the split (G, G') is a c-split.

Finally we show that (G, G') is of type I. Since s is an internal vertex of T and since every leaf in T is a species according to *(ii)* of 2.1, we get that $|G \setminus \{s\}| \geq 1$ and $|G'| \geq 1$ because $s \in G$. Finally the species s

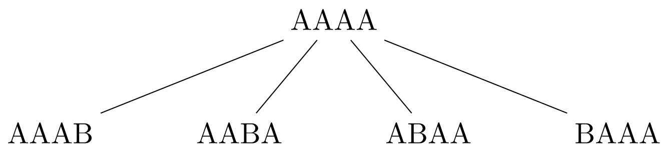
has the character states necessary to connect G and G', hence (G, G') is a c-split of type I in S. This contradicts the assumption that all c-splits are of type II, so we can not have any species $s \in S$ as a non-leaf. $\square$

Let us now look at an example concerning type II splits.

Example 2.11. Consider the species set

$$S = \begin{Bmatrix} AAAB, \\ AABA, \\ ABAA, \\ BAAA \end{Bmatrix}$$

One perfect phylogeny is given by $T_S =$



The example illustrates the concept of type II splits because there is no existing species which can act as a common ancestor to the others. Whichever subset we choose will not contain a species that is compatible with that set and its complement. So we need to infer the common ancestor $AAAA$.

From now on we only concern ourselves with the case where S has no type I splits, we may assume so for every S unless stated otherwise. Now since all the species are leaves in any perfect phylogeny, in order to connect subsets of S together we must do this through vertices in T that we infer, meaning ones that are not species. What properties would we require from such a vertex $x \in A_1 \times \cdots \times A_m$? Same as before this vertex is meant to be the connection between two subsets of species G, G', and so must at least be compatible with them in the sense of Definition 2.7. This time however, we are inferring the connection and so the state of such a connection on any $c \in D(G)$ is not forced. Therefor we impose that the state on those characters does not deviate from those of the elements of G. This leads to the following definition.

Definition 2.12 (Subphylogeny)**.** *A* ***subphylogeny*** $\widetilde{T}_G$ *of* $G \subset S$ *is a perfect phylogeny of* G *containing a vertex* x *that is compatible with* G *(with respect to* S*), such that* $x_c \in D_c(G)$ *for all* $c \in D(G)$*. The vertex* x *is called the* ***connection*** *of* $\widetilde{T}_G$*.*

In the previous section, on type I splits, we introduced the concept of a connecting species, and now we have introduced the concept of a connection. What is not explicitly mentioned in [AFB94] is that we can think of these as the *roots* of the respective trees, and it is convenient to think of them in this way when we build larger phylogenies from smaller ones. We should keep in mind that connecting two phylogenies together is always done at the roots, or connections, hence the name.

Recall that $D_c(G)$ denotes the set of distinguished character states of G. In other words the vertex x should have the character state of some species of G on each distinguished character. Now we want to show the importance of subphylogenies by relating them to the concept we are after, namely perfect phylogenies. To do so we will first need the following helpful Lemma.

Lemma 2.13. *Let the species set S be such that it has a perfect phylogeny T_S. For any edge $\{u, v\}$ in T_S let T_1 be the subtree of $T \setminus \{u, v\}$ containing u and let G_1 be the set $S \cap T_1$. Then there exists a tree $\widetilde{T}_1'$ such that it is a subphylogeny for G_1 with u as connection.*

Proof. Let T_2 be the subtree of $T \setminus \{u, v\}$ containing v and let $G_2 := S \setminus G_1$. It follows immediately that T_1 is a perfect phylogeny for G_1 since T_1 is a connected subtree of T_S. Now take any $c \in D(G_1)$ such that $u_c \notin D_c(G_1)$ and consider the set $B_c(G_1)$ of all $b \in V(T_1)$ such that $b_c = u_c$. Then take any $d \in V(T_1) \setminus B_c(G_1)$ such that d is a neighbour of some element of $B_c(G_1)$. If such a d does not exist then this would mean that $u_c = x_c$ for all $x \in V(G_1)$, contradicting that $u_c \notin D_c(G_1)$. Recall from the previous section that we may safely assume that each character state of each vertex is also the character state of some species. So either $d_c \in D_c(G_1)$ or $d_c \in D_c(G_2)$. However, the latter could not happen because we also have $u_c \neq d_c$, so if d_c occurs as a state in both T_1 and T_2 with no path between them, then condition (iii) would not be satisfied for T_S; hence $d_c \in D_c(G_1)$. Now let $\widetilde{T}_1'$ be the tree obtained by setting each b_c to be d_c, which gives the vertex u the desired property of having $u_c \in D_c(G_1)$ for all $c \in D(G_1)$ Thus the tree $\widetilde{T}_1'$ is a subphylogeny of G_1 with u as connection. □

Lemma 2.13 is not written separately as a Lemma in [AFB94] but rather as part of the proof of the next result. We have decided to state it separately because it will be used in multiple upcoming results such as the following useful fact.

Lemma 2.14. *S has a perfect phylogeny $\Leftrightarrow$ There exists a split (G, G') such that both G, G' have subphylogenies.*

Proof. ($\Leftarrow$) Let (G, G') be a split such that G, G' both have subphylogenies and let $\widetilde{T}, \widetilde{T}'$ be the respective subphylogenies with connections $x \in \widetilde{T}, x' \in \widetilde{T}'$. We may connect the trees $\widetilde{T}, \widetilde{T}'$ in different ways to obtain a perfect phylogeny of S. If $D(G) = \emptyset$ then $M(G) = M(G') = C$ and we must then have that the sequence defining the vertex x is the same as the sequence defining x'. So we simply identify $x = x'$ as vertices. Otherwise we add an edge between x, x' and the resulting tree will be a perfect phylogeny of S. Since the condition (i) of 2.1 holds for $\widetilde{T}, \widetilde{T}'$ it will hold for T_S. Since we also have not added any new vertices to obtain T_S, condition (ii) holds as well. Finally if there are no distinguishing characters of either partition then (iii) holds due to how x, x' are defined on their common characters. If instead $D(G) \neq \emptyset$ then condition (iii) still holds for each character of $D(G)$ since $\widetilde{T}, \widetilde{T}'$ are phylogenies, and for each character of $M(G)$ the condition holds due to how x, x' are defined.

($\Rightarrow$) Assume S has a perfect phylogeny T_S. Firstly we may assume that any element of $V(T_S)$ which is not in S has degree of at least 3. Otherwise by condition (ii) such an element v would need to have $deg(v) = 2$, and then we would get another perfect phylogeny by deleting v and adding an edge between the neighbours of v. The resulting tree satisfies (iii) because any subtree that passes through v will not be broken. The same is not true if $deg(v) > 2$.

Take any edge $\{u, v\} \in T_S$, then Lemma 2.13 gives us a subphylogeny for the subtree $\widetilde{T}_1 \subset T_S \setminus \{u, v\}$ containing u, applying Lemma 2.13 again with $\{u, v\}$ provides us a subphylogeny for the tree $\widetilde{T}_2 \subset T_S \setminus \{u, v\}$ which contains v. $\square$

Lemma 2.14 firstly guarantees us that pursuing subphylogenies is meaningful in an attempt of finding perfect phylogenies, since if we find a split with subphylogenies for each partition then they can be glued together to form a perfect phylogeny. Secondly, it tells us that if $G \subset S$ has no type II splits (G_1, G_2) with subphylogenies, then we may stop since there will not be a perfect phylogeny for G, and by Lemma 2.6 there will not be one for S. In the final algorithm of [AFB94], the subphylogenies are constructed exclusively from c-partitions, which is justified by the following Lemma.

Lemma 2.15. *Let $G \subset S$ be a partition. Then G has a subphylogeny if and only if there exists pairwise disjoint c-partitions $G_1, \ldots, G_k$ and a vector x such that*

- *$G = G_1 \cup \cdots \cup G_k$.*
- *x is compatible with each $G, G_1, \ldots, G_k$.*
- *Each G_i has a subphylogeny.*

Proof. ($\Leftarrow$) Let $\widetilde{T}_1, \ldots, \widetilde{T}_k$ be the respective subphylogenies for $G_1, \ldots, G_k$ with connections $x^1, \ldots, x^k$. Let $\widetilde{T}$ be the tree obtained by starting with x and $\widetilde{T}_1, \ldots, \widetilde{T}_k$, and adding edges $(x^1, x), \ldots, (x^k, x)$ to each $\widetilde{T}_i$. Since each $\widetilde{T}_i$ is a perfect phylogeny and x is compatible with each x^i, $\widetilde{T}$ is also a perfect phylogeny for G. If $D(G) = \emptyset$ then x is also a connection of G and so $\widetilde{T}$ is a subphylogeny of G.

Assume that $D(G) \neq \emptyset$ then consider any $c \in D(G)$. If $c \notin D(G_i)$ for any i then for each G_i there exists some G_j with $i \neq j$ such that G_i and G_j share a character state α on c. Since both G_i and G_j are compatible with x, and since $\alpha \in M_c(G_i)$ and $\alpha \in M_c(G_j)$, we must have that $x_c = \alpha$ and hence x has a character state on c of some species in G which makes x a connection of $\widetilde{T}$. If instead $c \in D(G_i)$ for some i then the connection x^i of G_i has $x^i_c \in D_c(G_i)$, so we may assign $x_c = x^i_c$ and get that x_c is the character state of some species in G. Again x is the connection of $\widetilde{T}$, which means that $\widetilde{T}$ is a subphylogeny of G.

