On the Comparison of LGT networks and Tree-based Networks

Bertrand Marchand[1], Nadia Tahiri[2], Olivier Tremblay-Savard[3], and Manuel Lafond[2]

[1] Université du Québec à Montréal, Canada
[2] Université de Sherbrooke, Canada
[3] University of Manitoba, Canada

Abstract. Phylogenetic networks are widespread representations of evolutionary histories for taxa that undergo hybridization or Lateral-Gene Transfer (LGT) events. There are now many tools to reconstruct such networks, but no clearly established metric to compare them. Such metrics are needed, for example, to evaluate predictions against a simulated ground truth. Despite years of effort in developing metrics, known dissimilarity measures either do not distinguish all pairs of different networks, or are extremely difficult to compute. Since it appears challenging, if not impossible, to create the ideal metric for all classes of networks, it may be relevant to design them for specialized applications. In this article, we introduce a metric on LGT networks, which consist of trees with additional arcs that represent lateral gene transfer events. Our metric is based on edit operations, namely the addition/removal of transfer arcs, and the contraction/expansion of arcs of the base tree, allowing it to connect the space of all LGT networks. We show that it is linear-time computable if the order of transfers along a branch is unconstrained but NP-hard otherwise, in which case we provide a fixed-parameter tractable (FPT) algorithm in the level. We implemented our algorithms and demonstrate their applicability on three numerical experiments.
Full online version: https://www.biorxiv.org/content/10.1101/2025.11.20.689557

1 Introduction

Phylogenetic networks are often seen as more accurate representations of evolution than trees, especially for species that undergo hybridization or lateral gene transfer, and one can even argue that the "tree of life" hypothesis should be updated to incorporate network-like structures [1]. Several approaches now aim to reconstruct networks, for example NeighborNet [2], PhyloNet [3], SNaQ [4], PhyNest [5], T-Rex [6], and more. Despite the importance of predicting such networks, there is still no clearly established method to evaluate the quality of the reconstructions. For phylogenetic *trees*, the Robinson-Foulds (RF) metric and extensions [7,8] are widely used, and despite some shortcomings are accepted as a good measure of similarity. Therefore, when a new tree prediction tool is developed, it can be evaluated against a gold standard or simulated dataset using RF, providing insights on its strengths and weaknesses.

In the case of networks, this is not so easy. Several network measures have been attempted in the last two decades, but all of them have drawbacks. This includes structural measures that compare small components of the networks, such as softwired or hardwired clusters [9,10,11], rooted triples and trinets [12,13,14], displayed trees [15], and others [9,16], as well as metrics based on counting paths, such as the μ-distance [17,18], which was recently generalized to semi-directed networks [19]. All of these metrics suffer from the fact that, in all generality, they may say two networks are the same when they are actually different (see e.g. [9,16]). Another family of measures are operational metrics, which are based on a number of operations needed to transform one network into the other (e.g., SPR [20,21,22], NNI [23,24], contractions [25], or cherry-picking [26,27] operations). However, these tend to be very hard to compute and, to our knowledge, not yet used frequently in practice. Overall, it appears extremely difficult to design a "one-size-fits-all" metric for networks, and it may be more relevant to design metrics that are tailored for specific applications.

Towards this goal, our aim in this paper is to compare networks in which reticulations represent lateral gene transfer (LGT) events. Specialized approaches can be used to reconstruct such networks, since the problem can be formulated in terms of starting with a species *tree*, and inserting arcs between branches to represent genetic exchanges between co-existing species. This results in what is called an *LGT network*, which consisting of the known original *base tree* and extra *transfer arcs* [28]. There are many LGT network reconstruction approaches: they can be built using tri-nets [29]; using so-called *Duplication-Transfer-Loss (DTL) reconciliation* [30,31]; using orthology and xenology data [32]; or using the presence/absence of character traits to predict clades that exchanged characters [33,34]. Moreover, LGT networks are a specialized form of *tree-based networks*, which also consist of base trees with additional arcs, but in which the base tree is unknown [35]. As tree-based networks form a popular class that generalizes many others [36], there is also a need to develop metrics for them.

Even in the case of LGT networks, there appears to be no ideal comparison metric. As a result, LGT network predictors have mostly been evaluated in an ad hoc manner in previous work. For example in [34], LGT networks were reconstructed to predict bacterial transfers, notably interphylum transfers between Proteobacteria and Actinobacteria, and the results were compared to published studies on this set of species. Unfortunately, this methodology is qualitative and does not easily generalize to other datasets. As already mentioned, another way of reconstructing LGT networks is by computing *DTL reconciliations* between species and gene trees [31,37]. These infer events on gene trees and the predicted gene transfers can be used to add LGT arcs on the species tree. These approaches typically measure the accuracy of the events on the gene trees, but not the accuracy of the underlying species LGT network — which is more relevant in some contexts such as transfer highway identification.

Our results. In this work, we address the aforementioned gaps by proposing a novel metric to compare LGT networks. We exploit the distinction between the base tree and the transfer arcs and compare these two components in an

independent fashion. The "base tree components" of the networks are compared using the established Robinson-Foulds metric, which reduces the problem to comparing only the "transfer components". An intuitive way of comparing the sets of transfer arcs in two networks would be to count the number of "same" transfers, but as we show this can be ambiguous. More specifically, we obtain the following results: We introduce a new dissimilarity measure d_{LGT} and show that it defines a metric space on the set of *all* LGT networks with the same leaf taxa. Our measure can also be extended to tree-based networks.

The metric is based on operations to transform one network into the other, and so it can be used to explore the space of all LGT networks on the same taxa (which is useful for stochastic and hill-climbing reconstruction methods that make local moves on networks, the currently dominant approach [4,5]). We obtain a complexity dichotomy: d_{LGT} can be computed in linear time if the order of transfers along a species branch does not matter, and NP-hard if it does. In the latter case, we show that d_{LGT} is fixed-parameter tractable (FPT) in the *level* (maximum number of reticulations in a biconnected component) of the input networks. We implemented our algorithms and perform three sets of experiments. First, we show on random simulated networks that our implementation is usable in practice for large networks (up to ~ 1800 vertices). We then show its usefulness in comparing LGT network predictions in two proofs-of-concept. In the first one, we compute d_{LGT} values between the outputs of the character-based methods presented in [34] and show quantitatively that transfer highway predictions are highly dependent on the approach and parameters used. In the other application, we look at networks predicted with a reconciliation tool (Ranger-DTL [31]), and show how our metric can be used to tune cost values within the reconciliation.

Due to space constraints, some of the content (proofs, pseudo-code…) has been deferred to the Appendix, or is only present in the full online version (https://www.biorxiv.org/content/10.1101/2025.11.20.689557).

2 LGT networks and Tree-based networks

We first introduce all mathematical notions required to define our metric. The terminology is based on the definition of LGT networks presented in [28,38]. In a directed graph, the in-neighbors of a vertex are called its *parents* and its out-neighbors are *children*. A *phylogenetic network*, or just *network* for short, is a *directed acyclic graph* that has a single vertex of indegree 0, called its *root*. We also require that each vertex of outdegree 0, called a *leaf*, has indegree 1. Note that we allow vertices of indegree 1 and outdegree 1, which we call *subdivision vertices*. Given a phylogenetic network N, we denote its set of vertices by $V(N)$, its set of arcs by $E(N)$, and its set of leaves by $L(N)$. The *contraction* of an arc (u, v) of a network N is an operation that identifies u and v, that is, it: adds the arc (w, u) for each in-neighbor w of v; adds the arc (u, w) for each out-neighbor w of v; removes v and its incident arcs. If v is a vertex with a single parent, contracting v means contracting the only arc entering v.

Two networks N_1, N_2 are *isomorphic*, denoted $N_1 \simeq N_2$, if $L(N_1) = L(N_2)$ and there exists a bijective function $\phi : V(N_1) \to V(N_2)$ such that: (1) for each

$\ell \in L(N_1)$, $\phi(\ell) = \ell$; (2) $(u, v) \in E(N_1)$ if and only if $(\phi(u), \phi(v)) \in E(N_2)$. Furthermore, the two networks are *homeomorphic* if they are isomorphic when we ignore subdivision vertices. That is, N_1 and N_2 are homeomorphic if N_1' and N_2' are isomorphic, where N_1' (resp. N_2') is obtained from N_1 (resp. N_2) by contracting each subdivision vertex.

Trees and clusters. A *tree* T is a special type of network in which the underlying undirected graph has no cycle. For a vertex v of T, we write $L_T(v)$ for the set of leaves descending from v, which we call the *cluster* of v in T. Note that two trees are isomorphic if and only if they have the same clusters, and each cluster occurs with the same multiplicity in both trees (recall that we allow subdivision vertices). Two trees are homeomorphic if and only if their sets of clusters are the same, without considering multiplicities.

LGT networks. An *LGT network* is a pair $\mathcal{N} = (N, (E_p, E_t))$ where $N = (V, E)$ is a network, (E_p, E_t) is a pair of subsets of E such that $E_p \cup E_t = E$ and $E_p \cap E_t = \emptyset$, with E_p the *principal arc set* and E_t the *transfer arc set*; and the subgraph $T_0(\mathcal{N}) = (V, E_p)$ consisting only of principal arcs is a tree with the same set of leaves as N. The tree $T_0(\mathcal{N})$ is called the *base tree* of $\mathcal{N}$ (note that $T_0(\mathcal{N})$ may contain subdivision vertices, see Figure 1). A vertex with at least two children in $T_0(\mathcal{N})$ is called a *tree vertex* (of either N or $T_0(\mathcal{N})$). Moreover, a vertex of N (or $T_0(\mathcal{N})$) that is incident to a transfer arc in N is called an *attachment point* (named as such since that endpoint that exists to attach a transfer). Since one goal of LGT networks is to distinguish between vertical and horizontal evolution, we assume the following.

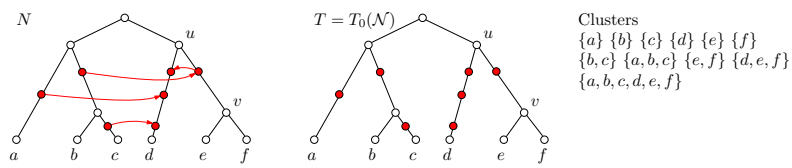

Fig. 1. Left is an LGT network $\mathcal{N} = N|T$. The base tree $T = T_0(\mathcal{N})$ is shown in the middle. In N, black arcs are those of the base tree T (their direction is omitted, they point downwards) and red arcs are transfer arcs. White vertices are tree vertices and filled red vertices are attachment points. The set of clusters of T is shown on the right. Multiplicities are ignored, but for instance $\{d\}$ occurs four times and $\{e, f\}$ two times.

Assumption 1. *In an LGT network $\mathcal{N}$, all attachment points are subdivision vertices in $T_0(\mathcal{N})$.*

The example in Figure 1 satisfies this assumption. Note, this is not required in the original definition of LGT networks. It always holds in binary LGT networks, but our networks are not necessarily binary.

It will often be more useful to refer to the base tree directly instead of using the set of arcs E_p and the tree T_0. We therefore use the notation $N|T$ to denote the unique LGT network $\mathcal{N}$ whose network is N and whose base tree $T_0(\mathcal{N})$ is T,

and stick with this alternate notation for the rest of the paper. We sometimes use the calligraphic $\mathcal{N}$ to denote an LGT network without specifying N and T. Note that the $N|T$ notation lets us deduce the arc partition of $\mathcal{N}$ with $E_p = E(T)$ and $E_t = E(N) \setminus E(T)$. Using this notation, given an LGT network $N|T$, we write $v \preceq_T u$ to indicate that v is a descendant of u in T.

A *tree pair* in $N|T$ is an ordered pair of vertices from $V(N)$, denoted $[u, v]$, such that u, v are tree vertices, $v \prec_T u$, and the directed path from u to v in T contains only attachment points, except u and v (we use brackets to distinguish tree pairs from arcs of T). Intuitively, a tree pair is just a branch linking tree vertices between which transfers occurred. Any attachment point on the $u - v$ path is said to be *on* $[u, v]$. In Figure 1, $[u, v]$ forms a tree pair, as well as $[u, d]$. Finally, we need a notion of identical LGT networks, since isomorphims may not preserve principal and transfer arcs. We say that two LGT networks $\mathcal{N}_1 = N_1|T_1, \mathcal{N}_2 = N_2|T_2$ are *LGT-isomorphic*, denoted $\mathcal{N}_1 \simeq_{LGT} \mathcal{N}_2$, if there exists an isomorphism ϕ between N_1 and N_2 such that ϕ is also an isomorphism between T_1 and T_2.

Tree-based networks. A network $N = (V, E)$ is a *tree-based network* if there *exists* a pair (E_p, E_s) such that $(N, (E_p, E_s))$ is an LGT network with the same leaves as N. In this case, (V, E_p) is called a *base tree* of N. This is equivalent to asking for a subset of arcs E_p such that (V, E_p) is a tree with the same set of leaves as N [35]. The main difference is that in an LGT network, the base tree is given, whereas a tree-based network may contain many base trees. The set of all base trees of a tree-based network N will be denoted $B(N)$. Thus, the set of all LGT networks contained in a tree-based network N is $\{N|T : T \in B(N)\}$.

Time consistent LGT networks. We say that an LGT network $N|T$ is *time-consistent* if their exists a labeling function $\lambda : V(N) \to \mathbb{N}^+$ such that: (1) for any transfer arc (u, v), $\lambda(u) = \lambda(v)$; and (2) for any other arc (u, v), $\lambda(u) < \lambda(v)$. In other words, λ represents time that increases from past to present. The arcs of the base tree go in strictly increasing times, as they represent vertical evolution, whereas arcs representing horizontal evolution require co-existence and thus have the same time. The LGT network in Figure 1 is easily seen to be time-consistent.

3 A metric for LGT networks and tree-based extensions

Given two LGT networks on the same leaves, we want to compare both their base trees and their transfer arcs. For the latter, we delete transfers to eliminate disagreements. As for base trees, they can be compared using established tree metrics, and we here use Robinson-Foulds. To avoid situations where non-attachment points become incident to transfer arcs, we only allow contraction of arcs whose two ends are not attachment points. Given an LGT network $N|T$, we thus define the two following operations (see Figure 2):

- the *contraction* of an arc (u, v) of $E(N)$, where u and v are both non-leaf tree vertices of $N|T$. Since u and v are not incident to a transfer, (u, v) is also in $E(T)$. This operation results in another LGT network $N'|T'$, where N' (resp. T') is obtained by contracting (u, v) in N (resp. in T);

– the *deletion* of a transfer arc (u, v) of $E(N) \setminus E(T)$. This operation results in another LGT network $N'|T'$ where N' is obtained by removing arc (u, v) from N, and contracting any resulting subdivision vertices of N (note, only u or v can become subdivision vertices); T' is obtained from T by contracting the same subdivision vertices that were contracted in N.

An LGT network $N'|T'$ is an *LGT reduction* of $N|T$ if $N'|T'$ can be obtained from $N|T$ by applying a sequence of contractions and transfer deletions, as defined above. We define $\delta(N|T, N'|T')$ as the number of operations required to transform $N|T$ into $N'|T'$. Given two LGT networks $\mathcal{N}_1 = N_1|T_1, \mathcal{N}_2 = N_2|T_2$, we say that $N|T$ is a *common LGT reduction* of $\mathcal{N}_1, \mathcal{N}_2$ if $N|T$ is LGT-isomorphic to an LGT reduction of both $\mathcal{N}_1$ and $\mathcal{N}_2$. The LGT distance is:

$$d_{LGT}(N_1|T_1, N_2|T_2) = \min_{N|T}(\delta(N_1|T_1, N|T) + \delta(N_2|T_2, N|T))$$

where the minimization is over all common LGT reductions $N|T$. When such an $N|T$ minimizes the above expression, it is called a *maximum common LGT reduction*. See Figure 2.

