Constructing a Consensus Phylogeny from a Leaf-Removal Distance

Cedric Chauve[1], Mark Jones[2], Manuel Lafond[3], Céline Scornavacca[4], and Mathias Weller[5]

[1]Department of Mathematics, Simon Fraser University, Burnaby, BC, Canada,, cedric.chauve@sfu.ca
[2]Delft Institute of Applied Mathematics, Delft University of Technology, P.O. Box 5, 2600 AA, Delft, the Netherlands, M.E.L.Jones@tudelft.nl
[3]Department of Mathematics and Statistics, University of Ottawa, 585 King Edward Ave., K1N 6N5 Ottawa, Canada, mlafond2@uOttawa.ca
[4]Institut des Sciences de l'Evolution – Université Montpellier, CNRS, IRD, EPHE, Place Eugène Bataillon, 34095 Montpellier, France, celine.scornavacca@umontpellier.fr
[5]Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, 34392 Montpellier Cedex 5 - France, mathias.weller@lirmm.fr

September 14, 2018

Abstract

Understanding the evolution of a set of genes or species is a fundamental problem in evolutionary biology. The problem we study here takes as input a set of trees describing possibly discordant evolutionary scenarios for a given set of genes or species, and aims at finding a single tree that minimizes the leaf-removal distance to the input trees. This problem is a specific instance of the general consensus/supertree problem, widely used to combine or summarize discordant evolutionary trees. The problem we introduce is specifically tailored to address the case of discrepancies between the input trees due to the misplacement of individual taxa. Most supertree or consensus tree problems are computationally intractable, and we show that the problem we introduce is also NP-hard. We provide tractability results in form of a 2-approximation algorithm and a parameterized algorithm with respect to the number of removed leaves. We also introduce a variant that minimizes the maximum number d of leaves that

are removed from any input tree, and provide a parameterized algorithm for this problem with parameter d.

1 Introduction

In the present paper, we consider a very generic computational biology problem: given a collection of trees representing, possibly discordant, evolutionary scenarios for a set of biological entities (genes or species – also called *taxa* in the following), we want to compute a single tree that agrees as much as possible with the input trees. Several questions in computational biology can be phrased in this generic framework. For example, for a given set of homologous gene sequences that have been aligned, one can sample *evolutionary trees* for this gene family according to a well defined posterior distribution and then ask how this collection of trees can be combined into a single gene tree, a problem known as *tree amalgamation* [13]. In phylogenomics, one aims at *inferring a species tree* from a collection of input trees obtained from whole-genome sequence data. A first approach considers gene families and proceeds by computing individual *gene trees* from a large set of gene families, and then combining this collection of gene trees into a unique species tree for the given set of taxa; this requires handling the discordant signal observed in the gene trees due to evolutionary processes such as gene duplication and loss [10], lateral gene transfer [14], or incomplete lineage sorting [12]. Another approach concatenates the sequence data into a single large multiple sequence alignment, that is then partitioned into overlapping subsets of taxa for which partial evolutionary trees are computed, and a unique species tree is then inferred by combining the resulting collection of partial trees [11].

For example, the Maximum Agreement Subtree (MAST) problem considers a collection of input trees[1], all having the same leaf labels and looks for a tree of maximum size (number of leaves), which agrees with each of the input trees. This problem is tractable for trees with bounded degree but NP-hard generally [1]. The MAST problem is a *consensus problem*, because the input trees share the same leaf labels set, and the output tree is called a *consensus* tree. In the *supertree framework*, the input trees might not all have identical label sets, but the output is a tree on the whole label set, called a *supertree*. For example, in the Robinson-Foulds (RF) supertree problem, the goal is to find a supertree that minimizes the sum of the RF-distances to the individual input trees [15]. One way to compute consensus trees and supertrees that is closely related to our work is to modify the collection of input trees minimally in such a way that the resulting modified trees all agree. For example, in the MAST problem, modifications of the input trees consist in removing a minimum number of taxa from the whole label set, while in the Agreement Supertree by Edge Contraction (AST-EC) problem, one is asked to contract a minimum number of edges of the input trees such that the resulting (possibly non-binary)

[1] All trees we consider here are uniquely leaf-labeled, rooted (*i.e.* are out-trees) and binary; see next section for formal definitions.

trees all agree with at least one supertree [7]; in the case where the input trees are all triplets (rooted trees on three leaves), this supertree problem is known as the Minimum Rooted Triplets Inconsistency problem [4]. The SPR Supertree problem considers a similar problem where the input trees can be modified with the Subtree-Prune-and-Regraft (SPR) operator [16].

In the present work, we introduce a new consensus problem, called `LR-Consensus`. Given a collection of input trees having the same leaf labels set, we want to remove a minimum number of leaves – an operation called a Leaf-Removal (LR) – from the input trees such that the resulting pruned trees all agree. Alternatively, this can be stated as finding a consensus tree that minimizes the cumulated *leaf-removal distance* to the collection of input trees. This problem also applies to tree amalgamation and to species tree inference from one-to-one orthologous gene families, where the LR operation aims at correcting the misplacement of a single taxon in an input tree.

In the next section, we formally define the problems we consider, and how they relate to other supertree problems. Next we show that the `LR-Consensus` problem is NP-hard and that in some instances, a large number of leaves need to be removed to lead to a consensus tree. We then provide a 2-approximation algorithm, and show that the problem is fixed-parameter tractable (FPT) when parameterized by the total number of LR. However, these FPT algorithms have impractical time complexity, and thus, to answer the need for practical algorithms, we introduce a variant of the `LR-Consensus` problem, where we ask if a consensus tree can be obtained by removing at most d leaves from each input tree, and describe an FPT algorithm with parameter d.

2 Preliminary notions and problems statement

Trees. All trees in the rest of the document are assumed to be rooted and binary. If T is a tree, we denote its root by $r(T)$ and its leaf set by $\mathcal{L}(T)$. Each leaf is labeled by a distinct element from a *label set* $\mathcal{X}$, and we denote by $\mathcal{X}(T)$ the set of labels of the leaves of T. We may sometimes use $\mathcal{L}(T)$ and $\mathcal{X}(T)$ interchangeably. For some $X \subseteq \mathcal{X}$, we denote by $lca_T(X)$ the *least common ancestor* of X in T. The subtree rooted at a node $u \in V(T)$ is denoted T_u and we may write $\mathcal{L}_T(u)$ for $\mathcal{L}(T_u)$. If T_1 and T_2 are two trees and e is an edge of T_1, grafting T_2 on e consists in subdividing e and letting the resulting degree 2 node become the parent of $r(T_2)$. Grafting T_2 above T_1 consists in creating a new node r, then letting r become the parent of $r(T_1)$ and $r(T_2)$. Grafting T_2 on T_1 means grafting T_2 either on an edge of T_1 or above T_1.

The leaf removal operation. For a subset $L \subseteq \mathcal{X}$, we denote by $T - L$ the tree obtained from T by removing every leaf labeled by L, contracting the resulting non-root vertices of degree two, and repeatedly deleting the resulting root vertex while it has degree one. The *restriction* $T|_L$ of T to L is the tree $T - (\mathcal{X} \setminus L)$, *i.e.* the tree obtained by removing every leaf *not* in L. A *triplet* is a rooted tree on 3 leaves. We denote a triplet R with leaf set $\{a, b, c\}$ by $ab|c$

if c is the leaf that is a direct child of the root (the parent of a and b being its other child). We say $R = ab|c$ is a triplet of a tree T if $T|_{\{a,b,c\}} = R$. We denote $tr(T) = \{ab|c : ab|c \text{ is a triplet of } T\}$.

We define a *distance function* d_{LR} between two trees T_1 and T_2 on the same label set $\mathcal{X}$ consisting in the minimum number of labels to remove from $\mathcal{X}$ so that the two trees are equal. That is,

$$d_{LR}(T_1, T_2) = \min\{|X| : X \subseteq \mathcal{X} \text{ and } T_1 - X = T_2 - X\}$$

Note that d_{LR} is closely related to the Maximum Agreement Subtree (MAST) between two trees on the same label set $\mathcal{X}$, which consists in a subset $X' \subseteq \mathcal{X}$ of maximum size such that $T_1|_{X'} = T_2|_{X'}$: $d_{LR}(T_1, T_2) = |\mathcal{X}| - |X'|$. The MAST of two binary trees on the same label set can be computed in time $O(n \log n)$, where $n = |\mathcal{X}|$ [5], and so d_{LR} can be found within the same time complexity.

Problem statements. In this paper, we are interested in finding a tree T on $\mathcal{X}$ minimizing the sum of d_{LR} distances to a given set of input trees.

`LR-Consensus`
Given: a set of trees $\mathcal{T} = \{T_1, \ldots, T_t\}$ with $\mathcal{X}(T_1) = \ldots = \mathcal{X}(T_t) = \mathcal{X}$.
Find: a tree T on label set $\mathcal{X}$ that minimizes $\sum_{T_i \in \mathcal{T}} d_{LR}(T, T_i)$.

We can reformulate the `LR-Consensus` problem as the problem of removing a minimum number of leaves from the input trees so that they are *compatible*. Although the equivalence between both formulations is obvious, the later formulation will often be more convenient. We need to introduce more definitions in order to establish this equivalence.

A set of trees $\mathcal{T} = \{T_1, \ldots, T_t\}$ is called *compatible* if there is a tree T such that $\mathcal{X}(T) = \bigcup_{T_i \in \mathcal{T}} \mathcal{X}(T_i)$ and $T|_{\mathcal{X}(T_i)} = T_i$ for every $i \in [t]$. In this case, we say that T *displays* $\mathcal{T}$. A list $\mathcal{C} = (\mathcal{X}_1, \ldots, \mathcal{X}_t)$ of subsets of $\mathcal{X}$ is a *leaf-disagreement* for $\mathcal{T}$ if $\{T_1 - \mathcal{X}_1, \ldots, T_t - \mathcal{X}_t\}$ is compatible. The *size* of $\mathcal{C}$ is $\sum_{i \in [t]} |\mathcal{X}_i|$. We denote by $AST_{LR}(\mathcal{T})$ the minimum size of a leaf-disagreement for $\mathcal{T}$, and may sometimes write $AST_{LR}(T_1, \ldots, T_t)$ instead of $AST_{LR}(\mathcal{T})$. A subset $\mathcal{X}' \subseteq \mathcal{X}$ of labels is a *label-disagreement* for $\mathcal{T}$ if $\{T_1 - \mathcal{X}', \ldots, T_t - \mathcal{X}'\}$ is compatible. Note that, if $\mathcal{T} = \{T_1, T_2\}$, then the minimum size of a label-disagreement for $\mathcal{T}$ is $d_{LR}(T_1, T_2)$. We may now define the `AST-LR` problem (see Figure 1 for an example).

`Agreement Subtrees by Leaf-Removals` (`AST-LR`)
Given: a set of trees $\mathcal{T} = \{T_1, \ldots, T_t\}$ with $\mathcal{X}(T_1) = \ldots = \mathcal{X}(T_t) = \mathcal{X}$.
Find: a leaf-disagreement $\mathcal{C}$ for $\mathcal{T}$ of minimum size.