($\Rightarrow$) Let $\widetilde{T}_G$ be a subphylogeny for G and let x be its connection. Our goal is to find sets $G_1, \ldots, G_k$ with the desired properties, the proof of the ($\Leftarrow$) part suggests that we should consider starting with x and its neighbours. So let $x_1, \ldots, x_k$ be the neighbours of x in $\widetilde{T}_G$ and let $\widetilde{T}_1, \ldots \widetilde{T}_k$ be the subtrees of $\widetilde{T}_G \setminus \{x\}$ containing $x_1, \ldots, x_k$ respectively. Take $G_i := S \cap \widetilde{T}_i$ for each $i = 1, \ldots, k$. Firstly we note that by Lemma 2.13 applied to each x_i, there exists a subphylogeny $\widetilde{T}'_i$ for each G_i. Next we note that each element of G is in some subtree $\widetilde{T}_i$, hence $G_1 \cup \cdots \cup G_k = G$. Since x is a connection of T_G, it is compatible with G. If G_i has a shared state on character c with $S \setminus G_i$ then any path between them must go through x, so x_c is forced and must be compatible with each G_i since $\widetilde{T}_G$ is a subphylogeny, and hence also a perfect phylogeny for which condition (iii) of Definition 2.1 is upheld.

Finally recall from our discussion in the previous section that we may safely assume that each vertex of any perfect phylogeny is distinct. So if we assume that $D(G_i) = \emptyset$ for some i, then x_i and x must be equal on each character, since they have exactly one character state in common on every $c \in C$. Hence $x_i = x$. But this contradicts the assumption that each vertex is distinct and that x_i is a neighbour of x. Then we must have that $D(G_i) \neq \emptyset$ for each i. This shows that all the $G_1, \ldots, G_k$ are c-splits which concludes the proof. □

So far, in this section dedicated to type II splits, we have made a definition of a suitable kind of perfect phylogeny that deals with this case, namely subphylogeny, and shown that we can restrict our search to c-splits, as well as showing that we can indeed stitch the subphylogenies together to form a perfect phylogeny. Behind all of this is the motivation for a recursive algorithm which works its way up from the simplest cases, that is what we are building towards. What we have at the moment is not enough to satisfy that goal. If G, G_1, G_2 are partitions of S with $G_1 \cup G_2 = G$ and $G_1 \cap G_2 = \emptyset$ then we could end up in one of three situations. Either both, or one, or none of G_1 and G_2 have subphylogenies. If none of G_1, G_2 have subphylogenies then we could simply keep searching among other partitions since by Lemma 2.14 there must be some split which has one if S is to have a perfect phylogeny. For the two other cases we formulate the following questions.

- Can we form a subphylogeny for G if G_1 and G_2 both have subphylogenies?
- What can we say about G if one of G_1 and G_2 has a subphylogeny?

The first question is answered by the following Lemma.

Lemma 2.16. *If $G, G_1, G_2 \subset S$ are partitions such that $G_1 \cup G_2 = G$ and $G_1 \cap G_2 = \emptyset$, and such that G_1, G_2 both have subphylogenies, then G has a subphylogeny.*

Note that although G_1 and G_2 are subsets of G, they are partitions with respect to all of S.

Proof. Let $\widetilde{T}_1$ and $\widetilde{T}_2$ with connections u and v be the respective subphylogenies of G_1 and G_2. Let x be a vertex such that $x_c \in M_c(G)$ for each $c \in M(G)$ and with $x_c = u_c$ for each $c \in D(G)$. The tree $\widetilde{T}$ is formed by taking $\widetilde{T}_1, \widetilde{T}_2$, adding x, and connecting them with the edges (u, x) and

(v, x). Then for any $c \in D(G)$, if $c \in D(G_1)$ then x has the character states necessary to be a connection since $u_c = x_c$ is the character state of some character in G_1. Assume then that $c \notin D(G_1)$. Then there must be some character state s_c that is common to both G_1 and G_2. This once again means that x_c is the character state of some character on G_1, this is because u must have the unique character state s_c on character c, so $x_c = u_c = s_c$.

Next we need to show that $\widetilde{T}$ is a perfect phylogeny. By $\widetilde{T_1}$ and $\widetilde{T_2}$ being subphylogenies and by x_c having the state of some species in $G_1 \cup G_2$ on each $c \in C$, it follows that $\widetilde{T}$ satisfies conditions (i) and (ii) of Definition 2.1. Now again since $\widetilde{T_1}, \widetilde{T_2}$ are subphylogenies, they uphold condition (iii) in and of themselves. What we need to check is that the addition of the connection x is sound with respect to the condition. Take any $c \in D(G)$, if $u_c = v_c$ then our assignment of $x_c = u_c$ assures that (iii) holds in this case. If $c \in M(G)$ and if $u_c = v_c$, then there must exist some $s \in G$ such that $s_c \in M_c(G)$. This species must either be in G_1 or G_2, so either $x_c = u_c$ or $x_c = v_c$, in both cases we are assured that $u_c = x_c = v_c$ so that (iii) holds.

Finally we note that if $D(G_1)$ or $D(G_2)$ is empty then we simply need to assign $x = u$ or $x = v$ respectively. $\square$

Now we arrive at the final missing piece before we can start describing the algorithm of [AFB94]. The following Lemma will give us an answer to the second question, which is sufficient for our purposes.

Lemma 2.17. *Assume that $G, G_1, G_2 \subset S$ are partitions such that $G_1 \cup G_2 = G$ and $G_1 \cap G_2 = \emptyset$. Suppose that G_1 has a subphylogeny $\widetilde{T_1}$, and that G has a subphylogeny $\widetilde{T}$ with connection x. If $\widetilde{T_1}$ is a subtree of $\widetilde{T}$ at x and if G_2 is not a c-partition, then the value of x_c on every $c \in D(G)$ is forced.*

Note, we mean subtree here in the sense that any path in $\widetilde{T}$ from any vertex of G_2 to x does not go through any vertex in $\widetilde{T_1}$. So any path from G_2 to G_1 has to go through x.

Proof. Assume that G_2 is not a c-partition, which would mean that $M(G_2) = C$, and take any $c \in D(G)$. What we want to show is that there exists some character state on c that is common to both G_1 and G_2, since any path going from G_1 to G_2 goes through x under our assumptions, this would imply that x_c is forced. Since $c \in M(G_2)$ then there must exist some $s \in G_2$ such that s_c is a character state in $S \setminus G_2$. If s_c is a character state in G_1 then we are done by what we mentioned earlier. If instead s_c is a character state in $S \setminus G$ then this would contradict that $c \in D(G)$, so this can not be the case. So x_c is forced. $\square$

Lemma 2.17 is the last result needed for the algorithm by [AFB94] and with it we may safely conclude this section to start describing the procedure.

2.3 Subphylogeny algorithm

We now arrive at the algorithms. As previously discussed, the problem of finding a perfect phylogeny can be broken down to dealing with type I and type II cases respectively. If we have no more cases of type I left in our procedure then we would like to produce a subphylogeny for any of the type II cases remaining, this is more or less the heart of the entire procedure. So we will begin by describing an algorithm dubbed SUBPHYLOGENY.

Algorithm 2.18 (SUBPHYLOGENY)**.**
Input: A partition $G \subset S$.

Output: A subphylogeny for G if one exists, otherwise *FAILURE.*

Setup: For every c-partition $G' \subset G$, a subphylogeny has already been constructed and recorded if one exists.

The algorithm:

```
1   SUBPHYLOGENY(G) :
2       if $|G| = 1$ then
3           $\widetilde{T}_G$ = the tree given by the single element of $G$
4           return $\widetilde{T}_G$
5       for each c-partition $G_1 \subset G$ such that $G_1$ has a subphylogeny $\widetilde{T}_1$ do
6           $G_2 = G \setminus G_1$
7           if $G_2$ is a c-partition with a subphylogeny then
8               $\widetilde{T}_G$ = subphylogeny of $G$ given by Lemma 2.16
9               return $\widetilde{T}_G$
10          $x$ = vector with forced states given by Lemma 2.17
11          Initialize $\widetilde{T}_G$ as $x$ together with $\widetilde{T}_1$ as a subtree
12          for each c-partition $H \subset G_2$ do
13              if $H$ has a subphylogeny $\widetilde{T}_H$ that is compatible with $x$ then
14                  $G_2 = G_2 \setminus H$
15                  $\widetilde{T}_G = \widetilde{T}_G \cup \widetilde{T}_H$ at connection $x$
16                  if $G_2 = \emptyset$ then return $\widetilde{T}_G$
17      return FAILURE
```

To clarify, the initialization in Line 9 is done by adding an edge between the connection of $\widetilde{T}_1$ and x, which can be done due to how a connection is defined. There are several things to immediately check with Algorithm 2.18 to ensure that it works as intended.

Lemma 2.19 (Correctness of SUBPHYLOGENY)**.** *Let $G \subset S$ be a partition and assume for each c-partition $G' \subset G$ that a subphylogeny of G' has been constructed if one exists. Then if G has a subphylogeny, the algorithm SUBPHYLOGENY constructs it, otherwise it returns FAILURE.*

Proof. If $|G| = 1$ then the tree consisting of the single species of G as a vertex is a subphylogeny of G and so the tree that is returned in Line 4 is indeed correct. So we may assume that $|G| > 1$.