Remark on exploring spaces of LGT networks. A common LGT reduction always exists if the leaf sets are the same, since on both networks we can delete every transfer arc, resulting in a tree, and then contract all remaining arcs to obtain a star tree. By defining the reverse of contractions and transfer deletions, one could therefore transform any LGT network into another (as long as leaves are the same). Thus our operations and their reverse can be used to traverse the space of all LGT networks, as claimed in the introduction.

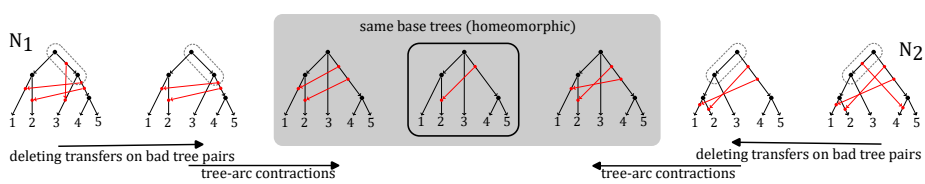

Fig. 2. Illustration of the different steps of an edition pathway between two networks $\mathcal{N}_1, \mathcal{N}_2$ (leftmost and rightmost networks). We want to contract "bad" tree pairs (encircled in $\mathcal{N}_1$ and $\mathcal{N}_2$, which respectively correspond to bad clusters $\{3, 4, 5\}$ and $\{1, 2, 3\}$). Because tree-arc contractions forbid attachment points, we must first delete the transfer arcs on those tree-pairs. Once bad tree pairs are removed, the base trees are homeomorphic (grey area), and one must then compute d_{TR}. As a side note, here $\mathcal{N}_1$ is not time-consistent because of the vertical transfer arc.

Extending to tree-based networks. Note, d_{LGT} translates naturally to tree-based networks, by asking for the base trees that minimize d_{LGT}. That is, if N_1 and N_2 are tree-based networks, the distance is:

$$d_{TB}(N_1, N_2) = \min_{T_1 \in B(N_1), T_2 \in B(N_2)} d_{LGT}(N_1|T_1, N_2|T_2).$$

3.1 A special case: the transfer reduction distance

A special case of d_{LGT} occurs when the base trees of the given LGT networks are the same (i.e., homeomorphic), and all disagreements are just between transfer arcs. An LGT network $N'|T'$ is a *transfer reduction* of $N|T$ if it is obtained from $N|T$ by applying a sequence of transfer deletions only. Observe that the number of transfer deletions needed to transform $N|T$ into $N'|T'$ is $|E(N) \setminus E(T)| - |E(N') \setminus E(T')|$. We say that $N|T$ is a *common transfer reduction* of two LGT networks $N_1|T_1$ and $N_2|T_2$ if $N|T$ is LGT-isomorphic to a transfer reduction of both $N_1|T_1$ and $N_2|T_2$ We can define the *transfer reduction* distance as

$$d_{TR}(N_1|T_1, N_2|T_2) = \min_{N|T}(|E(N_1) \setminus E(T_1)| + |E(N_2) \setminus E(T_2)| - 2|E(N) \setminus E(T)|)$$

where the minimization is over all common transfer reductions $N|T$. An $N|T$ that minimizes this expression is called a *maximum common transfer reduction*. Note that a common transfer reduction does not always exist, and in fact it exists if and only if T_1 and T_2 are homeomorphic.

3.2 Useful properties of d_{LGT}

We show that in order to compute d_{LGT}, it suffices to compare the base tree and the transfer arcs separately. We first define a weighted RF (wRF) distance for the tree part. Let $\mathcal{N}_1 = N_1|T_1$ and $\mathcal{N}_2 = N_2|T_2$ be two LGT networks. A vertex v of N_1 is *bad* (w.r.t. $\mathcal{N}_1$ and $\mathcal{N}_2$) if v is a tree vertex and the cluster $L_{T_1}(v)$ is not a cluster of T_2. Likewise, a vertex v of N_2 is bad if it is a tree vertex and $L_{T_2}(v)$ is not a cluster of T_1. Now let v be a non-root tree vertex of either network, and let u be the unique tree vertex such that $[u, v]$ is a tree-pair (recall, tree-pairs have only attachment points between them). Assign the weight $w(v)$ as the number of transfer arcs that have an endpoint on $[u, v]$, plus one. Then the wRF distance is defined as the sum of weights of bad vertices, i.e.,

$$wRF(N_1|T_1, N_2|T_2) = \sum_{v \in V(N_1): v \text{ is bad}} w(v) + \sum_{v \in V(N_2): v \text{ is bad}} w(v).$$

In Figure 2, each network has a bad vertex of weight two, so $wRF(\mathcal{N}_1, \mathcal{N}_2) = 4$. For the transfer component, let (u, v) be a transfer arc of $N_1|T_1$. Let $[u_1, u_2]$ be the unique tree pair that u is on, and let $[v_1, v_2]$ be the unique tree pair that v is on. We say that (u, v) is a *bad transfer* if one of u_2 or v_2 is a bad vertex. We also say that (u, v) is a *doubly bad transfer* if both u_2, v_2 are bad. Apply the analogous definitions to the transfers of $N_2|T_2$. In Figure 2, each network has one bad transfer, but no doubly bad ones.

Theorem 1. *Given two LGT networks $\mathcal{N}_1 = N_1|T_1, \mathcal{N}_2 = N_2|T_2$ on the same leafsets,*

$$d_{LGT}(\mathcal{N}_1, \mathcal{N}_2) = wRF(\mathcal{N}_1, \mathcal{N}_2) + d_{TR}(N_1^*|T_1^*, N_2^*|T_2^*) - D(\mathcal{N}_1, \mathcal{N}_2), \text{ where}$$

- *$N_1^*|T_1^*$ (resp. $N_2^*|T_2^*$) is the LGT network obtained from $N_1|T_1$ (resp. $N_2|T_2$) by applying a transfer deletion on each of its bad transfers (including doubly bad ones), and then contracting every bad vertex.*

– $D(\mathcal{N}_1, \mathcal{N}_2)$ *is the number of doubly bad transfer arcs that are in* $\mathcal{N}_1$ *or in* $\mathcal{N}_2$.

In essence, wRF counts the number of bad transfers and tree-arc contractions that must be performed, and d_{TR} adds the transfer deletions needed after (wRF double-counts doubly bad transfers so $D(\mathcal{N}_1, \mathcal{N}_2)$ is subtracted). Day's algorithm can be used to compute wRF in linear time [39]. The core of the problem is therefore to compute d_{TR}. We conclude this section by showing that d_{LGT} is a true mathematical metric. Recall that this requires showing that: (1) $d_{LGT}(\mathcal{N}_1, \mathcal{N}_2) = 0$ if and only if the two networks are LGT-isomorphic; (2) the distance is symmetric; and (3) it satisfies the triangle inequality.

Theorem 2. *The* d_{LGT} *distance satisfies all conditions of a metric.*

4 The complexity of computing d_{TR} (and d_{LGT})

We show that there is a dichotomy in the complexity of computing d_{TR}, depending on whether we care about the order of transfers along a tree pair. That is, suppose we have an LGT network $N|T$ and that there is a tree pair $[u, v]$ with more than one attachment points on it. This assumes an ordering of the transfers that occurred between u and v. However, in many cases, this order is unknown and very difficult to predict accurately, and it may be sufficient to contract all these attachment points into one due to uncertainty. This leads to LGT networks with at most one attachment point per tree pair. This matters, because we can compute d_{TR} in linear time under this assumption, and otherwise computing d_{TR} is NP-hard. This is true even if there are at most 3 attachment points on tree pairs, leaving the case of 2 attachment points open. Unless stated otherwise, we now deal with pairs of LGT networks whose base trees are homeomorphic. To simplify notation, we shall assume that two networks have the same set of tree vertices (instead of relying on an homeomorphism). Thus for two LGT networks $N_1|T_1, N_2|T_2$, we may assume that $[u, v]$ is a tree pair of T_1 if and only if $[u, v]$ is a tree pair of T_2 (though the number of attachment points may vary).

4.1 One attachment point per tree pair

Let $N|T$ be an LGT network, possibly non-binary. We define the set of tree pairs that share the two ends of a transfer, as follows:

$$tr(N|T) = \{([u_1, u_2], [v_1, v_2]) : \exists \text{ transfer } (u, v) \text{ s.t. } u \text{ is on } [u_1, u_2],\ v \text{ is on } [v_1, v_2]\}.$$

In words, $([u_1, u_2], [v_1, v_2]) \in tr(N|T)$ means that some transfer has its tail between u_1 and u_2, and its head between v_1 and v_2. Note, $tr(N|T)$ contains ordered pairs (of tree pairs), and $([v_1, v_2], [u_1, u_2])$ is different. Since we assume that tree vertices of $\mathcal{N}_1$ and $\mathcal{N}_2$ are the same, we can compare $tr(\mathcal{N}_1)$ and $tr(\mathcal{N}_2)$ directly. When there is at most one attachment point per tree pair, it suffices to delete transfers whose corresponding pair in the tr set is not in the other. We can thus use Day's algorithm for the wRF part [39], and compute the symmetric difference for the transfer part. Recall that $A \triangle B = (A \setminus B) \cup (B \setminus A)$.

Theorem 3. *Let $\mathcal{N}_1 = N_1|T_1$, $\mathcal{N}_2 = N_2|T_2$ be two LGT networks such that T_1, T_2 are homeomorphic. Suppose that for any tree pair $[u, v]$ of either network, at most one attachment point is on $[u, v]$. Then:*

$$d_{TR}(\mathcal{N}_1, \mathcal{N}_2) = |tr(\mathcal{N}_1) \triangle tr(\mathcal{N}_2)|.$$

Consequently, $d_{LGT}(\mathcal{N}_1, \mathcal{N}_2)$ can be computed in time $O(m_1 + m_2)$, where $m_1 = |E(N_1)|$, $m_2 = |E(N_2)|$.

Beyond one attachment point, the symmetric difference $tr(\mathcal{N}_1) \triangle tr(\mathcal{N}_2)$ may fail because the order may disagree. Figure 3 illustrates this phenomenon, which is exploited in the NP-hardness proof that follows.

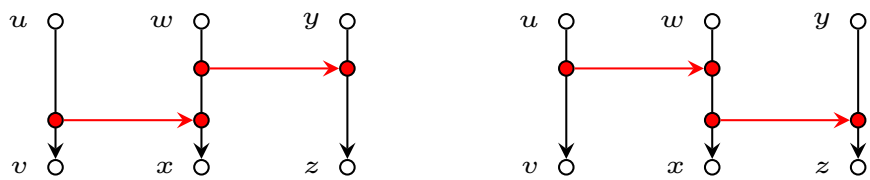

Fig. 3. With more than one attachment point per tree pair, one may have $tr(\mathcal{N}_1) = tr(\mathcal{N}_2)$ even though the networks are different. We have $tr(\mathcal{N}_1) = tr(\mathcal{N}_2) = \{([u, v], [w, x]), ([w, x], [y, z])\}$, and yet the two LGT networks cannot be isomorphic.

4.2 Three attachment points, when the order of transfers matters

We show that computing d_{LGT} is NP-hard, even if the input is restricted to time-consistent networks. The hardness arises when many transfers are in reverse order as in Figure 3. Such pairs of transfers are incompatible and we can keep at most one in each network. When many such incompatibilities arise, we have to find a maximum number of pairwise-compatible transfers, a difficult problem. Inspired by [40], our reduction is from 3-SAT, so that maximally compatible transfers correspond to satisfying a formula.

The reduction (see Figure 4). In 3-SAT, we receive a boolean formula ϕ over variables $x_1, \ldots, x_n$, with ϕ a conjunction of clauses $c_1, \ldots, c_m$, each containing three literals (a literal is a variable x or its negation $\overline{x}$). We must decide if there is a variable assignment that makes every clause *true*. From ϕ, we construct time-consistent LGT networks $N_1|T_1, N_2|T_2$ and an integer k, such that ϕ is satisfiable $\Leftrightarrow$ $d_{TR}(N_1|T_1, N_2|T_2) \leq k$.

For a variable x, we write $d(x)$ for the number of clauses in which x appears. We denote the clauses that contain x as $c_{x,0}, c_{x,1}, \ldots, c_{x,d(x)-1}$, ordered arbitrarily. To construct $N_1|T_1$ and $N_2|T_2$, the any base tree will do, so we let T be any tree with $\sum_{i=1}^{n} 2d(x_i) + m$ leaves. Denote by E the set of arcs of T whose head is a leaf. To obtain T_1 and T_2, we start from T and add attachment points only to arcs in E (this is for time-consistency). The arcs in E will therefore become tree pairs. Given a transfer arc $t = (u, v)$, we denote $t.\text{tail} = u$ and $t.\text{head} = v$.

Variable gadget. We describe variable gadgets in $N_1|T_1$. The construction of the base tree starts from T. For each variable x, choose $2d(x)$ arcs in E

that we denote $e_x[0], e'_x[0], e_x[1], e'_x[1], \ldots, e_x[d(x)-1], e'_x[d(x)-1]$ (note, each variable has its own distinct arcs). Subdivide each of these arcs twice to create two attachment points per arc. For $i \in [0; d(x)-1]$, we introduce a transfer arc $t_x[i]$ from the highest attachment point on $e_x[i]$ to the highest attachment point on $e'_x[i]$. These represent assigning x to true. Then, for all $i \in [0; d(x)-1]$, add a transfer arc $f_x[i]$ from the lowest attachment point on $e'_x[i]$ to the lowest attachment point on $e_x[i+1]$ (we take $i+1$ modulo $d(x)$). In $N_2|T_2$, we do the same, except that we **reverse** the order of attachment points of $t_x[i]$ and $f_x[i]$. That is, the $f_x[i]$'s take the highest attachment points from $e'_x[i]$ to $e_x[i+1]$, and the $t_x[i]$'s take the lowest from $e_x[i]$ to $e'_x[i]$.

Clause gadget. For each clause c of ϕ, we denote the variables that occur in c as $x_{c,0}, x_{c,1}, x_{c,2}$, with the only constraint that in this ordering, all positive literals appear before all negative literals. Let us now get back to modifying $N_1|T_1$. For each clause c, choose an arc e_c of E that is not used for a variable gadget or for another clause gadget. Add three attachment points $v_c[0], v_c[1], v_c[2]$ on e_c, in this order from top to bottom. For $j \in \{0,1,2\}$, consider the j-th variable $x := x_{c,j}$. Let i be the rank of c among the clauses in which x appears, i.e., $c = c_{x,i}$. If x appears as a positive literal in c, we add an attachment point $a_{c,x}$ above $t_x[i]$.head on $e'_x[i]$. If x appears as a negative literal in c, $a_{c,x}$ is added below $f_x[i]$.tail on $e'_x[i]$. We add a transfer arc from $a_{c,x}$ to $v_c[j]$. In $N_2|T_2$, add attachment points $v_c[2], v_c[1], v_c[0]$ on e_c (so, in reverse order). For $j \in \{0,1,2\}$ and variable $x := x_{c,j}$, let i be such that $c = c_{x,i}$. Add attachment point $a_{c,x}$ on $e'_x[i]$ above $t_x[i]$.head and below $f_x[i]$.tail (whether x is positive or not). Then add a transfer arc from $a_{c,x}$ to $v_c[j]$.