Lemma 1. *Let $\mathcal{T} = \{T_1, \ldots, T_t\}$ be a set of trees on the same label set $\mathcal{X}$, with $n = |\mathcal{X}|$. Given a supertree T such that $v := \sum_{T_i \in \mathcal{T}} d_{LR}(T, T_i)$, one can compute in time $O(tn \log(n))$ a leaf-disagreement $\mathcal{C}$ of size at most v. Conversely, given a leaf-disagreement $\mathcal{C}$ for $\mathcal{T}$ of size v, one can compute in time $O(tn \log^2(tn))$ a supertree T such that $\sum_{T_i \in \mathcal{T}} d_{LR}(T, T_i) \le v$.*

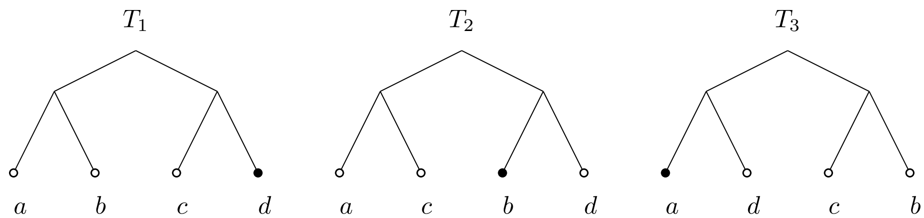

Figure 1: Example instance $\mathcal{T} = \{T_1, T_2, T_3\}$ of `AST-LR` with label set $\mathcal{X} = \{a, b, c, d\}$. The list $(\mathcal{X}_1 = \{d\}, \mathcal{X}_2 = \{b\}, \mathcal{X}_3 = \{a\})$ is a leaf-disagreement for $\mathcal{T}$ of size 3.

From Lemma 1[2] both problems share the same optimality value, the NP-hardness of one implies the hardness of the other and approximating one problem within a factor c implies that the other problem can be approximated within a factor c. We conclude this subsection with the introduction of a parameterized variant of the `AST-LR` problem.

`AST-LR-d`
Input: a set of trees $\mathcal{T} = \{T_1, \ldots, T_t\}$ with $\mathcal{L}(T_1) = \ldots = \mathcal{L}(T_t) = \mathcal{X}$, and an integer d.
Question: Are there $\mathcal{X}_1, \ldots, \mathcal{X}_t \subseteq \mathcal{X}$ such that $|\mathcal{X}_i| \leq d$ for each $i \in [t]$, and $\{T_1 - \mathcal{X}_1, \ldots, T_t - \mathcal{X}_t\}$ is compatible?

We call a tree T^* a *solution* to the `AST-LR-d` instance if $d_{LR}(T_i, T^*) \leq d$ for each $i \in [t]$.

Relation to other supertree/consensus tree problems. The most widely studied supertree problem based on modifying the input trees is the SPR Supertree problem, where arbitrarily large subtrees can be moved in the input trees to make them all agree (see [16] and references there). The interest of this problem is that the SPR operation is very general, modelling lateral gene transfer and introgression. The LR operation we introduce is a limited SPR, where the displaced subtree is composed of a single leaf. An alternative to the SPR operation to move subtrees within a tree is the Edge Contraction (EC) operation, that contracts an edge of an input tree, thus increasing the degree of the parent node. This operation allows correcting the local misplacement of a full subtree. `AST-EC` is NP-complete but can be solved in $O((2t)^p tn^2)$ time where p is the number of required EC operations [7].

Compared to the two problems described above, an LR models a very specific type of error in evolutionary trees, that is the misplacement of a single taxon (a single leaf) in one of the input trees. This error occurs frequently in reconstructing evolutionary trees, and can be caused for example by some evolutionary process specific to the corresponding input tree (recent incomplete lineage sorting, or recent lateral transfer for example). Conversely, it is not well

[2] All missing proofs are provided in Appendix.

adapted to model errors, due for example to ancient evolutionary events that impacts large subtrees. However, an attractive feature of the LR operation is that computing the LR distance is equivalent to computing the `MAST` cost and is thus tractable, unlike the SPR distance which is hard to compute. This suggests that the `LR-Consensus` problem might be easier to solve than the SPR Supertree problem, and we provide indeed several tractability results. Compared to the `AST-EC` problem, the `AST-LR` problem is naturally more adapted to correct single taxa misplacements as the EC operation is very local and the number of EC required to correct a taxon misplacement is linear in the length of the path to its correct location, while the LR cost of correcting this is unitary. Last, `LR-Consensus` is more flexible than the `MAST` problem as it relies on modifications of the input trees, while with the way `MAST` corrects a misplaced leaf requires to remove this leaf from all input trees. This shows that the problems `AST-LR` and `AST-LR-d` complement well the existing corpus of gene trees correction models.

3 Hardness and approximability of `AST-LR`

In this section, we show that the `AST-LR` problem is NP-hard, from which the `LR-Consensus` hardness follows. We then describe a simple factor 2 approximation algorithm. The algorithm turns out to be useful for analyzing the worst case scenario for `AST-LR` in terms of the required number of leaves to remove, as we show that there are `AST-LR` instances that require removing about $n - \sqrt{n}$ leaves in each input tree.

NP-hardness of `AST-LR`

We assume here that we are considering the decision version of `AST-LR`, *i.e.* deciding whether there is a leaf-disagreement of size at most ℓ for a given ℓ. We use a reduction from the `MinRTI` problem: given a set $\mathcal{R}$ of rooted triplets, find a subset $\mathcal{R}' \subset \mathcal{R}$ of minimum cardinality such that $\mathcal{R} \setminus \mathcal{R}'$ is compatible. The `MinRTI` problem is NP-Hard [4] (even $W[2]$-hard and hard to approximate within a $O(\log n)$ factor). Denote by $MINRTI(\mathcal{R})$ the minimum number of triplets to remove from $\mathcal{R}$ to attain compatibility. We describe the reduction here.

Let $\mathcal{R} = \{R_1, \ldots, R_t\}$ be an instance of `MinRTI`, with the label set $L := \bigcup_{i=1}^{t} \mathcal{X}(R_i)$. For a given integer m, we construct an `AST-LR` instance $\mathcal{T} = \{T_1, \ldots, T_t\}$ which is such that $MINRTI(\mathcal{R}) \leq m$ if and only if $AST_{LR}(\mathcal{T}) \leq t(|L| - 3) + m$.

We first construct a tree Z with additional labels which will serve as our main gadget. Let $\{L_i\}_{1 \leq i \leq t}$ be a collection of t new label sets, each of size $(|L|t)^{10}$, all disjoint from each other and all disjoint from L. Each tree in our `AST-LR` instance will be on label set $\mathcal{X} = L \cup L_1 \cup \ldots \cup L_t$. For each $i \in [t]$, let X_i be any tree with label set L_i. Obtain Z by taking any tree on t leaves

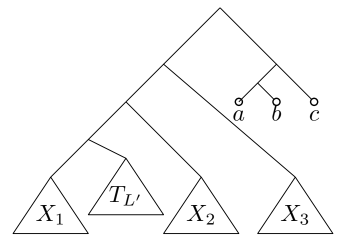

Figure 2: Construction of the tree T_1 for an instance $\mathcal{R} = \{R_1, R_2, R_3\}$ of `MinRTI` in which $R_1 = ab|c$.

$l_1, \ldots, l_t$, then replacing each leaf l_i by the X_i tree (*i.e.* l_i is replaced by $r(X_i)$). Denote by $r_Z(X_i)$ the root of the X_i subtree in Z.

Then for each $i \in [t]$, we construct T_i from R_i as follows. Let $L' = L \setminus \mathcal{X}(R_i)$ be the set of labels not appearing in R_i, noting that $|L'| = |L| - 3$. Let $T_{L'}$ be any tree with label set L', and obtain the tree Z_i by grafting $T_{L'}$ on the edge between $r_Z(X_i)$ and its parent. Finally, T_i is obtained by grafting R_i above Z_i. See Figure 2 for an example. Note that each tree T_i has label set $\mathcal{X}$ as desired. Also, it is not difficult to see that this reduction can be carried out in polynomial time. This construction can now be used to show the following.

Theorem 1. *The `AST-LR` and `LR-Consensus` problems are NP-hard.*

The idea of the proof is to show that in the constructed `AST-LR` instance, we are "forced" to solve the corresponding `MinRTI` instance. In more detail, we show that $MINRTI(\mathcal{R}) \leq m$ if and only if $AST_{LR}(\mathcal{T}) \leq t(|L|-3)+m$. In one direction, given a set $\mathcal{R}'$ of size m such that $\mathcal{R} \setminus \mathcal{R}'$ is compatible, one can show that the following leaf removals from $\mathcal{T}$ make it compatible: remove, from each T_i, the leaves $L' = L \setminus \mathcal{X}(R_i)$ that were inserted into the Z subtree, then for each $R_i \in \mathcal{R}'$, remove a single leaf in $\mathcal{X}(R_i)$ from T_i. This sums up to $t(|L|-3)+m$ leaf removals. Conversely, it can be shown that there always exists an optimal solution for $\mathcal{T}$ that removes, for each T_i, all the leaves $L' = L \setminus \mathcal{X}(R_i)$ inserted in the Z subtree, plus an additional single leaf l from m trees $T_{i_1}, \ldots, T_{i_m}$ such that $l \in L$. The corresponding triplets $R_{i_1}, \ldots, R_{i_m}$ can be removed from $\mathcal{R}$ so that it becomes compatible.

Approximating `AST-LR` and bounding worst-case scenarios

Given the above result, it is natural to turn to approximation algorithms in order to solve `AST-LR` or `LR-Consensus` instances. It turns out that there is a simple factor 2 approximation for `LR-Consensus` which is achieved by interpreting the problem as finding a median in a metric space. Indeed, it is not hard to see that d_{LR} is a metric (over the space of trees on the same label set $\mathcal{X}$). A direct consequence, using an argument akin to the one in [9, p.351], is the following.

Theorem 2. *The following is a factor 2 approximation algorithm for `LR-Consensus`: return the tree $T \in \mathcal{T}$ that minimizes $\sum_{T_i \in \mathcal{T}} d_{LR}(T, T_i)$.*

Theorem 2 can be used to lower-bound the 'worst' possible instance of AST-LR. We show that in some cases, we can only keep about $\sqrt{|\mathcal{X}|}$ leaves per tree. That is, there are instances for which $AST_{LR}(\mathcal{T}) = \Omega(t(n - \sqrt{n}))$, where t is the number of trees and $n = |\mathcal{X}|$. The argument is based on a probabilistic argument, for which we will make use of the following result [3, Theorem 4.3.iv].

Theorem 3 ([3])**.** *For any constant $c > e/\sqrt{2}$, there is some n_0 such that for all $n \geq n_0$, the following holds: if T_1 and T_2 are two binary trees on n leaves chosen randomly, uniformly and independently, then $\mathbb{E}[d_{LR}(T_1, T_2)] \geq n - c\sqrt{n}$.*

Corollary 1. *There are instances of AST-LR in which $\Omega(t(n - \sqrt{n}))$ leaves need to be deleted.*

The above is shown by demonstrating that, by picking a set $\mathcal{T}$ of t random trees, the expected optimal sum of distances $\min_T \sum_{T_i \in \mathcal{T}} d_{LR}(T, T_i)$ is $\Omega(t(n - \sqrt{n}))$. This is not direct though, since the tree T^* that minimizes this sum is not itself random, and so we cannot apply Theorem 3 directly on T^*. We can however, show that the tree $T' \in \mathcal{T}$ obtained using the 2-approximation, which is random, has expected sum of distances $\Omega(t(n - \sqrt{n}))$. Since T^* requires, at best, half the leaf deletions of T', the result follows. Note that finding a non-trivial upper bound on $AST_{LR}(\mathcal{T})$ is open.

