Predicting Horizontal Gene Transfers with Perfect Transfer Networks

Alitzel López Sánchez
Computer Science Department, Université de Sherbrooke, Canada

Manuel Lafond
Computer Science Department, Université de Sherbrooke, Canada

Abstract

Horizontal gene transfer inference approaches are usually based on gene sequences: parametric methods search for patterns that deviate from a particular genomic signature, while phylogenetic methods use sequences to reconstruct the gene and species trees. However, it is well-known that sequences have difficulty identifying ancient transfers since mutations have enough time to erase all evidence of such events. In this work, we ask whether character-based methods can predict gene transfers. Their advantage over sequences is that homologous genes can have low DNA similarity, but still have retained enough important common motifs that allow them to have common character traits, for instance the same functional or expression profile. A phylogeny that has two separate clades that acquired the same character independently might indicate the presence of a transfer even in the absence of sequence similarity.

We introduce perfect transfer networks, which are phylogenetic networks that can explain the character diversity of a set of taxa. This problem has been studied extensively in the form of ancestral recombination networks, but these only model hybridation events and do not differentiate between direct parents and lateral donors. We focus on tree-based networks, in which edges representing vertical descent are clearly distinguished from those that represent horizontal transmission. Our model is a direct generalization of perfect phylogeny models to such networks. Our goal is to initiate a study on the structural and algorithmic properties of perfect transfer networks. We then show that in polynomial time, one can decide whether a given network is a valid explanation for a set of taxa, and show how, for a given tree, one can add transfer edges to it so that it explains a set of taxa.

2012 ACM Subject Classification Applied computing → Molecular evolution

Keywords and phrases Horizontal gene transfer, tree-based networks, perfect phylogenies, character-based, gene-expression, indirect phylogenetic methods

Digital Object Identifier 10.4230/LIPIcs.WABI.2022.3

Funding *Alitzel López Sánchez*: acknowledges financial support from the programme de bourses d'excellence en recherche from the University of Sherbrooke.
Manuel Lafond: acknowledges financial support from the Natural Sciences and Engineering Research Council (NSERC) and the Fonds de Recherche du Québec Nature et technologies (FRQNT).

Acknowledgements The authors would like to thank the reviewers for their helpful comments and for pointing out paper [37].

1 Introduction

Evolution has historically been seen as a tree-like process in which genetic material is inherited through vertical descent. However, it is now established that co-existing species from most kingdoms of life, if not all, have exchanged genetic material laterally through hybridation or horizontal gene transfer (HGT). The latter is well-known to occur routinely between procaryotes [31, 49], but is believed to have affected eucaryotes as well [30, 25]. HGT is also known to occur between viruses and their hosts [27], between mitochondria and the nucleus [2], and between tumor cells [51].

Since HGTs play a significant role in shaping evolution, several bioinformatics approaches have been developed to identify them. Most of these can be classified as either parametric or phylogenetic. Parametric methods are based on the sequence of one genome of interest and attempt to find DNA regions that exhibit a signature that is different from the rest of the genome (see [42]). Phylogeny-based methods consist of taking a set of taxa (e.g. genes and/or species), reconstructing their phylogenetic tree, and inferring the unseen transfer locations on the tree. A common way of achieving this is through *reconciliation*, which aims to explain the discrepancies between a gene tree and a species tree by finding where the gene duplication, loss, or transfer events occurred [5, 13, 23, 11]. Finding a most parsimonious reconciliation under this model is NP-hard [50, 32, 28], mainly because the inferred transfer locations need to be *time-consistent*, meaning that they must occur between species that may have co-existed. In addition, recent fundamental approaches propose to identify pairwise gene relationships to infer transfers. For instance, irregularities in the pairwise gene distances can pinpoint to possible transfers [47], or predictions of orthologs, paralogs, and xenologs can help reconstructing a gene tree and a species *network* that explain these relationships [19, 24, 33, 29].

The above approaches are all based on gene sequences in one way or another, either to reconstruct the phylogenies or to infer pairwise relationships. However, it is well-known that sequences have their limits for predicting HGT, especially in the case of ancient transfers [8]. In this work, we ask whether *character-based* approaches can instead be used to predict HGT on a phylogeny. A character is a generic term to denote a trait that a taxa may possess or not, which can be morphological or molecular. A common example of character-based data is gene expression, where a trait corresponds to whether a species expresses a gene or not in a condition of interest [10, 43, 40]. A major advantage of using gene expression profiles, and possibly other character traits, over sequence data comes when highly divergent sequences are involved. In [1], the authors used expression to recover phylogenetic signals better than using only sequence similarity measures. This could be because the necessary information to coordinate the folding or function of proteins is encoded in a small number of conserved fragments, in which case the two homologous proteins can share a small percentage of sequence similarity. This can be leveraged to detect HGTs that are hard to find using sequences, since one could hypothesize that two clades that started expressing the same gene independently could have acquired this behavior by transfer.

The task in this setting is, given a set of characters $\mathcal{C}$ and a set of taxa $\mathcal{S}$ that each possess a subset of $\mathcal{C}$, to explain the diversity of $\mathcal{S}$ in a phylogeny. Ideally, $\mathcal{S}$ can be explained by a tree in which taxa that possess a common character form a clade, in which case the tree is called a *perfect phylogeny* [6, 16, 4, 26]. When no such perfect phylogeny exists, transfers may be required to explain the data. We point out that recently, character-based methods have resurfaced in tumor phylogenetics, where they are used to represent whether a tumor clone has acquired a somatic mutation or not [12, 41, 34, 46].

Before gene expression and other character-based data can be used to predict HGT, appropriate models and algorithmic frameworks need to be devised. To our knowledge, character-based approaches have mostly been used to detect *hybridation* events, where two or more species recombinate to produce an hybrid offspring. In the most popular models, a set of taxa is explained by an *ancestral recombination graph* (ARG), which is an acyclic directed graph in which nodes with multiple parents represent hybrids, and nodes with a single parent represent vertical descent [52, 22, 21]. The task of finding recombination events is different from that of finding HGTs. Recombinations create offsprings whose genetic content is merged from the parents without vertical descent being involved directly. As a result, there is no donor/recipient relationship. In the case of transfers, it is important to distinguish which traits were acquired vertically from the parent, and which traits were given by a donor.

Another model called perfect phylogenetic networks (PPN) was also introduced in [37, 36] to study the evolution of languages, but can also be used for biological characters. The model asks whether each character is compatible with some tree displayed by a given network. PPNs can be used to infer transfers, but since each character can choose a different parental edge in each reticulation, there is no obvious distinction between vertical descent and horizontal transmission. Also note that PPNs assume that all transfer edges are bidirectional.

In fact, the class of *tree-based networks* has been introduced recently for this purpose [18, 39]. This class of phylogenetic networks captures the idea of having a species tree on which a set of transfer highways were "attached". The *base tree* indicates where vertical descent occurred and the attached transfer edges clearly show where genetic material could have been exchanged.

Our contributions. We introduce *perfect transfer networks* (PTN), which are tree-based networks that can explain how each character was acquired/transferred in a given set of taxa. Our model is a direct generalization of perfect phylogenies to networks, as we use the same set of evolutionary rules. That is, we require that in the network, a character acquired by an ancestral species is never lost by its vertical descendants as in the Camin-Sokal parsimony model [9], and that each character has a unique origin. Additionally, a character can only be transferred horizontally on the edges that are explicitly labeled as transfers. It is worth mentioning that in [3], the authors study an HGT inference framework in which characters that admit a perfect phylogeny are ignored, whereas characters that do not are treated as evidence of transfers. Our work can be seen as an effort to formalize this idea.

We then study the structural and algorithmic aspects of PTNs. We first show that PTNs have two equivalent definitions that are both generalizations of perfect phylogenies. We then distinguish PTNs from recombination networks formally by showing that some taxa sets are explained by different networks depending on the model. As for the algorithmic aspects, we study the question of whether a given tree-based network can explain the characters of a set of taxa and provide a simple, polynomial-time algorithm for the problem. We finally study the tree completion problem where, given a tree, we are asked to add transfers to it so that it explains the input taxa. We show that any tree can explain any set of taxa, even if the characters at the ancestral nodes of the tree are constrained by the input. We then conclude with a discussion on open problems, including the problem of adding a minimum number of transfers to a tree to make it explain a set of taxa.

2 Preliminaries

In this section, we describe the standard phylogenetic notions used in the paper, and then define our perfect transfer network model.

2.1 Phylogenetic networks and tree-based networks

For an integer n, we use the notation $[n] = \{1, \dots, n\}$. All graphs in this work are directed and loopless. A directed graph G is *connected* if the underlying undirected graph of G is connected. A *binary phylogenetic network*, or simply a *network* for short, is a directed acyclic graph $G = (V, E)$ such that either $|V| = 1$, or such that G satisfies the following conditions:

- there is a set of vertices with in-degree 1 and out-degree 0, called *leaves*.
- there is a unique vertex with in-degree 0, called the *root*.

- every other vertex has either in-degree 1 and out-degree 2 (*tree nodes*), in-degree 2 and out-degree 1 (*reticulation nodes*), or in-degree 1 and out-degree 1 (*subdivision nodes*).

We say that $(u, v) \in E$ is a *tree edge* if v is a tree node or a leaf. Note that the usual definition of a network forbids subdivision nodes. We allow them only because it simplifies some of the definitions and proofs.

For a network G, we write $\rho(G)$ for the root of G and $L(G)$ for the leaves. If $|V(G)| = 1$, then we define $\rho(G)$ as the single vertex of G and consider that $L(G) = \{\rho(G)\}$. If σ is a bijection from $L(G)$ to a set $\mathcal{S}$, we call σ an *$\mathcal{S}$-map for G*, or just an *$\mathcal{S}$-map* if G is understood. We say that $u \in V(G)$ *reaches* a node $v \in V(G)$ if there exists a directed path from u to v in G. We denote by $R_u(G)$ the set of nodes that u reaches in G, and we note that $u \in R_u(G)$. For a subset W of $V(G)$, we denote by $G[W]$ the subgraph of G induced by W. We will also denote by $G - W$ the graph obtained by the removal of W from $V(G)$ and all of its incident edges. In other words, $G - W = G[V(G) \setminus W]$.

A *tree* T is a network whose underlying undirected graph has no cycles. We say that $W \subseteq V(T)$ *forms a subtree of* T if $T[W]$ is a tree. We say that a vertex $v \in V(T)$ is an *ancestor* of $u \in V(T)$ if v is on the path from $\rho(T)$ to u. In this case, we will call u a *descendant* of v. Note that v is an ancestor and descendant of itself. For $v \in V(T)$, we will use $T(v)$ to refer to the subtree of T rooted at v (that is, $T(v)$ contains v and all of its descendants).