If G has a subphylogeny then there are some different cases to consider. Firstly by Lemma 2.15 there are c-partitions that are compatible with x and have subphylogenies, so if we are going to find a subphylogeny of G then we are justified in searching for c-partitions as we do in Line 5. If $G_2 = G \setminus G_1$ is another c-partition with a subphylogeny then by Lemma 2.16 we may combine them to form a subphylogeny for G, as is returned in Line 9. If instead G_2 is not a c-partition and if x is a connection for a possible subphylogeny $\widetilde{T}_G$, then x is compatible with G and has x_c forced for each $c \in D(G)$, as given by Lemma 2.17. So the states of x are completely determined in that case and we may initialize such a vector together with $\widetilde{T}_1$ as we do in Lines 10 and 11.

The loop starting at Line 12 will connect c-partitions compatible with x to $\widetilde{T}_G$ until all species left to consider are already a vertex of $\widetilde{T}_G$, by Lemma 2.15 this does indeed produce a subphylogeny for G.

We also argue that any subphylogeny for G with $\widetilde{T}_1$ as a subtree, which is not constructed in a previous step of the algorithm, is constructed in the loop of Line 12. Firstly note that the tree G_2 is finite and will either shrink if some c-partition H has a subphylogeny that is compatible with x, or remain unchanged if no such H is found, in either case the loop of Line 12 will terminate, and if it terminated while $G_2 =$ then a subphylogeny was found. So if there exists some subphylogeny with $\widetilde{T}_1$ as a subtree at x that is not found in this loop, that must mean that the loop terminated with G_2 nonempty. We argue that this could not happen, or in other words that G_2 must contain some c-partition H with a subphylogeny, and that is compatible with x. If G has a perfect phylogeny then by Lemma 2.6, $G_2 \cup \{x\}$ has a perfect phylogeny $\widetilde{T}_{G_2 \cup \{x\}}$. Let y be any neighbour of x in $\widetilde{T}_{G_2 \cup \{x\}}$, let $\widetilde{T}_y$ be the subtree of $\widetilde{T}_{G_2 \cup \{x\}} \setminus \{x\}$ which contains y, and let J

be the set of species contained in T_y.

Firstly we note that J is a partition since it is the set of species of a subtree of a perfect phylogeny, and therefor for each character in $c \in M(J)$ there can be at most one character state in common between J and S. Secondly we recall that we can safely assume that each vertex is distinct. In that case J is a c-partition, the argument for this is the same as the argument made in the ($\Rightarrow$) part of Lemma 2.15, if $D(J) = \emptyset$ then $x = y$ which contradicts the uniqueness. Finally by Lemma 2.13 we get that J is indeed a subphylogeny with y as a connection. So if G has a subphylogeny then in each iteration of the loop of Line 12 we remove such a subset J from G_2 until there is nothing left. This shows that the algorithm does produce a subphylogeny as required.

We have established that the non-FAILURE returns of SUBPHYLOGENY work as intended, if Line 17 is reached then either there are no G_1 that fit the criteria for the loop in 5, or each such choice of G_1 failed to produce a subphylogeny for G. Since we may safely assume that species are distinct and since we assumed $|G| > 1$, the first case can not occur. In the second case there is no subphylogeny of G by Lemma 2.15. $\square$

With Algorithm 2.18 we now have the last ingredient needed to describe the full procedure. Examples where this algorithm is being run will be given later once we have described the full procedure, which will be covered next.

2.4 Phylogeny algorithm

We arrive now at the algorithm which is the culmination of all the previous sections, and the main result of [AFB94]. The recursive algorithm which is called PHYLOGENY, first handles type I splits while leaving the type II cases to SUBPHYLOGENY. The case of type I is dealt with recursively as we will describe now.

Algorithm 2.20.
Input: A species set S with $|S| = n$.

Output: A perfect phylogeny for S if one exists, otherwise $FAILURE$.

```
PHYLOGENY(S) :
if $|S| = 1$ then
    $T_S$ = the tree given by the single element of $S$
    return $T_S$
if there exists type I c-split $(G_1, G_2)$ then
    $s$ = connecting species of $(G_1, G_2)$
    Call PHYLOGENY($G_1 \cup \{s\}$) and PHYLOGENY($G_2 \cup \{s\}$)
    if both calls succeed then
        Combine the result into a perfect phylogeny $T_S$
        return $T_S$
    else return FAILURE
else
    $size = 1$
    while $size \leq n - 1$ do
        for each c-partition $G$ such that $|G| = size$ do
            Call SUBPHYLOGENY(G)
            if G has a subphylogeny $\widetilde{T}_G$, record $\widetilde{T}_G$ and its connection
        endfor
        size = size + 1
    endwhile
    Pick any $s \in S$
    if $G = S \setminus \{s\}$ has a subphylogeny $\widetilde{T}_G$ then
        $x$ = the connection of $\widetilde{T}_G$
        $T_S = \widetilde{T}_G \cup \{s\}$ at the connection $x$
        return $T_S$
    else return FAILURE
endif
```

Remark. As previously discussed it is safe to assume that the species are distinct, and we should do so. Consider for example the case where S consists of a single species with a multiplicity of 2. In that case the algorithm would return FAILURE since there are no c-splits.

Finally what remains is to show that PHYLOGENY works as intended.

Theorem 2.21. *The procedure PHYLOGENY will construct a perfect phylogeny for S if one exists, otherwise it will return FAILURE.*

Proof. The tree consisting of a single vertex as a species is a perfect phylogeny for that lone species, so if $|S| = 1$ then a perfect phylogeny is correctly returned in Line 4. Assume now that $|S| > 1$. There are two cases to consider, either S has type I splits or not.

If S has a type I c-split (G_1, G_2) with connecting species s, then by Lemma 2.14 every subset has one, so we can call PHYLOGENY with $G_1 \cup \{s\}$ and $G_2 \cup \{s\}$. If both calls succeed then the perfect phylogenies can be combined by merging the two vertices s in each tree to one vertex with all the combined neighbours and edges, as is done in Line 10. If any of the calls fail then there is no perfect phylogeny for S. Note that the line 4 will only be reached on the first call of PHYLOGENY and not on any subsequent recursive calls due to how type I splits are defined.

If all splits are of type II, then we first record all subphylogenies of c-partitions $G \subset S$ such that $|G| < |S|$, by Lemma 2.19 this produces exactly all subphylogenies of these c-partitions. Now we note two things, firstly that the tree consisting of any one species s is a subphylogeny as discussed in the proof of Lemma 2.19. Secondly, by Lemma 2.14 we have that if $\{s\}$ and $S \setminus \{s\}$ both have subphylogenies then we may combine these to a perfect phylogeny for S, as is returned in Line 25.

So in both cases the returned tree is a perfect phylogeny for S, what remains is to show that PHYLOGENY does not return FAILURE if there exists a perfect phylogeny. Assume that S has a perfect phylogeny, then again there are two cases to consider. If S has a type I split then we will call PHYLOGENY again, which will either result in more recursive calls or there will be no type I splits.

Assume there are no type I splits and pick any $s \in S$. By Lemma

2.10 the species s is a leaf in any perfect phylogeny of S, so the split $(\{s\}, S \setminus \{s\})$ is a c-split in S. Recall that we may assume each s to be distinct, so if s is a leaf then at least one character must be distinct, otherwise the neighbour of s must be identical to s. If s were not a leaf in some perfect phylogeny of S, then it might be that two children of s have common states with s on partially non-overlapping characters of C. In that case none of the children need necessarily be identical to s, but the union of both could eliminate the possibility of s to have distinct characters.

A subphylogeny for $\{s\}$ is $\{s\}$ itself as a single vertex. A subphylogeny for $S \setminus \{s\}$ must exist since S has a perfect phylogeny T_S, then the existence is assured by Lemma 2.13. Finally we note that the subphylogeny for $S \setminus \{s\}$ must have been constructed in some iteration of the loop in Line 14. This shows that PHYLOGENY(S) returns a phylogeny for S if and only if it has one. □

With this we are finished on the part of constructing perfect phylogenies. For those wanting to better understand how the algorithm runs we recommend working through Example A.1 in Appendix A.

3 Gene Reconciliation

In Chapter 2 we described a process which constructs a perfect phylogenetic tree if it exists, for a set of sequences. That study, although interesting in and of itself, can be put in a broader context to provide insight in to the study of protein functions. By considering DNA or protein sequences we may compare the resulting phylogenetic tree with a known species tree in a process called *reconciliation*, to gain a more complete evolutionary history of the sequences. A simple algorithm for gene reconciliation is given by C.M. Zmasek and S.R. Eddy [ZE01], our goal is to give a clear understanding of that algorithm.

In the framework of constructing phylogenetic trees, as in Chapter 2, sequences were sometimes referred to as species. We should now think of species as merely in the biological sense. The sequences we will consider as members of our produced perfect phylogenetic tree will be called genes. We consider these to carry two different pieces of information; firstly a sequence of proteins that define the gene, and secondly a biological species in which the gene has been observed. Any perfect phylogenetic tree with genes as input will be called a gene tree and given any set of genes (or gene tree) G, we refer to the set of biological species which harbor the genes in G as $S(G)$.

When we have obtained a gene tree G for a set of genes, whether it is by using PHYLOGENY or by any other method, we want to consider a known evolutionary species tree S of the species $S(G)$ which harbor the genes found in G. The reason is that we can use S to give us more information about G, more specifically we would like to know if the vertices of G branch off due to duplication events or speciation events. The details of what these mean biologically will not be covered here, although to highlight their significance we mention that duplication and speciation has implications for the study of protein functions. We are soon going to define them mathematically.