4 Fixed-parameter tractability of AST-LR and AST-LR-d.

An alternative way to deal with computational hardness is parameterized complexity. In this section, we first show that AST-LR is fixed-parameter-tractable with respect to $q := AST_{LR}(\mathcal{T})$. More precisely, we show that AST-LR can be solved in $O(12^q n^3)$ time, where $n := |\mathcal{X}|$. We then consider an alternative parameter d, and show that finding a tree T^*, if it exists, such that $d_{LR}(T_i, T^*) \leq d$ for every input tree T_i, can be done in $O(c^d d^{3d}(n^3 + tn \log n))$ time for some constant c.

4.1 Parameterization by q

The principle of the algorithm is the following. It is known that a set of trees $\mathcal{T} = \{T_1, \ldots, T_t\}$ is compatible if and only if the union of their triplet decomposition $tr(\mathcal{T}) = \bigcup_{T_i \in \mathcal{T}} tr(T_i)$ is compatible [2]. In a step-by-step fashion, we identify a conflicting set of triplets in $tr(\mathcal{T})$, each time branching into the (bounded) possible leaf-removals that can resolve the conflict. We stop when either $tr(\mathcal{T})$ is compatible after the performed leaf-removals, or when more than q leaves were deleted.

We employ a two phase strategy. In the first phase, we eliminate direct conflicts in $tr(\mathcal{T})$, *i.e.* if at least two of $ab|c, ac|b$ and $bc|a$ appear in $tr(\mathcal{T})$, then we recursively branch into the three ways of choosing one of the 3 triplets, and remove one leaf in each T_i disagreeing with the chosen triplet (we branch into the three possible choices, either removing a, b or c). The chosen triplet is locked in $tr(\mathcal{T})$ and cannot be changed later.

When the first phase is completed, there are no direct conflicts and $tr(\mathcal{T})$ consists in a *full set of triplets* on $\mathcal{X}$. That is, for each distinct $a, b, c \in \mathcal{X}$, $tr(\mathcal{T})$ contains exactly one triplet on label set $\{a, b, c\}$. Now, a full set of triplets is not necessarily compatible, and so in the second phase we modify $tr(\mathcal{T})$, again deleting leaves, in order to make it compatible. Only the triplets that have not been locked previously can be modified. This second phase is analogous to the FPT algorithm for *dense* `MinRTI` presented in [8]. The dense `MinRTI` is a variant of the `MinRTI` problem, introduced in Section 3, in which the input is a full set of triplets and one has to decide whether p triplets can be deleted to attain compatibility.

Theorem 4 ([8])**.** *A full set of triplets $\mathcal{R}$ is compatible if and only if for any set of four labels $\{a, b, c, d\}$, $\mathcal{R}$ does not contain the subset $\{ab|c, cd|b, bd|a\}$ nor the subset $\{ab|c, cd|b, ad|b\}$.*

One can check, through an exhaustive enumeration of the possibilities, that there are only four ways to correct a conflicting set of triplets R_1, R_2, R_3 where $R_1 = ab|c, R_2 = cd|b, R_3 \in \{bd|a, ad|b\}$. We can: (1) transform R_1 to $bc|a$; (2) transform R_1 to $ac|b$; (3) transform R_2 to $bd|c$; (4) transform R_3 to $ab|d$. This leads to a $O(4^p n^3)$ algorithm for solving dense `MinRTI`: find a conflicting set of four labels, and branch on the four possibilities, locking the transformed triplet each time.

For the second phase of `AST-LR`, we propose a slight variation of this algorithm. Each time a triplet R is chosen and locked, say $R = ab|c$, the trees containing $ac|b$ or $bc|a$ must loose a, b or c. We branch into these three possibilities. Thus for each conflicting 4-set, there are four ways of choosing a triplet, then for each such choice, three possible leaves to delete from a tree. This gives 12 choices to branch into recursively. Algorithm 1 is described in detail in the Appendix and its analysis yields the following.

Theorem 5. *`AST-LR` can be solved in time $O(12^q tn^3)$.*

Although Theorem 5 is theoretically interesting as it shows that `AST-LR` is in FPT with respect to q, the 12^q factor might be too high for practical purposes, motivating the alternative approach below.

4.2 Parameterization by maximum distance d

We now describe an algorithm for the `AST-LR-d` problem, running in time $O(c^d d^{3d}(n^3 + tn \log n))$ that, if it exists, finds a solution (where here c is a constant not depending on d nor n).

We employ the following branch-and-bound strategy, keeping a candidate solution at each step. Initially, the candidate solution is the input tree T_1 and, if T_1 is indeed a solution, we return it. Otherwise (in particular if $d_{LR}(T_1, T_i) > d$ for some input tree T_i), we branch on a set of "leaf-prune-and-regraft" operations on T_1. In such an operation, we prune one leaf from T_1 and regraft it somewhere else. If we have not produced a solution after d such operations, then we halt this branch of the algorithm (as any solution must be reachable from T_1 by at most d operations). The resulting search tree has depth at most d. In order to bound the running time of the algorithm, we need to bound the number of "leaf-prune-and-regraft" operations to try at each branching step. There are two steps to this: first, we bound the set of candidate leaves to prune, second, given a leaf, we bound the number of places where to regraft it. To bound the candidate set of leaves to prune, let us call a leaf x *interesting* if there is a solution T^*, and minimal sets $X_1, X_i \subseteq \mathcal{X}$ of size at most d, such that (a) $T_1 - X_1 = T^* - X_1$, (b) $T_i - X_i = T^* - X_i$, and (c) $x \in X_1 \setminus X_i$, where T_i is an arbitrary input tree for which $d_{LR}(T_1, T_i) > d$. It can be shown that an interesting leaf x must exist if there is a solution. Moreover, though we cannot identify x before we know T^*, we can nevertheless construct a set S of size $O(d^2)$ containing all interesting leaves. Thus, in our branching step, it suffices to consider leaves in S.

Assuming we have chosen the correct x, we then bound the number of places to try regrafting x. Because of the way we chose x, we may assume there is a solution T^* and $X_i \subseteq \mathcal{X}$ such that $|X_i| \leq d$, $T_i - X_i = T^* - X_i$ and $x \notin X_i$. Thus we may treat T_i as a "guide" on where to regraft x. Due to the differences between T_1, T_i and T^*, this guide does not give us an exact location in T_1 to regraft x. Nevertheless, we can show that the number of candidate locations to regraft x can be bounded by $O(d)$. Thus, in total we have $O(d^3)$ branches at each step in our search tree of depth d, and therefore have to consider $O((O(3^d))^d) = O(c^d d^{3d})$ subproblems.

Theorem 6. *AST-LR-d can be solved in time $O(c^d d^{3d}(n^3 + tn \log n))$, where c is a constant not depending on d or n.*

5 Conclusion

To conclude, we introduced a new supertree/consensus problem, based on a simple combinatorial operator acting on trees, the Leaf-Removal. We showed that, although this supertree problem is NP-hard, it admits interesting tractability results, that compare well with existing algorithms. Future research should explore if various simple combinatorial operators, that individually define relatively tractable supertree problems (for example LR and EC) can be combined into a unified supertree problem while maintaining approximability and fixed-parameter tractability.

Acknowledgement: MJ was partially supported by Labex NUMEV (ANR-10-LABX-20) and Vidi grant 639.072.602 from The Netherlands Organization for Scientific Research (NWO). CC was supported by NSERC Discovery Grant 249834. CS was partially supported by the French Agence Nationale de la Recherche Investissements dAvenir/Bioinformatique (ANR-10-BINF-01-01, ANR-10- BINF-01-02, Ancestrome). ML was supported by NSERC PDF Grant.

References

[1] A. Amir and D. Keselman. Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms. *SIAM J. Comput.*, 26:1656–1669, 1997.

[2] D. Bryant. *Building trees, hunting for trees, and comparing trees.* PhD thesis, Bryant University, 1997.

[3] D. Bryant, A. McKenzie, and M. Steel. The size of a maximum agreement subtree for random binary trees. *Dimacs Series in Discrete Mathematics and Theoretical Computer Science*, 61:55–66, 2003.

[4] J. Byrka, S. Guillemot, and J. Jansson. New results on optimizing rooted triplets consistency. *Discrete Appl. Math.*, 158:1136–1147, 2010.

[5] R. Cole, M. Farach-Colton, R. Hariharan, T. M. Przytycka, and M. Thorup. An $O(n\log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM J. Comput.*, 30:1385–1404, 2000.

[6] Y. Deng and D. Fernández-Baca. Fast Compatibility Testing for Rooted Phylogenetic Trees. In *Combinatorial Pattern Matching 2016*, volume 54 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 12:1–12:12, 2016.

[7] D. Fernández-Baca, S. Guillemot, B. Shutters, and S. Vakati. Fixed-parameter algorithms for finding agreement supertrees. *SIAM J. Comput.*, 44:384–410, 2015.

[8] S. Guillemot and M. Mnich. Kernel and fast algorithm for dense triplet inconsistency. *Theoret. Comput. Sci.*, 494:134–143, 2013.

[9] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology.* Cambridge university press, 1997.

[10] M. Hellmuth, N. Wieseke, M. Lechner, H.-P. Lenhof, M. Middendorf, and P. F. Stadler. Phylogenomics with paralogs. *Proc. Natl. Acad. Sci. USA*, 112:2058–2063, 2015.

[11] E. D. Jarvis et al. Whole-genome analyses resolve early branches in the tree of life of modern birds. *Science*, 346:1320–1331, 2014.

[12] C. Scornavacca and N. Galtier. Incomplete lineage sorting in mammalian phylogenomics. *Sys. Biol.*, 66:112–120, 2017.

[13] C. Scornavacca, E. Jacox, and G. J. Szollösi. Joint amalgamation of most parsimonious reconciled gene trees. *Bioinformatics*, 31:841–848, 2015.

[14] G. J. Szollösi, B. Boussau, S. S. Abby, E. Tannier, and V. Daubin. Phylogenetic modeling of lateral gene transfer reconstructs the pattern and relative timing of speciations. *Proc. Natl. Acad. Sci. USA*, 109:17513–17518, 2012.

[15] P. Vachaspati and T. Warnow. FastRFS: fast and accurate robinson-foulds supertrees using constrained exact optimization. *Bioinformatics*, 33:631–639, 2017.

[16] C. Whidden, N. Zeh, and R. G. Beiko. Supertrees based on the subtree prune-and-regraft distance. *Sys. Biol.*, 63:566–581, 2014.

A Omitted proofs

Here we give proofs for several results whose proofs were omitted in the main paper. Note that the proof of Theorem 6 is deferred to its own section.

Lemma 1 *(restated). Let $\mathcal{T} = \{T_1, \ldots, T_t\}$ be a set of trees on the same label set $\mathcal{X}$. Then, given a supertree T such that $v := \sum_{T_i \in \mathcal{T}} d_{LR}(T, T_i)$, one can compute in time $O(tn \log n)$ a leaf-disagreement $\mathcal{C}$ of size at most v, where $n = |\mathcal{X}|$. Conversely, given a leaf-disagreement $\mathcal{C}$ for $\mathcal{T}$ of size v, one can compute in time $O(tn \log^2(tn))$ a supertree T such that $\sum_{T_i \in \mathcal{T}} d_{LR}(T, T_i) \leq v$.*

Proof. In the first direction, for each $T_i \in \mathcal{T}$, there is a set $X_i \subseteq \mathcal{X}$ of size $d_{LR}(T, T_i)$ such that $T_i - X_i = T - X_i$. Moreover, X_i can be found in time $O(n \log n)$. Thus $(X_1, \ldots, X_t)$ is a leaf-disagreement of the desired size and can be found in time $O(tn \log n)$. Conversely, let $\mathcal{C} = (X_1, \ldots, X_t)$ be a leaf-disagreement of size v. As $\mathcal{T}' = \{T_1 - X_1, \ldots, T_t - X_t\}$ is compatible, there is a tree T that displays $\mathcal{T}'$, and it is easy to see that the sum of distances between T and $\mathcal{T}'$ is at most the size of $\mathcal{C}$. As for the complexity, it is shown in [6] how to compute in time $O(tn \log^2(tn))$, given a set of trees $\mathcal{T}'$, a tree T displaying $\mathcal{T}'$ if one exists. □ □

We next consider the case where $\mathcal{T}$ consists only of two trees.