A network $G = (V, E)$ is a **tree-based network** [38] if G has no subdivision nodes, and there is a partition $\{E_S, E_T\}$ of E such that the subgraph $\mathcal{T}_G = (V, E_S)$ is a tree with the same set of leaves as G, which is called the *support tree* of G. The edges in E_S are called *support edges* and the edges in E_T are called *transfer edges.* Note that $\mathcal{T}_G$ contains subdivision nodes, unless E_T is empty. The tree obtained from $\mathcal{T}_G$ by suppressing its subdivision nodes is called the *base tree* of G (suppressing a subdivision node u with parent p and child v consists of removing u and adding an edge from p to v). Roughly speaking, a tree-based network G can be obtained by starting with a tree and inserting transfer edges into it.

As mentioned in the introduction, networks should be *biologically-feasible* in terms of time. We define a *time-consistent map* over a tree-based network G with support edges E_S and transfer edges E_T as a function $\tau : V \to \mathbb{R}$ such that:

- for every $(u, v) \in E_S$, $\tau(u) > \tau(v)$.
- for every $(u, v) \in E_T$, $\tau(u) = \tau(v)$.

We say that G is a *time-consistent tree-based network* if there exists a time consistent map for G [17]. Note that the existence of a time-consistent map on a network implies that it is tree-based [35] (but the converse does not necessarily hold). In the following sections, we will assume that all the tree-based networks are time-consistent without explicit mention.

2.2 Perfect transfer networks

We now propose to extend the *perfect phylogeny* model to tree-based networks. Let $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ be a set of taxa and $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ a set of characters. We view a taxa S_i as the set of characters that it possesses, so that for each $i \in [n]$, S_i is a subset of $\mathcal{C}$. Our goal is to explain the character diversity of $\mathcal{S}$ using its evolutionary history. Given a tree-based network G with $\mathcal{S}$-map σ, we want to know where each character appeared in G under the conditions that each character has a single origin, that it cannot be lost once acquired, and that it can be transferred. Throughout the phylogenetic literature, requiring a single origin is called the *homoplasy-free* assumption (or sometimes the “no parallel evolution” or “no convergent evolution”), which states that characters cannot arise independently in unrelated lineages [45, 48]. HGT is not considered to be a cause of homoplasy, but of course homoplasy can occur even in the presence of HGT. Nonetheless, this assumption has historically been used as a first step towards more complex models (see e.g. [44]).

To formalize this, given a tree-based network G, a $\mathcal{C}$*-labeling* of G is a function $l : V(G) \to 2^{\mathcal{C}}$ that maps each node of G to the subset of characters that it possesses (here, $2^{\mathcal{C}}$ represents the powerset of $\mathcal{C}$). For a character $c \in \mathcal{C}$, we will denote by $V_c(l) = \{v \in V(G) : c \in l(v)\}$ the set of nodes that possess character c, and we denote by $\overline{V}_c(l) = V(G) \setminus V_c(l)$ the nodes that do not have it. If l is clear from the context, then we may simply write V_c and $\overline{V}_c$.

Our evolutionary requirements are encapsulated in the following definition.

▶ **Definition 1** (Perfect transfer networks). *Let $\mathcal{S}$ be a set of taxa on characters $\mathcal{C}$, let $G = (V, E_S \cup E_T)$ be a tree-based network, and let σ be an $\mathcal{S}$-map for G. We say that a $\mathcal{C}$-labeling l of G* explains *$\mathcal{S}$ if the following conditions hold:*

- *for each $v \in L(G)$, $l(v) = \sigma(v)$;*
- *for each support edge $(u, v) \in E_S$, $c \in l(u)$ implies that $c \in l(v)$ (never lost once acquired);*
- *for each $c \in \mathcal{C}$, there exists a unique node $v \in V_c(l)$ that reaches every node of $V_c(l)$ in $G[V_c(l)]$ (single origin).*

Furthermore, we call the pair (G, σ) a perfect transfer network *(PTN) for $\mathcal{S}$ if there exists a $\mathcal{C}$-labeling of G that explains $\mathcal{S}$.*

See Figure 1.2 for an example of a PTN. Later on in Theorem 2, we will show that Definition 1 is similar to a connectedness requirement known as *convexity* on each character, see [21]. Notice that if G is a tree, then every edge is a support edge and Definition 1 coincides with the definition of a perfect phylogeny from [21]. If $G = (V, E_S \cup E_T)$ is a tree-based network, the definition does not explicitly state what can or cannot be done with transfer edges. The way to see this is that the definition *does not forbid* ancestral taxa from using transfer edges. That is, if $(u, v) \in E_T$, then u can transmit any subset of its characters to v horizontally. We are interested in the following algorithmic problems:

- *The PTN-recognition problem:* given a tree-based network G with $\mathcal{S}$-map σ, is (G, σ) a PTN for $\mathcal{S}$? That is, does there exist a $\mathcal{C}$-labeling of G that explains $\mathcal{S}$? See Figure 1.4 for an example of a network that is not a PTN.
- *The tree-completion problem:* given a tree T with $\mathcal{S}$-map σ, does there exist a PTN (G, σ) for $\mathcal{S}$ such that T is the base tree of G? See Figure 1.2.
- *The tree-minimization problem:* given a tree T with $\mathcal{S}$-map σ, find a PTN (G, σ) for $\mathcal{S}$ such that T is the base tree of G, and such that the number of transfer edges in G is minimized. See Figure 1.3.

We show that the $\mathcal{C}$-labeling problem can be solved in time $O(|\mathcal{C}||V(G)|^2)$. For the tree-completion problem, we provide a more general result: any tree with any given pre-labeling can explain any set of taxa. To be more specific, for any given tree T and any $\mathcal{C}$-labeling of T that satisfies the *never lost once acquired* condition, one can always explain $\mathcal{S}$ by adding transfers in a time-consistent manner while preserving the given labeling. This motivates the need for the minimization problem, which leads to several open problems.

3 Properties of the perfect transfer model

Before delving into the algorithms, we study our model a bit more in-depth. First, we provide an alternate definition of perfect transfer networks in terms of character connectedness. This

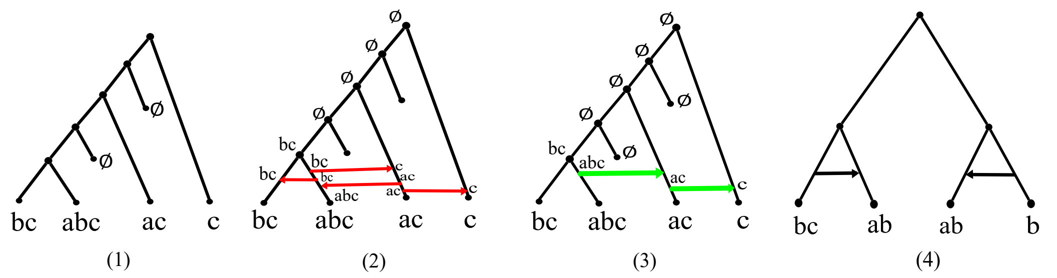

Figure 1 (1) A species tree T with a set of taxa $\mathcal{S}$ on characters $\mathcal{C} = \{a, b, c\}$. (2) A PTN with T as base tree that explains $\mathcal{S}$. Red arrows represent transfer edges. (3) Another PTN with T as base tree that also explains $\mathcal{S}$. Note that this PTN requires two less transfers. (4) A tree-based network G with $\mathcal{S}-$map σ for which no labeling can explain $\mathcal{S}$.

definition is sometimes easier to deal with in our proofs, and is akin to perfect phylogenies that also admit a similar equivalent definition. Second, we ensure that our model does not reinvent the wheel by explicitly stating its differences with the recombination model.

▶ **Theorem 2.** *Let $G = (V, E_S \cup E_T)$ be a tree-based network with $\mathcal{S}$-map σ. Then a $\mathcal{C}$-labeling l of G explains $\mathcal{S}$ if and only if the following conditions hold:*

- *for each $v \in L(G)$, $l(v) = \sigma(v)$;*
- *for each $c \in \mathcal{C}$, $G[V_c(l)]$ is connected and contains a unique node of in-degree 0;*
- *for each $c \in \mathcal{C}$, either $V = V_c(l)$, or $\mathcal{T}_G[\overline{V}_c(l)]$ is connected and contains $\rho(G)$.*

Proof. ($\Rightarrow$) Suppose that l is a $\mathcal{C}$-labeling of G that explains $\mathcal{S}$. We argue that the three conditions stated in the theorem are true. By definition, $l(v) = \sigma(v)$ for each $v \in L(G)$ holds. For the other conditions, let $c \in \mathcal{C}$. Let v be the unique node of $V_c(l)$ that reaches every node in $G[V_c(l)]$, which is guaranteed to exist by Definition 1. Because $G[V_c(l)]$ is acyclic, no node other than v can reach v in $G[V_c(l)]$. This implies that v has in-degree 0 in $G[V_c(l)]$. Moreover, $G[V_c(l)]$ is connected since v reaches all of its nodes. If $\overline{V}_c(l)$ is empty, then the third condition also holds, so assume this is not the case. Let us now focus on $\mathcal{T}_G[\overline{V}_c(l)]$. Observe that because characters are never lost once acquired, l satisfies the property that for every $(u, w) \in E_S$, $w \in \overline{V}_c(l)$ implies that $u \in \overline{V}_c(l)$. This in turn implies that for any $u \in \overline{V}_c(l)$, every ancestor of u in $\mathcal{T}_G$ is in $\overline{V}_c(l)$, including the root $\rho(G)$. Since we assume that $\overline{V}_c(l)$ is non-emtpy, it follows that $\rho(G) \in \overline{V}_c(l)$. It also follows that $\mathcal{T}_G[\overline{V}_c(l)]$ is connected because all of its nodes have a path to $\rho(G)$.