By looking at G and comparing it to S we gain a more complete picture of the history being told by G in terms of duplications. Loosely speaking, we *reconcile* G by embedding it in S and in doing so we can see which genes should be marked as duplication and which should be marked as speciation. Before we proceed to talk about duplications, let us first explicitly define the embedding function that is used in [ZE01].

Note. For the remainder of this chapter we consider all phylogenetic trees, both G and S, to be rooted and binary. We also assume that the observed genes which have been used to construct G, are leaves in G.

Definition 3.1. *Let G be a gene tree and let S be a known species tree of $S(G)$, the species which harbor the genes in G.*

i. *Let T be any rooted tree. Then define $\sigma : T \to T$ by letting $\sigma(\alpha)$ be the set of leaves of T which have α as ancestor.*

ii. *Define $\gamma : G \to S$ by letting $\gamma(g) \mapsto S(\sigma(g))$*

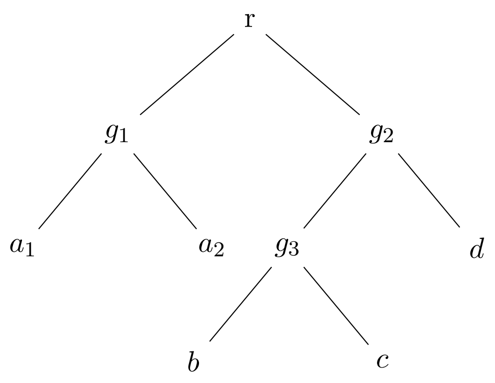
iii. *Let $M : G \to S$ be such that $M(g)$ is the lowest vertex in S satisfying $\gamma(g) \subseteq \sigma(M(g))$. Then M is called the* ***embedding*** *or* ***mapping*** *function of G and S.*

In other words, we map each gene $g \in G$ to the lowest species in S which has as descendants (including itself) at least all the species names found in the descendants of g. Let us illustrate Definition 3.1 with an example.

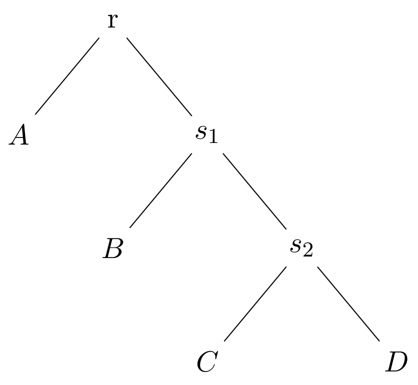
Example 3.2. Consider the set of genes

$$\{a_1, a_2, b, c, d\}$$

contained as leaves in the tree $G =$



with $A := S(\{a_1\}) = S(\{a_2\})$, $B := S(\{b\})$, $C := S(\{c\})$, and $D := S(\{d\})$. The corresponding species set consists of $\{A, B, C, D\}$, let the species tree be given by $S =$



i) In general for any leaf $\alpha \in G$ we get that $\sigma(\alpha) = \alpha$. For the non-leaf vertices we get $\sigma(g_1) = \{a_1, a_2\}$, $\sigma(g_3) = \{b, c\}$, $\sigma(g_2) = \{b, c, d\}$ and finally $\sigma(r) = \{a_1, a_2, b, c, d\}$.

ii) For the leaf-vertices $\alpha \in G$ we get that $\gamma(\alpha) = S(\sigma(\alpha)) = S(\alpha)$ by what we described in *i*). For the non-leaf vertices we get $\gamma(g_1) = \{A\}$, $\gamma(g_3) = \{B, C\}$, $\gamma(g_2) = \{B, C, D\}$ and finally $\gamma(r) = \{A, B, C, D\}$.

iii) Firstly we notice same as above that each leaf gets mapped to each corresponding species. Let us look at the non-leaf vertices. The node g_1 has $\gamma(g_1) = \{A\}$ and we can see that the lowest species in S that contains A is the species A itself. For g_3 we get that s_1 is the lowest species that has B and C as descendants. Similarly s_1 is the lowest species with B, C and D as descendants. This gives us that $M(g_1) = A$, $M(g_2) = s_1$, $M(g_3) = s_1$, and finally $M(r) = r$.

The embedding function M now gives us a clear way to define exactly what a gene duplication is.

Definition 3.3. *Let G be a gene tree with species tree S and mapping function M. Let $g \in G$ be any non-leaf vertex and let g_1 and g_2 be the two children of g. Then g is a* ***duplication*** *if and only if $M(g) = M(g_1)$ or $M(g) = M(g_2)$. Otherwise it is a* ***speciation****.*

We should note that Definition 3.3 gives one way (among many) to reconcile G with S and that this definition assumes a minimal number of duplications as well as placing those duplications as far down in S as possible. In Example 3.2 we get that g_1 and g_2 are duplications since $M(g_1) = M(a_1)$ and $M(g_2) = M(g_3)$.

Once the mapping function has been calculated then assigning the duplications is straightforward. What about determining the function M? One approach would be to compute $\gamma(g)$ and $\sigma(s)$ for each $g \in G$ and $s \in S$. However, this is needlessly expensive. We notice that both γ and σ depend only on the leaves of each respective tree, and that both these functions are increasing in the sense that $f(E_1) \subseteq f(E_2)$ if $E_1 \subseteq E_2$, where f can be either of the two functions. Finally we notice that each leaf g of G gets mapped to the single species $S(g)$. This leads us to think that if we traverse G recursively in the right order, then we can assign duplications in a simple way without having to calculate M explicitly first. Let us see how this is done.

Algorithm 3.4 (Reconciliation)**.**
Input: A rooted binary gene tree G, and a rooted binary species tree S.

Output: An assignment of duplication or speciation to each $g \in G$.

Setup: Number the vertices of S in preorder traversal with $root = 1$. For each leaf $g \in G$, assign $M(g)$ to be the number of the species $S(g)$.

Recursion:

```
1  Traverse non-leaves of $G$ in postorder with $g$ being the current vertex:
2      Let $g_1$ and $g_2$ be the two children of $g$
3      set $\alpha_1 = M(g_1)$
4      set $\alpha_2 = M(g_2)$
5      while $\alpha_1 != \alpha_2$:
6          if $\alpha_1 > \alpha_2$:
7              set $\alpha_1$ = parent vertex of $\alpha_1$
8          else:
9              set $\alpha_2$ = parent vertex of $\alpha_2$
10     set $M(g) = \alpha_1$
11     if $(M(g) == M(g_1))$ or $(M(g) == M(g_2))$:
12         $g$ is a duplication
13     else:
14         g is a speciation
```

Due the traversal being done in postorder, the recursion will start at the near bottom and work upward from there. Since the labeling in S was done in preorder, the recursion will move up through S to find the lowest (highest label) common ancestor of $M(g_1)$ and $M(g_2)$. While doing so it also assigns the values of duplication or speciation, which is possible due to the fact that the dynamical programming approach taken here works from the bottom up. Example A.2 in Appendix A shows a detailed computation of the algorithm.

With this we can now conclude the topic of reconciliation and we will move on to discuss how we can merge the two ideas that have been presented so far.

4 Incompatibilities

4.1 Problem Formulation

We started in Chapter 2 by considering sequences which represent biological information of species, such as proteins or DNA. After having carefully introduced the necessary theory, we presented the algorithm of [AFB94] which takes the sequences as an input and outputs a perfect phylogeny tree of the sequences. Next, in Chapter 3 we presented the method by [ZE01] to compare the phylogeny tree to a known species tree to give significant annotations to the vertices in the phylogeny.

The output of the first algorithm is roughly the input of the second one. However, there are some incompatibilities between them. We recall from Chapter 3 that we assume all gene trees G to be *rooted* and *binary.* There are cases in which Algorithm 2.20 produces non-binary trees, see Example A.1. The algorithm also does not assign any root vertex. Furthermore Algorithm 3.4 requires that the observed sequences are *leaves*, which is not guaranteed by PHYLOGENY. We make the following explicit formulation.

Problem:
There are the following incompatibilities between PHYLOGENY 2.20 and Reconciliation 3.4.

- Reconciliation requires a rooted tree.
- Reconciliation requires a binary tree.
- Reconciliation requires observed sequences to be leaves.

We address these problems in the following section.

4.2 Solutions

To start, observe that any tree can be made in to a rooted tree with any vertex as the root. Despite this, we can not merely assign an arbitrary root in the setup-phase of the algorithm. Firstly the choice of root, although can be made arbitrarily, is certainly not unessential for the outcome with regards to duplications. We will not make an analysis of how to attain the best choice. However, we will show that the choice can be incorporated in to the algorithm. Secondly, in order to determine duplications we require an orientation in the tree at any recursion step. In other words, it is not enough that we have *one* global root assigned, we need to have local orientations that remain consistent across different iterations of the recursion. In our discussion of the type I and II cases we mentioned that one can think of connections or connecting sequences as the root. With this in mind we can introduce a method of labeling the vertices so that we can keep track of the hierarchy when we need to.

Recall from our discussion on constructing perfect phylogenies that we can add or remove copies of any sequence. If the connecting sequence is in the gene set then we can use the above fact to create enough copies of the connection that we keep a binary structure on the tree while having the connection be a leaf. Let us formalize how this refinement can be done.