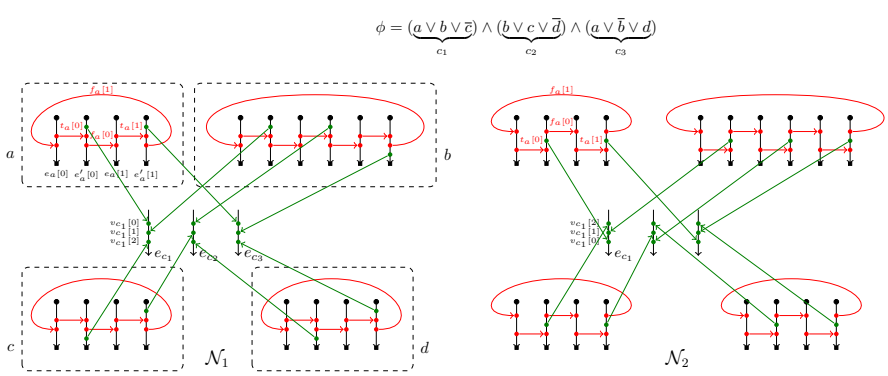

Fig. 4. Illustration of the reduction used in the proof of Theorem 4 (NP-completeness of d_{LGT}).

We denote $N = \sum_{i=1}^{n} d(x_i)$. We show that ϕ is satisfiable if and only if $d_{TR}(N_1|T_1, N_2|T_2) \leq 2N + 4m$.

Theorem 4. *Computing d_{LGT} is NP-complete, even on binary time-consistent networks with at most 3 transfers per attachment point.*

Proof (sketch). Time-consistency can be established by providing an explicit time-labeling, see Appendix. For the equivalence with the satisfiability problem, first assume that ϕ is satisfied by some assignment. If variable x is assigned true, then we delete all $f_x[i]$ arcs and keep only the $t_x[i]$ arcs, in both networks. Do the opposite if x is false. This amounts to $2N$ deletions (N per network). Then for every clause c, take a variable x that satisfies it, and on both networks delete every transfer arc that enters e_c, except the arc from $a_{c,x}$ to $v_c[j]$ (where $j \in \{0, 1, 2\}$). Thus for each clause we perform two transfer deletions, which amounts to $2m$ deletions per network. The total is $2N+4m$, as desired. One can see in the Appendix (Figure 9) that doing these deletions on both networks results in the same transfer reduction. In the other direction, assume that $d(N_1|T_1, N_2|T_2) \leq 2N + 4m$. One can argue that any solution must essentially mimic the previous direction. In an x gadget, we cannot keep both a $t_x[i]$ and $f_x[i]$ arc because their orders are reversed in the two networks. The best we can do is keep all $t_x[i]$ or all $f_x[i]$ arcs. The ones we keep correspond to an assignment of x. In a c gadget, we can keep at most one transfer arc going on e_c because the order of attachment points are reversed in the two networks. To perform at most $2N + 4m$ deletions, one such arc must be kept, which corresponds to the variable that can satisfy c.

This reduction can also be adapted to prove the NP-hardness of d_{TB}, the distance on tree-based networks. The idea is to add a gadget to N_1 and N_2 to constraint their base trees, such that transfer arcs are the same.

Theorem 5. *Computing d_{TB} is NP-hard.*

4.3 Parameterized algorithms and pre-processing rules

We first observe that to compute d_{TR} in practice, and thus d_{LGT}, one can design a simple pre-processing rule as follows.

Transfer cleaning rule. Given two networks $\mathcal{N}_1 = N_1|T_1$ and $\mathcal{N}_2 = N_2|T_2$, with T_1, T_2 homeomorphic, if $\mathcal{N}_1$ has a transfer from tree pair $[a, b]$ to tree pair $[c, d]$, but $\mathcal{N}_2$ has no such transfer, then we delete this transfer immediately (and remember to count it in the distance).

The transfer cleaning rule is easily seen to be correct by observing that no edge contraction nor transfer deletion may introduce new transfer arcs between tree pairs of $\mathcal{N}_2$. We note that this pre-processing rule is critical in achieving the practical scalability results presented in the experiments section. After cleaning, we can test all ways of deleting transfers.

Proposition 1. *The distance $d_{LGT}(\mathcal{N}_1, \mathcal{N}_2)$ can be computed in time $O(2^t \cdot m)$, where t (resp. m) is the number of transfer arcs (resp. arcs) in $\mathcal{N}_1$ and $\mathcal{N}_2$.*

On binary networks, we can do better by handling matching blobs in the two networks independently (see appendix and [11] for a definition of the level).

Theorem 6. *Given two binary LGT networks $\mathcal{N}_1, \mathcal{N}_2$ both of level at most ℓ, one can compute $d_{TR}(\mathcal{N}_1, \mathcal{N}_2)$ and thus $d_{LGT}(\mathcal{N}_1, \mathcal{N}_2)$ in time $O(4^\ell \cdot m^2)$, where m is the total number of edges in both networks.*

5 Experiments

All of the experiments use an implementation of the algorithms presented in Section 4.1 and 4.2, publicly-available at https://github.com/bmarchand/transfer_edition_distance. The algorithms were implemented in Rust and are available in Python with PyO3 (https://github.com/PyO3/pyo3).

We remark that for the first experiment, other measures designed to compare phylogenetic networks (reviewed in the introduction [9,10,11,12,13,14,15,16,17]) are not applicable, as the task is specifically to compare LGT networks, in which a distinction between tree arcs and transfer arcs is made. To illustrate the problem, consider the fact that a given tree-based network N may give rise to two different LGT networks $N|T$ and $N|T'$ if T, T' are two different base trees. The metrics mentioned above do not distinguish these situations. While this argument does not apply to the second and third experiment, we note that in both of them, the networks predicted by the methods are tree-based, but not necessarily orchard networks [26] (see Figure 12 in Appendix for more details). To our knowledge, none of the existing implemented metrics [17,3] are capable of distinguishing all tree-based networks.

5.1 Benchmarks on random LGT networks

Overview. We first evaluate the computational scalability of our metric on random LGT networks, using the simulator from [38]. It outputs binary LGT networks, potentially with several transfers along a tree-pair, with their orders specified. Our results indicate that, although d_{LGT} is NP-hard to compute, Algorithm 1 (Appendix) is fast in practice up to very large network sizes. Indeed, Figure 5 shows that under reasonable parameter choices, two output networks of size $\simeq 1800$ may be compared in $\lesssim 0.1$ seconds.

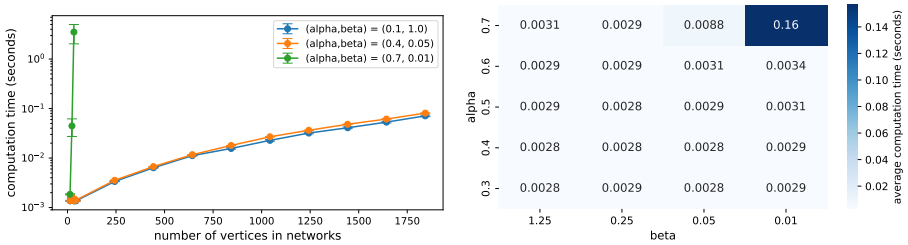

Fig. 5. Run-time benchmark of the computation of our metric on random LGT networks.(left) For several combinations of (α, β), average computation time of the distance between two random networks, as a function of the number of vertices. (right) For a fixed network size (200), table giving the average computation time of the distance between two random networks, for various combinations of (α, β). Note that $\alpha = 0.7$ is higher than values explored in [38] and arguably leads to unrealistic networks.

Simulator. The simulator in [38] takes three parameters: the "number of steps" N (each step adds two vertices to the network, see below), a transfer probability α, and a "level coefficient" β. Starting with a rooted tree with two

leaves, at each step a random choice is made between a *speciation* and a *transfer*, with α the probability of a transfer. If "speciation" is picked, a leaf is chosen uniformly at random and a new leaf is attached on its parent arc. If "transfer" is picked, then two leaves l and l' are chosen and a transfer is added on their parent arcs. Parameter β influences the choice of l and l', with a higher β increasing the chances of choosing two leaves in different biconnected components. In essence, a higher α increases the number of transfers, and β controls the *level* (the maximum number of reticulations in a biconnected component).

Results. Figure 5 shows that our algorithm is fast on a wide range of parameters, except when $\alpha = 0.7$ and $\beta = 0.01$. This can be seen both on Figure 5 (a), where $(\alpha, \beta) = (0.7, 0.01)$ exhibits a stronger exponential scalin, and on Figure 5 (b) for a fixed network size. We note that $\alpha = 0.7$ is beyond the range of values explored in [38], and may be considered unrealistically high (recall that $\alpha = 0.7$ means that, in expectation, 70% of events are transfers, and 30% speciations). Even then, hard instances are only produced if $\beta \leq 0.01$, i.e., when transfers are $100\times$ more likely between two leaves whose parents are in the same biconnected component, producing "blobs" with exceedingly many transfers.

5.2 Comparison of character-based transfer reconstructions

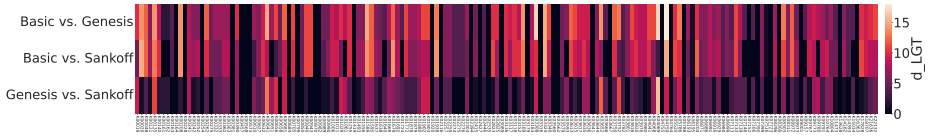

Fig. 6. Matrix of distances between reconstructed phylogenetic networks, using the character-based methods presented in [34], dubbed "Basic", "Sankoff" and "Genesis". Each column of the matrix corresponds to a character. For each character, each method reconstructs a network. We observe that Genesis and Sankoff tend to give network reconstructions that are closer to one another, compared with Basic. In [34], only qualitative comparisons had been reported.

Some models such as *perfect phylogenetic networks* [41] and *perfect transfer networks* [33,42] can predict transfers on a species tree by explaining the presence/absence of given characters at the leaves (here, characters could be genes, functions, traits, ...). We revisit the experiments and data used in one of the latest iterations of this line of work [34] to illustrate a use-case of d_{LGT}. The authors of [34] were only able to compare their predictions qualitatively, and here our metric may be used to compare predictions *quantitatively*.

Overview. Given a species tree and characters at the leaves, [34] considers three different approaches for reconstructing the history of individual characters. For each character, they assume a *single origin* and *losses* and *transfers* as evolutionary events. A first approach is called *Basic labeling*, and infers a character at an internal vertex if and only if it is present in all of its descendants. The assumption is then that maximal subtrees containing a character are

involved in an HGT. Such a labeling has sometimes been used in hypothesis-testing, where biologists want to check whether a gene could have emerged in a de novo fashion, or was rather acquired through HGT (see [43,44]). The two other approaches, *Genesis* and *Sankoff*, are more sophisticated dynamic programming schemes computing ancestral assignments of the characters minimizing the costs of losses and transfers. Details can be found in [34]. For each prediction method and each character, the set of inferred transfers yields an LGT network with the species tree as base tree (plus subdivisions). Note that the inferred transfers along a tree pair are *unordered*, so we may use Theorem 3.

Data. We use the same data as in [34]. It consists of a set of 180 functional characters from the KEGG database (Kyoto Encyclopedia of Genes and Genomes [45]) within a set of 45 bacterial species. We take the species tree connecting these species from the NCBI Taxonomy Browser [46].

Results. In [34], reconstructed networks were combined into sets of "transfer highways", that were then compared regardless of their weights (i.e. regardless of how many transfers use the highway). Having a metric such as d_{LGT} allows for a finer quantitative comparison. On Figure 6, we report computations of d_{LGT} values between network predictions reconstructed with each of the three methods. For each of the 180 KEGG characters mentioned above (x-axis on Figure 6), the reconstructions are compared in a pairwise fashion: Basic vs. Genesis, Basic vs. Sankoff, and Genesis vs. Sankoff (y-axis on Figure 6). We note that the methods "Genesis" and "Sankoff" yield predictions that are much closer to one another, compared to "Basic". While this analysis does not tell which method is more accurate, it indicates that the choice of predictor significantly alters the results. Hence, care must be taken when making biological conclusions from HGT predictions as they may be highly dependent on the method used. Nevertheless, these results are an example of how a metric may be used to "cluster" predictions into similar groups, so that a consensus may be identified.

A full pairwise comparison of the networks predicted by each method for a subset of characters is also given in Appendix, Figure 13.

5.3 Optimizing reconciliation parameters for transfer prediction

Overview. Reconciliation addresses the following problem: given a species tree S and a gene tree G, with a map from $L(G)$ to $L(S)$, find an evolutionary history on G that explains their differences, typically with duplication, loss, and transfer events. This is often done parsimoniously by minimizing the weights of inferred events. A crucial problem in this context is to know what cost to associate to each allowed type of event (see e.g. [48]). We show how metrics may help tune the parameters of reconciliation tools, to optimize for transfer inference.

Experimental Protocol. A set of $N = 100$ histories, each consisting of a species tree with $L = 50$ leaves and a gene tree evolving inside it were simulated with Asymmetree [47], using rates for duplications, losses, and transfers of respectively 0.5, 0.5 and 0.1 (see the documentation of Asymmetree for details). This simulator includes speciations, duplications, losses and transfers events. By adding arcs corresponding to the transfers in the simulation to the species tree, we get a "ground truth" network. We also give each pair of species tree and gene

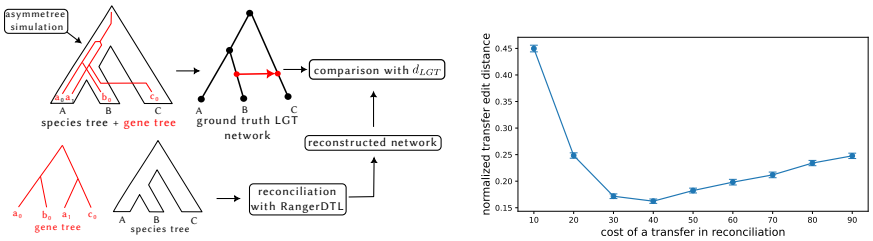

Fig. 7. (left) A set of species tree along with a gene tree evolving within it (with duplications, transfers and losses), are simulated using Asymmetree [47]. Adding the transfers to the species tree yields a ground-truth LGT network. A reconciliation of each gene-tree/species-tree pair is also computed using RangerDTL [31], yielding a reconstructed LGT network. (right) Average normalized d_{LGT} values between the reconstructed and ground truth networks, as a function of the cost associated to transfer events in the reconciliation (the costs of duplications and losses is kept constant). This plot allows the identification of cost values that minimize the average reconstruction distance, offering a way to calibrate reconciliation costs.

tree to RangerDTL [31], a leading tool for reconciliation. Its output is a mapping of each vertex of the gene tree into the species tree, along with a labeling with one of the events (speciation, duplication, transfer). We run RangerDTL $p = 20$ times per pair of species/gene trees, to get a variety of optimal reconciliations (there may be many). Likewise, by adding the transfers onto the species tree, we get a "reconstructed LGT network", see Figure 7 (a). Note that both the ground truth and reconstructed networks are unordered, as described in Section 4.1, i.e. with at most one attachment point per tree pair. In the (rare) case that two transfer events take place between the same tree pair, we treat it as a single transfer. RangerDTL also supports cost values for losses, duplications and transfers (the default being 1,2 and 3). We fix the costs of duplications and losses in RangerDTL to 10, and use transfer costs ranging from 10 to 100. Figure 7 (b) shows normalized d_{LGT} values between the reconstructed and ground truth networks, as a function of the cost of transfers, averaged over N and p. species as base tree and (2) are unordered. Formally, for the normalization we compute $\frac{d_{LGT}(\mathcal{N}_1,\mathcal{N}_2)}{|tr(\mathcal{N}_1)|+|tr(\mathcal{N}_2)|} = \frac{|tr(\mathcal{N}_1)\Delta tr(\mathcal{N}_2)|}{|tr(\mathcal{N}_1)|+|tr(\mathcal{N}_2)|}$.