($\Leftarrow$) Suppose that l satisfies all the conditions of the theorem. We show that all properties of Definition 1 hold. For each $v \in L(G)$, we know that $l(v) = \sigma(v)$. For the other conditions, let $c \in \mathcal{C}$. First suppose for contradiction that there is a support edge $(u, v) \in E_S$ such that $u \in V_c(l)$ but $v \in \overline{V}_c(l)$. Thus $\overline{V}_c(l)$ is not empty, in which case $G[\overline{V}_c(l)]$ is connected and contains $\rho(G)$. The path in the support tree $\mathcal{T}_G$ from $\rho(G)$ to v goes through u. But $\rho(G) \in \overline{V}_c(l)$, which is a contradiction since $\mathcal{T}_G[\overline{V}_c(l)]$ is connected, but here u disconnects $\rho(G)$ from v in $\mathcal{T}_G$. Thus the condition of never losing acquired characters holds. Now let v be the unique node of $V_c(l)$ of in-degree 0 in $G[V_c(l)]$. Assume that there is some $u \in V_c(l)$ that v does not reach in $G[V_c(l)]$. Let P_u be the set of nodes of $V_c(l)$ that reach u in $G[V_c(l)]$. Because $G[V_c(l)]$ is acyclic, P_u must contain a node w of in-degree 0, contradicting that v is the unique node of in-degree 0. Thus v reaches every node in $G[V_c(l)]$ and, because it has in-degree 0, it is the unique such node. Thus the single origin condition is satisfied. ◀

Perfect transfer networks versus recombination networks

Before we move on, it is important to put our model in perspective with recombination networks which, to our knowledge, is the closest to our work. In this model, the indices of the characters $\mathcal{C} = \{c_1, \ldots, c_m\}$ determine an ordering of the characters. For a string B, $B[j]$ denotes its j-th character and $B[i..j]$ its substrings containing positions from i to j. Each taxa $S_i \subseteq \mathcal{C}$ can be represented as an m-bit string $\beta(S_i)$ in which $\beta(S_i)[j] = 1$ if and only if S_i possesses character c_j. Given an m-bit string B and an odd integer d, we say that B is a *d-crossover* of two other m-bit strings X and Y if there are indices $i_1, \ldots, i_d$ such that $B = X[1..i_1]Y[i_1 + 1..i_2]X[i_2 + 1..i_3] \ldots Y[i_d + 1..n]$. If d is even, the definition of a d-crossover is the same except that the last substring is $X[i_d + 1..n]$. For a network G and $\mathcal{S}$-map σ, a *binary $\mathcal{C}$-labeling* of G is a function f in which $f(v)$ is an m-bit binary string for each $v \in V(G)$, such that $f(\rho(G))$ only contains 0s. A binary $\mathcal{C}$-labeling f of G *explains $\mathcal{S}$ with d-crossovers* if $f(v) = \beta(\sigma(v))$ for each $v \in L(G)$, and the following holds:

- for each reticulation node v with parents u and w, $f(v)$ is a d-crossover of $f(u)$ and $f(w)$;
- for each tree edge (u, v), $f(v)$ is obtained from $f(u)$ by flipping some 0 positions to 1;
- for each $i \in [m]$, there is at most one tree edge (u, v) with $f(u)[i] = 0$ but $f(v)[i] = 1$.

We say that (G, σ) is an *ancestral recombination graph* (ARG) for $\mathcal{S}$ with d-crossovers if there exists a binary $\mathcal{C}$-labeling of G that explains $\mathcal{S}$ with d-crossovers. We denote by $ARG_d(\mathcal{S})$ the set of all ARGs for $\mathcal{S}$ with d-crossovers. We also denote $ARG_\infty(\mathcal{S}) = \bigcup_{d=1}^{\infty} ARG_d(\mathcal{S})$.

In the literature, single crossovers are modeled with $d = 1$ and have been studied extensively. The $d = 2$ case is often referred to as double crossovers and can also model gene conversion. It was stated in [21] that $d > 6$ is rarely considered in practice.

The most obvious difference between ARGs and PTNs is that ARGs were introduced to model hybridation events, whereas we created PTNs to model horizontal gene transfer. More specifically, ARGs do not differentiate between the two parents of a reticulation node, whereas in our model, one parent only transmits genetic content via vertical descent whereas the other does so via HGT. In fact, ARGs do not need to be tree-based networks. Although this is a fundamental difference, it is also interesting to ask whether, among the class of tree-based networks, the data explanation depends on the model.

To formalize this, we write $PTN(\mathcal{S})$ for the set of perfect transfer networks for $\mathcal{S}$. The next result shows that ARGs with d-crossovers are incomparable with PTNs unless we allow an arbitrary number of crossovers. We emphasize that even though infinite crossovers can emulate transfers, they still cannot distinguish vertical from horizontal inheritance.

▶ **Proposition 3.** *The following relationships between PTNs and ARGs hold:*

- *for any fixed $d \geq 1$, there exists a set of taxa $\mathcal{S}$ on $d + 2$ characters such that $PTN(\mathcal{S}) \setminus ARG_d(\mathcal{S})$ is non-empty;*
- *there exists a set of taxa $\mathcal{S}$ on two characters such that $ARG_1(\mathcal{S}) \setminus PTN(\mathcal{S})$ is non-empty;*
- *for any set of taxa $\mathcal{S}$, $PTN(\mathcal{S}) \subseteq ARG_\infty(\mathcal{S})$.*

The proof can be found in the Appendix. For (1) we show in Figure 2 an example of a network G such that for any set of characters $\mathcal{C}$, (G, σ) is in $PTN(\mathcal{S})$ but not in $ARG_d(\mathcal{S})$. Equivalently, for (2) in Figure 2 we present a network G that can be explained by using single crossovers but for which there is no $\mathcal{C}-$labeling that explains $\mathcal{S}$.

Perfect transfer networks versus perfect phylogenetic networks

The model that is the closest to ours is Perfect Phylogenetic Networks (PPN), as stated in [36, Definition 12.1.2]. The idea is that a network contains several evolutionary trees, and

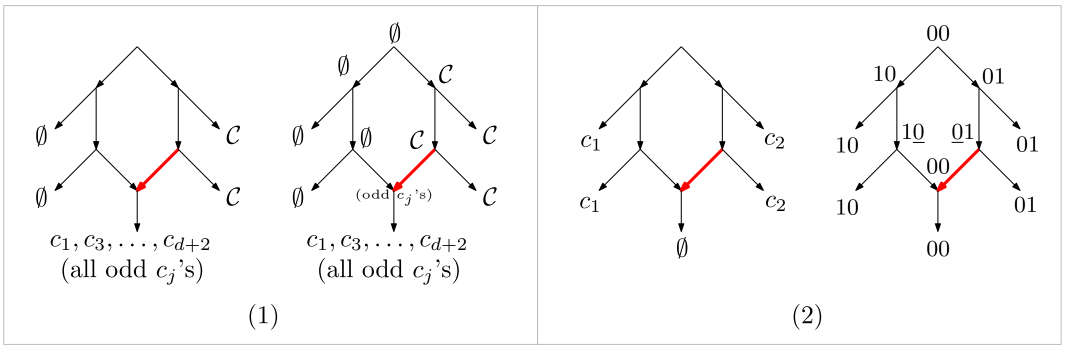

Figure 2 (1) Left: a network G with 5 taxa at the leaves. The fat red edge is a transfer edge. Two taxa have no character, two have all characters of $\mathcal{C}$, and one has only the odd numbered taxa (we assume odd d in the figure). Right: a $\mathcal{C}$-labeling that explains G (the reticulation receives the odd characters from the transfer edge). This network has no explanation with d-crossovers. (2) Left: a network G with 5 taxa on character set $\mathcal{C} = \{c_1, c_2\}$. Right: a binary $\mathcal{C}$-labeling with single crossovers that explains G. This network is not a PTN.

a character could evolve in any one of those trees. More specifically, given a network G and a tree T, we say that T is *displayed* by G if T can be obtained by successively removing reticulation edges and suppressing subdivision nodes. A network G is a PPN for a set of characters $\mathcal{C}$ if, for each $c \in \mathcal{C}$, G displays a tree T that can be labeled so that the nodes that contain c form a connected subtree of T, as well as the nodes that do not contain c.

Figure 3 shows that PPNs can be different from PTNs. The main reason is that PPNs consider all displayed trees as equal, whereas PTNs have a clearly defined base tree.

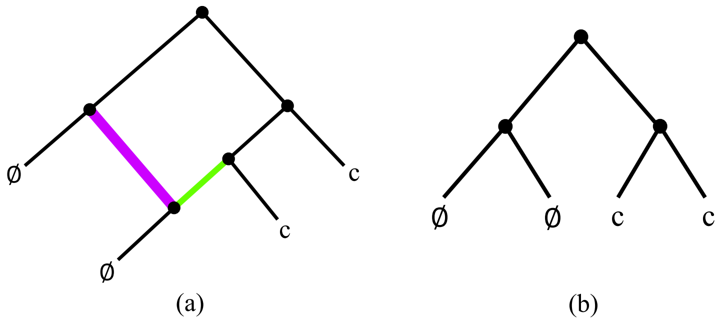

Figure 3 (a) A PPN network G with one character c where the two colored edges enter a reticulation node. (b) A tree displayed by G that admits a labeling such that every node that contains c, or not, forms a connected component. Note that this tree can be obtained by the removal of the green edge and suppression of the resulting subdivision node. As for PTNs, assume that the fat purple edge is a transfer edge. Then a solution for PTNs would not consider the removal of the green edge, in which case no $\mathcal{C}$-labeling is possible.

Also note that it is NP-hard to decide whether a given network G is a PPN for a given set of characters $\mathcal{C}$ [36]. Later on, we will show that the analogous recognition problem for PTNs can be solved in polynomial time. However, our result holds for binary character states and the hardness proof for PPNs relies on the character states being non-binary, and it is unclear whether the hardness is preserved for binary character states.

4 Algorithmic Problems

We first study the problem of deciding whether a given network with known transfers can explain a set of taxa $\mathcal{S}$. We then study the tree completion problem, where we must add the transfers to explain the taxa.

4.1 The PTN recognition problem

The first problem in this section is analogous to the perfect phylogeny problem, where we must find a labeling of all the inner nodes so that the network correctly represents the evolution of a given set of species. This is not as trivial as in the tree case, since there may be multiple options for the originator of a character $c \in \mathcal{C}$.

The PTN recognition problem.
Input. A set of taxa $\mathcal{S}$ on characters $\mathcal{C}$ and a tree-based network $G = (V, E_S \cup E_T)$ with $\mathcal{S}$-map σ.
Output. A $\mathcal{C}$-labeling of G that explains $\mathcal{S}$, if one exists.

Importantly, the above problem formulation lets us assume that we have knowledge of support and transfer edges. We first need some intermediate results.

▶ **Lemma 4.** *Let $G = (V, E_S \cup E_T)$ be a tree-based network with $\mathcal{S}$-map σ, and let l be a $\mathcal{C}$-labeling of G that explains $\mathcal{S}$. Let $u \in \overline{V}_c(l)$. Then for each ancestor v of u in $\mathcal{T}_G$, we have $v \in \overline{V_c}(l)$ as well.*

Proof. Suppose that $u \in \overline{V}_c(l)$ has an ancestor v in $\mathcal{T}_G$ such that but $v \in V_c(l)$. Consider the unique path of $\mathcal{T}_G$ from v to u, namely the sequence $(v, v_1, v_2, \ldots, v_k, u)$. Due to the *never lost once acquired* requirement of Definition 1 $\{v_1, v_2, \ldots, v_k\} \subseteq V_c(l)$. In this way the edge (v_k, u) will represent the loss of character c, a contradiction. ◀

In other words, any node that has at least one descendant in $\overline{V_c}(l)$ should be in $\overline{V_c}(l)$ in every possible $\mathcal{C}-$labeling l that explains $\mathcal{S}$. Since we must have $l(v) = \sigma(v)$ for each leaf v, we can already deduce that if v is a leaf such that $c \notin \sigma(v)$, then no ancestor of v in the support tree can possess c. This is a property similar to character state changes in the Camin-Sokal parsimony method [15]. We thus define the following subset of nodes that are forced to not have c:

$$F_c(G, \sigma) = \{v \in V(G) : \exists w \in L(\mathcal{T}_G(v)) \text{ such that } c \notin \sigma(w)\ \}$$

If G and σ are clear from the context, we will write F_c instead of $F_c(G, \sigma)$. Recall that for a network G, $R_v(G)$ denotes the set of nodes reachable from v. The following characterization of PTNs will let us recognize them easily.