Let G be a set of genes and assume that $s \in G$ is a connection or connecting sequence. If s is problematic, meaning that it violates one of the conditions mentioned in the problem formulation, then we would like a way to alter s. First we describe the handful of different ways in which s can be problematic.

A. s is in G and has one child.

B. s is in G and has two or more children.

C. s is not in G and has one child.

D. s is not in G and has three or more children.

In a process we call **refinement**, we alter s depending on which case it belongs to. The result of one such alteration may in some cases not be enough. We perform the steps multiple times until the result is no longer problematic. To perform the process we need to have an underlying orientation to keep track of parents and children. Starting with 0, each time we consider a new connection or connecting sequence, we assign it a higher integer than any previously assigned. Using this, we know that lower labels are higher up in the hierarchy while unlabeled or higher labeled vertices are lower. Since the refinement changes the hierarchy, it will be convenient to explicitly resolve what happens to the labels. We explain the details of the procedure now. In the following, let l denote the label of s.

Scenario A: We add a vertex s_1 that is a copy of s and transfer any parent and child edges from s to s_1. We let s be a child of s_1. This ensures that s is a leaf and that s_1 has two children. Vertex s_1 is given label l while s is relabeled with $l + 1$

Scenario B: We add two copies s_1 and s_2. Then we transfer all the child edges from s to s_2 while transferring parent edges to s_1. We let s and s_2 be children of s_1. Vertices s_1 and s_2 are labeled l and $l + 2$ respectively, while s is relabeled with $l + 1$. In this case s is turned in to a leaf and s_2 is a non-leaf copy of s which acts as a connecting sequence in its place. See Figure 2. The result will depend on the number of children that s had before we refined it. If there were two children, then no further refinements are needed since the new connection s_2 is not in G. If there were three or more children then the vertex s_2 would be in scenario D and we refine it.

Scenario C: In this case the vertex s does not need to exist and we can remove it. If s has a parent then we connect the child of s with that parent.

Scenario D: We add a copy s_1 of s and connect s as a parent of s_1. Then we transfer all but one of the child edges from s to s_1 and give s_1 the label $l + 1$. If s_1 still has three or more children then we refine s_1.

Otherwise: If a vertex does not fall in to any of the previous categories then we do not need to perform any operation, and the refinement is done.

Unless otherwise stated, when we say that a sequence or vertex was refined then we mean that it was altered as a result of refinement. Keep in mind that the order of when we label and when we refine is important; we label the connections on the way down in recursion depth, and we refine on the way up. This operation raises some other issues. Recall from Chapter 2 that if G has a type I split (G_1, G_2) with perfect phylogenies T_1 and T_2 respectively

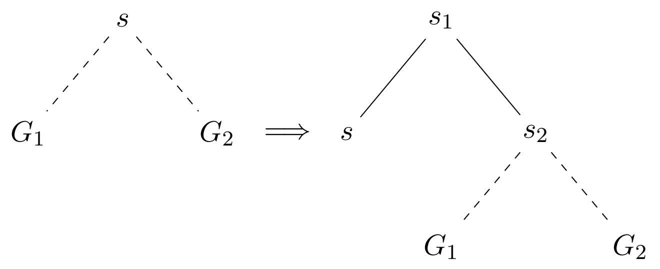


Figure 2: Scenario B refinement of a connecting sequence s for the split (G_1, G_2). In the tree on the left, we need s to be a leaf, so we add s_1 and s_2 and connect them as in the tree on the right.

at connection g, then we can merge these by identifying g in T_1 with g in T_2. The same process would not directly work in our current context because we could easily end up with a result that is again problematic, furthermore if g has been altered through refinement then that poses the question of which version we should merge. However, we can avoid these issues by carefully choosing where to merge the trees after any refinement. We do the following.

```
Merge($T_1, T_2$):
    if $g$ was refined in $T_1$ or $T_2$:
        $A$ = any one of $T_1$ or $T_2$ in which $g$ was refined
        $B = \{T_1, T_2\} \setminus A$
        $g_1$ = the parent copy of $g \in A$
        Identify $g_1 \in A$ with $g \in B$
```

Recall that by "refined", we mean that g was altered through refinement. Since T_1 and T_2 are phylogenies for the respective partitions of a type I split, the connection g is assumed to be in G and so is never problematic of scenario C. The label of g can be kept from either of the previous versions without relevance. An illustration of the procedure is shown in Figure 3. By how the final algorithm will be constructed, g will in fact be refined in one or two of the trees T_1 and T_2.

Finally we discuss the reconciliation. For any perfect phylogeny T_G, the embedding of T_G in a known species tree S of the species $S(G)$ is heavily dependant on the topological structure of T_G. Since we change the structure when we refine the connections, we only consider finding the mapping M and reconciling the sequences after we have connected trees together, and

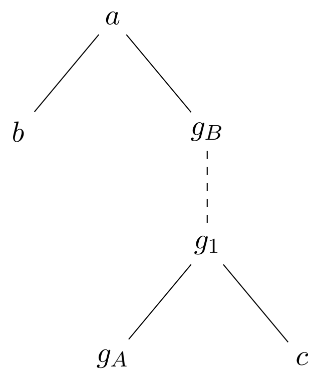


Figure 3: Merging of two perfect phylogenies after a refinement. The tree $A := \{g_1, g_A, c\}$ and the tree $B := \{a, b, g_B\}$ with connection g, denoted by g_A and g_B for each tree respectively. The dashed line between the trees indicates the vertices that are identified. Since g has been refined in A producing the extra vertex g_1, we identify this vertex with g_B when we merge the trees together.

after having refined the result at the connection g. Due to the refinement, there might be several non-reconciled copies of g and so we would possibly need to reconcile several of these vertices. For the binary subtree T_g rooted at g, we do the following assuming that S is the appropriate species set and is set up as in Algorithm 3.4.

```
Reconcile($T_g, S$):
    Traverse non-reconciled non-leaf vertices of $T_g$ in postorder:
        With $g$ as the current vertex, and $S$ as species tree:
            Do lines 2 to 14 in Algorithm 3.4
```

With this, we are now ready to present the main result.

5 Main Result

Now that we have introduced all the topics needed and discussed how they can be put together, we are ready to present the resulting algorithm **RecPhylogeny** in detail.

Algorithm 5.1 (RecPhylogeny)**.**
Input: A set of genes G with $|G| = n$ and a known binary rooted species tree S of the set of species $S(G)$.

Output: A reconciled binary rooted perfect phylogeny T_G of G if one exists, otherwise FAILURE.

Setup: Initiate $l = 0$ as label. Set up S as is needed for Algorithm 3.4.

```
RecPhylogeny(G) :
if $|G| = 1$ then
    $T_G$ = the tree given by the single element of $G$
    return $T_G$
if there exists type I c-split $(G_1, G_2)$ then
    $s$ = connecting sequence of $(G_1, G_2)$
    Give $s$ label $l$ and set $l = l + 1$
    Call RecPhylogeny($G_1 \cup \{s\}$) and RecPhylogeny($G_2 \cup \{s\}$)
    if both calls succeed with phylogenies $T_1$ and $T_2$ then
        Merge($T_1, T_2$) to a perfect phylogeny $T_G$
        $g$ = the lowest label vertex of $T_G$
        $T_g$ = the subtree of $T_G$ with root at $g$
        Reconcile($T_g, S$)
        return $T_G$
    else return FAILURE
else
    $size = 1$
    while $size \leq n - 1$ do
        for each c-partition $G'$ such that $|G'| = size$ do
            Call SUBPHYLOGENY($G'$)
            if $G'$ has a subphylogeny $\widetilde{T}_{G'}$ then
                Record $\widetilde{T}_{G'}$ and its connection
        endfor
        size = size + 1
```

```
25        endwhile
26        Pick any $s \in G$
27        if $G' = G \setminus \{s\}$ has a subphylogeny $\widetilde{T}_{G'}$ then
28            $x =$ the connection of $\widetilde{T}_{G'}$
29            $T_G = \widetilde{T}_{G'} \cup \{s\}$ at the connection $x$
30            Traverse non-reconciled vertices of $T_G$ with lowest label as root:
31                $g =$ current vertex of the traversal
32                if $g$ has no label then give $g$ label $l$ and set $l = l + 1$
33                Refine $T_G$ at $g$
34                $T_g =$ subtree of $T_G$ with lowest label copy of $g$ as root
35                Reconcile($T_g, S$)
36            return $T_G$
37        else return FAILURE
38    endif
```

After a run of RecPhylogeny, we can mark the vertex with label 0 as global root. Although we recursively run RecPhylogeny, the labelings done in Lines 7 and 32 must persist across any recursion depth because the very purpose of the labels is to keep track of the hierarchical structure of the tree. Since the structure is determined by recursion depth, the label must remain global. Furthermore, note that for the purpose of reconciliation, we only need to know which vertex is the root; if we start at the root then we can assign *some* order to the children of the current vertex and traverse down in postorder (whether the tree is binary or not). Because of this, it is not strictly necessary to give labels to copies in a refinement, though we have found it convenient to do so. After merging trees in Line 10, the lowest label vertex in T_G may have gained more descendants. So although we have already reconciled it, we need to do so again as is done in Line 13. The refinement in Line 33 ensures that the tree is binary before we reconcile. Once completed, the procedure provides us with a binary tree that has been reconciled, with the lowest label vertex as the root. An example run of RecPhylogeny is given in the Appendix, Example A.3. With this we are finished with presenting the main result. We conclude this thesis in the following section with a discussion on different approaches for optimizing the result in future work.