Results. Figure 7 (b) displays the normalized distance as a function of the transfer cost chosen in the reconciliation. We observe the presence of an optimal cost of 40, yielding reconstructions closest to the ground truth. Such a setup may be used to tune the parameters of RangerDTL (or other similar tools), prior to applications in cases where the ground truth is not known.

6 Discussion and Conclusion

Future research directions could include methods for computing the *consensus* of more than two networks, for instance through a *median* under our distance. One could also explore other popular choices of edit operations for the species tree, such as *subtree-prune and regraft* (SPR) or *nearest-neighbor interchange* (NNI). It should also be noted that the complexity of d_{TR} when tree pairs have

at most 2 attachment points is open. Another algorithmic question of interest is whether computing our metric is fixed-parameter tractable in d_{LGT}, e.g., with a complexity that is exponential in d_{LGT} only and not the sizes of the networks. To finish, one could also explore generalizations of our metric that would take into account the "distance" between disagreeing transfers, to better distinguish a slightly misplaced transfer from one further away.

References

1. W Ford Doolittle and Eric Bapteste. Pattern pluralism and the tree of life hypothesis. *Proceedings of the National Academy of Sciences*, 104(7):2043–2049, 2007.
2. David Bryant and Vincent Moulton. Neighbor-net: an agglomerative method for the construction of phylogenetic networks. *Molecular biology and evolution*, 21(2):255–265, 2004.
3. Dingqiao Wen, Yun Yu, Jiafan Zhu, and Luay Nakhleh. Inferring phylogenetic networks using phylonet. *Systematic biology*, 67(4):735–740, 2018.
4. Claudia Solís-Lemus, Paul Bastide, and Cécile Ané. Phylonetworks: a package for phylogenetic networks. *Molecular biology and evolution*, 34(12):3292–3298, 2017.
5. Sungsik Kong, David L Swofford, and Laura S Kubatko. Inference of phylogenetic networks from sequence data using composite likelihood. *Systematic Biology*, 74(1):53–69, 2025.
6. Alix Boc, Hervé Philippe, and Vladimir Makarenkov. Inferring and validating horizontal gene transfer events using bipartition dissimilarity. *Systematic biology*, 59(2):195–211, 2010.
7. David F Robinson and Leslie R Foulds. Comparison of phylogenetic trees. *Mathematical biosciences*, 53(1-2):131–147, 1981.
8. Yu Lin, Vaibhav Rajan, and Bernard ME Moret. A metric for phylogenetic trees based on matching. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(4):1014–1022, 2011.
9. Gabriel Cardona, Mercè Llabrés, Francesc Rosselló, and Gabriel Valiente. Metrics for phylogenetic networks i: Generalizations of the robinson-foulds metric. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(1):46–61, 2008.
10. Daniel H Huson, Regula Rupp, Vincent Berry, Philippe Gambette, and Christophe Paul. Computing galled networks from real data. *Bioinformatics*, 25(12):i85–i93, 2009.
11. Daniel H Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.
12. Jesper Jansson, Nguyen Bao Nguyen, and Wing-Kin Sung. Algorithms for combining rooted triplets into a galled phylogenetic network. *SIAM Journal on Computing*, 35(5):1098–1121, 2006.
13. Jesper Jansson and Wing-Kin Sung. Inferring a level-1 phylogenetic network from a dense set of rooted triplets. *Theoretical Computer Science*, 363(1):60–68, 2006.
14. Katharina T Huber and Vincent Moulton. Encoding and constructing 1-nested phylogenetic networks with trinets. *Algorithmica*, 66:714–738, 2013.
15. Daniel H Huson and Celine Scornavacca. A survey of combinatorial methods for phylogenetic networks. *Genome biology and evolution*, 3:23–35, 2011.
16. Gabriel Cardona, Mercè Llabrés, Francesc Rosselló, and Gabriel Valiente. Metrics for phylogenetic networks ii: Nodal and triplets metrics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(3):454–469, 2008.

17. Gabriel Cardona, Joan Carles Pons, Gerard Ribas, and Tomás Martínez Coronado. Comparison of orchard networks using their extended μ-representation. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 21(3):501–507, 2024.
18. Allan Bai, Péter L Erdős, Charles Semple, and Mike Steel. Defining phylogenetic networks using ancestral profiles. *Mathematical Biosciences*, 332:108537, 2021.
19. Michael Maxfield, Jingcheng Xu, and Cécile Ané. A dissimilarity measure for semidirected networks. *IEEE Transactions on Computational Biology and Bioinformatics*, 2025.
20. Benjamin L Allen and Mike Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of combinatorics*, 5(1):1–15, 2001.
21. Magnus Bordewich and Charles Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Annals of combinatorics*, 8(4):409–423, 2005.
22. Magnus Bordewich, Simone Linz, and Charles Semple. Lost in space? generalising subtree prune and regraft to spaces of phylogenetic networks. *Journal of theoretical biology*, 423:1–12, 2017.
23. Katharina T Huber, Simone Linz, Vincent Moulton, and Taoyang Wu. Spaces of phylogenetic networks from generalized nearest-neighbor interchange operations. *Journal of Mathematical Biology*, 72(3):699–725, 2016.
24. Philippe Gambette, Leo Van Iersel, Mark Jones, Manuel Lafond, Fabio Pardi, and Celine Scornavacca. Rearrangement moves on rooted phylogenetic networks. *PLoS computational biology*, 13(8):e1005611, 2017.
25. Bertrand Marchand, Nadia Tahiri, Shohreh Golpaigani Fard, Olivier Tremblay-Savard, and Manuel Lafond. Finding maximum common contractions between phylogenetic networks. *Algorithms for Molecular Biology*, 20(1):18, 2025.
26. Kaari Landry, Aivee Teodocio, Manuel Lafond, and Olivier Tremblay-Savard. Defining phylogenetic network distances using cherry operations. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 20(3):1654–1666, 2022.
27. Kaari Landry and Olivier Tremblay-Savard. Fast calculation of cherry distance on level-1 orchard networks: Optimization, heuristic and implementation. In *RECOMB International Workshop on Comparative Genomics*, pages 157–177. Springer, 2025.
28. Gabriel Cardona, Joan Carles Pons, and Francesc Rosselló. A reconstruction problem for a class of phylogenetic networks with lateral gene transfers. *Algorithms for Molecular Biology*, 10(1):28, 2015.
29. Gabriel Cardona and Joan Carles Pons. Reconstruction of lgt networks from tri-lgt-nets. *Journal of Mathematical Biology*, 75(6):1669–1692, 2017.
30. Edwin Jacox, Cedric Chauve, Gergely J Szöllősi, Yann Ponty, and Celine Scornavacca. eccetera: comprehensive gene tree-species tree reconciliation using parsimony. *Bioinformatics*, 32(13):2056–2058, 2016.
31. Mukul S Bansal, Manolis Kellis, Misagh Kordi, and Soumya Kundu. Ranger-dtl 2.0: rigorous reconstruction of gene-family evolution by duplication, transfer and loss. *Bioinformatics*, 34(18):3214–3216, 2018.
32. 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*, 2022.
33. Alitzel López Sánchez and Manuel Lafond. Predicting horizontal gene transfers with perfect transfer networks. *Algorithms for Molecular Biology*, 19(1):6, 2024.

34. Alitzel López Sánchez, Guillaume E Scholz, Peter F Stadler, and Manuel Lafond. A sankoff-rousseau-like algorithm for minimizing lateral gene transfers and losses on single origin characters. In *RECOMB International Workshop on Comparative Genomics*, pages 69–86. Springer, 2025.
35. Andrew R Francis and Mike Steel. Which phylogenetic networks are merely trees with additional arcs? *Systematic biology*, 64(5):768–777, 2015.
36. Sungsik Kong, Joan Carles Pons, Laura Kubatko, and Kristina Wicke. Classes of explicit phylogenetic networks and their biological and mathematical significance. *Journal of Mathematical Biology*, 84(6):47, 2022.
37. 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.
38. Joan Carles Pons, Celine Scornavacca, and Gabriel Cardona. Generation of level-k k lgt networks. *IEEE/ACM transactions on computational biology and bioinformatics*, 17(1):158–164, 2019.
39. William HE Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of classification*, 2(1):7–28, 1985.
40. Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.
41. Luay Nakhleh, Donald A Ringe, and Tandy Warnow. Perfect phylogenetic networks: A new methodology for reconstructing the evolutionary history of natural languages. *Language*, 81(2):382–420, 2005.
42. Alitzel López Sánchez and Manuel Lafond. Galled perfect transfer networks. In *RECOMB International Workshop on Comparative Genomics*, pages 24–43. Springer, 2024.
43. Longjun Wu and J David Lambert. Clade-specific genes and the evolutionary origin of novelty; new tools in the toolkit. In *Seminars in Cell & Developmental Biology*, volume 145, pages 52–59. Elsevier, 2023.
44. Yen-Wen Wang, Jaqueline Hess, Jason C Slot, and Anne Pringle. De novo gene birth, horizontal gene transfer, and gene duplication as sources of new gene families associated with the origin of symbiosis in amanita. *Genome biology and evolution*, 12(11):2168–2182, 2020.
45. Minoru Kanehisa. The kegg database. In *'In silico'simulation of biological processes: Novartis Foundation Symposium 247*, volume 247, pages 91–103. Wiley Online Library, 2002.
46. Conrad L Schoch, Stacy Ciufo, Mikhail Domrachev, Carol L Hotton, Sivakumar Kannan, Rogneda Khovanskaya, Detlef Leipe, Richard Mcveigh, Kathleen O'Neill, Barbara Robbertse, et al. Ncbi taxonomy: a comprehensive update on curation, resources and tools. *Database*, 2020:baaa062, 2020.
47. David Schaller, Marc Hellmuth, and Peter F Stadler. Asymmetree: A flexible python package for the simulation of complex gene family histories. *Software*, 1(3):276–298, 2022.
48. Laurent Bulteau and Mathias Weller. Parameterized algorithms in bioinformatics: an overview. *Algorithms*, 12(12):256, 2019.
49. Marc Hellmuth, David Schaller, and Peter F Stadler. Clustering systems of phylogenetic networks. *Theory in Biosciences*, 142(4):301–358, 2023.

Appendix: complete proofs

A Proofs for Section 3 (A metric for LGT networks and tree-based extensions)

Theorem 1. *Given two LGT networks $\mathcal{N}_1 = N_1|T_1, \mathcal{N}_2 = N_2|T_2$ on the same leafsets,*

$$d_{LGT}(\mathcal{N}_1, \mathcal{N}_2) = wRF(\mathcal{N}_1, \mathcal{N}_2) + d_{TR}(N_1^*|T_1^*, N_2^*|T_2^*) - D(\mathcal{N}_1, \mathcal{N}_2), \text{ where}$$

- *$N_1^*|T_1^*$ (resp. $N_2^*|T_2^*$) is the LGT network obtained from $N_1|T_1$ (resp. $N_2|T_2$) by applying a transfer deletion on each of its bad transfers (including doubly bad ones), and then contracting every bad vertex.*
- *$D(\mathcal{N}_1, \mathcal{N}_2)$ is the number of doubly bad transfer arcs that are in $\mathcal{N}_1$ or in $\mathcal{N}_2$.*

Proof. First notice that for any LGT network $N|T$, by applying a transfer deletion or base tree arc contraction to create another LGT network $N'|T'$, we cannot create a new clusters in the base tree, i.e., every cluster of T' is also in T. In fact, transfer deletions preserve all clusters, and a base tree arc contraction of (u, v) removes the cluster of v and preserves every other cluster.

Now let v be a bad vertex of $N_1|T_1$, and let u be the unique vertex such that $[u, v]$ is a tree pair. Notice that any LGT reduction of $\mathcal{N}_1$ that contains the cluster $L_{T_1}(v)$ cannot be isomorphic to an LGT reduction of $N_2|T_2$. By our above observations, this means that v needs to be contracted eventually. Before this can be achieved, every transfer arc with an attachment point on $[u, v]$ needs to be deleted. Counting these deletions, plus the contraction of (u, v), amounts to $w(v)$ operations. This is true for every bad vertex, and one can easily see that any minimum sequence of operations can be altered, if necessary, so that it first deletes every bad transfer arc, then contracts every bad vertex with its parent. The same must be done with $N_2|T_2$. The number of operations needed to perform this is

$$\sum_{v \in V(N_1) \cup V(N_2) : v \text{ is bad}} w(v) - D(\mathcal{N}_1, \mathcal{N}_2) = wRF(\mathcal{N}_1, \mathcal{N}_2) - D(\mathcal{N}_1, \mathcal{N}_2)$$

where we subtract $D(\mathcal{N}_1, \mathcal{N}_2)$ because in the summation, every doubly bad transfer arc is counted twice (and those that are bad but not doubly bad are counted once). Once this step is done, we obtain LGT reductions $N_1^*|T_1^*$ and $N_2^*|T_2^*$ of $N_1|T_1$ and $N_2|T_2$, respectively, as described in our statement. Note that T_1^* and T_2^* now have the same clusters. They are thus homeomorphic, and all that remains is to apply transfer deletions to make the LGT networks make isomorphic. By definition, this requires $d_{TR}(N_1^*|T_1^*, N_2^*|T_2^*)$ additional operations. This shows that any minimum sequence must perform at least the number of operations in our statement, and that number can easily be seen to also be an upper bound, since it corresponds to deleting bad transfers, contracting bad vertices, and handling remaining transfers.

Theorem 2. *The d_{LGT} distance satisfies all conditions of a metric.*

Proof. The fact that $d_{LGT}(N_1|T_1, N_2|T_2) = 0 \Leftrightarrow N_1|T_1 \simeq_{LGT} N_2|T_2$ holds by definition, since a common LGT reduction can be reached by 0 operations if and only if the two networks are already LGT-isomorphic. Symmetry is also easily verified.