▶ **Lemma 5.** *Let G be a tree-based network with $\mathcal{S}$-map σ. Then (G, σ) is a perfect transfer network for $\mathcal{S}$ if and only if for every character $c \in \mathcal{C}$, $G - F_c$ contains a node v such that $R_v(G - F_c)$ contains every leaf in $L(G - F_c)$.*

Proof. ($\Rightarrow$) Let l be a $\mathcal{C}-$labeling of G that explains $\mathcal{S}$, and let $c \in \mathcal{C}$. By Lemma 4, we must have $F_c \subseteq \overline{V}_c(l)$. The resulting graph $G - F_c$ thus contains all the nodes in $V_c(l)$. Due to the single origin requirement of our definition, $G - F_c$ contains a node v that is able to reach all the leaves in $V_c(l)$, which includes all the leaves of $G - F_c$.

($\Leftarrow$) To show the converse, we build a $\mathcal{C}-$labeling l in the following way: for every $c \in \mathcal{C}$, let v be a node such that its reachable set in $G - F_c$ contains every leaf in $L(G - F_c)$, and let us call v the *origin* of c. Then for $u \in V(G)$, we put $c \in l(u)$ if and only if $u \in R_v(G - F_c)$. That is, the origin of c is v and it is transmitted to every node that v reaches in $G - F_c$. We claim that l satisfies all the conditions of Theorem 2.

Let $u \in L(G)$ and let us first argue that $l(u) = \sigma(u)$. Let $c \in \mathcal{C}$ and let v be the chosen origin of c in l. If $c \notin \sigma(u)$, then $u \in F_c$ and v does not reach u in $G - F_c$ (because u is simply not in $G - F_c$), and by our construction, $c \notin l(u)$. If $c \in \sigma(u)$, then $u \notin F_c$ and v reaches u in $G - F_c$, in which case we put $c \in l(u)$. It follows that $l(u) = \sigma(u)$. We now argue on the connectedness requirements of the $G[V_c(l)]$ subgraphs. Again, let $c \in \mathcal{C}$ and let v be the chosen origin of c in l. Since $V_c(l)$ consists of nodes reachable from v, it is evident that v will be the only node in $G[V_c(l)]$ whose in-degree is 0. It is also evident that $(G - F_c)[V_c(l)]$ is connected, since $V_c(l)$ only contains nodes reachable from v in $G - F_c$. This implies that $G[V_c(l)]$ is also connected, since we cannot disconnect the $V_c(l)$ subgraph by putting back the nodes of F_c into $G - F_c$.

It remains to argue the third condition of Theorem 2. First assume that the origin v of c is equal to $\rho(G)$. Then every leaf must possess c, as otherwise if some leaf u would satisfy $c \notin \sigma(c)$, then $u \in F_c$ and the root would also be in F_c, by its definition. It follows that F_c is empty. Then our construction puts $V_c(l) = V(G)$ since $\rho(G)$ reaches every node in $G - F_c = G$. In this case, the third condition of the theorem holds. So we may assume that $v \neq \rho(G)$. In this case, v does not reach $\rho(G)$ in $G - F_c$ since no node reaches the root (except the root itself). Thus we may assume that $\overline{V}_c(l) \neq \emptyset$ and contains the root. Suppose for a contradiction that $\mathcal{T}_G[\overline{V_c}(l)]$ is disconnected, i.e. there exist some node $u \in \overline{V}_c(l)$ that $\rho(G)$ cannot reach in $G[\overline{V}_c(l)]$. Then in $\mathcal{T}_G$, u has an ancestor $w \in V_c(l)$, which implies that the origin, v, can reach w in $G - F_c$. This implies that $w \notin F_c$, which in turn implies that no node on the from from w to u in $\mathcal{T}_G$ can be in F_c. This means that the path from w to u exists in $G - F_c$. Thus in $G - F_c$, v reaches u through w, and so we would have put $c \in l(u)$, a contradiction. Hence, $\mathcal{T}_G[\overline{V_c}(l)]$ is connected and contains $\rho(G)$. ◀

The proof of the previous lemma implies the following verification algorithm:

▶ **Theorem 6.** *Algorithm 1 correctly solves the* PTN recognition *problem in time* $O(|\mathcal{C}||V(G)|^2)$.

Proof. We first argue that the algorithm is correct. By Lemma 5, it suffices to check, for every $c \in \mathcal{C}$, that some node v reaches every leaf in $G - F_c$. Moreover, when this is the case, the proof of Lemma 5 shows that we can add c to every node in $R_v(G - F_c)$ to obtain a labeling that explains $\mathcal{S}$. If the algorithm finds such a v, this is exactly the labeling that it applies. So we must argue that when such a v exists, the algorithm will find it.

For that, we claim that the verification in line 12 of Algorithm 1 is enough to find the node required by Lemma 5. That is, we do not need to check every node v of $\widehat{G_c} = G - F_c$, but only those in the set X, i.e. the nodes of in-degree 0. Suppose that there is $y \in V(\widehat{G_c})$ that reaches every leaf in $L(\widehat{G_c})$. If $y \in X$, we are done. Otherwise, because $\widehat{G_c}$ is acyclic, there must be $v \in X$ that reaches y (it can be found by following by iteratively following in-neighbors starting from y until such a node is reached). It follows that $R_y(\widehat{G_c}) \subseteq R_v(\widehat{G_c})$ and that v also reaches every leaf. Thus, it suffices to check every source in $\widehat{G_c}$.

Let us now argue the complexity. For a character $c \in \mathcal{C}$, computing F_c can be done in a post-order traversal of $\mathcal{T}_G$ in time $O(|V(G)|)$. Checking whether the resulting network is connected or not can be done in linear time too. Finding the nodes of in-degree 0 takes linear-time and, for each $O(|V(G)|)$ of these nodes, computing $R_v(\widehat{G_c})$ takes time $O(|V(G)|)$. Thus each character requires time $O(|V(G)|^2)$, and thus our algorithm solves the problem in $O(|\mathcal{C}||V(G)|^2)$. ◀

Algorithm 1 Check if a given tree-based network G explains $\mathcal{S}$.

```
function findLabeling($G, \sigma, \mathcal{S}$)
    Set $l(v) = \sigma(v)$ for every leaf $v \in L(G)$.
    Set $l(v) = \emptyset$ for all $v \in V(G) \setminus L(G)$.
    while $l$ not NULL and $\mathcal{C}$ is not empty do
        Pick a character $c \in \mathcal{C}$.
        Compute $F_c(G, \sigma)$.
        Obtain the pruned network $\widehat{G_c} = G - F_c(G, \sigma)$
        if $\widehat{G_c}$ is not connected then
            Set $l = $ NULL.
        else
            Let $X$ be the set of all nodes in $\widehat{G_c}$ with in-degree 0.
            if there is $v \in X$ such that $L(\widehat{G_c}) \subseteq R_v(\widehat{G_c})$ then
                Set $l(u) = l(u) \cup \{c\}$ for all $u$ in $R_v(\widehat{G_c})$.
            else
                Set $l = $ NULL.
        Remove $c$ from $\mathcal{C}$
    return $l$.
```

4.2 The tree-completion problem

We now turn to the problem of predicting transfer locations on a given species tree. As mentioned in [38], species trees depict vertical inheritance and thus serve as basis for our networks, and HGT events can be seen as additional evolutionary events that happen on the way. This motivates the feasibility question: given a set of taxa $\mathcal{S}$ and a species tree T on $\mathcal{S}$, can we add transfer arcs to T so that the resulting network explains $\mathcal{S}$? Moreover, can we do this so that the resulting network is time-consistent? One possibility is that a *universal tree-based network*, which contains all phylogenetic trees on a given set of n leaves [7], could explain any set of characters. However, a base tree needs to be specified in our model, and it is not clear that such a choice always exists in a universal network.

Nevertheless, it turns out that there is no feasibility problem. Indeed, it is relatively easy to show that, for a set of taxa $\mathcal{S}$, any tree T on $\mathcal{S}$ can be complemented with transfers to become a PTN. One way to achieve this is to add a transfer edge between the parent edge of every pair of leaves, from which point it can be argued that the network is a PTN. In fact, we can show a stronger statement: if T is "pre-labeled" in any way that acquired characters are never lost, then we can add transfers to T to explain $\mathcal{S}$ while preserving the given pre-labeling.

Formally, for a tree T, a $\mathcal{C}$-labeling λ of T is called a *no-loss $\mathcal{C}$-labeling* if, for any edge $(u, v) \in E(T)$ and any $c \in \mathcal{C}$, it holds that $c \in l(u)$ implies $c \in l(v)$. Recall that if G is a tree-based network, then the base tree T of G is obtained by removing transfer arcs and suppressing subdivision nodes. Hence, $V(T) \subseteq V(G)$, and we use $V(G) \cap V(T)$ to explicitly refer to the nodes of G that are also in the base tree. We have the following problem.

The Pre-labeled Tree Completion problem.

Input. A set of taxa $\mathcal{S}$ on characters $\mathcal{C}$, a tree T with $\mathcal{S}$-map σ, and a no-loss $\mathcal{C}-$labeling λ of T such that $\lambda(v) = \sigma(v)$ for each $v \in L(T)$.

Output. A time-consistent tree-based network G such that T is the base tree of G, and a $\mathcal{C}$-labeling l of G that explains $\mathcal{S}$ and such that $l(v) = \lambda(v)$ for every $v \in V(G) \cap V(T)$.

We will now prove that this is always possible, even with the time-consistency constraint. One interest of allowing a pre-labeling is that one can start with any hypothesis on where the characters appeared on a tree, and transfers can explain that hypothesis. Perhaps the most natural pre-labeling is the Fitch-like labeling where, for each character c, we add c in every maximal subtree whose leaves have the character. To be more specific, if v is a leaf, $\lambda(v) = \sigma(v)$ and otherwise, let u, w be the children of v, then $\lambda(v) = \lambda(u) \cap \lambda(w)$. This corresponds to the reasonable hypothesis that characters always appear at maximal clades that have it, and we provide and algorithm that can explain this. Again, this is one example of a possible pre-labeling, and our algorithm can explain any other that has the no-loss property, even if characters are again after the lowest common ancestor of the leaves that have this character.