Lemma 2. *Let T_1, T_2 be two trees on the same label set $\mathcal{X}$. Then $AST_{LR}(T_1, T_2) = d_{LR}(T_1, T_2)$. Moreover, every optimal leaf-disagreement $\mathcal{C} = (\mathcal{X}_1', \mathcal{X}_2')$ for T_1 and T_2 can be obtained in the following manner: for every label-disagreement $\mathcal{X}'$ of size $d_{LR}(T_1, T_2)$, partition $\mathcal{X}'$ into $\mathcal{X}_1', \mathcal{X}_2'$.*

Proof. Let $\mathcal{X}' \subset \mathcal{X}$ such that $|\mathcal{X}'| = d_{LR}(T_1, T_2)$ and $T_1 - \mathcal{X}' = T_2 - \mathcal{X}'$. Then clearly, for any bipartition $(\mathcal{X}_1', \mathcal{X}_2')$ of $\mathcal{X}'$, $T_1' := T_1 - \mathcal{X}_1'$ and $T_2' := T_2 - \mathcal{X}_2'$ are compatible, since the leaves that T_1' and T_2' have in common yield the same subtree, and leaves that appear in only one tree cannot create incompatibility. In particular, $AST_{LR}(T_1, T_2) \leq d_{LR}(T_1, T_2)$.

Conversely, let $\mathcal{C} = (\mathcal{X}_1', \mathcal{X}_2')$ be a minimum leaf-disagreement. We have $\mathcal{X}_1' \cap \mathcal{X}_2' = \emptyset$, for if there is some $\ell \in \mathcal{X}_1' \cap \mathcal{X}_2'$, then ℓ could be reinserted into one of the two trees without creating incompatibility. Thus $\mathcal{C}$ is a bipartition of $\mathcal{X}' = \mathcal{X}_1' \cup \mathcal{X}_2'$. Moreover, we must have $T_1 - \mathcal{X}' = T_2 - \mathcal{X}'$, implying $|\mathcal{X}'| \geq d_{LR}(T_1, T_2)$. Combined with the above inequality, $|\mathcal{X}'| = d_{LR}(T_1, T_2)$, and the Lemma follows. □

It follows from Lemma 2 that any optimal label-disagreement $\mathcal{X}'$ can be turned into an optimal leaf-disagreement, which is convenient as $\mathcal{X}'$ can be found in polynomial time. We will make heavy use of this property later on.

Note that the same type of equivalence does not hold when 3 or more trees are given, *i.e.* computing a MAST of three trees does not necessarily yield a leaf-disagreement of minimum size. Consider for example the instance $\mathcal{T} = \{T_1, T_2, T_3\}$ in Figure 1. An optimal leaf-disagreement for $\mathcal{T}$ has size 2 and consists of any pair of distinct leaves. On the other hand, an optimal leaf-disagreement for $\mathcal{T}$ has size 3, and moreover each leaf corresponds to a different label.

Theorem 1 *(restated). The* `AST-LR` *and* `LR-Consensus` *problems are NP-hard.*

Proof. We begin by restating the reduction from `MinRTI` to `AST-LR`.

Let $\mathcal{R} = \{R_1, \ldots, R_t\}$ be an instance of `MinRTI`, with the label set $L := \bigcup_{i=1}^{t} \mathcal{X}(R_i)$. For a given integer m, we construct an `AST-LR` instance $\mathcal{T} = \{T_1, \ldots, T_t\}$ which is such that $MINRTI(\mathcal{R}) \leq m$ if and only if $AST_{LR}(\mathcal{T}) \leq t(|L| - 3) + m$.

We first construct a tree Z with additional labels which will serve as our main gadget. Let $\{L_i\}_{1 \leq i \leq t}$ be a collection of t new label sets, each of size $(|L|t)^{10}$, all disjoint from each other and all disjoint from L. Each tree in our `AST-LR` instance will be on label set $\mathcal{X} = L \cup L_1 \cup \ldots \cup L_t$. For each $i \in [t]$, let X_i be any tree with label set L_i. Obtain Z by taking any tree on t leaves $l_1, \ldots, l_t$, then replacing each leaf l_i by the X_i tree (*i.e.* l_i is replaced by $r(X_i)$). Denote by $r_Z(X_i)$ the root of the X_i subtree in Z.

Then for each $i \in [t]$, we construct T_i from R_i as follows. Let $L' = L \setminus \mathcal{X}(R_i)$ be the set of labels not appearing in R_i, noting that $|L'| = |L| - 3$. Let $T_{L'}$ be any tree with label set L', and obtain the tree Z_i by grafting $T_{L'}$ on the edge between $r_Z(X_i)$ and its parent. Finally, T_i is obtained by grafting R_i above Z_i. See Figure 2 for an example. Note that each tree T_i has label set $\mathcal{X}$ as desired. Also, it is not difficult to see that this reduction can be carried out in polynomial time.

We now show that $MINRTI(\mathcal{R}) \leq m$ if and only if $AST_{LR}(\mathcal{T}) \leq t(|L| - 3) + m$.

($\Rightarrow$) Let $\mathcal{R}' \subset \mathcal{R}$ such that $|\mathcal{R}'| \leq m$ and $\mathcal{R}^* := \mathcal{R} \setminus \mathcal{R}'$ is compatible, and let $T(\mathcal{R}^*)$ be a tree displaying $\mathcal{R}^*$. Note that $|\mathcal{R}^*| \geq t - m$. We obtain a `AST-LR` solution by first deleting, in each $T_i \in \mathcal{T}$, all the leaves labeled by $L \setminus \mathcal{X}(R_i)$ (thus T_i becomes the tree obtained by grafting R_i above Z). Then for each deleted triplet $R_i \in \mathcal{R}'$, we remove any single leaf of T_i labeled by some element in $\mathcal{X}(R_i)$. In this manner, no more than $t(|L| - 3) + m$ leaves get deleted. Moreover, grafting $T(\mathcal{R}^*)$ above Z yields a tree displaying the modified set of trees, showing that they are compatible.

($\Leftarrow$) We first argue that if $\mathcal{T}$ admits a leaf-disagreement $\mathcal{C} = (\mathcal{X}_1, \ldots, \mathcal{X}_t)$ of size at most $t(|L|-3)+m$, then there is a better or equal solution that removes, in each T_i, all the leaves labeled by $L \setminus \mathcal{X}(R_i)$ (*i.e.* those grafted in the Z_i tree). For each $i \in [t]$, let $T'_i = T_i - \mathcal{X}_i$, and denote $\mathcal{T}' = \{T'_1, \ldots, T'_t\}$. Suppose that there is some $i \in [t]$ and some $\ell \in L \setminus \mathcal{X}(R_i)$ such that $\ell \in \mathcal{X}(T'_i)$.

We claim that $\ell \notin \mathcal{X}(T'_j)$ for every $i \neq j \in [t]$. Suppose otherwise that $\ell \in \mathcal{X}(T'_j)$ for some $j \neq i$. Consider first the case where $\ell \notin \mathcal{X}(R_j)$. Note that by the construction of Z_i and Z_j, for every $x_i \in \mathcal{X}(X_i) \cap \mathcal{X}(T'_i) \cap \mathcal{X}(T'_j)$ and every $x_j \in \mathcal{X}(X_j) \cap \mathcal{X}(T'_i) \cap \mathcal{X}(T'_j)$, T'_i contains the $\ell x_i | x_j$ triplet whereas T'_j contains the $\ell x_j | x_i$ triplet. Since these triplets are conflicting, no supertree can contain both and so no such x_i, x_j pair can exist, as we are assuming that a supertree for T'_i and T'_j exists. This implies that one of $\mathcal{X}(X_i) \cap \mathcal{X}(T'_i) \cap \mathcal{X}(T'_j)$ or $\mathcal{X}(X_j) \cap \mathcal{X}(T'_i) \cap \mathcal{X}(T'_j)$ must be empty. Suppose without loss of generality that the former is empty. Then each $x_i \in X_i$ must have been deleted in at least one of T_i or T_j. As $|\mathcal{X}(X_i)| = (|L|t)^{10} > t(|L|-3)+m$, this contradicts the size of the solution $\mathcal{C}$. In the second case, we have $\ell \in \mathcal{X}(R_j)$. But this time, if there are $x_i \in \mathcal{X}(X_i) \cap \mathcal{X}(T'_i) \cap \mathcal{X}(T'_j)$ and $x_j \in \mathcal{X}(X_j) \cap \mathcal{X}(T'_i) \cap \mathcal{X}(T'_j)$, then T'_j contains the $x_i x_j|\ell$ triplet, again conflicting with the $\ell x_i|x_j$ triplet found in T_i. As before, we run into a contradiction since too many X_i or X_j leaves need to be deleted. This proves our claim.

We thus assume that ℓ only appears in T'_i. Let $R_j \in \mathcal{R}$ such that $\ell \in \mathcal{X}(R_j)$, noting that ℓ does not appear in T'_j. Consider the solution $\mathcal{T}''$ obtained from $\mathcal{T}'$ by removing ℓ from T'_i, and placing it back in T'_j where it originally was in T_j. Formally this is achieved by replacing, in the leaf-disagreement $\mathcal{C}$, $\mathcal{X}_i$ by $\mathcal{X}_i \cup \{\ell\}$ and $\mathcal{X}_j$ by $\mathcal{X}_j \setminus \{\ell\}$. Since ℓ still appears only in one tree, no conflict is created and we obtain another solution of equal size. By repeating this process for every such leaf ℓ, we obtain a solution in which every leaf labeled by $L \setminus \mathcal{X}(R_i)$ is removed from T'_i. We now assume that the solution $\mathcal{T}'$ has this form.