Let us focus on the triangle inequality. We want to show that for any triple of LGT networks $\mathcal{N}_1 = N_1|T_1, \mathcal{N}_2 = N_2|T_2, \mathcal{N}_3 = N_3|T_3$ on the same leaves, we have

$$d_{LGT}(\mathcal{N}_1, \mathcal{N}_3) \leq d_{LGT}(\mathcal{N}_1, \mathcal{N}_2) + d_{LGT}(\mathcal{N}_2, \mathcal{N}_3).$$

Let $\mathcal{N}_{12} = N_{12}|T_{12}$ be a maximum common LGT reduction of $\mathcal{N}_1, \mathcal{N}_2$ and let $\mathcal{N}_{13} = N_{13}|T_{13}$ be a maximum common LGT reduction of $\mathcal{N}_2, \mathcal{N}_3$. Observe that to reach a common LGT reduction of $\mathcal{N}_1$ and $\mathcal{N}_3$, we can first transform $\mathcal{N}_1$ into $\mathcal{N}_{12}$ and $\mathcal{N}_3$ into $\mathcal{N}_{23}$, and then find a common LGT reduction of the two LGT reductions. Therefore we get:

$$\begin{aligned} d_{LGT}(\mathcal{N}_1, \mathcal{N}_3) &\leq \delta(\mathcal{N}_1, \mathcal{N}_{12}) + \delta(\mathcal{N}_3, \mathcal{N}_{23}) + d_{LGT}(\mathcal{N}_{12}, \mathcal{N}_{23}) \\ &= d_{LGT}(\mathcal{N}_1, \mathcal{N}_2) - \delta(\mathcal{N}_2, \mathcal{N}_{12}) + d_{LGT}(\mathcal{N}_2, \mathcal{N}_3) - \delta(\mathcal{N}_2, \mathcal{N}_{23}) + d_{LGT}(\mathcal{N}_{12}, \mathcal{N}_{23}) \\ &= d_{LGT}(\mathcal{N}_1, \mathcal{N}_2) + d_{LGT}(\mathcal{N}_2, \mathcal{N}_3) + (d_{LGT}(\mathcal{N}_{12}, \mathcal{N}_{23}) - \delta(\mathcal{N}_2, \mathcal{N}_{12}) - \delta(\mathcal{N}_2, \mathcal{N}_{23})) \,. \end{aligned}$$

We next claim that $d_{LGT}(\mathcal{N}_{12}, \mathcal{N}_{23}) \leq \delta(\mathcal{N}_2, \mathcal{N}_{12}) + \delta(\mathcal{N}_2, \mathcal{N}_{23})$, which will prove the triangle inequality. To see this, we view $\mathcal{N}_{12}$ and $\mathcal{N}_{23}$ as obtained from $\mathcal{N}_2$ by applying operations, which lets us assume that $V(N_{12}) \subseteq V(N_2)$ and $V(N_{23}) \subseteq V(N_2)$ (this is without loss of generality, it just lets us avoid introducing an isomorphism in the notation). Also note that we may assume that to go from $\mathcal{N}_2$ to $\mathcal{N}_{12}$, all transfer deletions are done first, followed by all tree arc contractions (this is because if a contraction is followed by a transfer deletion, we can just move the transfer deletion before since contractions do not create nor destroy transfers). The same holds for turning $\mathcal{N}_2$ into $\mathcal{N}_{12}$. Let A_{12} (resp. A_{23}) be the set of transfer arcs deleted from $\mathcal{N}_2$ to $\mathcal{N}_{12}$ (resp. $\mathcal{N}_{23}$). Likewise, let C_{12} (resp. C_{23}) be the set of arcs of the base tree contracted from $\mathcal{N}_2$ to $\mathcal{N}_{12}$ (resp. $\mathcal{N}_{23}$) after all transfer deletions were applied. Then, consider the LGT reduction $\mathcal{N}' = N'|T'$ obtained from $\mathcal{N}_2$ by deleting every transfer arc in $A_{12} \cup A_{23}$, and then contracting every base tree arc in $C_{12} \cup C_{23}$ (note that these arcs have no attachment points since we delete all transfer arcs first).

Notice that $\mathcal{N}'$ is an LGT reduction of $\mathcal{N}_{12}$, since we can start from $\mathcal{N}_{12}$ and delete every arc from A_{23} that it still contains, and then contract the base tree arcs from C_{23} that remain (note, these are well-defined since we assume that $V(N_{12}), V(N_{23}) \subseteq V(N_2)$). Likewise, $\mathcal{N}'$ can be obtained from $\mathcal{N}_{23}$ by deleting the transfer arcs in A_{12} and contracting the tree arcs in C_{12}. Thus, $\mathcal{N}'$ is a common LGT reduction of $\mathcal{N}_{12}$ and $\mathcal{N}_{23}$. The distance $\delta(\mathcal{N}_{12}, \mathcal{N}')$ is at most $\delta(\mathcal{N}_2, \mathcal{N}_{23})$ since we apply at most $|A_{23}| + |C_{23}|$ operations. Likewise, $\delta(\mathcal{N}_{23}, \mathcal{N}')$ is at most $\delta(\mathcal{N}_2, \mathcal{N}_{12})$. We thus get the desired claim, since

$$d(\mathcal{N}_{12}, \mathcal{N}_{23}) \leq \delta(\mathcal{N}_{12}, \mathcal{N}') + \delta(\mathcal{N}_{23}, \mathcal{N}') \leq \delta(\mathcal{N}_2, \mathcal{N}_{12}) + \delta(\mathcal{N}_2, \mathcal{N}_{23}).$$

B Proofs for Section 4 (The complexity of computing d_{TR} (and d_{LGT}))

Theorem 3. *Let $\mathcal{N}_1 = N_1|T_1$, $\mathcal{N}_2 = N_2|T_2$ be two LGT networks such that T_1, T_2 are homeomorphic. Suppose that for any tree pair $[u, v]$ of either network, at most one attachment point is on $[u, v]$. Then:*

$$d_{TR}(\mathcal{N}_1, \mathcal{N}_2) = |tr(\mathcal{N}_1) \triangle tr(\mathcal{N}_2)|.$$

Consequently, $d_{LGT}(\mathcal{N}_1, \mathcal{N}_2)$ can be computed in time $O(m_1 + m_2)$, where $m_1 = |E(N_1)|$, $m_2 = |E(N_2)|$.

Proof. We first prove our identity on $d_{TR}(\mathcal{N}_1, \mathcal{N}_2)$, and then argue the linear time complexity.

Proof that $d_{TR}(\mathcal{N}_1, \mathcal{N}_2) = |tr(\mathcal{N}_1) \triangle tr(\mathcal{N}_2)|$. Consider the LGT network $\mathcal{N}_1' = N_1'|T_1'$ obtained from $\mathcal{N}_1$ by deleting every transfer arc that is not in $\mathcal{N}_2$. Formally, this means we delete every transfer arc (u, v) of $\mathcal{N}_1$ such that u is on $[u_1, u_2]$, v is on $[v_1, v_2]$ and $([u_1, u_2], [v_1, v_2]) \in tr(\mathcal{N}_1) \setminus tr(\mathcal{N}_2)$. Let $\mathcal{N}_2' = N_2'|T_2'$ be obtained from $\mathcal{N}_2$ analogously, by deleting every transfer arc corresponding to an element of $tr(\mathcal{N}_2) \setminus tr(\mathcal{N}_1)$. One can easily see that every such transfer must be deleted to obtain a common transfer reduction, because transfer deletions cannot create new transfers between tree pairs, so $|tr(\mathcal{N}_1) \triangle tr(\mathcal{N}_2)|$ is a lower bound on $d_{LGT}(\mathcal{N}_1, \mathcal{N}_2)$. We argue that it is also an upper bound.

We claim that $\mathcal{N}_1' = N_1'|T_1'$ and $\mathcal{N}_2' = N_2'|T_2'$ are LGT-isomorphic, which proves this upper bound. Let us first prove that T_1' and T_2' are isomorphic. Since T_1 and T_2 were assumed to be homeomorphic, and since transfer deletions can only remove attachment points in those trees, and not tree vertices, we know that T_1' and T_2' are still homeomorphic. Thus $[u, v]$ is a tree pair of T_1' if and only if $[u, v]$ is a tree pair of T_2'. If the two trees are not isomorphic, it would thus mean that there is some tree pair $[u, v]$ of T_1' with one attachment point on it, but $[u, v]$ in T_2' has no attachment point on it (or the roles of T_1' and T_2' could be swapped, with the same arguments). This means that every transfer arc of $\mathcal{N}_1$ with an endpoint on $[u, v]$ is in $tr(\mathcal{N}_1) \setminus tr(\mathcal{N}_2)$, because if $tr(\mathcal{N}_2)$ had the same transfer arc, it would still be in $\mathcal{N}_2'$ and the attachment point on $[u, v]$ would have remained. There is therefore a contradiction, since every transfer arc of $\mathcal{N}_1$ with an attachment point on $[u, v]$ should be deleted, and the attachment point should have been contracted. We therefore deduce that T_1' and T_2' are isomorphic.

Let us assume for simplicity that $V(T_1') = V(T_2')$, and that $(u, v) \in E(T_1') = E(T_2')$ (we say for simplicity here, because the correspondence between vertices should technically rely on an isomorphism, which we omit). It remains to show that (u, v) is a transfer arc of $\mathcal{N}_1'$ if and only if (u, v) is a transfer arc of $\mathcal{N}_2'$. This follows from the fact that there is at most one attachment point between each tree pair. In more detail, if, for transfer arc (u, v) of $\mathcal{N}_1$, u is on $[u_1, u_2]$ and v is on $[v_1, v_2]$, then u must be the single attachment point on $[u_1, u_2]$ and v the single attachment point on $[v_1, v_2]$. Because the pair $([u_1, u_2], [v_1, v_2])$ is in both $tr(\mathcal{N}_1)$ and $tr(\mathcal{N}_2)$, there is also a transfer arc in $\mathcal{N}_2'$ whose tail is on $[u_1, u_2]$ and whose head is on $[v_1, v_2]$. There is only one option for the the tail and head of this transfer, which are u and v, the single attachment points between the two tree

pairs. Thus (u, v) is also in $\mathcal{N}_2'$. One can argue in the same manner that every transfer in $\mathcal{N}_2'$ is in $\mathcal{N}_1'$, and therefore the two networks are LGT-isomorphic. This shows that $d_{TR}(\mathcal{N}_1, \mathcal{N}_2) = |tr(\mathcal{N}_1) \triangle tr(\mathcal{N}_2)|$.

Proof of complexity. Now let us consider the time complexity of computing $d_{LGT}(N_1|T_1, N_2|T_2)$. The reader may refer to the skeleton of the algorithm in Algorithm 1 and Algorithm 2, shown later in the appendix. We assume that all vertices of N_1 and N_2 are uniquely labeled with an integer between 1 and $n_1 + n_2$, where $n_i = |V(N_i)|$ for $i \in \{1, 2\}$. Note that $n_1 + n_2$ is in $O(m_1 + m_2)$. We assume that we have access to both base trees T_1 and T_2, and that the list of transfer arcs of both networks are stored separately from the base trees.

The algorithm performs the following tasks:

1. Find the bad vertices (corresponding to bad clusters) and compute $wRF(\mathcal{N}_1, \mathcal{N}_2)$. To do this, we take the base trees T_1, T_2 and contract their attachment points, which can be done in time $O(n_1 + n_2)$ (since attachment points are subdivision vertices, each contraction just removes two arcs and adds another). On the two resulting trees, we run Day's algorithm, which in addition to computing the standard RF distance also finds the bad vertices in linear time $O(n_1 + n_2)$ [39]. We can use a table with $n_1 + n_2$ bits, one entry for each possible vertex, with a 0 at position i if vertex i is not bad, and a 1 if it is bad. In this way, we can know in constant time whether a vertex is bad when traversing the base trees. We can then easily calculate the weight of bad vertices by just counting the number of transfers on the attachment point above them, and calculate the wRF component in linear time (the counts can be obtained by traversing the lists of transfers and incrementing a counter for their two respective ends).
2. Delete all the bad transfer arcs. For this we traverse the base trees and, whenever a bad vertex u is encountered, for each attachment point v just above u, we delete all transfer arcs that include v from the list of transfers. If we encounter a transfer arc that was already deleted, it is doubly bad and we can increment $D(\mathcal{N}_1, \mathcal{N}_2)$.
   Doing all the necessary deletions and calculating $D(\mathcal{N}_1, \mathcal{N}_2)$ in linear time requires a bit of care. We can proceed as follows. We take the list of bad vertices and sort it. Since vertex indices are at most $n_1 + n_2$, sorting takes linear time with bucket sort, for instance. We denote this list by B. Then, the lists of transfers of $\mathcal{N}_1$ and $\mathcal{N}_2$ store pairs of the form (i, j), where i, j are integers bounded by n_1+n_2. The two lists have at most m_1+m_2 elements. We can thus use radix sort to sort the two lists of transfers in time $O(m_1+m_2+n_1+n_2)$, which is $O(m_1+m_2)$ (using lexicographic order). We can then obtain the list of bad transfers bad_{tail} whose tail is in B, by maintaining an index in B and an index in the transfer list, and retaining each bad transfer whose first coordinate is a the current position in B, and advancing appropriate indices (an example of this strategy is used in Algorithm 4). Similarly, we can obtain the list of bad transfers bad_{head} whose head is in B, by sorting the transfer lists with respect to the second coordinate and repeating the process. Next, we compute the size of the intersection of bad_{tail} and bad_{head}

to obtain $D(\mathcal{N}_1, \mathcal{N}_2)$ (to get this size, sort the two lists and do the two-index trick again). Finally, we can remove the bad transfers from our lists, again by sorting and, in a single pass, deleting every element that occurs in bad_{tail}, then doing a second pass to delete those that are in bad_{head}. Every step described here takes time $O(m_1 + m_2)$. It is then not difficult to detect the now unused attachment points in both base trees and contract them.

3. Contract all bad vertices. If we do a pre-order traversal of the base trees and contract bad vertices when they are encountered, each vertex of either base trees changes parent at most once, so this takes time $O(n_1 + n_2)$.
4. Compute $d_{TR}(\mathcal{N}_1, \mathcal{N}_2) = |tr(\mathcal{N}_1) \triangle tr(\mathcal{N}_2)|$ on the resulting networks. For this, we represent a transfer arc (u, v) as two tree pairs $[u_1, u_2]$ and $[v_1, v_2]$. For each network and each transfer arc (u, v), we just look at the parent and child of u in the base tree to get $[u_1, u_2]$, and do the same on v to get $[v_1, v_2]$ (here, we use the assumption that each tree pair has at most one attachment point). We store $tr(\mathcal{N}_1)$ and $tr(\mathcal{N}_2)$ in two lists, which contain elements of the form $([i, j], [k, l])$. These are quadruples of integers between 1 and $n_1 + n_2$, and again can be sorted in linear time with radix sort. Once $tr(\mathcal{N}_1)$ and $tr(\mathcal{N}_2)$ are sorted, we can maintain an index on each list to compute the size of their intersection in linear time, and thus get the size of the symmetric difference. This is detailed in Algorithm 4.

Since each step takes time $O(m_1 + m_2)$ to calculate, we achieve the desired complexity.

Theorem 4. *Computing d_{LGT} is NP-complete, even on binary time-consistent networks with at most 3 transfers per attachment point.*

Proof. We prove the correctness of the reduction given above.

Time-consistency. We first verify that the networks are time-consistent. We build time maps λ_1 for $N_1|T_1$ and λ_2 to $N_2|T_2$. First note that we only add attachment points to arcs in E, which are arcs of the initial tree T whose heads are leaves. Thus, if we take the subgraph induced by the ancestors of the tails of arcs in E (inclusively) in either $N_1|T_1$ or $N_2|T_2$, we have a tree. It is easy to assign times in λ_1 and λ_2 such that all tails of arcs in E have time less than 0 (their ancestors will have negative times, which we allow for simplicity). It therefore suffices to assign a time above 0 to attachment points in a time-consistent manner.

In $N_1|T_1$, assign in λ_1 a time of 4 to the two ends of every $t_x[i]$ transfer arc, and a time of 5 to the two ends of each $f_x[i]$ transfer arc. Now consider a clause c and recall that on arc e_c, we added three attachment points $v_c[0], v_c[1], v_c[2]$. Those correspond to the variables $x_{c,0}, x_{c,1}, x_{c,2}$, which we recall are ordered so that positive literals appear first. Assign a time to the $v_c[j]$ vertices that correspond to positive literals a time in $\{1, 2, 3\}$, and a time to those for negative vertices a time of 6 or more (by our ordering, this can be done in a time-consistent manner). Finally, for each transfer arc whose head is a $v_c[j]$ vertex, assign the tail the same time as $v_c[j]$.

Let us argue that λ_1 labeling is time-consistent. By construction, the ends of each transfer arc have equal times. Then, for $i \in [0; d(x) - 1]$, there are exactly

2 attachment points on $e_x[i]$, namely $t_x[i]$.tail and $f_x[i-1]$.head, with $t_x[i]$.tail above $f_x[i]$.head. Those have time-labels by λ_1 equal to 4 and 5, respectively, which is time-consistent. On $e_x[i']$ for $i \in [0; d(x)-1]$, there are exactly 3 attachment points: $t_x[i]$.head followed by $f_x[i]$.tail, again assigned respective times 4 and 5. There is a third attachment point $a_{c,x}$, whose time is either in $\{1, 2, 3\}$ or greater than 5, corresponding respectively to a positive occurrence of x in c (in which case $a_{c,x}$ is above $t_x[i]$.head) and a negative occurrence of x in c (in which case $a_{c,x}$ is below $f_x[i]$). In both cases, the ordering of time labels is respected. Our construction of λ_1 shows that for each tree pair corresponding to an arc of E, the times of the attachment points go in increasing order, and it follows that $N_1|T_1$ is time-consistent.