We define a transfer operation between two nodes u and v on a tree-based network G. We write $G\blacktriangle(u, v)$ to denote the tree-based network obtained by subdividing the respective incoming edges of u and v in $\mathcal{T}_G$, thereby creating new parent u' for u and v' for v, and adding the transfer edge (u', v').

It is important to point out that in a no-loss labeling, there can be multiple ancestral species that acquire a specific character that for the first time. In our algorithm, this property will also be maintained in the constructed network, and we will use the following notion:

▶ **Definition 7.** *Let G be a tree-based network with a no-loss $\mathcal{C}-$labeling λ. Let (u, v) be an edge of $\mathcal{T}_G$. We will say that v is a **first-appearance node for** c if it holds that $u \in \overline{V_c}(l)$ and every descendant of v in $\mathcal{T}_G$ belongs to $V_c(l)$.*

Note that if λ is a no-loss labeling, a first-appearance node for c cannot have a first-appearance node for c as a descendant. We can now describe our algorithm. The first step is to make G a copy of the given tree T, and then we add transfer edges to G. Note that in our problem, the given tree has no time map, and deciding where to put the transfers on T in a time-consistent manner becomes surprisingly complicated. For this reason, before adding any transfer, we start by constructing a time consistent map τ for G. It is easy to do this in such a way that no two vertices have the same time.

From that point, we look at each character c and their set of first-appearance nodes $a_1, \ldots, a_k$, ordered in decreasing order of age, and we greedily connect them using transfers. The τ that we constructed dictates the order of connections, in the sense that each a_i is assumed to transfer c to the younger a_{i+1} node. This is achieved by finding a descendant of a_i that could have co-existed between a_{i+1} and its parent. The τ map is also used to assign a time to the nodes created by transfers.

▶ **Lemma 8.** *The network returned by Algorithm 2 is time-consistent under the time map τ.*

▶ **Lemma 9.** *Let G and l be the network and $\mathcal{C}$-labeling returned by Algorithm 2, respectively, on input $T, \mathcal{S}$, and λ. Then T is the base tree of G. Furthermore, l explains $\mathcal{S}$ and satisfies $l(v) = \lambda(v)$ for every $v \in V(G) \cap V(T)$.*

The proofs of lemma 8 and lemma 9 can be found in the Appendix.

▶ **Theorem 10.** *Algorithm 2 solves the* Pre-labeled Tree Completion *problem correctly in time $O(|\mathcal{C}|^2|\mathcal{S}| + |\mathcal{C}||\mathcal{S}|^2)$.*

Proof. By Lemma 8, the network G output by the algorithm has T as its base tree and is time-consistent. Then by Lemma 9, the output labeling l preserves the pre-labeling λ and explains $\mathcal{S}$. Thus, the output is correct.

Algorithm 2 Place an edge between all the first appearance trees.

```
1  function TransferAdditionGreedy($(T, S, \lambda)$)
2      Let $G = (V, E_S \cup E_T)$ be the tree-based network such that $V(G) = V(T)$,
         $E_s = E(T)$ and $E_T = \emptyset$.
3      Let $l$ be the $\mathcal{C}$-labeling in which $l(v) = \lambda(v)$ for all $v \in V(G)$.
4      Let $\tau$ be a time-consistent map for $G$ in which every node has a distinct time.
5      for $c \in \mathcal{C}$ do
6          Compute $A_c$, the set of first appearance nodes of character $c$ in $\mathcal{T}_G$.
7          Let $X_c = (a_1, a_2, \ldots, a_k)$ be the ordering of $A_c$ such that $\tau(a_i) \geq \tau(a_{i+1})$ for
             all $i \in [k-1]$.
8          for $i \in [k-1]$ do
9              Let $a'_i$ be the parent of $a_i$ in $\mathcal{T}_G$
10             Let $a'_{i+1}$ be the parent of $a_{i+1}$ in $\mathcal{T}_G$.
11             if $a_i$ has no descendant $w$ in $\mathcal{T}_G$ such that $(w, a'_{i+1}) \in E_T$ then
12                 Look for $(w', w) \in E(\mathcal{T}_G(a'_i))$ such that $\tau(w') > \tau(a_{i+1})$ and
                     $\tau(w) \leq \tau(a_{i+1})$
13                 Apply the transformation $G\blacktriangle(w, a_{i+1})$.
14                 Let $\hat{w}$ and $\hat{a}_{i+1}$ be the new parents of $w$ and $a_{i+1}$, respectively, in $\mathcal{T}_G$
15                 Set $l(\hat{w}) = (l(w) \cap l(w')) \cup \{c\}$ and $(l(\hat{a}_{i+1}) = l(a'_{i+1}) \cap l(a_{i+1})) \cup \{c\}$.
16                 Set $\tau(\hat{w}) = \tau(\hat{a}_{i+1}) = \dfrac{\min(\tau(w'), \tau(a'_{i+1})) + \tau(a_{i+1})}{2}$
17     return$(G, l)$
```

As for the running time, the complexity is dominated by the main *while* loop over $c \in \mathcal{C}$. Note that first appearance nodes partition the leaves of $G[V_c(l)]$, and so each A_c has at most $|\mathcal{S}|$ elements and, for each c we add at most $|\mathcal{S}|$ transfers. It follows that the final network G output by the algorithm has at most $O(|\mathcal{C}||\mathcal{S}| + |V(T)|)$ nodes. Let us denote $n = |V(G)| \leq |\mathcal{C}||\mathcal{S}| + |V(T)|$. For a given $c \in \mathcal{C}$, computing the first appearance nodes can be done in time $O(n)$ and sorting them takes time $O(n \log n)$. Then for each of the $O(n)$ nodes in X_c, we must find a descendant w in time $O(n)$. The other operations take constant time. Thus, each iteration of the main loop takes time $O(n \log n + n^2) = O(n^2)$. This is repeated $O(|\mathcal{C}|)$ times, and thus the complexity is $O(|\mathcal{C}|n^2) = O(|\mathcal{C}| \cdot (|\mathcal{C}||\mathcal{S}| + |V(T)|))$. The claimed complexity follows from $|V(T)| \in O(|\mathcal{S}|)$ since T is a tree on leafset $\mathcal{S}$. ◀

4.3 On minimizing the number of transfers

We have shown that any tree whose leaves are mapped to $\mathcal{S}$ can explain $\mathcal{S}$. This means that if only $\mathcal{S}$ is known, the question of whether there exists a network that explains $\mathcal{S}$ is uninteresting: it suffices to take any tree on leaf set $\mathcal{S}$ and run Algorithm 2.

This motivates two optimization formulations:

1. The Minimum PTN problem: given a set of taxa $\mathcal{S}$ on character set $\mathcal{C}$, find a PTN for $\mathcal{S}$ with a minimum number of transfers;
2. The Minimum Perfect Transfer Completion problem: given a tree T with $\mathcal{S}$-map σ, add a minimum number of transfers to T so that it becomes a PTN for $\mathcal{S}$.

We leave the complexity status of these as open problems. However, it is interesting to put Algorithm 2 in perspective with the Minimum Perfect Transfer Completion problem. The most natural pre-labeling λ to use is the one where, for each $c \in \mathcal{C}$, we put $c \in \lambda(v)$ for each

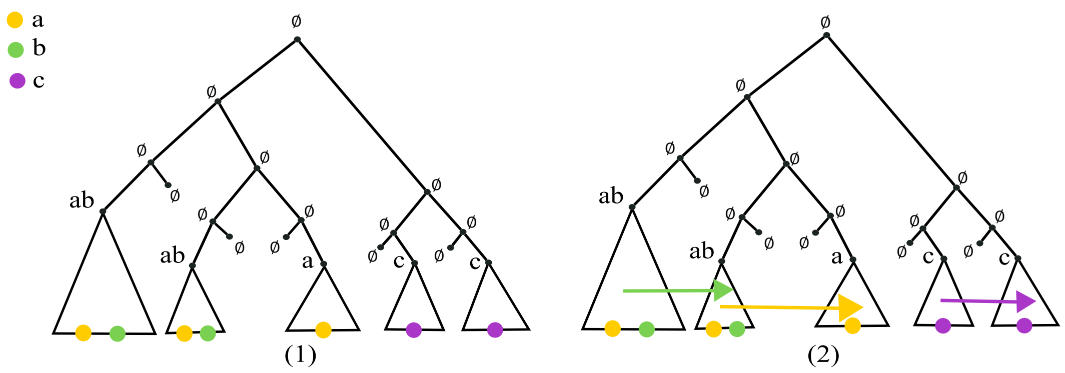

Figure 4 An example of a solution that will be given by our greedy algorithm (1) The given instance T with $\mathcal{C} = \{a, b, c\}$. Triangles represent subtrees and colors represent the characters that can be found on them (2) One possible solution output by Algorithm 2. The green arrow joins the first two subtrees that contain $\{a, b\}$, the yellow arrows joins the second subtree with the subtree that contains only $\{a\}$ and the final arrow connects two subtrees that contain only $\{c\}$.

node whose descending leaves *all* have c. Let us call this the Fitch-labeling. Note that for a minimization problem, an α-approximation is an algorithm whose solution is always at most α times the optimal.

▶ **Proposition 11.** *Suppose that Algorithm 2 is given $T, \mathcal{S}$, and the Fitch-labeling λ. Then Algorithm 2 does not always output an optimal solution to the* Minimum Perfect Transfer Completion *problem. However, it is a $|\mathcal{C}|$-approximation.*

Proof. To see that Algorithm 2 can be suboptimal, consider Figure 5 and the explanation in the caption.

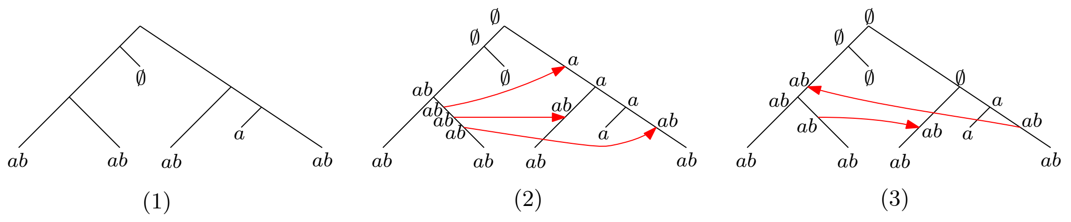

Figure 5 The greedy Algorithm 2 can add more transfers than the minimum. (1) A network G whose leaves are on two characters a and b. (2) One possible solution output by Algorithm 2. The left clade is chosen to originate a and a transfer is greedily added above all the a's on the right side. Then, the left clade is also chosen to originate b, but for that two more separate transfers are added. (3) A solution that is somewhat less natural, but with one less transfer.