6 Conclusion and Future Work

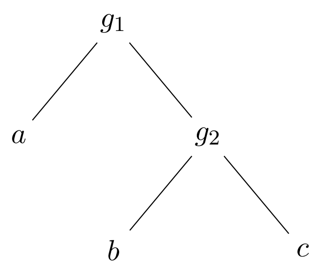
The result presented in Chapter 5 showed that it is possible to merge the two algorithms for a combined procedure, and in arriving at this we also detailed in Chapter 2 some justifications for choices made in [AFB94].

The main focus of our work has been in producing **one** combined Algorithm for phylogeny and reconciliation, however, the result presented in 5.1 is far from optimal. By measuring the number of duplications in reconciled phylogenetic trees, one could compare the quality of different results. A natural continuation to this work is to implement the methods here, to empirically test the quality of the method on real or synthesized data.

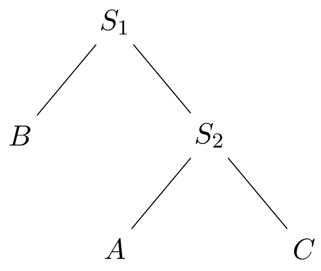
The assignment of duplications is dependant, among other things, on the topology of the tree T_G, the position of the genes, and the choice of root. So consider the following choices we made. If a vertex s of T_G is problematic then we made the choice in scenario B to place s as close to the first copy as possible, and in scenario D we expand the vertex s as much as possible in an unbalanced way. Both of these choices affect the structure in a possibly erroneous way. Authors Durand et al., [DHV06] give a method of correcting topological errors on a rooted gene tree with respect to minimizing duplications. In our case the choice of root is made arbitrarily, which also influences the outcome. Methods by P. Górecki and O. Eulenstein [GE12] provide simultaneous rooting and error-correction. The impact of various choices of root on the outcome of a more general reconciliation model is studied by S. Kundu and M.S. Bansal [KB18].

Part of the reason for the refinement procedure we introduced is that the process of PHYLOGENY leads to non-binary vertices in some cases. The authors Y. Zheng and L. Zhang [ZZ17] consider optimal reconciliation with non-binary gene-trees by expanding non-binary vertices in a way that is consistent with minimizing duplications. It could very well be possible to expand Algorithm 5.1 by incorporating these methods, or parts of them, in the procedure, for an optimized algorithm.

Finally, consider the following observation. For any non-leaf vertex g of a binary rooted perfect phylogeny, let g_1 and g_2 denote the children of g. With γ as in Definition 3.1, if $\gamma(g_1) \cap \gamma(g_2) \neq \emptyset$ then g will be marked as duplication when reconciled with any species tree. However, the converse statement does not hold in general. Consider the gene tree $G =$



and the corresponding species tree $S =$



with capital letters in S as the species of the corresponding letters in G. Then g_1 is a duplication since $M(g_1) = M(g_2) = S_2$. However, we have $\gamma(a) \cap \gamma(g_2) = \emptyset$. Authors K. M. Swenson et al [SDEM12] denote these types of vertices as **non-apparent duplications** and provide methods for eliminating them by restructuring the gene-tree.

A Appendix

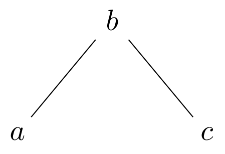
Example A.1 (Phylogeny algorithm)**.** First let us note the output of a commonly occurring case in most runs of the algorithm. If the input S of PHYLOGENY is two distinct species s and t, then the output will be a tree consisting of the two species as vertices connected by an edge. Firstly there can not be any type I c-splits because of the size requirement. Secondly we will call SUBPHYLOGENY on s and t respectively which will return s and t as trivial subphylogenies. Then we will connect either s or t with its complement in S which produces the same result.

Now consider the sequences given by the table

Denotation	Sequence
a	1000000
b	1100000
c	1200000
d	0021110
e	0012110
f	0011210
g	0011120
h	0031111
i	0013111
j	0011311

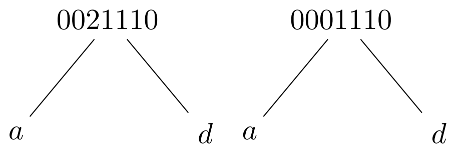
and let $S := \{a, b, c, d, e, f, g, h, i, j\}$. To simplify the notation we will omit brackets and commas when referring to sets of S. A bar over the set indicates the set complement with regards to the set S of the current running of PHYLOGENY. So in PHYLOGENY($defghij$), we would write $\overline{hij}$ to mean $\{d, e, f, g\}$.

PHYLOGENY(S):
Since $|S| > 1$ we move on to see if there are any type I c-splits in S and see that $(abc, defghij)$ is such; character 1 and 7 are common with at most one common state, a is compatible with both partitions, and $D(abc) \neq \emptyset$.
PHYLOGENY(abc):
We see that (a, bc) is a type I c-split with b as connecting species.
PHYLOGENY(ab) and **PHYLOGENY(bc)**:
By our above discussion this will return trees consisting of a, b and b, c respectively. We attach these together at the connection b and get the following tree.



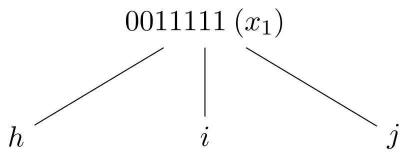
PHYLOGENY($adefghij$):
One can verify that there are no type I c-splits in $adefghij$. With $size = 1$ there are the eight trivial subphylogenies, one for each singleton c-partition. With $size = 2$ we get each of the sets $ad, ae, \ldots, aj$ as c-partitions, we will show SUBPHYLOGENY(ad) only as the others are done in an analogous way.

SUBPHYLOGENY(ad):
We choose the c-partition $G_1 = a$ which has a subphylogeny, and let $G_2 = d$. Then we note that also G_2 has a subphylogeny, so we return the combined tree as given by Lemma 2.16, depending on our choice of u and v in the Lemma the connection will be a different vertex. Either of the following trees could be the result.



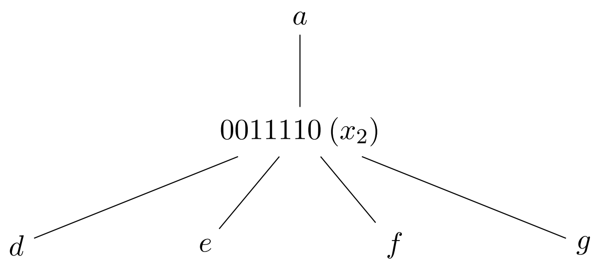
For $size = 3$ we get the c-partition hij, so we run SUBPHYLOGENY on this partition.

SUBPHYLOGENY(hij):
Let $G_1 = h$ which has a subphylogeny $\widetilde{T}_h$, so that $G_2 = ij$ which does not have a subphylogeny. Then $x_1 = 0011111$ is the vector with forced states as given by Lemma 2.17. We set $\widetilde{T}_G$ to be x_1 with $\widetilde{T}_h$ as a subtree. Now since i is a c-partition with a subphylogeny $\widetilde{T}_i$ that is compatible with x, we may remove i from G_2 and connect $\widetilde{T}_h$ with $\widetilde{T}_G$ at the connection x. We do similarly with j, and after having done so, G_2 is now empty. We return the tree $\widetilde{T}_G$ which now looks like the following.

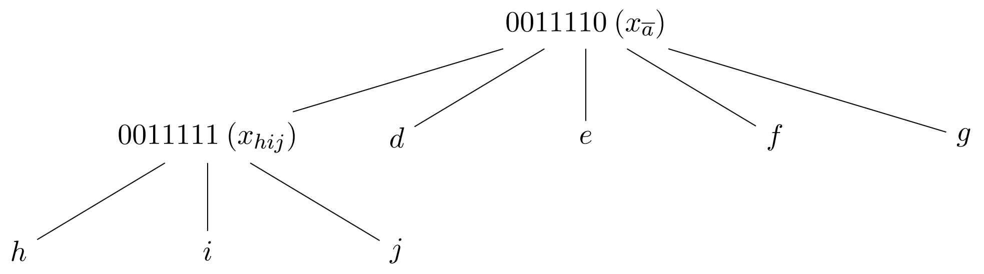


There are no other c-partitions of size 3. For $size = 4$ it is once again possible to check that there does not exist any c-partitions, however with $size = 5$ we have one c-partition $adefg$.

SUBPHYLOGENY($adefg$):
The case is analogous to the previous one of hij and produces the following tree.