Consider the subset $\mathcal{R}' = \{R_i \in \mathcal{R} : |\mathcal{X}(T'_i) \cap \mathcal{X}(R_i)| < 3\}$, that is those triplets R_i for which the corresponding tree T_i had a leaf removed outside of the Z_i tree. By the form of the $\mathcal{T}'$ solution, at least $t(|L|-3)$ removals are done in the Z_i trees, and as only m removals remain, $\mathcal{R}'$ has size at most m. We show that $\mathcal{R} \setminus \mathcal{R}'$ is a compatible set of triplets. Since $\mathcal{T}'$ is compatible, there is a tree T that displays each $T'_i \in \mathcal{T}'$, and since each triplet of $\mathcal{R} \setminus \mathcal{R}'$ belongs to some T'_i, T also displays $\mathcal{R} \setminus \mathcal{R}'$. This concludes the proof. □ □

Theorem 2 *(restated). The following is a factor* 2 *approximation algorithm for* `LR-Consensus`*: return the tree $T \in \mathcal{T}$ that minimizes $\sum_{T_i \in \mathcal{T}} d_{LR}(T, T_i)$.*

Proof. Let T^* be an optimal solution for `LR-Consensus`, *i.e.* T^* is a tree minimizing $\sum_{T_i \in \mathcal{T}} d_{LR}(T_i, T^*)$, and let T be chosen as described in the theorem statement. Moreover let T' be the tree of $\mathcal{T}$ minimizing $d_{LR}(T', T^*)$. By the triangle inequality,

$$\sum_{T_i \in \mathcal{T}} d_{LR}(T', T_i) \leq \sum_{T_i \in \mathcal{T}} (d_{LR}(T', T^*) + d_{LR}(T^*, T_i)) \leq 2 \sum_{T_i \in \mathcal{T}} d_{LR}(T^*, T_i)$$

where the last inequality is due to the fact that $d_{LR}(T', T^*) \leq d_{LR}(T^*, T_i)$ for all i, by our choice of T'. Our choice of T implies $\sum_{T_i \in \mathcal{T}} d_{LR}(T, T_i) \leq \sum_{T_i \in \mathcal{T}} d_{LR}(T', T_i) \leq 2 \sum_{T_i \in \mathcal{T}} d_{LR}(T_i, T^*)$. □ □

Corollary 1 *(restated). There are instances of* `AST-LR` *in which $\Omega(t(n - \sqrt{n}))$ leaves need to be deleted.*

Proof. Let $\mathcal{T} = \{T_1, \ldots, T_t\}$ be a random set of t trees chosen uniformly and independently. For large enough n, the expected sum of distances between each pair of trees is

$$\mathbb{E}\left[\sum_{1\leq i<j\leq t} d_{LR}(T_i,T_j)\right] = \sum_{1\leq i<j\leq t} \mathbb{E}[d_{LR}(T_i,T_j)] \geq \binom{t}{2}(n-c\sqrt{n})$$

for some constant c, by Theorem 3. Let $S := \min_T \sum_{i=1}^{t} d_{LR}(T,T_i)$ be the random variable corresponding to the minimum sum of distances. By Theorem 2, there is a tree $T' \in \mathcal{T}$ such that $\sum_{i=1}^{t} d_{LR}(T',T_i) \leq 2S$. We have

$$\begin{aligned}\sum_{1\leq i<j\leq t} d_{LR}(T_i,T_j) &\leq \sum_{1\leq i<j\leq t} d_{LR}(T_i,T') + d_{LR}(T',T_j)\\ &= (t-1)\sum_{i=1}^{t} d_{LR}(T_i,T')\\ &\leq (t-1)2S\end{aligned}$$

Since, in general for two random variables X and Y, always having $X \leq Y$ implies $\mathbb{E}[X] \leq \mathbb{E}[Y]$, we get

$$\binom{t}{2}(n-c\sqrt{n}) \leq \mathbb{E}\left[\sum_{1\leq i<j\leq t} d_{LR}(T_i,T_j)\right] \leq \mathbb{E}[(t-1)2S] = 2(t-1)\mathbb{E}[S]$$

yielding $\mathbb{E}[S] \geq t/4(n-c\sqrt{n}) = \Omega(t(n-\sqrt{n}))$, and so there must exist an instance $\mathcal{T}$ satisfying the statement. □

Theorem 5 *(restated).* `AST-LR` *can be solved in time* $O(12^q tn^3)$.

Proof. We provide an algorithm, Algorithm 1, for `AST-LR`, and prove its correctness and complexity.

We first argue that the algorithm is correct. First observe that the algorithm only returns TRUE when a conflict-free set of triplets is attained without deleting more than q leaves, and so there are no false-positives. Moreover, it is not hard to see that the first phase of the algorithm tries every possible way of obtaining a full set of triplets from $tr(\mathcal{T})$ using at most q leaf removals. Indeed, for every set of 3 labels a, b, c that are present in a direct conflict, the algorithm branches into the 3 ways of locking $ab|c$, $ac|b$ or $bc|a$ and for each tree T_i disagreeing with the chosen triplet, all three ways of removing a leaf to agree with the chosen triplet are tested. In a similar manner, for each dense set D of triplets that is attained, each way of freeing D from conflicting 4-sets is evaluated (not only in the ways of choosing a triplet to resolve the conflict, but also in the ways of removing leaves from the trees of $\mathcal{T}$ so that they agree with the locked triplets). It follows that if $tr(\mathcal{T})$ can be made compatible by deleting at most q leaves, then some leaf in the branch tree created by the algorithm will return TRUE, and so there are no false-negatives.

Algorithm 1 Recursive `AST-LR` FPT algorithm.

procedure MASTRL($\mathcal{T}, q, phase, F$)
$\mathcal{T}$ is the set of input trees, q is the maximum number of leaves to delete, $phase$ is the current phase number (initially 1), F is the set of locked triplets so far
  if $q < 0$ or F contains conflicting triplets **then**
    Return FALSE
  else if there is $ab|c \in F$ and a tree $T_i \in \mathcal{T}$ such that $ac|b \in tr(T_i)$ or $bc|a \in T_i$ **then**
    Branching: If one of the following calls returns True
      MASTRL($(\mathcal{T} \setminus \{T_i\}) \cup \{T_i - \{a\}\}, q-1, phase, F$) //remove a from T_i
      MASTRL($(\mathcal{T} \setminus \{T_i\}) \cup \{T_i - \{b\}\}, q-1, phase, F$) //remove b from T_i
      MASTRL($(\mathcal{T} \setminus \{T_i\}) \cup \{T_i - \{c\}\}, q-1, phase, F$) //remove c from T_i
    then Return True, otherwise Return False
  else if $phase = 1$ **then**
    if there are $a, b, c \in \mathcal{X}$ such that at least 2 of $ab|c, ac|b$ or $bc|a$ appear in $\mathcal{T}$ **then**
      Branching: If one of the following calls returns True
        MASTRL($\mathcal{T}, q, phase, F \cup \{ab|c\}$)
        MASTRL($\mathcal{T}, q, phase, F \cup \{ac|b\}$)
        MASTRL($\mathcal{T}, q, phase, F \cup \{bc|a\}$)
      then Return True, otherwise Return False
    else
      Return MASTRL($\mathcal{T}, q, 2, F$) //enter phase 2
  else if $phase = 2$ **then**
    if there is a conflicting set $\{a, b, c, d\}$ in $tr(\mathcal{T}) \cup F$ **then**
      Branching: If one of the following calls returns True
        MASTRL($\mathcal{T}, q, phase, F \cup \{ac|b\}$)
        MASTRL($\mathcal{T}, q, phase, F \cup \{bc|a\}$)
        MASTRL($\mathcal{T}, q, phase, F \cup \{bd|c\}$)
        MASTRL($\mathcal{T}, q, phase, F \cup \{ab|d\}$)
      then Return True, otherwise Return False
    else
      Return True //There are no conflicts $\Rightarrow tr(\mathcal{T}) \cup F$ is compatible

As for the complexity, when the algorithm enters the 'else if' block of line 4, it branches into 3 cases that decrement q. When it enters the 'if' block of line 11, it branches into 3 cases but q is not decremented. However each of these 3 recursive calls immediately leads to the 'else if' block on line 4, and so this case can be seen as branching into 9 cases. Similarly, when the algorithm enters the 'if' block of line 19, it branches into 4 cases, each of which leads to the 3 subcases following line 4. Thus 12 cases are considered. Therefore, the branching tree created by the algorithm has degree at most 12 and depth at most q, and so at most 12^q cases are considered. Finally, each call to the algorithm requires time $O(tn^3)$ since this is the time required to identify conflicting sets of triplets within the t trees. □ □

B Leaf Prune-and-Regraft Moves

Here we introduce the notion of leaf prune-and-regraft (LPR) moves, which will be used in the proof of Theorem 6, and which may be of independent interest. In an LPR move, we prune a leaf from a tree and then regraft it another location (formal definitions below). LPR moves provide an alternate way of characterizing the distance function d_{LR} - indeed, we will show that $d_{LR}(T_1, T_2) \leq k$ if and only if there is a sequence of at most k LPR moves transforming T_1 into T_2.

Definition 1. *Let T be a tree on label set $\mathcal{X}$. A LPR move on T is a pair (ℓ, e) where $\ell \in \mathcal{X}$ and $e \in \{E(T - \{\ell\}), \perp\}$. Applying (ℓ, e) consists in grafting ℓ on the e edge of $T - \{\ell\}$ if $e \neq \perp$, and above the root of $T - \{\ell\}$ if $e = \perp$.*

An LPR sequence $L = ((\ell_1, e_1), \dots, (\ell_k, e_k))$ *is an ordered tuple of LPR moves, where for each $i \in [k]$, (ℓ_i, e_i) is an LPR move on the tree obtained after applying the first $i-1$ LPR moves of L. We may write $L = (\ell_1, \dots, \ell_k)$ if the location at which the grafting takes place does not need to be specified. We say that L turns T_1 into T_2 if, by applying each LPR move of L in order on T_1, we obtain T_2.*

See Figure 4 for an example of an LPR sequence.

In the following statements, we assume that T_1 and T_2 are two trees on label set $\mathcal{X}$. We exhibit an equivalence between leaf removals and LPR sequences, then show that the order of LPR moves in a sequence do not matter in terms of turning one tree into another - in particular any leaf can be displaced first.

Lemma 3. *There is a subset $X \subseteq \mathcal{X}$ such that $T_1 - X = T_2 - X$ if and only if there is an LPR sequence $(x_1, x_2, \dots, x_k)$ turning T_1 into T_2 such that $X = \{x_1, \dots, x_k\}$.*

Proof. If $T_1 = T_2$ then the proof is trivial, so we will assume this is not the case. We prove the lemma by induction on $|X|$.

For the base case, suppose that $X = \{x\}$. If $T_1 - X = T_2 - X$, then let $T_m = T_1 - X = T_2 - X$. We find an LPR move (x, e) with $e \in E(T_m) \cup \{\perp\}$ turning T_1 into T_2. Observe that T_2 can be obtained by grafting x on T_m, either

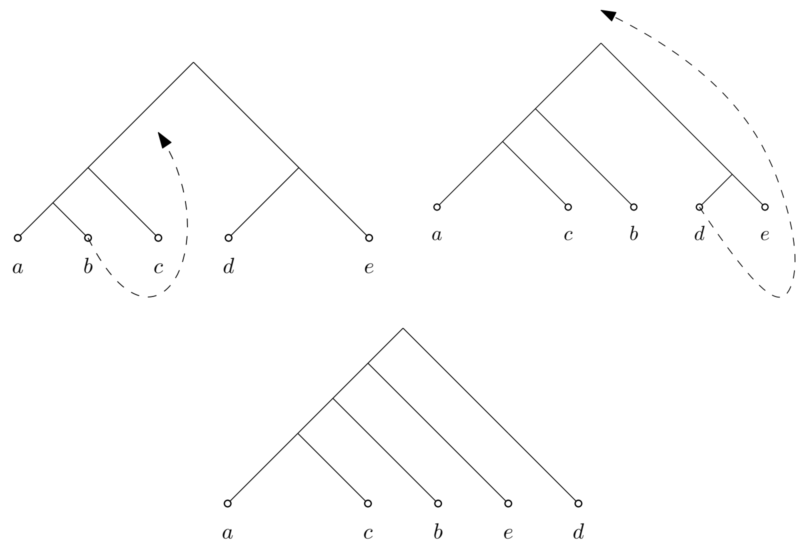

Figure 3: Sequence of trees showing the LPR sequence $L = ((b, f), (d, \perp))$, where f is the edge between the root and the least common ancestor of a and c in the first tree.

on an edge uv, in which case we set $e = uv$, or above the root, in which case we set $e = \perp$. Since $T_m = T_1 - \{x\}$, it follows that (x, e) is an LPR move turning T_1 into T_2. In the other direction, assume there is an LPR move (x, e) turning T_1 into T_2. Observe that for any tree T' derived from T_1 by an LPR move using x, $T' - \{x\} = T_1 - \{x\}$. In particular, $T_2 - \{x\} = T_1 - \{x\}$ and we are done.