Now let us build time map λ_2 for $N_2|T_2$. For each $i \in [0, d(x)-1]$, set the time of both ends of $f_x[i]$ to 1, and the time of both ends of $t_x[i]$ to 5. Then for each clause c and arc e_c, the attachment points are, in order, $v_c[2], v_c[1], v_c[0]$, which we assign respective times $2, 3, 4$. The tail of each transfer ending at one of these $v_c[j]$ vertices is assigned the same time as $v_c[j]$.

To see that this makes $N_2|T_2$ time-consistent, for $e_x[i]$ with $i \in [0; d(x)-1]$, the attachment points occuring on this arc have times 1 and 5. For $e'_x[i]$, $a_{c,x}$ is below $f_x[i]$.tail and above $t_x[i]$.head, and is assigned one of the times in $\{2, 3, 4\}$. This is between 1 and 5, and so the times along $e'_x[i]$ are in increasing order. This concludes the proof of time-consistency.

We next show that ϕ is satisfiable if and only if a common transfer reduction can be achieved using a total of $2N + 4m$ deletions.

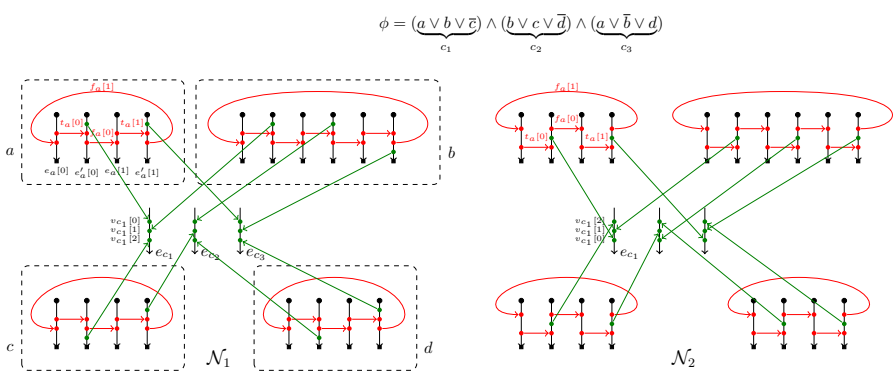

Fig. 8. Example of the reduction (same figure as in the main text).

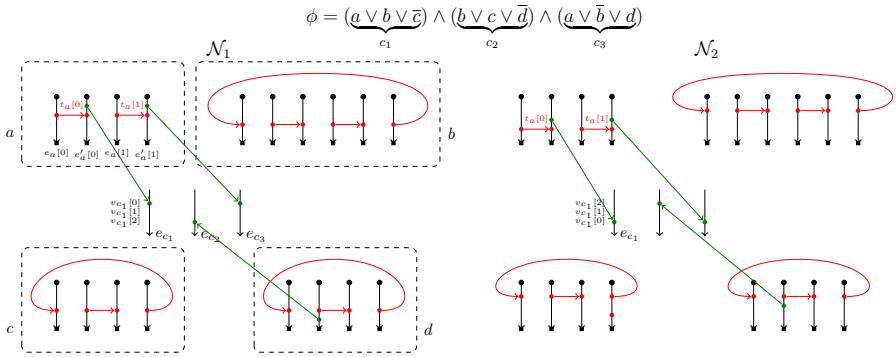

Fig. 9. The solution obtained in the first direction, corresponding the the satisfying assignment that puts $a = true, b = c = d = false$. Here, c_1 and c_3 are satisfied by $a = true$ and c_2 by $d = false$.

ϕ satisfiable $\Rightarrow$ existence of a large common reduction. Consider an assignment of each variable to true or false that satisfies ϕ. We show how to delete transfers from both networks, as illustrated in Figure 9. If a variable x is set to true, we remove the transfers $f_x[i]$ for all $i \in [0; d(x)-1]$ in both networks. If it is set to false, likewise we remove $t_x[i]$ for $i \in [0; d(x)-1]$ in both networks. This amounts to $2\sum_x d(x) = 2N$ deletions. Then, for each clause c of ϕ, we pick a variable x whose assignment satisfies ϕ (as true if x appears positively, as false otherwise) and remove in both networks all transfers arriving at e_c, except the one originating at $a_{c,x}$. For each clause, a total of 4 transfer deletion occurs this way in both networks (we delete two incoming transfers per clause per network). We have thus performed $2N + 4m$ deletions.

We now argue that after these deletions on $N_1|T_1$ and $N_2|T_2$, we have achieved the same LGT network. Note that the two networks are identical above the tails of E, so we must only consider the transfer arcs between the tree pairs corresponding to arcs of E. To start with, for each variable x, either the set $\{t_x[i]\}_{i\in[0;d(x)-1]}$ or $\{f_x[i]\}_{i\in[0;d(x)-1]}$ is kept, and the same choice is made in both networks, following the variable assignments. Therefore, between the tree pairs corresponding to the $e_x[i]$ and $e'_x[i]$ arcs, for all x, there is in the reductions of both networks exactly one transfer between $e_x[i]$ and $e'_x[i]$ for $i \in [0; d(x)-1]$ in the former case, and between $e'_x[i]$ and $e_x[i+1]$ in the latter (addition modulo $d(x)$). Moreover, if x is the variable chosen to satisfy c, and c is the i-th clause in which x appears, there is also $a_{c,x}$ kept on $e'_x[i]$. If x appears positively in c, then $a_{c,x}$ is above $t_x[i]$.head on $e'_x[i]$ in N_1. This is also the case in N_2, regardless of whether x appears positively or negatively in c. Likewise, if x appears negatively in c, then $a_{c,x}$ is below $f_x[i]$.tail in N_1. This is also the case in N_2, again regardless of whether x appears positively or negatively in c. As for the edges e_c for each clause c, exactly 1 attachment point remains on it in each network, and is the end-point of a transfer originating from the same arc $e_x[i]$, with x the variable that was picked to witness that c is satisfied, and with c the i-th clause

in which x appears. Therefore, the reductions applied to N_1 and N_2 have given two networks in which transfers occur between exactly the same set of pairs of edges. In addition, the ordering of the attachment points of these transfers is also the same in both networks. Overall, a common reduction has been reached, with a total of $2N + 4m$ transfer deletions.

Existence of a large common reduction $\Rightarrow$ ϕ satisfiable. Conversely, suppose that a common transfer reduction M has been obtained from $N_1|T_1$ and $N_2|T_2$ with less than $2N + 4m$ transfer deletions. We need to show that ϕ is satisfiable. This direction is a bit more complicated and we introduce a useful tool before proceeding.

Transferring above other transfers. Let $N|T$ be an LGT network. We call a pair of attachment points uv a *transfer pair* if either (u, v) or (v, u) is a transfer arc of N. Note, the uv notation emphasizes that the direction is ignored. Given two tree pairs $[x, y]$ and $[z, t]$ of the base tree T, and $e = [p, q]$ another tree pair of T, we say that $[x, y]$ is *has a transfer above* $[z, t]$ *in* e if there are 2 transfer pairs ab, cd such that (1) a is on $[x, y]$ (2) c is on $[z, t]$ (3) b and d are on e with $d \prec_T b$. We denote this relationship by $[x, y] \Rightarrow_e [z, t]$. Note that we may have both $[x, y] \Rightarrow_e [z, t]$ and $[z, t] \Rightarrow_e [x, y]$ in a network with base tree T. This notion is illustrated on Figure 10.

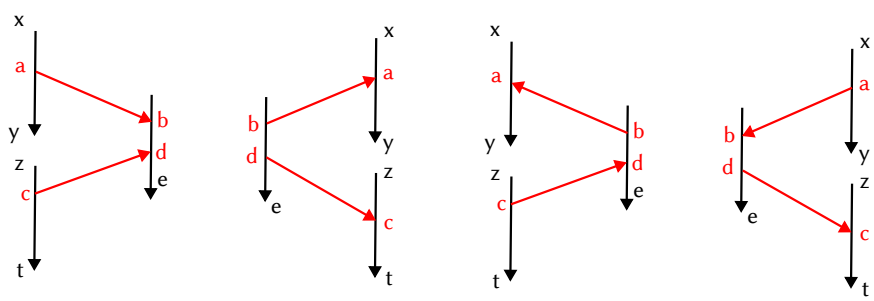

Fig. 10. The different ways the relation $[x, y] \Rightarrow_e [z, t]$ can materialize. $[x, y]$ and $[z, t]$ essentially play the role of labels of the end points of the transfers. With this point of view, the relation $[x, y] \Rightarrow_e [z, t]$ simply states that "label $[x, y]$ is above label $[z, t]$ in e".

When taking reductions of a network, the following property stating that $\Rightarrow_e$ relations cannot be created will be useful:

Property 1. Let $\mathcal{N}' = N'|T'$ be a transfer reduction of $\mathcal{N} = N|T$. If we do not have $[x, y] \Rightarrow_e [z, t]$ in $N|T$, then we also do not have $[x, y] \Rightarrow_e [z, t]$ in $N'|T'$.

Proof. We prove the result if $\mathcal{N}$, $\mathcal{N}'$ differ by 1 transfer deletion, and the result follows by induction. We show the contraposition of our statement, i.e., that given two tree pairs $[x, y]$ and $[z, t]$ such that $[x, y] \Rightarrow_e [z, t]$ for some tree pair e in $\mathcal{N}'$, we also have $[x, y] \Rightarrow_e [z, t]$ in $\mathcal{N}$. Let (u, v) be the transfer that is deleted to go from $\mathcal{N}$ to $\mathcal{N}'$. We also denote s_1t_1 and s_2t_2 transfer pairs in $N'|T$ such

that s_1 is on $[x, y]$, t_1 is on e, s_2 is on $[z, t]$, t_2 is on e and $t_2 \preceq_T t_1$ in $\mathcal{N}'$ (such transfer pairs exist since $[x, y] \Rightarrow_e [z, t]$ in $\mathcal{N}'$). To finish, we denote the transfer arc sets of $\mathcal{N}$ and $\mathcal{N}'$ by E_t and E'_t, respectively. We get that $E'_t = E_t \setminus \{(u, v)\}$, and $\{(s_1, t_1), (s_2, t_2)\} \subseteq E'_t \subseteq E_t$. We also have $t_2 \preceq_{T'} t_1$ in $\mathcal{N}$. Indeed, recall that to go from $\mathcal{N}'$ to $\mathcal{N}$, two arcs of $E(T')$ are possibly subdivided to create u and v, and then an arc is added between u and v. Whether one of the subdivided arcs is on the path from t_1 to t_2 in T' or not, we have $t_1 \preceq_T t_2$ in $\mathcal{N}$ as well. Overall, $[x, y] \Rightarrow_e [z, t]$ in $\mathcal{N}$ as well. □

Let us now go back to our main proof. Property 1 will be used to show that if $[x, y] \Rightarrow_e [z, t]$ holds in $\mathcal{N}_1$ but not in $\mathcal{N}_2$, then we must apply a deletion that makes $[x, y] \Rightarrow_e [z, t]$ become false in $\mathcal{N}_1$ (since the property says that the $\Rightarrow_e$ relation can never be created in $\mathcal{N}_2$).

We first argue that any common transfer reduction of $\mathcal{N}_1$ and $\mathcal{N}_2$ requires *at least* $2N + 4m$ deletions. Let us first note that a transfer arc $t_x[i]$ is deleted in $\mathcal{N}_1$ if and only if that same transfer arc $t_x[i]$ is deleted in $\mathcal{N}_2$, since this is the only transfer arc between the tree pairs of $e_x[i]$ and $e'_x[i]$. The same holds for the $f_x[i]$ arcs.

Next, note that for any variable x and for $i \in [0; d(x)-1]$, $e_x[i] \Rightarrow_{e'_x[i]} e_x[i+1]$ in $\mathcal{N}_1$ (as $t_x[i]$.head is above $f_x[i]$.tail on $e'_x[i]$ in $\mathcal{N}_1$), and not the reverse, whereas $e_x[i+1] \Rightarrow_{e'_x[i]} e_x[i]$ in $\mathcal{N}_2$ ($t_x[i]$.head is below $f_x[i]$.tail on $e'_x[i]$ in $\mathcal{N}_2$), and not the reverse (as usual, additions are to be taken modulo $d(x)$). In both cases, the reverse relation is not true, so if both $t_x[i]$ and $f_x[i]$ are kept in a common transfer reduction $\mathcal{N} = N|T$ of $\mathcal{N}_1$ and $\mathcal{N}_2$, we would have both $e_x[i] \Rightarrow_{e'_x[i]} e_x[i+1]$ and $e_x[i+1] \Rightarrow_{e'_x[i]} e_x[i]$ in $\mathcal{N}$, contradicting Property 1. Likewise, for $i \in [0; d(x)-1]$, if both $f_x[i]$ and $t_x[i+1]$ are kept in a common transfer reduction $\mathcal{N}$, then we have both $e'_x[i] \Rightarrow_{e_x[i+1]} e'_x[i+1]$ and $e'_x[i+1] \Rightarrow_{e_x[i+1]} e'_x[i]$ in $\mathcal{N}$, contradicting Property 1.

In other words, any two transfers that have their attachment point on the same tree pair are incompatible: one of the two must be deleted. This yields at least $\sum_{i=1}^{n} d(x_i) = N$ deletions in the variable gadgets in each network. In addition, we show that if exactly N deletions occur in a variable gadget associated to x, then either $\{t_x[i]\}_{i \in [0; d(x)-1]}$ or $\{f_x[i]\}_{i \in [0; d(x)-1]}$ is entirely kept. If exactly N deletions occur, then for each x and for each $i \in [0; d(x) - 1]$, either $t_x[i]$ or $f_x[i]$ is removed but not both. Likewise, either $f_x[i]$ or $t_x[i + 1]$ must be deleted, but not both. Suppose $t_x[0]$ is kept, then $f_x[d(x) - 1]$ and $f_x[0]$ are removed. This implies $t_x[1]$ is kept. Iterating this argument shows that exactly $\{t_x[i]\}_{i \in [0; d(x)-1]}$ is kept. We can show similarly that if only N deletions occur and $f_x[0]$ is kept in the clause gadget of x, then $\{f_x[i]\}_{i \in [0; d(x)-1]}$ is kept.

As for the clause gadget, consider a clause c, and two variables $x = x_{c,j}$ and $x' = x_{c,j'}$ appearing in c. We suppose w.l.o.g. that x is before x', that is $j < j'$. We denote i and i' the integers such that c is the i-th clause in which x appears, and the i'-th clause in which x' appears (so, $c = c_{x,i} = c_{x',i'}$). Note that we have $e'_x[i] \Rightarrow_{e_c} e'_{x'}[i']$ in $\mathcal{N}_1$ and not the reverse, while the opposite is true in $\mathcal{N}_2$, i.e., $e'_{x'}[i'] \Rightarrow_{e_c} e'_x[i]$ in $\mathcal{N}_2$ because we reversed the order of attachment points on e_c. As this is true for any two variables x, x' that appear in c, it implies through Property 1 that at most 1 transfer entering e_c may be kept in each network. It

implies at least $2m$ deletions in each network. In addition, if only $2m$ deletions occur, exactly one transfer arriving in each e_c must be kept.