Let us argue that it is a $\mathcal{C}$-approximation and let OPT denote the number of transfers added by an optimal solution. Let c be such that in the input T, the number of first-appearance nodes in A_c for c under λ is maximum. Notice that all nodes of T that do not descend from a first-appearance node cannot contain c in any solution by Lemma 4 (since they have a descendant not possessing c). Therefore, any solution for T must add at least $|A_c| - 1$ transfers to T to be able to connect the first-appearance subtrees. Thus $OPT \geq |A_c| - 1$. Algorithm 2 adds exactly $|A_c| - 1$ such transfers when it considers c in its *while* loop (noting that the number of first appearance nodes for c never increases in the algorithm). Since $|A_c|$ is maximum, it follows that Algorithm 2 adds a total of at most $|\mathcal{C}|(|A_c| - 1) \leq |\mathcal{C}|OPT$ transfers. ◀

The cases in which Algorithm 2 is suboptimal appear to have a common cause. The algorithm tends to add transfers as high as possible in the tree, whereas the optmal solution would add more transfers lower in the tree, but with the advantage of being reusable by more characters. For instance in Figure 5, a is greedily transferred by itself and two more separate transfers are required for b, whereas the optimal solution reuses the same transfers for both a and b. It appears difficult to determine when to add more transfers lower and when not to, which leads us to the following.

▶ **Conjecture 12.** *The* Minimum Perfect Transfer Completion *problem is NP-hard.*

We must reckon that a $|\mathcal{C}|$-approximation is far from interesting. We conclude by suggesting a candidate approximation algorithm that combines both algorithms presented here. Suppose that, given T and Fitch-labeling λ, we obtain G and l from Algorithm 2. A simple heuristic post-processing step can then be applied to detect unneeded transfers. That is, for each transfer edge (u, v) of G, consider the network $G - (u, v)$ obtained by removing (u, v) and the resulting subdivision nodes. Then, run Algorithm 1 to check if $G - (u, v)$ is a PTN for $\mathcal{S}$. If so, we know that (u, v) can safely be removed and we repeat. We try every such transfer edge until all of them are necessary. With this modified algorithm, we were unable to generate instances with more than twice as many transfers as the optimal solution. This suggests that it might not be far from optimal.

▶ **Conjecture 13.** *Algorithm 2, combined with the above post-processing step, achieves a constant factor approximation.*

5 Conclusion

In this contribution we have introduced *perfect transfer networks* (PTNs) as a model that aims to combine the structural properties of tree-based networks with the classical notion of perfect characters. In the absence of HGT events, PTNs coincide with the notion of perfect phylogenies. To better understand these, we studied their properties, stated the main differences between them and recombination networks which is to our knowledge the closest model related to ours. Additionally, we explored several algorithmic challenges that result from this model with potential applications for HGT inference methods using character-based information that does not rely on sequence similarities. To our knowledge, this is the first theoretical work that attempts to extend the notion of perfect phylogenies to tree-based networks for the inference of HGT events.

Although widely used throughout the literature, perfect characters impose strong restrictions for our model, since each character is allowed to change of state at most once. A potential extension of our model would be to include different models for character state changes as in *Dollo parsimony* [14]. This model allows losing an acquired character but not gaining it twice. This assumption has been shown to be more suitable for complex characters such as restriction sites and introns [15]. Another possible extension of our model would be to include expression levels of the different characters instead of discrete changes which is a problem similar to that of multi-state perfect phylogenies [20].

Several other questions seem to be appealing for future work. Most importantly, since finding experimental datasets that use gene expression seems like a challenging task, the generation of *in-silico* datasets to test our algorithms seems to be a pertinent solution. Nevertheless, to our knowledge there is no simulation framework that combines evolutionary histories with gene expression data. Therefore, a future direction for this project could also be the design of a simulation environment that is able to generate this type of data.

References

1 Patrick A Alexander, Yanan He, Yihong Chen, John Orban, and Philip N Bryan. The design and characterization of two proteins with 88% sequence identity but different structure and function. *Proceedings of the National Academy of Sciences*, 104(29):11963–11968, 2007.

2 Yoann Anselmetti, Nadia El-Mabrouk, Manuel Lafond, and Aïda Ouangraoua. Gene tree and species tree reconciliation with endosymbiotic gene transfer. *Bioinformatics*, 37(Supplement_1):i120–i132, 2021.

3 Eliran Avni and Sagi Snir. A new phylogenomic approach for quantifying horizontal gene transfer trends in prokaryotes. *Scientific reports*, 10(1):1–14, 2020.

4 Vineet Bafna, Dan Gusfield, Giuseppe Lancia, and Shibu Yooseph. Haplotyping as perfect phylogeny: A direct approach. *Journal of Computational Biology*, 10(3-4):323–340, 2003.

5 Mukul S Bansal, Eric J Alm, and Manolis Kellis. Efficient algorithms for the reconciliation problem with gene duplication, horizontal transfer and loss. *Bioinformatics*, 28(12):i283–i291, 2012.

6 Hans L Bodlaender, Mike R Fellows, and Tandy J Warnow. Two strikes against perfect phylogeny. In *International Colloquium on Automata, Languages, and Programming*, pages 273–283. Springer, 1992.

7 Magnus Bordewich and Charles Semple. A universal tree-based network with the minimum number of reticulations. *Discrete Applied Mathematics*, 250:357–362, 2018.

8 Luis Boto. Horizontal gene transfer in evolution: facts and challenges. *Proceedings of the Royal Society B: Biological Sciences*, 277(1683):819–827, 2010.

9 Joseph H. Camin and Robert R. Sokal. A method for deducing branching sequences in phylogeny. *Evolution*, 19(3):311, September 1965. doi:10.2307/2406441.

10 Gjalt De Jong. Phenotypic plasticity as a product of selection in a variable environment. *The American Naturalist*, 145(4):493–512, 1995.

11 Mattéo Delabre, Nadia El-Mabrouk, Katharina T Huber, Manuel Lafond, Vincent Moulton, Emmanuel Noutahi, and Miguel Sautie Castellanos. Evolution through segmental duplications and losses: a super-reconciliation approach. *Algorithms for Molecular Biology*, 15(1):1–15, 2020.

12 Gianluca Della Vedova, Murray Patterson, Raffaella Rizzi, and Mauricio Soto. Character-based phylogeny construction and its application to tumor evolution. In *Conference on Computability in Europe*, pages 3–13. Springer, 2017.

13 Jean-Philippe Doyon, Celine Scornavacca, K Yu Gorbunov, Gergely J Szöllősi, Vincent Ranwez, and Vincent Berry. An efficient algorithm for gene/species trees parsimonious reconciliation with losses, duplications and transfers. In *RECOMB international workshop on comparative genomics*, pages 93–108. Springer, 2010.

14 James S. Farris. Phylogenetic Analysis Under Dollo's Law. *Systematic Biology*, 26(1):77–88, March 1977. doi:10.1093/sysbio/26.1.77.

15 Joseph Felsenstein. *Inferring phylogenies*. Sunderland, Mass. : Sinauer Associates, 2004.

16 David Fernández-Baca. The perfect phylogeny problem. In *Steiner Trees in Industry*, pages 203–234. Springer, 2001.

17 Andrew Francis, Charles Semple, and Mike Steel. New characterisations of tree-based networks and proximity measures. *Advances in Applied Mathematics*, 93:93–107, 2018. doi:10.1016/j.aam.2017.08.003.

18 Andrew R Francis and Mike Steel. Which phylogenetic networks are merely trees with additional arcs? *Systematic Biology*, 64(5):768–777, 2015.

19 Manuela Geiß, John Anders, Peter F Stadler, Nicolas Wieseke, and Marc Hellmuth. Reconstructing gene trees from fitch's xenology relation. *Journal of Mathematical Biology*, 77(5):1459–1491, 2018.

20 Dan Gusfield. The multi-state perfect phylogeny problem with missing and removable data: Solutions via integer-programming and chordal graph theory. *Journal of Computational Biology*, 17(3):383–399, March 2010. doi:10.1089/cmb.2009.0200.

21 Dan Gusfield. *ReCombinatorics: the algorithmics of ancestral recombination graphs and explicit phylogenetic networks*. MIT press, 2014.

22 Dan Gusfield, Satish Eddhu, and Charles Langley. Optimal, efficient reconstruction of phylogenetic networks with constrained recombination. *Journal of Bioinformatics and Computational Biology*, 2(01):173–213, 2004.

23 Marc Hellmuth, Katharina T Huber, and Vincent Moulton. Reconciling event-labeled gene trees with mul-trees and species networks. *Journal of Mathematical Biology*, 79(5):1885–1925, 2019.

24 Marc Hellmuth, Carsten R Seemann, and Peter F Stadler. Generalized fitch graphs II: Sets of binary relations that are explained by edge-labeled trees. *Discrete Applied Mathematics*, 283:495–511, 2020.

25 Julie C Dunning Hotopp. Horizontal gene transfer between bacteria and animals. *Trends in genetics*, 27(4):157–163, 2011.

26 Leo Van Iersel, Mark Jones, and Steven Kelk. A third strike against perfect phylogeny. *Systematic Biology*, 68(5):814–827, 2019.

27 Nicholas AT Irwin, Alexandros A Pittis, Thomas A Richards, and Patrick J Keeling. Systematic evaluation of horizontal gene transfer between eukaryotes and viruses. *Nature microbiology*, 7(2):327–336, 2022.

28 Edwin Jacox, Mathias Weller, Eric Tannier, and Celine Scornavacca. Resolution and reconciliation of non-binary gene trees with transfers, duplications and losses. *Bioinformatics*, 33(7):980–987, 2017.

29 Mark Jones, Manuel Lafond, and Celine Scornavacca. Consistency of orthology and paralogy constraints in the presence of gene transfers. *Peer Community in Mathematical and Computational Biology*, 2012.

30 Patrick J Keeling and Jeffrey D Palmer. Horizontal gene transfer in eukaryotic evolution. *Nature Reviews Genetics*, 9(8):605–618, 2008.

31 Eugene V Koonin, Kira S Makarova, and L Aravind. Horizontal gene transfer in prokaryotes: quantification and classification. *Annual Reviews in Microbiology*, 55(1):709–742, 2001.

32 Misagh Kordi and Mukul S Bansal. On the complexity of duplication-transfer-loss reconciliation with non-binary gene trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 14(3):587–599, 2015.

33 Manuel Lafond and Marc Hellmuth. Reconstruction of time-consistent species trees. *Algorithms for Molecular Biology*, 15(1):1–27, 2020.