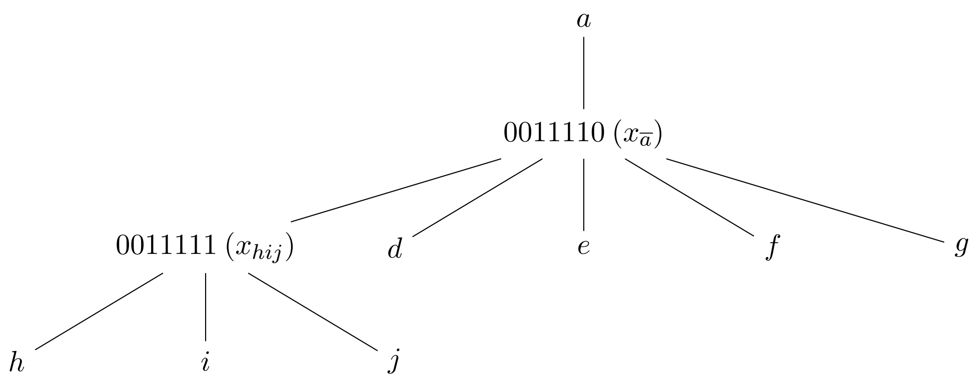
The orientation of a is kept upward to remind us that it is the connecting species in this run of PHYLOGENY. For $size = 6$ there are no c-partitions. Finally for $size = 7$ we get the c-partitions $\overline{a}, \overline{d}, \ldots, \overline{j}$. We will look at the first one.
SUBPHYLOGENY($\overline{a}$):
Take the c-partition $G_1 = hij$ which has the subphylogeny $\widetilde{T}_{hij}$ and connection $x_{hij} = 0011111$. Then $G_2 = defg$ does not have any subphylogeny, so we let $x_{\overline{a}}$ be the vertex with forced states 0011110 and form the tree $\widetilde{T_{\overline{a}}}$ with $x_{\overline{a}}$ and the tree $\widetilde{T}_{hij}$ as a subtree at $x_{\overline{a}}$. Finally we look at the c-partitions of G_2 which have subphylogenies compatible with $x_{\overline{a}}$, these are d, e, f and g. We attach each of these to connection $x_{\overline{a}}$ and return $\widetilde{T_{\overline{a}}}$, the following tree.



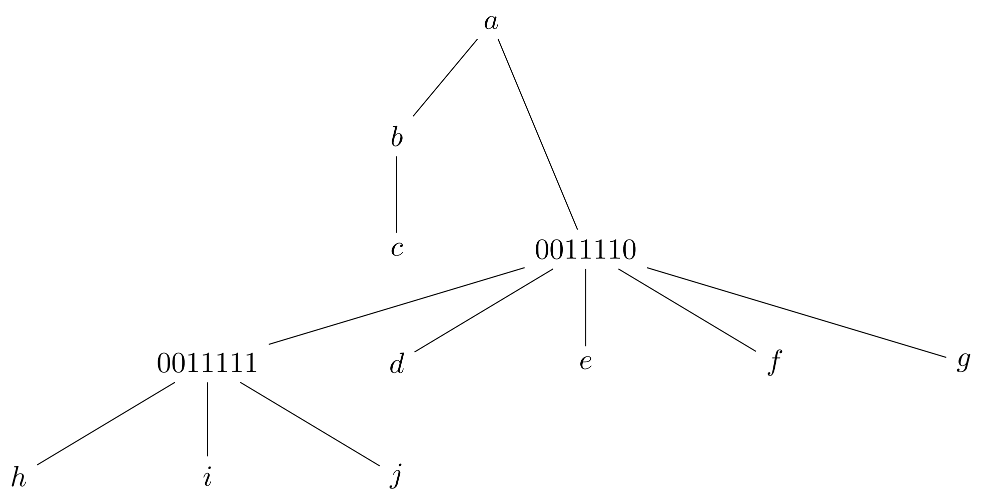
SUBPHYLOGENY for each of $\overline{d}, \overline{e}, \overline{f}, \overline{g}$ will be the same in every regard except for which leaves are attached to 0011110. For the case of $\overline{h}, \overline{i}$ and $\overline{j}$ the difference will be that instead of taking $G_1 = hij$ we will take $G_1 = adefg$, otherwise it will be done in a similar manner, so we omit these.

One thing to note is that our choice of $G_1 = hij$ is arbitrary, we could have chosen $G_1 = h$ for example. In that situation however, we would have attached i and j to x_{hij} but then been forced to choose a different c-partition as $G_2 = defg \neq \emptyset$ and there are no more c-partitions compatible with x_{hij}.

Now we are finished with the loop that produces the subphylogenies and can go on to the final step. We take an arbitrary species, say a, and consider $\overline{a}$. Since $\overline{a}$ has a subphylogeny $\widetilde{T_{\overline{a}}}$ we can connect $\widetilde{T_{\overline{a}}}$ with $\{a\}$ at the connection $x_{\overline{a}}$. The result is the following tree.

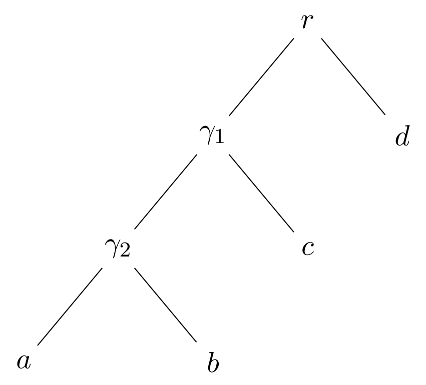


This ends the call of **PHYLOGENY(*adefghij*)**. Now finally we may combine the two resulting outputs into

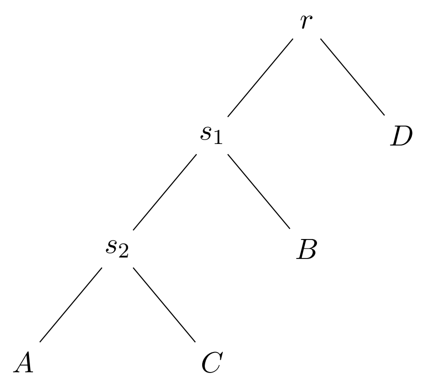


which is the output of **PHYLOGENY**(S).

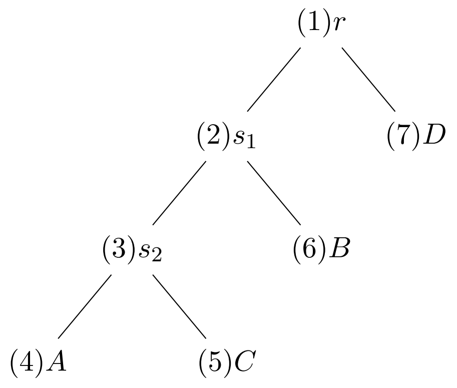
Example A.2 (Reconciliation algorithm)**.** Consider the gene tree $G =$



with corresponding species $A, \ldots, D$ respectively for each gene $a, \ldots, d$. Let the species tree be given by $S =$



After the setup, the tree S looks like



with numberings indicated in paranthesis and with initial M given by the following table.

g	$M(g)$	spec/dup
a	A	-
b	B	-
c	C	-
d	D	-
γ_1	?	?
γ_2	?	?
r	?	?

Starting our traversal in postorder, the first vertex we consider is $g = \gamma_2$ with $g_1 = a$ and $g_2 = b$. Then we set $\alpha_1 = A$ and $\alpha_2 = B$. Since $\alpha_1 < \alpha_2$ we set $\alpha_2 = s_1$, then $\alpha_1 > \alpha_2$ so we set $\alpha_1 = s_2$. We now have that $\alpha_1 > \alpha_2$ so we set $\alpha_1 = s_1$, this finishes the loop since $\alpha_1 = \alpha_2$. We set $M(\gamma_2) = s_1$ and since $(M(\gamma_2) \neq M(g_1)$ and $M(\gamma_2) \neq M(g_2)$ we get that γ_2 is a speciation.

Continuing the postorder traversal with $g = \gamma_1$ we set $\alpha_1 = s_1$ and $\alpha_2 = B$. Since $\alpha_1 < \alpha_2$ we set $\alpha_2 = s_1$ which ends the loop. We set $M(\gamma_1) = s_1 = M(\gamma_2)$ and hence γ_1 is marked as a duplication.

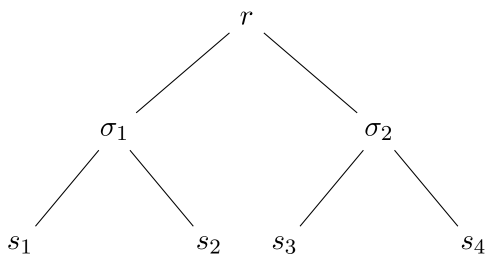
Finally we let $g = r$ and set $\alpha_1 = M(\gamma_1) = s_1$ and $\alpha_2 = D$. Since $\alpha_1 < \alpha_2$ we set $\alpha_2 = r$. Now we have that $\alpha_1 > \alpha_2$ so we set $\alpha_1 = r$ which ends the loop. This gives us that $M(r) = r$ and is marked as a speciation. The complete table of g, M, and speciation/duplication is given by the following.

g	$M(g)$	spec/dup
a	A	-
b	B	-
c	C	-
d	D	-
γ_1	s_1	duplication
γ_2	s_1	speciation
r	r	speciation

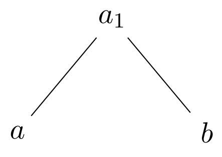
Example A.3 (RecPhylogeny)**.** Consider the the sequence set $G = \{a, b, c, d, e, f, g, h, i, j\}$ as in Example A.1, with the addition of sequence set $S(G) = \{s_1, s_2, s_3, s_4\}$ where $S(\{a, d, h\}) = s_1$, $S(\{b, e, i\}) = s_2$, $S(\{c, f\}) = s_3$ and $S(\{g, j\}) = s_4$. We have the following table.