For the induction step, assume that $|X| > 1$ and that the claim holds for any X' such that $|X'| < |X|$. If $T_1 - X = T_2 - X$, then define $T_m = T_1 - X$, and let x be an arbitrary element of X. We will first construct a tree T_1' such that $T_1 - \{x\} = T_1' - \{x\}$ and $T_1' - (X \setminus \{x\}) = T_2 - (X \setminus \{x\})$.

Observe that $T_2 - (X \setminus \{x\})$ can be obtained by grafting x in T_m. Let $e = uv$ if this grafting takes place on an edge of T_m with v being the child of u, or $e = \perp$ if x is grafted above T_m, and in this case let $v = r(T_m)$. Let $v' = v_{T_1 - \{x\}}$ be the node in $T_1 - \{x\}$ corresponding to v.

Let T_1' be derived from $T_1 - \{x\}$ by grafting x onto the edge between v' and its parent if v' is non-root, and grafting above v' otherwise. It is clear that $T_1 - \{x\} = T_1' - \{x\}$. Furthermore, by our choice of v' we have that $T_1' - (X \setminus \{x\}) = T_2 - (X \setminus \{x\})$.

Now that we have $T_1 - \{x\} = T_1' - \{x\}$ and $T_1' - (X \setminus \{x\}) = T_2 - (X \setminus \{x\})$, by the inductive hypothesis there is an LPR sequence turning T_1 into T_1' consisting of a single move (x, e), and an LPR sequence $(x_1, x_2, \ldots, x_{k-1})$ turning T_1' into T_2 such that $\{x_1, \ldots, x_{k'}\} = (X \setminus \{x\})$. Then by concatenating these two sequences, we have an LPR sequence $(x_1, x_2, \ldots, x_k)$ turning T_1 into T_2 such that $X = \{x_1, \ldots, x_k\}$.

For the converse, suppose that there is an LPR sequence $(x_1, x_2, \ldots, x_k)$ turning T_1 into T_2 such that $X = \{x_1, \ldots, x_k\}$. Let T_1' be the tree derived from T_1 by applying the first move in this sequence. That is, there is an LPR move (x_1, e) turning T_1 into T_1', and there is an LPR sequence $(x_2, \ldots, x_k)$ turning T_1' into T_2. Then by the inductive hypothesis $T_1 - \{x_1\} = T_1' - \{x_1\}$ and $T_1' - \{x_2, \ldots, x_k\} = T_2 - \{x_2, \ldots, x_k\}$. Thus, $T_1 - X = T_1' - X = T_2 - X$, as required. □ □

Lemma 4. *If there is an LPR sequence $L = (x_1, \ldots, x_k)$ turning T_1 into T_2, then for any $i \in [k]$, there is an LPR sequence $L' = (x_1', \ldots, x_k')$ turning T_1 into T_2 such that $x_1' = x_i$ and $\{x_1, \ldots, x_k\} = \{x_1', \ldots, x_k'\}$.*

Proof. Consider again the proof that if $T_1 - X = T_2 - X$ then there is an LPR sequence $(x_1, \ldots x_k)$ turning T_1 into T_2 such that $X = \{x_1, \ldots, x_k\}$ (given in the proof of Lemma 3). When $|X| > 1$, we construct this sequence by concatenating the LPR move (x, e) with an LPR sequence of length $|X| - 1$, where x is an arbitrary element of X. As we could have chosen any element of X to be x, we have the following: If $T_1 - X = T_2 - X$ then for each $x \in X$, there is an LPR sequence $(x_1, \ldots, x_k)$ turning T_1 into T_2 such that $X = \{x_1, \ldots, x_k\}$ and $x_1 = x$.

Thus our proof is as follows: Given an LPR sequence $L = (x_1, \ldots, x_k)$ turning T_1 into T_2 and some $i \in [k]$, Lemma 3 implies that $T_1 - \{x_1, \ldots, x_k\} = T_2 - \{x_1, \ldots, x_k\}$. By the observation above, this implies that there is an LPR sequence $(x_1', \ldots, x_k')$ turning T_1 into T_2 such that $\{x_1, \ldots, x_k\} = \{x_1', \ldots, x_k'\}$ and $x_1' = x$. □ □

C Proof of Theorem 6

This section makes use of the concept of LPR moves, which are introduced in the previous section. As discussed in the main paper, we employ a branch-and-bound style algorithm, in which at each step we alter a candidate solution by pruning and regrafting a leaf. That is, we apply an LPR move.

The technically challenging part is bound the number of possible LPR moves to try. To do this, we will prove Lemma 7, which provides a bound on the number of leaves to consider, and Lemma 10, which bounds the number of places a leaf may be regrafted to.

Denote by $tr(T)$ the set of rooted triplets of a tree T. Two triplets $R_1 \in tr(T_1)$ and $R_2 \in tr(T_2)$ are *conflicting* if $R_1 = ab|c$ and $R_2 \in \{ac|b, bc|a\}$. We denote by $conf(T_1, T_2)$ the set of triplets of T_1 for which there is a conflicting triplet in T_2. That is, $conf(T_1, T_2) = \{ab|c \in tr(T_1) : ac|b \in tr(T_2) \text{ or } bc|a \in tr(T_2)\}$. Finally we denote by $confset(T_1, T_2) = \{\{a, b, c\} : ab|c \in conf(T_1, T_2)\}$, *i.e.* the collection of 3-label sets formed by conflicting triplets. Given a collection $C = \{S_1, \ldots, S_{|C|}\}$ of sets, a *hitting set* of C is a set S such that $S \cap S_i \neq \emptyset$ for each $S_i \in C$.

Lemma 5. *Let $X \subseteq \mathcal{X}$. Then $T_1 - X = T_2 - X$ if and only if X is a hitting set of $confset(T_1, T_2)$.*

Proof. It is known that for two rooted trees T_1, T_2 that are leaf-labelled and binary, $T_1 = T_2$ if and only if $tr(T_1) = tr(T_2)$ [2]. Note also that $tr(T - X) = \{ab|c \in tr(T_1) : X \cap \{a, b, c\} = \emptyset\}$ for any tree T and $X \subseteq \mathcal{X}$.

Therefore we have that $T_1 - X = T_2 - X$ if and only if $tr(T_1 - X) = tr(T_2 - X)$, which holds if and only if for every $a, b, c \in \mathcal{X} \setminus X$, if $ab|c \in tr(T_1)$ then $ab|c \in tr(T_2)$. This in turn occurs if and only if X is a hitting set for $confset(T_1, T_2)$. □ □

In what follows, we call $X \subseteq \mathcal{X}$ a *minimal disagreement* between T_1 and T_2 if $T_1 - X = T_2 - X$ and for any $X' \subset X$, $T_1 - X' \neq T_2 - X'$.

Lemma 6. *Suppose that $d < d_{LR}(T_1, T_2) \leq d' + d$ with $d' \leq d$, and that there is a tree T^* and subsets $X_1, X_2 \subseteq \mathcal{X}$ such that $T_1 - X_1 = T^* - X_1$, $T_2 - X_2 = T^* - X_2$ and $|X_1| \leq d', |X_2| \leq d$. Then, there is a minimal disagreement X between T_1 and T_2 of size at most $d + d'$ and $x \in X$ such that $x \in X_1 \setminus X_2$.*

Proof. Let $X' = X_1 \cup X_2$. Observe that $T_1 - X' = T^* - X' = T_2 - X'$ and $|X'| \leq d + d'$. Letting X be the minimal subset of X' such that $T_1 - X = T_2 - X$, we have that X is a minimal disagreement between T_1 and T_2 and $|X| \leq d + d'$. Furthermore as $|X| \geq d_{LR}(T_1, T_2) > d$, $|X \setminus X_2| > 0$, and so there is some $x \in X$ with $x \in X \setminus X_2 = X_1 \setminus X_2$. □ □

We are now ready to state and prove Lemma 7.

Lemma 7. *Suppose that $d_{LR}(T_1, T_2) \leq d$ for some integer d. Then, there is some $S \subseteq \mathcal{X}$ such that $|S| \leq 8d^2$, and for any minimal disagreement X between T_1 and T_2 with $|X| \leq d$, $X \subseteq S$. Moreover S can be found in time $O(n^2)$.*

We will call S as described in Lemma 7 a *d-disagreement kernel* between T_1 and T_2. Thus Lemma 6 essentially states that if T_1 isn't a solution and $d_{LR}(T_1, T_2) > d$, then for T_1 to get closer to a solution, there is a leaf x in the $d_{LR}(T_1, T_2)$-disagreement kernel that needs to be removed and regrafted in a location that T_2 'agrees with'. Lemma 7 in turn gives us a set S of size at most $8d^2$ such that the desired x must be contained in S.

Proof. By Lemma 5, it is enough to find a set S such that S contains every minimal hitting set of $confset(T_1, T_2)$ of size at most d.

We construct S as follows.

Let X be a subset of $\mathcal{X}$ of size at most d such that $T_1 - X = T_2 - X$. As previously noted, this can found in time $O(n \log n)$ [5].

For notational convenience, for each $x \in X$ we let x_1, x_2 be two new labels, and set $X_1 = \{x_1 : x \in X\}$, $X_2 = \{x_2 : x \in X\}$. Thus, X_1, X_2 are disjoint "copies" of X. Let T_1' be derived from T_1 by replacing every label from X with the corresponding label in X_1, and similarly let T_2' be derived from T_2 by replacing every label from X with the corresponding label in X_2.

Let T_J be a tree with label set $(\mathcal{X} \setminus X) \cup X_1 \cup X_2$ such that $T_J - X_2 = T_1'$ and $T_J - X_1 = T_2'$. The tree T_J always exists and can be found in polynomial time. Intuitively, we can start from T_1', and graft the leaves of X_2 where T_2

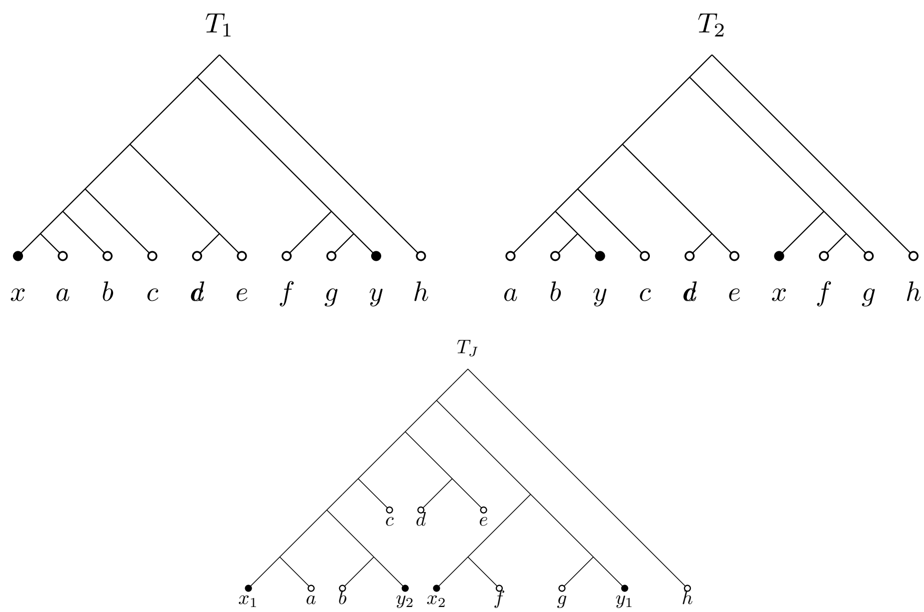

Figure 4: Construction of the tree T_J, given two trees T_1, T_2, with $X = \{x, y\}$ such that $T_1 - X = T_2 - X$.