34 Salem Malikic, Farid Rashidi Mehrabadi, Simone Ciccolella, Md Khaledur Rahman, Camir Ricketts, Ehsan Haghshenas, Daniel Seidman, Faraz Hach, Iman Hajirasouliha, and S Cenk Sahinalp. Phiscs: a combinatorial approach for subperfect tumor phylogeny reconstruction via integrative use of single-cell and bulk sequencing data. *Genome research*, 29(11):1860–1877, 2019.

35 Yukihiro Murakami. *On Phylogenetic Encodings and Orchard Networks*. PhD thesis, TU Delft, 2021.

36 Luay Nakhleh. *Phylogenetic networks*. PhD thesis, The University of Texas at Austin, 2004.

37 Luay Nakhleh, Don Ringe, and Tandy Warnow. Perfect phylogenetic networks: A new methodology for reconstructing the evolutionary history of natural languages. *Language*, 81(2):382–420, 2005. URL: `http://www.jstor.org/stable/4489897`.

38 Joan Carles Pons, Charles Semple, and Mike Steel. Tree-based networks: characterisations, metrics, and support trees. *Journal of Mathematical Biology*, 78(4):899–918, October 2018. `doi:10.1007/s00285-018-1296-9`.

39 Joan Carles Pons, Charles Semple, and Mike Steel. Tree-based networks: characterisations, metrics, and support trees. *Journal of Mathematical Biology*, 78(4):899–918, 2019.

40 Beatriz Pontes, Raúl Giráldez, and Jesús S Aguilar-Ruiz. Configurable pattern-based evolutionary biclustering of gene expression data. *Algorithms for Molecular Biology*, 8(1):1–22, 2013.

41 Dikshant Pradhan and Mohammed El-Kebir. On the non-uniqueness of solutions to the perfect phylogeny mixture problem. In *RECOMB International Conference on Comparative Genomics*, pages 277–293. Springer, 2018.

42 Matt Ravenhall, Nives Škunca, Florent Lassalle, and Christophe Dessimoz. Inferring horizontal gene transfer. *PLoS Computational Biology*, 11(5):e1004095, 2015.

43 Arun Rawat, Georg J Seifert, and Youping Deng. Novel implementation of conditional co-regulation by graph theory to derive co-expressed genes from microarray data. In *BMC Bioinformatics*, volume 9, pages 1–9. Springer, 2008.

44 Don Ringe, Tandy Warnow, and Ann Taylor. Indo-european and computational cladistics. *Transactions of the Philological Society*, 100(1):59–129, 2002.

45 Michael J Sanderson and Larry Hufford. *Homoplasy: the recurrence of similarity in evolution*. Elsevier, 1996.

46 Palash Sashittal, Simone Zaccaria, and Mohammed El-Kebir. Parsimonious clone tree reconciliation in cancer. In *Leibniz International Proceedings in Informatics, LIPIcs*, volume 201, page 9. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021.

47 David Schaller, Manuel Lafond, Peter F Stadler, Nicolas Wieseke, and Marc Hellmuth. Indirect identification of horizontal gene transfer. *Journal of Mathematical Biology*, 83(1):1–73, 2021.

48 Charles Semple and Mike Steel. Tree reconstruction from multi-state characters. *Advances in Applied Mathematics*, 28(2):169–184, 2002.

49 Christopher M Thomas and Kaare M Nielsen. Mechanisms of, and barriers to, horizontal gene transfer between bacteria. *Nature Reviews Microbiology*, 3(9):711–721, 2005.

50 Ali Tofigh, Michael Hallett, and Jens Lagergren. Simultaneous identification of duplications and lateral gene transfers. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(2):517–535, 2010.

51 Catalina Trejo-Becerril, Enrique Pérez-Cárdenas, Lucía Taja-Chayeb, Philippe Anker, Roberto Herrera-Goepfert, Luis A Medina-Velázquez, Alfredo Hidalgo-Miranda, Delia Pérez-Montiel, Alma Chávez-Blanco, Judith Cruz-Velázquez, et al. Cancer progression mediated by horizontal gene transfer in an in vivo model. *PloS One*, 7(12):e52754, 2012.

52 Lusheng Wang, Kaizhong Zhang, and Louxin Zhang. Perfect phylogenetic networks with recombination. *Journal of Computational Biology*, 8(1):69–78, 2001.

A Appendix

Proof of Proposition 3

Proof. For (1), consider the network G in Figure 2.1. For any $\mathcal{C}$, the labeled network shows that (G, σ) is in $PTN(\mathcal{S})$. Put $\mathcal{C} = \{c_1, \dots, c_{d+2}\}$. We argue that (G, σ) is not in $ARG_d(\mathcal{S})$. Note that in any binary $\mathcal{C}$-labeling of G that explains $\mathcal{S}$, the parent of each $\emptyset$ leaf taxa must be the string $00\dots0$, as otherwise they would need to transmit a 1 to these leaves. Likewise, the parent of the upper $\mathcal{C}$ leaf taxon must be $11\dots1$. If not, there would be a $0-1$ flip on the edge leading to the upper $\mathcal{C}$. But, we would also need another $0-1$ flip somewhere on the path to the lower $\mathcal{C}$ taxon, which is not allowed in ARGs. So, the parents of the single reticulation have labels $00\dots0$ and $11\dots1$. Moreover, the root is labeled $00\dots0$, and thus all the possible $0-1$ flips occur between the root and its second child. We cannot have any new $0-1$ flip, and so the reticulation must be labeled $0101\dots01$ to be able to transmit the required characters to the odd-characters leaf (or the last character is 0 if d is even). Whether d is odd or even, there are $d+2$ characters and we must alternate each of them between the parents. This requires $d+1$ crossovers, and so $G \notin ARG_d(\mathcal{S})$.

For (2), consider the network G in Figure 2.2. There are only two characters c_1, c_2. The right side of the figure shows that G can be explained under single crossovers. However, we can argue that (G, σ) is not a perfect transfer network. This is because if there is a

$\mathcal{C}$-labeling l that explains the taxa at the leaves, the parent of each c_1 leaf must contain c_1. But by the *never lose once acquired* condition, c_1 should then be transmitted vertically to the leaf that has no character, a contradiction. Thus $(G, \sigma) \notin PTN(\mathcal{S})$.

For (3), let $(G, \sigma) \in PTN(\mathcal{S})$. Let l be a $\mathcal{C}$-labeling of G that explains $\mathcal{S}$. To show that $(G, \sigma) \in ARG_\infty(\mathcal{S})$, we need to modify l slightly. Specifically, we ensure that characters never have their first appearance on a reticulation node. Let v be a reticulation node of G with parents u, w such that (u, v) is a support edge and (w, v) is a transfer edge. Suppose that there is $c_j \in \mathcal{C}$ such that $c_j \notin l(u), c_j \notin l(w)$, but $c_j \in l(v)$. It follows from Theorem 2 that v is the unique node of in-degree 0 in $G[V_{c_j}(l)]$. Consider the labeling l' obtained by taking l, but applying $l'(v) = l(v) \setminus \{c_j\}$. Let v' be the unique child of v. One can easily see that $G[V_{c_j}(l')]$ is connected and that v' is its unique node of in-degree 0. Moreover, $\mathcal{T}_G[\overline{V}_{c_j}(l')]$ is the same as $\mathcal{T}_G[\overline{V}_{c_j}(l)]$ but with v added. It is still connected since v is a child of $u \in \overline{V}_{c_j}(l)$, and it still contains the root. Hence by Theorem 2, l' also explains $\mathcal{S}$.

By applying the above argument to every character, we obtain a labeling l that explains $\mathcal{S}$ such that for any character c_j, the first appearance of c_j under l does not occur at a reticulation node. Assume that l has this property.

Consider the binary $\mathcal{C}$-labeling f of G in which, for each $v \in V(G)$, we put $f(v)[j] = 1$ if and only if $c_j \in l(v)$. We claim that f explains $\mathcal{S}$ with m-crossovers, where $m = |\mathcal{C}|$. First consider a reticulation node v with parents u and w, where (u, v) is a support edge and (w, v) is a transfer edge. We must argue that $f(v)$ is an m-crossover of $f(u)$ and $f(w)$. Because under l, no character first appears on a reticulation, we have that for each $c_j \in l(v)$, either $c_j \in l(u)$ or $c_j \in l(w)$. Moreover, for each $c_j \notin l(v)$, we must have $c_j \notin l(u)$ (by the *never lose once acquired* condition). In terms of f, this means that for each $j \in [m]$, if $f(v)[i] = 1$ then one of u or w also has a 1 in position j, and if $f(v) = 0$, then $f(u)[j] = 0$. It follows that $f(v)$ can be obtained with at most m crossovers from its parents.

Second, it is not hard to see that for each character c_j, there is at most one tree edge (u, v) that flips c_j from 0 to 1 under f. Indeed, if there were two such edges, then under l, there would be two trees nodes that possess c_j but not their (unique) parent. Thus $G[V_{c_j}(l)]$ would have two nodes of in-degree 0, contradicting Theorem 2. Third, the condition that characters are not lost after being acquired implies that, for every tree edge (u, v), $f(v)$ can be obtained from $f(u)$ by flipping 0s to 1s, but not vice-versa (this follows from the fact that tree edges are support edges). Thus, f explains $\mathcal{S}$ and we conclude that $(G, \sigma) \in ARG_\infty(\mathcal{S})$. ◀

Proof of Lemma 8

Proof. We argue that at any moment during the execution of the algorithm, τ is a time-consistent map of G. We prove this by induction on the number of iterations undertaken. Notice that as a base case, the statement is initially true before entering the main *for* loop. Now assume that we have inserted $j - 1$ transfer edges and that τ is time-consistent for G after these insertions. Consider the j-th transfer edge inserted into G. This transfer edge is $(\hat{w}, \hat{a}_{i+1})$, which are created between w and its parent w' in the support tree T', and between a_{i+1} and its parent a'_{i+1} in T', respectively.

We first claim that (w', w) as used in the algorithm exists in the subtree $T(a_i)$, where a_i is the node that precedes a_{i+1} in X_c. We know that $\tau(a_i) \geq \tau(a_{i+1})$. Suppose that $\tau(a_i) = \tau(a_{i+1})$. Let a'_i be the parent of a_i, then by induction hypotesis, since the network is time consistent we have that $\tau(a'_i) > \tau(a_i) = \tau(a_{i+1})$ and so $w' = a'_i$ and $w = a_i$. In this case, we do have $\tau(w') > \tau(a_{i+1})$ and $\tau(w) \leq \tau(a_{i+1})$. Now, suppose that $\tau(a_i) > \tau(a_{i+1})$. Note that we can always order the vertices of $T(a_i)$ with respect to τ in such a way that

$\tau(a_i) \geq v$ for all $v \in V(T(a_i))$. By induction hypothesis, every time we pick a descendant y of a_i, $\tau(y) < \tau(a_i)$, so w can be found by iteratively following the descendants of a_i and choosing the first one whose time is at most $\tau(a_{i+1})$.