Finally, let x be a variable in which we kept all $t_x[i]$ arcs. One can see that if, in the gadget for x, we kept an arc of $\mathcal{N}_1$ from some $a_{c,x}$ vertex to some $v_c[j]$ vertex, then x must occur positively in c. This is because if the occurrence was negative, $a_{c,x}$ would be below $t_x[i]$.head in $\mathcal{N}_1$ but above $t_x[i]$.head in $\mathcal{N}_2$ and we would again violate Property 1. For the same reasons, if all $f_x[i]$ arcs were kept instead, then all remaining arcs starting from $a_{c,x}$ vertices correspond to negative occurrences. Thus in a common transfer reduction, all clause arcs e_c whose incoming transfer arcs are from the same x gadget have the same type of occurrence, all positive or all negative. This means that we can assign $x = true$ if the $t_x[i]$'s were kept and $x = false$ if the $f_x[i]$'s were kept. Since every e_c arc has an incoming transfer from some x gadget that corresponds to satisfying c, this assignment shows that ϕ is satisfiable.

Proof that computing d_{TB} is NP-hard.

Construction of the proof of hardness for d_{TB}. We now modify the reduction described in Section 4.2 to show that d_{TB} is NP-hard. Letting ϕ be an instance of 3-SAT, we let $\mathcal{N}_1 = N_1|T_1$ and $\mathcal{N}_2 = N_2|T_2$ be the LGT networks constructed from ϕ as described previously. Recall that both have a base tree that originated from a tree T that contains a set of arcs E, all having their head vertex being a leaf, to which we added attachment points.

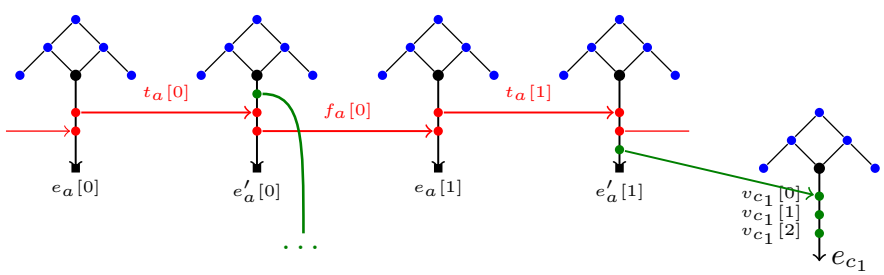

Fig. 11. Modification of the gadgets of $N_1|T_1$ for a variable a and a clause c_1. For each tail u of an arc in E with parent p_u (p_u is not shown): we remove the arc (p_u, u), we add the subgraph consisting of the five blue vertices as shown, then add the root of that subgraph as a child of p_u. In each such subgraph, the two dangling vertices are newly introduced leaves.

Let us obtain a tree-based network N_1' from N_1. Let (u, v) be an arc in E, and note that $[u, v]$ is a tree-pair of $N_1|T_1$. Now make the transformation to N_1 that is illustrated in Figure 11. That is, denoting by p_u the parent of u in N_1 (and T_1), the arc (p_u, u) is replaced with a subgraph with 5 vertices where u is now a reticulation (in the figure, the possible u vertices are the larger black circles — p_u is not shown, but instead of having u as a child it would have the root of the

inserted subnetwork in blue as a child). Denote by N_1' the network obtained from N_1 after applying this transformation to every (u, v) in E. Likewise, denote by N_2' the network obtained from N_2 by applying the same transformations. It is not difficult to show that N_1' and N_2' are tree-based networks. Moreover, the point of this transformation is that the set of transfer arcs of our original reduction are enforced.

Lemma 1. *Let (x, y) be a transfer arc of $N_1|T_1$. Then for any base tree T_1' in $B(N_1')$, the arc (x, y) is a transfer arc of $N_1'|T_1'$. Likewise, if (x, y) is a transfer arc of $N_2|T_2$, then for any base tree T_2' in $B(N_2')$, the arc (x, y) is a transfer arc of $N_2'|T_2'$.*

Proof. We prove the result for $N_1|T_1$ first. Let T_1' be a base tree of N_1'. Let $e_c \in E$ be an arc of the initial tree corresponding to a clause c, and let $e_c = (u, v)$. Note that $[u, v]$ is a tree pair of $N_1|T_1$ on which we added attachment points $v_j[0], v_j[1], v_j[2]$. Because in N_1', u is a reticulation, the arc $(u, v_j[0])$ must be in the base tree T_1' of N_1' (recall that by definition, the base tree must have the same leaves as N_1', and so every non-leaf of N_1' needs to have at least one child in the base tree). Thus, the other arc entering $v_c[0]$, namely some $(a_{c,x}, v_c[0])$ arc, cannot also be in the base tree and thus it must be a transfer of $N_1'|T_1'$. Following the same logic, $(v_c[0], v_c[1])$ must be in T_1' and the other arc entering $v_c[1]$ is a transfer arc, and the same holds for $v_c[2]$. Thus, the transfer arcs with an end on e_c are the same in $N_1|T_1$ and $N_1'|T_1'$.

We next argue that each $t_x[i]$ arc cannot be in the base tree. Consider the arc $e_x'[i] = (u, v)$ and note that $t_x[i]$.head is on the tree pair corresponding to $e_x'[i]$. By construction, in N_1 either $t_x[i]$.head has u as a parent, or has an attachment point $a_{c,x}$ for a clause c as a parent. In the former case, $(u, t_x[i]$.head$)$ must be in the base tree because u has one child, implying that $t_x[i]$ cannot also be in the base tree. In the latter case, the arc from $a_{c,x}$ to e_c was argued to be a transfer, so $(a_{c,x}, t_x[i]$.head$)$ must be in the base tree since $a_{c,x}$ needs a child, implying in turn that $t_x[i]$ cannot be in the base tree. In all cases, $t_x[i]$ must be a transfer arc. Now consider a $f_x[i]$ arc. In N_1, $f_x[i]$.head has $t_x[i+1]$.tail as a parent. Since we now know that $t_x[i]$ is not in the base tree, then $(t_x[i]$.tail$, f_x[i]$.head$)$ is in the base tree and $f_x[i]$ must be a transfer arc. Thus, all transfer arcs of $N_1|T_1$ are also transfers of $N_1'|T_1'$.

For $N_2|T_2$, we use similar ideas to argue that any base tree of N_2 results in the same transfer arcs. The transfer arcs entering e_c cannot be in any base tree of N_2 for the same reasons as before. Each $f_x[i]$ cannot be in the base tree because each reticulation u on the upper end of $e_x[i+1]$ must have $f_x[i]$.head as its child. Finally, each $t_x[i]$ cannot be in the base tree, because in N_2' the parent of $t_x[i]$.head is some $a_{c,x}$ attachment point, which has its arc going into e_c a transfer and which therefore needs $t_x[i]$.head as its child, preventing $t_x[i]$ from being in the base tree.

Theorem 5. *Computing d_{TB} is NP-hard.*

Proof. We propose a reduction from 3-SAT. Given an instance ϕ of 3-SAT, first construct the networks $\mathcal{N}_1 = N_1|T_1$ and $\mathcal{N}_2 = N_2|T_2$ from Section 4.2, and

then apply the modifications described above to obtain tree-based networks N_1' and N_2', resulting in the d_{TB} instance. We claim that $d_{LGT}(N_1|T_1, N_2|T_2) = d_{TB}(N_1', N_2')$, which is sufficient to prove NP-hardness since it implies ϕ is satisfiable if and only if $d_{TB}(N_1', N_2') \leq 2N + 4m$ as before.

Let us first show that $d_{TB}(N_1', N_2') \geq d_{LGT}(N_1|T_1, N_2|T_2)$. Let T_1', T_2' be base trees of N_1, N_2 such that $d_{LGT}(N_1'|T_1', N_2'|T_2')$ is minimum. Let D_1 be the set of transfer arcs deleted from $N_1'|T_1'$ and let D_2 be those deleted from $N_2'|T_2'$ to obtain LGT networks $\mathcal{N}_1''$ and $\mathcal{N}_2''$ that form a maximum common LGT reduction (note, contractions could have been performed, but we don't even count them). Recall that by Lemma 1, all transfer arcs in $N_1|T_1$ and $N_2|T_2$ are also in $N_1'|T_1'$ and $N_2'|T_2'$, respectively. Consider the LGT networks obtained from $N_1|T_1$ by deleting all of its transfers that are also in D_1, and the LGT network obtained from $N_2|T_2$ by deleting all those that are also in D_2, resulting in new LGT networks $\mathcal{N}_1^*$ and $\mathcal{N}_2^*$. As a result, the subgraph induced by the descendants of tails of elements from E is identical in $\mathcal{N}_1^*$ and in $\mathcal{N}_1''$, and identical in $\mathcal{N}_2^*$ and in $\mathcal{N}_2''$ (recall that N_1', N_2' are obtained by replacing the arcs incoming into the tails of elements from E by a subgraph, so everything "below" tails of E was unaltered, and thus we get the same subgraphs in these parts after making the same deletions). Moreover, since $\mathcal{N}_1''$ and $\mathcal{N}_2''$ are LGT-isomorphic, the subgraphs induced by the descendants of tails of elements from E are identical in both LGT networks, and thus the same holds for $\mathcal{N}_1^*$ and $\mathcal{N}_2^*$. It follows that $\mathcal{N}_1^*$ and $\mathcal{N}_2^*$ are themselves LGT-isomorphic. Since we made at most $|D_1| + |D_2| \leq d_{TB}(N_1', N_2')$ transfer deletions to reach $\mathcal{N}_1^*$ and $\mathcal{N}_2^*$, we get $d_{TB}(N_1', N_2') \geq d_{LGT}(N_1|T_1, N_2|T_2)$.

We show the converse inequality $d_{TB}(N_1', N_2') \leq d_{LGT}(N_1|T_1, N_2|T_2)$. In N_1', we take the base tree T_1' with the same transfer arcs as $\mathcal{N}_1$, and for each reticulation above E we choose the transfer arc arbitrarily. For N_2', we take the base tree T_2' with the same transfer arcs as $\mathcal{N}_2$, and for the reticulations above E we make the same arbitrary choice as in T_1'. It then becomes easy to see that a sequence of transfer deletions that makes $\mathcal{N}_1$ and $\mathcal{N}_2$ LGT-isomorphic can also be used to make $N_1'|T_1'$ and $N_2'|T_2'$ LGT-isomorphic. This yields $d_{TB}(N_1', N_2') \leq d_{LGT}(N_1|T_1, N_2|T_2)$.

Proof that d_{LGT} is FPT in the number of transfers.

We recall our proposition of interest here.

Proposition 1. *The distance $d_{LGT}(\mathcal{N}_1, \mathcal{N}_2)$ can be computed in time $O(2^t \cdot m)$, where t (resp. m) is the number of transfer arcs (resp. arcs) in $\mathcal{N}_1$ and $\mathcal{N}_2$.*

Proof. Using the ideas in the proof of Theorem 3, one may calculate wRF and $D(\mathcal{N}_1, \mathcal{N}_2)$ in linear time, and it remains to state how to calculate $d_{TR}(\mathcal{N}_1, \mathcal{N}_2)$. Suppose that $\mathcal{N}_1$ has t_1 transfer arcs and $\mathcal{N}_2$ has t_2. One may iterate over all 2^{t_1} subsets of transfers of $\mathcal{N}_1$ and all 2^{t_2} sets of transfers of $\mathcal{N}_2$ in time $2^{t_1} \times 2^{t_2} = 2^{t_1+t_2} = 2^t$. For each subset, we can obtain transfer reductions $\mathcal{N}_1', \mathcal{N}_2'$ in linear time (technically, we must copy each network before doing so, which just adds another linear time term to the complexity). Then, one needs to check whether the two reductions are LGT-isomorphic. This may also be done

in linear time as follows. We already know at this point that the two networks are homeomorphic, so we only need to check that the transfer arcs are identical. For that, one may simply store, for each transfer arc (u, v): (1) the tree pairs $[u_1, u_2]$ and $[v_1, v_2]$ that u and v are on; (2) the rank of u on $[u_1, u_2]$ and the rank of v on $[v_1, v_2]$ (i.e., the number of attachment points to traverse on the tree pair to reach the vertex). We then ensure LGT-isomorphic by checking, for each transfer arc (u, v) in either network, that the corresponding transfer arc with the same extra information exists in the other network. To do this, assuming that vertices are labeled 1 to $n_1 + n_2$, with $n_i = |V(N_i)|$ as in the proof of Theorem 3, we can store each transfer arc in the form $([i, j], [k, l], r, r')$, where r, r' are the aforementioned ranks. This is a sextuple with integers bounded by $n_1 + n_2$. These can be sorted in linear time using radix sort, and then we can check that the lists of transfers are identical.

Theorem 6. *Given two binary LGT networks $\mathcal{N}_1, \mathcal{N}_2$ both of level at most ℓ, one can compute $d_{TR}(\mathcal{N}_1, \mathcal{N}_2)$ and thus $d_{LGT}(\mathcal{N}_1, \mathcal{N}_2)$ in time $O(4^\ell \cdot m^2)$, where m is the total number of edges in both networks.*

Proofs for Theorem 6 (FPT in the level)

If the input networks are binary, we can do better and obtain an FPT algorithm in the *level* of the input networks [11]. To define this notion, first, an arc (u, v) of an LGT network $N|T$ is a *bridge* if deleting (u, v) from N makes its underlying undirected graph disconnected. We call v a bridge-end. A *blob* of N is a maximal subgraph that contains no bridge. The *level* of $\mathcal{N}$ (and N) is the maximum number of reticulations in a blob of N. In binary networks, the level can be much lower than the number of transfers, making it a useful parameter in practice.

We say that a pair of LGT networks $\mathcal{N}_1 = N_1|T_1, \mathcal{N}_2 = N_2|T_2$. forms a *clean instance* if the transfer cleaning rule is not applicable on them. We can obtain a clean instance by applying the transfer cleaning rule until exhaustion.

Note that a transfer arc in either network cannot be a bridge, since there is always an undirected cycle that contains a transfer arc. For the same reason, a bridge-end cannot be an attachment point. Therefore, bridge-ends are tree vertices and have a single parent. This will be useful in the lemmas that build up to Theorem 6.

Lemma 2. *Suppose that $\mathcal{N}_1 = N_1|T_1, \mathcal{N}_2 = N_2|T_2$ is a clean instance. Then v is a bridge-end of N_1 if and only if v is a bridge-end of N_2.*

Proof. Let (u, v) be a bridge of N_1. As mentioned just above, v is a tree vertex of $\mathcal{N}_1$ and thus v is also in $\mathcal{N}_2$. If v is also a bridge-end of N_2, we are done, so assume otherwise for contradiction. Thus in the undirected version of N_2, the root can reach some vertex w descending from v, say through a path P. Without loss of generality, we may assume that w is a tree vertex, since otherwise we redefine w to be its next descendant that is a tree vertex. Thus w is also a vertex of N_1.

We can partition P so that it is equal to $P_1 - Q_1 - P_2 - Q_2 - \ldots - Q_{h-1} - P_h$ where: (1) each P_i is a subpath of T_2 that starts and ends at a tree vertex (so

it uses no transfer arc); (2) each Q_i starts with the last vertex p_i of P_i, goes to an attachment point on an edge incident to p_i, takes a transfer arc (in either direction), and goes to one of the tree vertices from the tree-pair that the transfer is on. We argue that in the undirected version of N_1, the root can also reach w without going through v. To see this, observe that for each P_i, the undirected version of T_1 also contains a path P'_i with the same ends that avoid v (we just traverse the same tree pairs as P_i, so P_i and P'_i only differ by the attachment points they go through). Likewise, for each Q_i, the undirected version of N_1 also has a path Q'_i with the same ends as Q_i, that uses a transfer, and whose interior vertices are all attachment points (the reason is that the instance is clean, so the tree pairs having the transfer arc used by Q_i also have a transfer in N_1). Moreover, Q'_i also avoids v. It follows that the sequence of edges formed by $P'_1 - Q'_1 - \dots Q'_{h-1} - P'_h$ goes from the root of N_1 to w without going through v. That sequence may not be a path because it could repeat vertices, but in this case we can remove the cycles in that sequence to obtain a path. We have reached a contradiction on (u, v) being a bridge of N_1, so the bridge-ends of N_1 are also bridge-ends of N_2. The converse holds using the same arguments. □

Recall that a blob of a network N is a maximal subgraph B of N that contains no bridge. A *pendant bridge of* B is a bridge (u, v) such that exactly one of u or v is in $V(B)$ (if v is not in it, then the bridge goes "out", and if u is not in it, then v is the root of the blob and the bridge is entering the blob). The set of pendant bridges of B is denoted $pend(B)$.