"wants" them to be. See Figure 4 for an example. Algorithm 2 gives a method for constructing T_J, and takes $O(n^2)$ time.

In addition, let L be the set of all labels in $\mathcal{X} \setminus X$ that are descended in T_J from $\textsc{lca}_{T_J}(X_1 \cup X_2)$, and let $R = \mathcal{X} \setminus (L \cup X)$. Thus, L, X, R form a partition of $\mathcal{X}$, and L, X_1, X_2, R form a partition of the labels of T_J.

For the rest of the proof, we call $\{x, y, z\}$ a *conflict triple* if $\{x, y, z\} \in confset(T_1, T_2)$.

We first observe that no triple in $confset(T_1, T_2)$ contains a label in R. Indeed, consider a triple $\{x, y, z\}$. Any conflict triple must contain a label from X, so assume without loss of generality that $x \in X, z \in R$. If $x \in X, y \in L, z \in R$, then we have that T_J contains the triplets $x_1y|z, x_2y|z$, and so T_1 and T_2 both contain $xy|z$, and $\{x, y, z\}$ is not a conflict triple. Similarly if $x, y \in X, z \in R$, then T_J contains the triplets $x_1y_1|z, x_2y_2|z$, and again $\{x, y, z\}$ is not a conflict triple. If $x \in X$ and $y, z \in R$, then the triplet on $\{x_1, y, z\}$ in T_J depends only on the relative positions in T_J of y, z and $\textsc{lca}_{T_J}(X_1 \cup X_2)$. Thus we get the same triplet if we replace x_1 with x_2, and so $\{x, y, z\}$ is not a conflict triple.

This concludes the proof that no triple in $confset(T_1, T_2)$ contains a label in R. Having shown this, we may conclude that any minimal disagreement between T_1 and T_2 is disjoint from R, and so our returned set S only needs to contain labels in $L \cup X$.

Now consider the tree $T^* = T_J|_{X_1 \cup X_2}$, *i.e.* the subtree of T_J restricted to the labels in $X_1 \cup X_2$. Thus in the example of Figure 4, T^* is the subtree of T_J

Algorithm 2 Algorithm to construct "Join tree" of T_1', T_2'

procedure JOIN-TREES($T_1', T_2', L', X_1', X_2'$)
T_1' is a tree on $L' \cup X_1'$, T_2' is a tree on $L' \cup X_2'$, $T_1'|_{L'} = T_2'|_{L'}$. Output: A tree T_J on $L' \cup X_1' \cup X_2'$ such that $T_J|_{L' \cup X_1'} = T_1'$ and $T_J|_{L' \cup X_2'} = T_2'$
    if $L' \cup X_1' = \emptyset$ **then**
        Return T_2'
    else if $L' \cup X_2' = \emptyset$ **then**
        Return T_1'
    else if $X_1' \cup X_2' = \emptyset$ **then**
        Return T_1'
    Set $r_1 =$ root of T_1', u, v the children of r_1
    Set $r_2 =$ root of T_2', w, z the children of r_2
    Set $X_{1u} =$ descendants of u in X_1', $L_{1u} =$ descendants of u in L'
    Set $X_{1v} =$ descendants of v in X_1', $L_{1v} =$ descendants of v in L'
    Set $X_{2w} =$ descendants of w in X_2', $L_{2w} =$ descendants of w in L'
    Set $X_{2z} =$ descendants of z in X_2', $L_{2z} =$ descendants of z in L'
    if $L_{1u} = L_{2w}$ and $L_{1v} = L_{2z}$ **then**
        Set $T_{left} =$ JOIN-TREES($T_1'|_{L_{1u} \cup X_{1u}}, T_2'|_{L_{1u} \cup X_{2w}}, L_{1u}, X_{1u}, X_{2w}$)
        Set $T_{right} =$ JOIN-TREES($T_1'|_{L_{1v} \cup X_{1v}}, T_2'|_{L_{1v} \cup X_{2z}}, L_{1v}, X_{1v}, X_{2z}$)
    else if $L_{1u} = L_{2z}$ and $L_{1v} = L_{2w}$ **then**
        Set $T_{left} =$ JOIN-TREES($T_1'|_{L_{1u} \cup X_{1u}}, T_2'|_{L_{1u} \cup X_{2z}}, L_{1u}, X_{1u}, X_{2z}$)
        Set $T_{right} =$ JOIN-TREES($T_1'|_{L_{1v} \cup X_{1v}}, T_2'|_{L_{1v} \cup X_{2w}}, L_{1v}, X_{1v}, X_{2w}$)
    else if $L_{1u} =_{\emptyset}$ **then**
        Set $T_{left} =$ JOIN-TREES($T_1'|_{X_{1u}}, T_2'|_{\emptyset}, \emptyset, X_{1u}, \emptyset$)
        Set $T_{right} =$ JOIN-TREES($T_1'|_{L' \cup X_{1v}}, T_2', L', X_{1v}, X_2'$)
    else if $L_{1v} = \emptyset$ **then**
        Set $T_{left} =$ JOIN-TREES($T_1'|_{L' \cup X_{1u}}, T_2', L', X_{1u}, X_2'$)
        Set $T_{right} =$ JOIN-TREES($T_1'|_{X_{1v}}, T_2'|_{\emptyset}, \emptyset, X_{1v}, \emptyset$)
    else if $L_{2w} = \emptyset$ **then**
        Set $T_{left} =$ JOIN-TREES($T_1'|_{\emptyset}, T_2'|_{X_{2w}}, \emptyset, \emptyset, X_{2w}$)
        Set $T_{right} =$ JOIN-TREES($T_1', T_2'|_{L' \cup X_{2z}}, L', X_1', X_{2z}$)
    else if $L_{2z} = \emptyset$ **then**
        Set $T_{left} =$ JOIN-TREES($T_1', T_2'|_{L' \cup X_{2w}}, L', X_1', X_{2w}$)
        Set $T_{right} =$ JOIN-TREES($T_1'|_{\emptyset}, T_2'|_{X_{2z}}, \emptyset, \emptyset, X_{2z}$)
    ▷ If none of the above cases holds, then $T_1'|_{L'} \neq T_2'|_{L'}$, contradicting the requirements on the input
    Set $T_J =$ the tree on $L' \cup X_1' \cup X_2'$ whose root has T_{left} and T_{right} as children.
    Return T_J.

spanned by $\{x_1, x_2, y_1, y_2\}$. We will now use the edges of T^* to form a partition of L, as follows. For any edge uv in T^* with u the parent of v, let $s(uv)$ denote the set of labels $y \in \mathcal{X}$ such that y has an ancestor which is an internal node on the path from u to v in T_J, but y is not a descendant of v itself. For example in Figure 4, if u is the least common ancestor of x_1, y_1 and v is the least common ancestor of x_1, y_2, then uv is an edge in T^* and $s(uv) = \{c, d, e\}$.

Observe that $\{s(uv) : uv \in E(T^*)\}$ forms a partition of L. (Indeed, for any $l \in L$, let u be the minimal element in T^* on the path in T_J between l and $\text{LCA}_{T_J}(X_1 \cup X_2)$ (note that u exists as $\text{LCA}_{T_J}(X_1 \cup X_2)$ itself is in T^*). As u is in T^*, both of its children are on paths in T_J between u and a child of u in T^*. In particular, the child of u which is an ancestor of l is an internal node on the path between u and v in T_J, for some child v of u in T^*, and l is not descended from v by construction. It is clear by construction that all $s(uv)$ are disjoint.)

The main idea behind the construction of S is that we will add X to S, together with $O(d)$ labels from $s(uv)$ for each edge uv in T^*. As the number of edges in T^* is $2(|X_1 \cup X_2| - 1) = O(d)$, we have the required bound of $O(d^2)$ on $|S|$.

So now consider $s(uv)$ for some edge uv in T^*. In order to decide which labels to add to S, we need to further partition $s(uv)$. Let $u = u_0 u_1 \dots u_t = v$ be the path in T_J from u to v. For each $i \in [t-1]$ (note that this does not include $i = 0$), we call the set of labels descended from u_i but not u_{i+1} a *dangling clade*. Observe that the dangling clades form a partition of $s(uv)$. Thus in the example of Figure 4, if u is the least common ancestor of x_1, y_1 and v is the least common ancestor of x_1, y_2, then for the edge uv the dangling clades are $\{c\}$ and $\{d, e\}$.

We now make the following observations about the relation between $s(uv)$ and triples in $confset(T_1, T_2)$.

Observation 1: if $\{x, y, z\}$ is a conflict triple and $x \in s(uv), y, z \notin s(uv)$, then $\{x', y, z\}$ is also a conflict triple for any $x' \in s(uv)$. (The intuition behind this is that there are no labels appearing 'between' x and x' that are not in $s(uv)$.)

Observation 2: for any triple $\{x, y, z\}$ with $x, y \in s(uv)$, $\{x, y, z\}$ is a conflict triple if and only if x, y are in different dangling clades and $z \in X$ with z_i descended from v, z_{3-i} not descended from u_1 for some $i \in [2]$ (recall that $z_1 \in X_1$ and $z_2 \in X_2$). To prove one direction, it is easy to see that if the conditions hold, then T_i displays either $xz|y$ or $yz|x$ (depending on which dangling clade appears 'higher'), and T_{3-i} displays $xy|z$. For the converse, observe first that $z \in X$ as X is a hitting set for $confset(T_1, T_2)$ and $x, y \notin X$. Then if xy are in the same dangling clade, we have that both T_1 and T_2 display $xy|z$. So x, y must be in different dangling clades. Next observe that each of z_1, z_2 must either be descended from v or not descended from u_1, as otherwise v would not be the child of u in T^*. If z_1, z_2 are both descended from v or neither are descended from u_1, then T_1 and T_2 display the same triplet on $\{x, y, z\}$. So instead one must be descended from v and one not descended from u_1, as required.

Using Observations 1 and 2, we now prove the following:

Observation 3: for any minimal disagreement X' between T_1 and T_2, one of the following holds:

- $X' \cap s(uv) = \emptyset$;
- $s(uv) \subseteq X'$;
- $s(uv) \setminus X'$ forms a single dangling clade.

To see this, let X' be any minimal hitting set of $confset(T_1, T_2)$ with $s(uv) \cap X' \neq \emptyset$ and $s(uv) \setminus X' \neq \emptyset$. As X' is minimal, any $x \in s(uv) \cap X'$ must be in a conflict triple $\{x, y, z\}$ with $y, z \notin X'$. As X is a hitting set for $confset(T_1, T_2)$, at least one of y, z must be in X. If $y, z \notin s(uv)$, then by Observation 1 $\{x', y, z\}$ is also a conflict triple for any $x' \in s(uv) \setminus X'$. But this is a contradiction as $\{x', y, z\}$ has no elements in X'. Then one of y, z must also be in $s(uv)$. Suppose without loss of generality that $y \in s(uv)$. We must also have that $z \in X$, as X is a hitting set for $confset(T_1, T_2)$ and $x, y \notin X$. By Observation 2, we must have that one of z_1, z_2 is descended from v, and the other is not descended from u_1. This in turn implies (again by Observation 2) that for any $x' \in s(uv) \setminus X'$, if x' and y are in different dangling clades then $\{x', y, z\}$ is a conflict triple. Again this is a contradiction as $\{x', y, z\}$ has no elements of X', and so we may assume that all elements of $s(uv) \setminus X'$ are in the same dangling clade.