Note that by adding the transfer edge $(\hat{w}, \hat{a}_{i+1})$, the times τ of every edge have remained unchanged and still satisfy the time-consistency definition, with the exception of the edges linking $w', \hat{w}$, and w, those linking $a'_{i+1}, \hat{a}_{i+1}$, and a_{i+1}, as well as the new transfer edge. Since $\tau(\hat{w}) = \tau(\hat{a}_{i+1})$ is made explicit on line 16, to conclude the proof it suffices to show that $\tau(w') > \tau(\hat{w}) > \tau(w)$ and that $\tau(a'_{i+1}) > \tau(\hat{a}_{i+1}) > \tau(a_{i+1})$. By induction we know that $\tau(w') > \tau(w)$ and that $\tau(a'_{i+1}) > \tau(a_{i+1})$ (this holds before and after the transfer insertion because they were not changed). Using these inequalities we have that:

$$\tau(\hat{w}) = \frac{\min(\tau(w'), \tau(a'_{i+1})) + \tau(a_{i+1})}{2} < \frac{\tau(w') + \tau(w')}{2} = \tau(w')$$

since $\min(\tau(w'), \tau(a'_{i+1})) \leq \tau(w')$ and $\tau(a_{i+1}) < \tau(w')$ both hold. Note that it is also true that

$$\tau(\hat{w}) > \frac{\tau(a_{i+1}) + \tau(a_{i+1})}{2} = \tau(a_{i+1}) \geq \tau(w)$$

since $\min(\tau(w'), \tau(a'_{i+1})) > \tau(a_{i+1})$ (because $\tau(w') > \tau(a_{i+1})$ and $\tau(a'_{i+1}) > \tau(a_{i+1})$ both hold). Using the same arguments, we have

$$\tau(\hat{a}_{i+1}) = \frac{\min(\tau(w'), \tau(a'_{i+1})) + \tau(a_{i+1})}{2} < \frac{\tau(a'_{i+1}) + \tau(a'_{i+1})}{2} = \tau(a'_{i+1})$$

and

$$\tau(\hat{a}_{i+1}) > \frac{\tau(a_{i+1}) + \tau(a_{i+1})}{2} > \tau(a_{i+1})$$ ◀

Proof of Lemma 9

Proof. Let $G = (V, E_S \cup E_T)$ be the network returned by the algorithm and let l be the returned labeling. To see that $l(v) = \lambda(v)$ for every node v in the base tree, it suffices to note that the algorithm never changes the labeling of a node initially present in T: it only assigns sets of characters to nodes created by transfer insertion operations. It is also easy to see that T is the base tree of G, since we start with a copy of T and only attach transfer edges to it.

We will argue that the returned $\mathcal{C}-$labeling l explains $\mathcal{S}$ using the conditions required by Definition 1. First, by requirement on the input λ, we know that for every $v \in L(G)$, $l(v) = \lambda(v) = \sigma(v)$.

Let us next show that for every character $c \in \mathcal{C}$ there exists a unique node $v \in V_c(l)$ that reaches every node in $G[V_c(l)]$. It is not hard to see that after handling a particular $c \in \mathcal{C}$ in the algorithm, this property will be satisfied for $V_c(l)$. However, it is not obvious that the subsequent iterations on other characters will not "break" this property for c. We thus prove the following statement. Assume that the algorithm handles the characters of $\mathcal{C}$ in order $c_1, \ldots, c_m$ in the main *while* loop. Then we claim that in the network G obtained after finishing the i-th iteration and handling $c_1, c_2, \ldots, c_i$, for every $j \leq i$, there exists a unique node $v \in V_{c_j}(l)$ that reaches every node in $G[V_{c_j}(l)]$. This shows the desired property since it will hold for $j = m$, i.e. for every character. As a base case, consider $i = 0$. Then it is true that for $j \leq i$, the desired node v exists (because there is no c_j to satisfy). Now, assume that $i > 0$ and that before entering the i-th iteration, the statement holds for every c_j with $j \leq i - 1$.

After we are done handling c_i on the $i-$th iteration we know that there exists a vertex $a_1 \in X_{c_i}$ such that $\tau(a_1) \geq \tau(a_h)$ for all $a_h \in X_{c_i}$. In particular, when equality holds for some a_h, it must be that $\tau(a_1) = \tau(a_2)$. In this case, when the algorithm iterates on a_1, we get $(w', w) = (a'_1, a_1)$ and the corresponding transformation yields $G\blacktriangle(a_1, a_2)$. In this case, we claim that the vertex $\hat{a}_1$ created by the subdivision of the edge $(a'_1, a_1) \in E_s$ will remain as the unique source for $V_{c_i}(l)$. To see this first note that a_1 has no incoming edge from $V_c(l)$ at the start of the iteration, because if that was not the case then $c \in l(a'_1)$ and so a_1 would not have been a first-appearance node in the first place. This in turn implies that $\hat{a}_1$ has no incoming edge after the i-th iteration, since all other transfer heads are added above $a_2, \ldots, a_k$. Additionally, $\hat{a}_1$ will become the first-appearance node for its corresponding subtree. After applying $G\blacktriangle(a_1, a_2)$, $\hat{a}_1$ now reaches the new parent $\hat{a}_2$ of a_2 and all its descendants in $\mathcal{T}_G$. Subsequently, we will now choose a descendant w of a_2 from which we will add a new transfer to the parent node $\hat{a}_3$ of a_3. In this way, $\hat{a}_1$ will also reach a_3 and all of its descendants. We will continue adding transfer operations in this way so that finally $\hat{a}_1$ will be able to reach the last vertex of X_{c_j}, a_k and all of its descendants. Note that any node of $V_{c_i}(l)$ is reachable by an element of A_{c_i}, thus after the $i-$th iteration $\hat{a}_1$ reaches every node in the $G[V_{c_i}(l)]$ subgraph.

On the other hand, when we have the strict inequality, i.e. when $\tau(a_1) > \tau(a_2)$, then the first chosen w is distinct from a_1, and using the same arguments, we see that a_1 is the unique origin for $V_{c_i}(l)$.

We must also argue that the i-th iteration does not "break" a c_j with $j < i$. Consider such a c_j. By induction, before the i-th iteration, $G[V_{c_j}(l)]$ had a unique origin v. Suppose that during the $i-$th iteration of the main loop we created some transformation $G\blacktriangle(w, a_h)$ after which some node, say z, that possesses c_j cannot be reached by v in the transformed graph. Let w', a'_h be the parents of w and a_h, respectively, before the addition of the transfer. Also let $\hat{w}$ and $\hat{a}_h$ be the nodes which were created by this transformation.

First assume that $z = \hat{w}$. Then $c_j \notin l(w')$, as otherwise if $c_j \in l(w')$, by induction, v would reach w' and thus also reach $\hat{w} = z$. But because $c_j \notin l(w')$, $c_j \notin l(w') \cap l(w)$ and so the algorithm would not have put c_j in $\hat{w} = z$, a contradiction. By the same argument, $z \neq \hat{a}_h$. Thus, z was present in G before the insertion of the transfer.

Next, assume that $c_j \notin l(\hat{w})$ and $c_j \notin l(\hat{a}_h)$. If v cannot reach z anymore, every path from v to z in $G[V_{c_j}(l)]$ must have been going through (w', w) or (a'_h, a_h) before the transfer insertion. But then, $c_j \in l(w') \cap l(w)$ and $c_j \in l(a'_h) \cap l(a_h)$, and the algorithm would have put $c_j \in l(\hat{w})$ and $c_j \in l(\hat{a}_h)$, a contradiction.

Then either $c_j \in l(\hat{w})$, $c_j \in l(\hat{a}_h)$ or $c_j \in l(\hat{w}) \cap l(\hat{a}_g)$. Assume $c_j \in l(\hat{w})$. By line 15, we know that this would only be possible if $c_j \in l(w') \cap l(w)$. So any path from v to z in $V_{c_j}(l)$ that used the (w', w) edge can now use the edges $(w', \hat{w}), (\hat{w}, w)$ to reach z. If $c_j \in l(\hat{a}_h)$ as well, the same idea applies, and we get that any path from v to z is still usable, albeit with either $\hat{w}$ or $\hat{a}_h$ as an additional vertex. So it must be that $c_j \notin l(\hat{a}_h)$, and that all paths used (a'_h, a_h). As before, this means that $c_j \in l(a'_h) \cap l(a_h)$ and that we should have $c_j \in l(\hat{a}_h)$, a contradiction. This covers the case $c_j \in l(\hat{w})$. The case $c_j \in l(\hat{a}_h)$ can be handled in the same manner.

We then show that for each support edge $(u, v) \in E_S, c \in l(u)$ implies that $c \in l(v)$. We argue that this property holds before and after any transfer edge is inserted. Notice that initially, when G is just a copy of T and l a copy of λ, $c \in l(u)$ implies $c \in l(v)$ because λ is a no-loss labeling. Now suppose inductively that the property holds before we insert some transfer $(\hat{w}, \hat{a}_{i+1})$ by line 13. It suffices to argue that the property holds on the support edges $(w', \hat{w}), (\hat{w}, w), (a'_{i+1}, \hat{a}_{i+1})$, and $(\hat{a}_{i+1}, a_{i+1})$, as defined in the algorithm, because no other support edge is modified. Let $c' \in l(w')$ (we distinguish c' from c, the latter being the c the algorithm is currently iterating on). Then by assumption that the property held before the transfer addition, we must have $c' \in l(w)$ and, because $c' \in l(w) \cap l(w')$, c' will be added to $l(\hat{w})$, as desired. The same argument holds for $c' \in l(a'_{i+1})$ and the fact that $c' \in l(\hat{a}_{i+1})$. Now let $c' \in l(\hat{w})$. We want to argue that $c' \in l(w)$. If $c' \in l(w')$, then again by assumption we have $c' \in l(w)$ as well and our property holds. So suppose that $c' \notin l(w')$. The algorithm puts $l(\hat{w}) = (l(w) \cap l(w')) \cup \{c\}$ where c is the character of the current iteration, which means that only $c = c'$ is possible. Notice that the algorithm chooses the edge (w, w') in the subtree $\mathcal{T}_G(a'_i)$, where a'_i has child a_i that is a first appearance node for c. By assumption, every descendant of a_i in the support tree possesses c, so only $w' = a'_i$ is possible. Thus $w = a_i$ and $c \in l(a_i) = l(w)$, as desired. Finally, let $c' \in l(\hat{a}_{i+1})$. If $c' \in a'_{i+1}$, by assumption $c' \in l(a_{i+1})$ and we are done. Otherwise, as the previous case we must have $c' = c$ and, since a_{i+1} is a first appearance for c, we have $c \in l(a_{i+1})$ as desired. ◀