Lemma 3. *Suppose that $\mathcal{N}_1 = N_1|T_1, \mathcal{N}_2 = N_2|T_2$ is a clean instance. Let B_1 (resp. B_2) be the blob of N_1 (resp. N_2) that contains the root. Then v is the end of a pendant bridge of B_1 if and only if v is the end of a pendant bridge end of B_2.*

Proof. Let (u, v) be a pending bridge of B_1, with v the pendant bridge-end, and recall that v is a tree vertex. Thus v is also in N_2. Suppose that B_2 does not have a pending bridge with end v. By Lemma 2, v is also a bridge-end of N_2, and so the arc (u', v) of N_2 whose head is v is a pending bridge of some blob B'_2 of N_2 that is different from the root blob B_2. Let (p, q) be the arc entering this blob B'_2 (this arc exists and is unique, see [49]). Then (p, q) is a bridge of N_2 and therefore q is a bridge-end of N_1, say on the bridge (p', q) by Lemma 2. In N_2, (p, q) separates the root from v and its parent, and because v descends from q in N_1 as well, in N_1 we have that (p', q) separates the root from v and its parent u. This is a contradiction, since in N_1 the vertex u should not be in the same blob as the root. This concludes the proof. □

We are now setup to describe a dynamic programming algorithm that processes each blob independently. To this end, for an LGT network $\mathcal{N} = N|T$ and a bridge-end u, let $\mathcal{N}(u) = N(u)|T(u)$ denote the subnetwork of $\mathcal{N}$ rooted at u, where $N(u)$ (resp. $T(u)$) is the subgraph of N (resp. of T) induced by u and all its descendants. Note, by the definition of a bridge, no transfer arc of $\mathcal{N}$ has one end in $N(u)$ and the other outside.

Similarly, if B is a blob of N rooted at some vertex u, let $N\langle B\rangle$ denote the subgraph induced by B and all the vertices that are bottom ends of a pendant bridge of B. Note that these ends are the leaves of $N\langle B\rangle$.

Lemma 4. *Let $\mathcal{N}_1 = N_1|T_1, \mathcal{N}_2 = N_2|T_2$ be* binary *LGT networks that form a clean instance, and whose blobs containing the root are respectively B_1 and B_2. Then*

$$d_{TR}(\mathcal{N}_1, \mathcal{N}_2) = d_{TR}(\mathcal{N}_1\langle B_1\rangle, \mathcal{N}_2\langle B_2\rangle) + \sum_{(u,v)\in pend(B_1)} d_{TR}(\mathcal{N}_1(v), \mathcal{N}_2(v)).$$

Consequently, d_{TR} and thus d_{LGT} can be computed in time $O(4^\ell m^2)$, where ℓ is the maximum level of $\mathcal{N}_1$ or $\mathcal{N}_2$ and $m = |E(N_1)| + |E(N_2)|$.

Proof. Let us first focus on the correctness of our claimed recurrences. By Lemma 3, the networks $\mathcal{N}_1\langle B_1\rangle$ and $\mathcal{N}_2\langle B_2\rangle$ have the same leaves. Hence, if we make those subnetworks isomorphic by transfer deletions, and then replace each of their leaves by isomorphic subnetworks, the result is an isomorphic network. Thus the recurrence gives a possible way to obtain a common transfer reduction, which justifies

$$d_{TR}(\mathcal{N}_1, \mathcal{N}_2) \leq d_{TR}(\mathcal{N}_1\langle B_1\rangle, \mathcal{N}_2\langle B_2\rangle) + \sum_{(u,v)\in pend(B_1)} d_{TR}(\mathcal{N}_1(v), \mathcal{N}_2(v)).$$

For the converse bound, let R_1, R_2 be a set of transfer arcs to remove from $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively, to make them isomorphic. These deletion sets must, in particular, make $\mathcal{N}_1\langle B_1\rangle, \mathcal{N}_2\langle B_2\rangle$ isomorphic, and for each end v of a pendant bridge, make $\mathcal{N}_1(v)$ and $\mathcal{N}_2(v)$ isomorphic. The aforementioned subnetworks share no transfer arc, so the subsets of R_1, R_2 within any such subnetwork are disjoint. This justifies

$$d_{TR}(\mathcal{N}_1, \mathcal{N}_2) \geq d_{TR}(\mathcal{N}_1\langle B_1\rangle, \mathcal{N}_2\langle B_2\rangle) + \sum_{(u,v)\in pend(B_1)} d_{TR}(\mathcal{N}_1(v), \mathcal{N}_2(v)).$$

As for the complexity, we first make the networks clean by applying the transfer cleaning rule until exhaustion. This is easy to do in linear time $O(m)$, using the techniques from Theorem 3. For concreteness, if we assume constant time operations on hash tables, we can find the transfers to delete in the preprocessing by hashing each transfer according to the tree pairs it is on, and then computing the symmetric difference in linear time. Once this is done, we can compute the d_{TR} entries by dynamic programming in a bottom-up fashion. It is easily seen that the only entries $d_{TR}(\mathcal{N}_1(v), \mathcal{N}_2(v))$ that will ever be required are those when v is the root of a blob (this includes the case $d_{TR}(\mathcal{N}_1, \mathcal{N}_2)$, where v is taken to be the root of both networks). There are $O(n) = O(m)$ such possible entries. For one specific entry $d_{TR}(\mathcal{N}_1(v), \mathcal{N}_2(v))$, we try deleting every subset of transfer arcs from the root blob of the first network, and every subset in the second. Since every reticulation is incident to at most one transfer arc in a binary LGT network, the number of transfer arcs is at most ℓ and there are thus

2^ℓ subsets to enumerate in each blob, for a total of 4^ℓ combinations. For each combination, we check whether applying the deletions leads to an isomorphic network, which adds linear time (see the proof of Proposition 1).

In summary, for each pair of blobs we spend time $O(4^\ell m)$ to enumerate the deletion sets and test isomorphism on each, and there are $O(m)$ entries to compute. The time for cleaning the networks does not impact this, so overall the time is $O(4^\ell m^2)$.

The complexity claimed in Theorem 6 then follows immediately from the previous lemma.

Algorithms for Section 4

Here we describe the algorithm from Proposition 1. The same ideas can be used for Theorem 6. The algorithm computes d_{LGT} as the sum of the weight of bad vertices (wRF), the number of extra transfer deletions to make (D_{tr}), and the number of doubly bad transfers (D). Bad vertices are found using Day algorithm. Algorithm 1 integrates these components and returns the combined contribution of tree disagreement and transfer disagreement.

Algorithm 1: getD_LGT($N_1|T_1, N_2|T_2$)

Data: LGT networks $N_1|T_1$ and $N_2|T_2$
Result: $d_{LGT}(N_1|T_1, N_2|T_2)$

$badVertices \leftarrow \emptyset$; $wRF \leftarrow 0$; $D_tr \leftarrow 0$; $D \leftarrow 0$;
$bad^1_{tail} \leftarrow \emptyset$; $bad^1_{head} \leftarrow \emptyset$; $bad^2_{tail} \leftarrow \emptyset$; $bad^2_{head} \leftarrow \emptyset$;
Compute $badVertices$ and wRF using Day's algorithm;
for $i \in \{1, 2\}$ **do**
  $bad^i_{tail} \leftarrow$ transfers of $N_i|T_i$ whose tail is on $[u, v]$ with v in $badVertices$;
  $bad^i_{head} \leftarrow$ transfers of $N_i|T_i$ whose head is on $[u, v]$ with v in $badVertices$;
  $D \leftarrow D + |bad^i_{tail} \cap bad^i_{head}|$; `// counting the doubly bad transfers`
  Delete from $N_i|T_i$ all transfer arcs in $bad^i_{tail} \cup bad^i_{head}$;
for $i \in \{1, 2\}$ **do**
  foreach $v \in V(T_i)$ *in pre-order* **do**
    if $v \in badVertices$ **then**
      Contract v in T_i;
if $N_1|T_1, N_2|T_2$ *have unordered transfers* **then**
  $D_tr \leftarrow getD_trUnordered(N_1|T_1, N_2|T_2)$;
else
  $D_tr \leftarrow getD_trOrdered(N_1|T_1, N_2|T_2)$;
return $wRF + D_tr - D$

When transfer arcs are treated as unordered, the distance reduces to counting the arcs that appear in exactly one of the two networks. In this setting, no additional structural constraints need to be verified once the relative complements are computed. Algorithm 2 therefore returns the transfer distance directly

as the total number of non-shared arcs. Note, the relative complement of A in B is the set difference $B \setminus A$. We can obtain the symmetric difference $A \triangle B$ by computing both relative complements $A \setminus B$ and $B \setminus A$. The algorithm for GetRelativeComplements is shown aferwards.

Algorithm 2: getD_trUnordered($N_1|T_1, N_2|T_2$)

Data: LGT networks $N_1|T_1$ and $N_2|T_2$ with unordered transfers where T_1 and T_2 are homeomorphic
Result: $d_{TR}(N_1|T_1, N_2|T_2)$

```
(tr_rc(N_1|T_1), tr_rc(N_2|T_2)) ← GetRelativeComplements(N_1|T_1,N_2|T_2);
return |tr_rc(N_1|T_1)| + |tr_rc(N_2|T_2)|
```

When transfers are ordered, our FPT algorithm simply explores all subsets of the remaining transfer arcs and finds the smallest pair of deletion sets that yields equivalent LGT structures, as shown in Algorithm 3.

Algorithm 3: getD_trOrdered($N_1|T_1, N_2|T_2$)

Data: LGT networks $N_1|T_1$ and $N_2|T_2$ with ordered transfers where T_1 and T_2 are homeomorphic
Result: $d_{TR}(N_1|T_1, N_2|T_2)$

```
min_d ← ∞;
//Pre-processing: remove transfers that have no chance of being kept;
(tr_rc(N_1|T_1), tr_rc(N_2|T_2)) ← GetRelativeComplements(N_1|T_1,N_2|T_2);
Delete transfer arcs from N_1|T_1 corresponding to the set tr_rc(N_1|T_1);
Delete transfer arcs from N_2|T_2 corresponding to the set tr_rc(N_2|T_2);
//Brute-force the subsets of remaining transfers to find the optimal deletion
  sets foreach R_1 ⊆ E(N_1) \ E(T_1) do
    foreach R_2 ⊆ E(N_2) \ E(T_2) do
      N'_1|T'_1 ← copy of N_1|T_1 with transfers from R_1 deleted;
      N'_2|T'_2 ← copy of N_2|T_2 with transfers from R_2 deleted;
      if N'_1|T'_1 ≃_LGT N'_2|T'_2 then
        min_d ← min(min_d, |R_1| + |R_2|);
return min_d
```

Algorithm 4 performs this computation of the relative complements, so that the returned pair of sets can be used to get the symmetric difference, or to delete the transfers in both networks. This is done by sorting both arc sets and scanning them in parallel to isolate the non-shared arcs. The transfers in $tr(N_i|T_i)$ have the form $([u_1, u_2], [v_1, v_2])$, which can be seen as a quadruple of integers between 0 and the sizes of the networks, in which case radix sort takes linear time.

Algorithm 4: getRelativeComplements($N_1|T_1$, $N_2|T_2$)

Data: LGT networks $N_1|T_1$ and $N_2|T_2$ with T_1, T_2 homeomorphic
Result: A pair $(tr_{rc}(N_1|T_1), tr_{rc}(N_2|T_2))$ corresponding to the relative complements between $N_1|T_1$ and $N_2|T_2$.

```
tr_ord(N1|T1) ← radixSort(tr(N1|T1)); tr_ord(N2|T2) ← radixSort(tr(N2|T2));
i ← 0; j ← 0;
tr_rc(N1|T1) ← ∅; tr_rc(N2|T2) ← ∅ ;  // To store the relative complements
while i < |tr_ord(N1|T1)| and j < |tr_ord(N2|T2)| do
    if tr_ord(N1|T1)[i] == tr_ord(N2|T2)[j] then   // tr_ord(N1|T1)[i]: ith pair
        i ← i + 1; j ← j + 1;
    else if tr_ord(N1|T1)[i] < tr_ord(N2|T2)[j] then
        tr_rc(N1|T1) ← tr_rc(N1|T1) + tr_ord(N1|T1)[i];
        i ← i + 1;
    else
        tr_rc(N2|T2) ← tr_rc(N2|T2) + tr_ord(N2|T2)[j];
        j ← j + 1;
while i ≤ |tr_ord(N1|T1)| do
    tr_rc(N1|T1) ← tr_rc(N1|T1) + tr_ord(N1|T1)[i];
    i ← i + 1;
while j ≤ |tr_ord(N2|T2)| do
    tr_rc(N2|T2) ← tr_rc(N2|T2) + tr_ord(N2|T2)[j];
    j ← j + 1;
return (tr_rc(N1|T1), tr_rc(N2|T2))
```

C Proofs for Section 5 (Experiments)

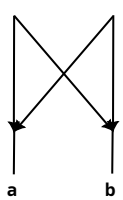

Fig. 12. The pattern shown the left is not allowed for the sub-class of orchard networks (see for instance [26] for a definition). Although it can be considered non-realistic (it is not time-consistent). Yet, the network prediction methods used in Sections 5.2 and 5.3 (namely [34] and [31]) could in principle return networks exhibiting this pattern, requiring comparison methods that are capable of comparing ideally all tree-based networks [36], and not just orchard networks.

Additional experiment data for Section 5

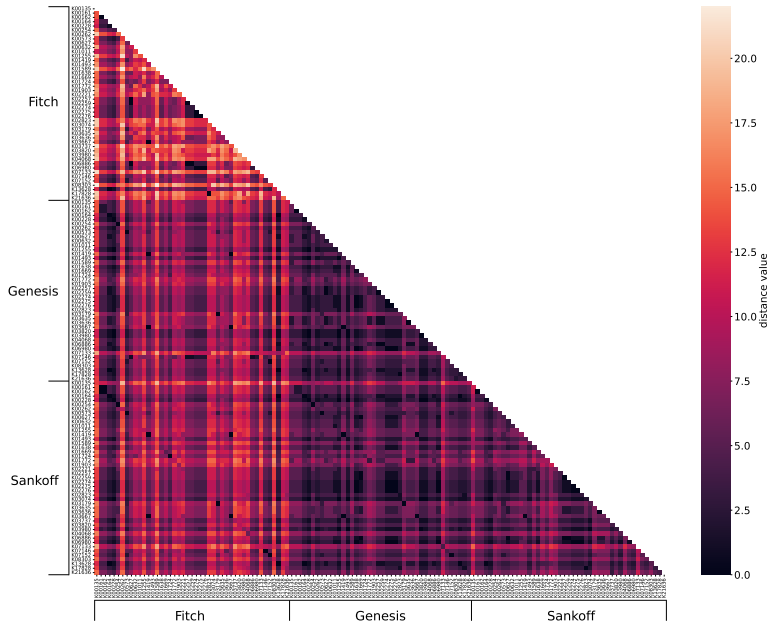

Fig. 13. Ancestral character reconstruction obtained using the Basic, Sankoff, and Genesis algorithms.