$S(E)$	$E \subset G$
s_1	a, d, h
s_2	b, e, i
s_3	c, f
s_4	g, j

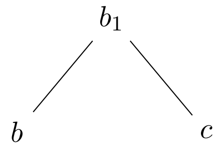
Let the species tree of $S(G)$ be given by $S =$



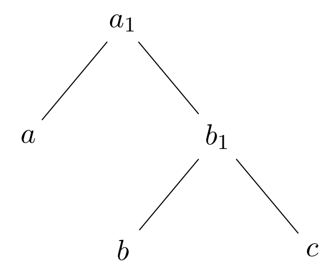
Then we may do roughly the same as in Example A.1 while applying the modified steps this time. To start, if we again choose the type I split $(abc, defghij)$ with a as connection, and then choose type I split (a, bc) with b as connection, then we call RecPhylogeny(ab) with a being considered higher up in the tree than b. Same as before, there are no more type I splits so we form the subphylogeny consisting of a connected with b. Now we traverse the tree ab to refine and reconcile it. Since a has the single child b, we refine it at a and get



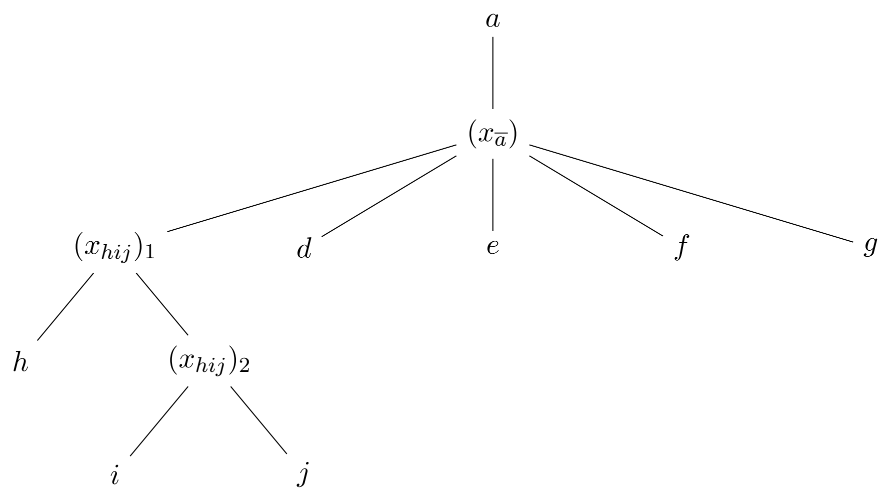
with $M(a_1) = \sigma_1$. Similarly for RecPhylogeny(bc), we refine at b and get the tree



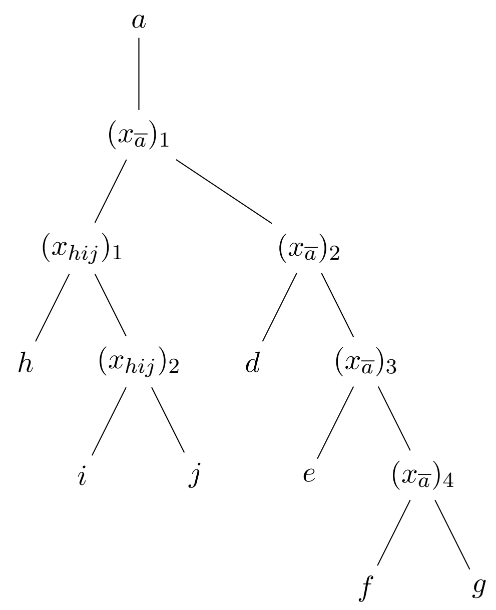
with $M(b_1) = r$. Merging these two together, we get. Since we have added more descendants to a_1, we need to reconcile that vertex again. This gives us $M(a_1) = r$.



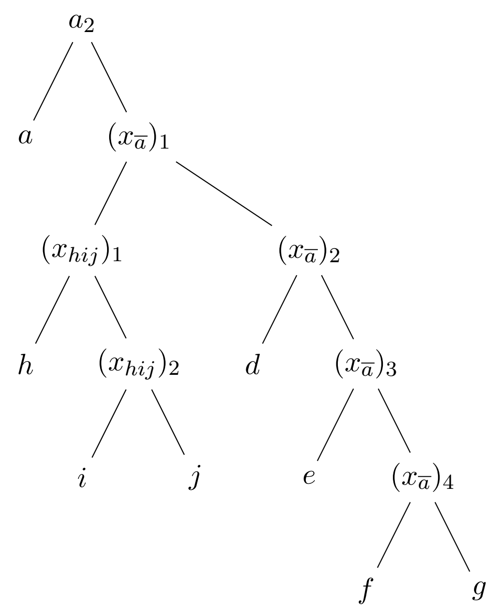
Moving on to RecPhylogeny($adefghij$), we get the same subphylogeny as before with a as connection, and := x_{hij} and $x_{\overline{x}}$ as connecting sequences. This time we traverse the tree to refine x_{hij} and get the tree



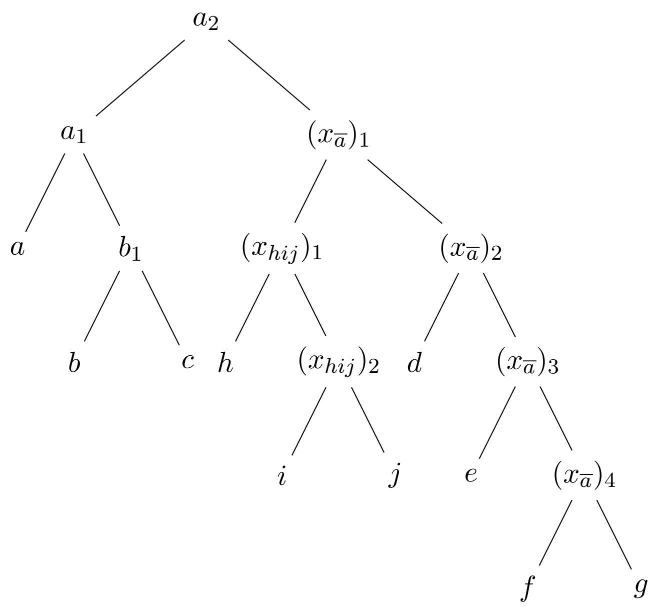
with $M((x_{hij})_2) = r$ and $M((x_{hij})_1) = r$. Then we move on in our traversal to refine and reconcile $x_{\overline{x}}$. We get the tree



with $M((x_{\overline{a}})_4) = \sigma_2$ and $M((x_{\overline{a}})_3) = M((x_{\overline{a}})_2) = M((x_{\overline{a}})_1) = r$ Finally we refine and reconcile a, we get the tree



with $M(a_2) = r$, which ends this run of RecPhylogeny($adefghij$). Now finally we merge the two trees of (abc) and ($adefghij$) together. This gives us



and since we have added more descendants to a_2, we need to reconcile it again. However, we already have $M(a_2) = r$ so there is no other possible lowest common ancestor which we could map a_2 to. Hence we return the above tree and this finishes this run of the algorithm. The following table shows M with speciation and duplication.

x	$M(x)$	spec/dup
b_1	r	speciation
a_1	r	duplication
a_1	r	duplication
$(x_{hij})_2$	r	speciation
$(x_{hij})_1$	r	duplication
$(x_{\overline{a}})_4$	σ_2	speciation
$(x_{\overline{a}})_3$	r	duplication
$(x_{\overline{a}})_2$	r	duplication
$(x_{\overline{a}})_1$	r	duplication

The end result of this artificial example contained a large amount of duplications, but this might not be the case for examples on real data.

References

[AFB94] Richa Agarwala and David Fernández-Baca. A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM Journal on Computing*, 23(6):1216–1224, 1994. arXiv:https://doi.org/10.1137/S0097539793244587, doi:10.1137/S0097539793244587.

[DHV06] Dannie Durand, Bjarni V. Halldórsson, and Benjamin Vernot. A hybrid micro–macroevolutionary approach to gene tree reconstruction. *Journal of Computational Biology*, 13(2):320–335, 2006. PMID: 16597243. arXiv:https://doi.org/10.1089/cmb.2006.13.320, doi:10.1089/cmb.2006.13.320.

[FBL03] David Fernández-Baca and Jens Lagergren. A polynomial-time algorithm for near-perfect phylogeny. *SIAM Journal on Computing*, 32(5):1115–1127, 2003. arXiv:https://doi.org/10.1137/S0097539799350839, doi:10.1137/S0097539799350839.

[GE12] Pawel Górecki and Oliver Eulenstein. Algorithms: simultaneous error-correction and rooting for gene tree reconciliation and the gene duplication problem. *BMC Bioinformatics*, 13(10):S14, Jun 2012. doi:10.1186/1471-2105-13-S10-S14.

[KB18] Soumya Kundu and Mukul S. Bansal. On the impact of uncertain gene tree rooting on duplication-transfer-loss reconciliation. *BMC Bioinformatics*, 19(9):290, Aug 2018. doi:10.1186/s12859-018-2269-0.

[SDEM12] Krister M. Swenson, Andrea Doroftei, and Nadia El-Mabrouk. Gene tree correction for reconciliation and species tree inference. *Algorithms for Molecular Biology*, 7(1):31, Nov 2012. doi:10.1186/1748-7188-7-31.

[ZE01] Christian Zmasek and Sean Eddy. A simple algorithm to infer gene duplication and speciation events on a gene tree. *Bioinformatics (Oxford, England)*, 17:821–8, 10 2001. doi:10.1093/bioinformatics/17.9.821.

[ZZ17] Yu Zheng and Louxin Zhang. Reconciliation with nonbinary gene trees revisited. *J. ACM*, 64(4), aug 2017. doi:10.1145/3088512.