It remains to show that every element of this dangling clade is in $s(uv) \setminus X'$. To see this, suppose there exists some $x \in X'$ in the same dangling clade as the elements of $s(uv) \setminus X'$. Once again we have that x is in some conflict triple $\{x, y, z\}$ with $y, z \notin X'$, and if $y, z \notin s(uv)$ then $\{x', y, z\}$ is also a conflict triple for any $x' \in s(uv) \setminus X'$, a contradiction. So we may assume that one of y, z is in $s(uv) \setminus X'$. But all elements of $s(uv) \setminus X'$ are in the same dangling clade as x, and so by Observation 2 $\{x, y, z\}$ cannot be a conflict triple, a contradiction. So finally we have that all elements of $s(uv) \setminus X'$ are in the same dangling clade and all elements of this clade are in $s(uv) \setminus X'$, as required.

With the proof of Observation 3 complete, we are now in a position to construct S. For any minimal hitting set X' of $confset(T_1, T_2)$ with size at most d, by Observation 3 either $X' \cap s(uv) = \emptyset$, or $s(uv) \subseteq X'$ (in which case $|s(uv)| \leq d$), or $s(uv) \setminus X'$ forms a single dangling clade C (in which case $|s(uv) \setminus C| \leq d$).

So add all elements of X to S. For all $uv \in E(T_J)$ and any dangling clade C of labels in $s(uv)$, add $s(uv) \setminus C$ to S if $|s(uv) \setminus C| \leq d$. Observe that this construction adds at most $2d$ labels from $s(uv)$ to S.

Thus, in total, we have that the size of S is at most $|X| + 2d|E(T_J)| \leq d + 2d(2(|X_1 \cup X_2| - 1)) \leq d + 2d(4d - 2) = 8d^2 - 3d \leq 8d^2$.

Algorithm 3 describes the full procedure formally. The construction of T_J occurs once and as noted above takes $O(n^2)$ time. As each other line in the algorithm is called at most n times and takes $O(n)$ time, the overall running time of the algorithm $O(n^2)$. $\square$ $\square$

The last ingredient needed for Theorem 6 is Lemma 10, which shows that if a leaf x of T_1 as described in Lemma 6 has to be moved, then there are not too

Algorithm 3 Algorithm to construct a d-disagreement kernel between T_1 and T_2

procedure DISAGREEMENT-KERNEL(d, T_1, T_2)

T_1 and T_2 are trees on $\mathcal{X}$,d an integer.

Output: A set $S \subseteq \mathcal{X}$ such that for every minimal disagreement X between T_1 and T_2 with $|X| \leq d$, $X \subseteq S$.

  Find X such that $|X| \leq d$ and $T_1 - X = T_2 - X$

  Set $S = X$

  Let X_1, X_2 be copies of X and replace T_1, T_2 with corresponding trees T_1', T_2' on $(\mathcal{X} \setminus X) \cup X_1, (\mathcal{X} \setminus X) \cup X_2$.

  Let $T_J =$ JOIN-TREES$(T_1', T_2', (\mathcal{X} \setminus X), X_1, X_2)$

  Let $T^* = T_J|_{X_1 \cup X_2}$

  for $uv \in E(T^*)$ **do**

    Let $u = u_0 u_1 \dots u_t = v$ be the path in T_J from u to v

    Let $s(uv) = \{l \in \mathcal{X} \setminus X : l$ is descended from u_1 but not from $v\}$

    Set $p = |s(uv)| - d$ ▷ Any clade C has $|C| \geq p$ iff $|s(uv) \setminus C| \leq d$

    for $i \in [t]$ **do**

      Set $C = \{l \in s(uv) : l$ is descended from u_i but not from $u_{i+1}\}$ ▷ C is a single 'dangling clade'

      if $|C| \geq p$ **then**

        Set $S = S \cup (s(uv) \setminus C)$

  Return S.

many ways to regraft it in order to get closer to T^*.

In the course of the following proofs, we will want to take observations about one tree and use them to make statements about another. For this reason it's useful to have a concept of one node "corresponding" to another node in a different tree. In the case of leaf nodes this concept is clear - two leaf nodes are equivalent if they are assigned the same label- but for internal nodes there is not necessarily any such correspondence. However, in the case that one tree is the restriction of another to some label set, we can introduce a well-defined notion of correspondence:

Given two trees T, T' such that $T' = T|_X$ for some $X \subseteq \mathcal{X}(T)$, and a node $u \in V(T')$, define the node u_T of T by $u_T = \textsc{lca}_T(\mathcal{L}_{T'}(u))$. That is, u_T is the least common ancestor, in T, of the set of labels belonging to descendants of u in T'. We call u_T the *node corresponding to u in T*.

We note two useful properties of u_T here:

Lemma 8. *For any $T, T', X \subseteq \mathcal{X}(T)$ such that $T' = T|_X$ and any $u, v \in V(T')$, u_T is an ancestor of v_T if and only if u is an ancestor of v.*

Proof. If u is an ancestor of v then $\mathcal{L}_{T'}(v) \subseteq \mathcal{L}_{T'}(u)$, which implies that u_T is an ancestor of v_T. For the converse, observe that for any $Z \subseteq X$, any label in X descending from $\textsc{lca}_T(Z)$ in T is also descending from $\textsc{lca}_{T'}(Z)$ in T'. In particular letting $Z = \mathcal{L}_{T'}(u)$, we have $\mathcal{L}_T(u_T) \cap X = \mathcal{L}_T(\textsc{lca}_T(Z)) \cap$

$X \subseteq \mathcal{L}_{T'}(\text{LCA}_{T'}(Z)) = \mathcal{L}_{T'}(\text{LCA}_{T'}(\mathcal{L}_{T'}(u))) = \mathcal{L}_{T'}(u) \subseteq \mathcal{L}_T(u_T) \cap X$. Thus $\mathcal{L}_{T'}(u) = \mathcal{L}_T(u_T) \cap X$ and similarly $\mathcal{L}_{T'}(v) = \mathcal{L}_T(v_T) \cap X$. Then we have that u_T being an ancestor of v_T implies $\mathcal{L}_T(v_T) \subseteq L_T(u_T)$, which implies that $\mathcal{L}_{T'}(v) = \mathcal{L}_T(v_T) \cap X \subseteq L_T(u_T) \cap X = \mathcal{L}_{T'}(u)$, which implies that u is an ancestor of v. □ □

Lemma 9. *For any T'',T',T and $Y \subseteq X \subseteq \mathcal{X}(T)$ such that $T' = T|_X$ and $T'' = T'|_Y$, $(u_{T'})_T = u_T$.*

Proof. It is sufficient to show that any node in $V(T)$ is a common ancestor of $\mathcal{L}_{T'}(\text{LCA}_{T'}(Z))$ if and only if it is a common ancestor of Z, where $Z = \mathcal{L}_{T''}(u)$ (as this implies that the least common ancestors of these two sets are the same). It is clear that if $v \in V(T)$ is a common ancestor of $\mathcal{L}_{T'}(\text{LCA}_{T'}(Z))$ then it is also a common ancestor of Z, as $Z \subseteq \mathcal{L}_{T'}(\text{LCA}_{T'}(Z))$. For the converse, observe that as $T' = T|_X$ and $Z \subseteq X$, any label in X descended from $\text{LCA}_{T'}(Z)$ in T' is also descended from $\text{LCA}_T(Z)$ in T. This implies $\mathcal{L}_{T'}(\text{LCA}_{T'}(Z)) \subseteq \mathcal{L}_T(\text{LCA}_T(Z))$, and so any common ancestor of Z in T is also a common ancestor of $\mathcal{L}_{T'}(\text{LCA}_{T'}(Z))$. □ □

We are now ready to state and prove Lemma 10

Lemma 10. *Suppose that $d < d_{LR}(T_1, T_2) \leq d' + d$ with $d' \leq d$, and that there are $X_1, X_2 \subseteq \mathcal{X}$, and a tree T^* such that $T_1 - X_1 = T^* - X_1, T_2 - X_2 = T^* - X_2, |X_1| \leq d', |X_2| \leq d$, and let $x \in X_1 \setminus X_2$. Then, there is a set P of trees on label set $\mathcal{X}$ that satisfies the following conditions:*

- *for any tree T' such that $d_{LR}(T', T^*) < d_{LR}(T_1, T^*)$ and T' can be obtained from T_1 by pruning a leaf x and regrafting it, $T' \in P$;*
- $|P| \leq 18(d + d') + 8$*;*
- *P can be found in time $O(n(\log n + 18(d + d') + 8))$.*

The idea behind the proof is as follows: by looking at a subtree common to T_1 and T_2, we can identify the location that T_2 "wants" x to be positioned. This may not be the correct position for x, but we can show that if x is moved too far from this position, we will create a large number of conflicting triplets between T_2 and the solution T^*. As a result, we can create all trees in P by removing x from T_1 and grafting it on one of a limited number of edges.

Proof. For the purposes of this proof, we will treat each tree T as "planted", *i.e.* as having an additional root of degree 1, denoted $r(T)$, as the parent of what would normally be considered the "root" of the tree. (That is, $r(T)$ is the parent of $\text{LCA}_T(\mathcal{X}(T))$. Note that trees are otherwise binary. We introduce $r(T)$ as a notational convenience to avoid tedious repetition of proofs - grafting a label above a tree T can instead be represented as grafting it on the edge between $r(T)$ and its child. For the purposes of corresponding nodes, if $T' = T - X$ then $(r(T'))_T = r(T)$. This allows us to assume that every node in T is a descendant of u_T for some node u in T'.

A naive method for constructing a tree in P is the following: Apply an LPR move (x, e) on T_1, such that x is moved to a position that T_2 "wants" x to be in. There are at least two problems with this method. The first is that, since T_1 and T_2 have different structures, it is not clear where in T_1 it is that T_2 "wants" x to be. We can partially overcome this obstacle by initially considering a subtree common to both T_1 and T_2. However, because T_2 will want to move leafs that will not be moved in T_1, it can still be the case that even though T_2 "agrees" with T^* on x, T_2 may want to put x in the "wrong" place, when viewed from the perspective of T_1. For this reason we have to give a counting argument to show that if x is moved "too far" from the position suggested by T_2, it will create too many conflicting triplets, which cannot be covered except by moving x. We make these ideas precise below.

Let P^* be the set of all trees T' such that $d_{LR}(T', T^*) < d_{LR}(T_1, T^*)$ and T' can be obtained from T_1 by an LPR move on x. Thus, it is sufficient to construct a set P such that $|P| \leq 18(d + d') + 8$ and $P^* \subseteq P$.

We first construct a set $X_m \subseteq \mathcal{X}$ such that $|X_m| \leq d + d', x \in X_m$, and $T_1 - X_m = T_2 - X_m$. Note that the unknown set $(X_1 \cup X_2)$ satisfies these properties, as $T_1 - (X_1 \cup X_2) = T^* - (X_1 \cup X_2) = T_2 - (X_1 \cup X_2)$, and so such a set X_m must exist. We can find X_m in time $O(n \log n)$ by applying MAST on $(T_1 - \{x\}, T_2 - \{x\})$ [5].

Now let T_m be the tree with labelset $\mathcal{X} \setminus X_m$ such that $T_m = T_1 - X_m = T_2 - X_m$. Note that for any T' in P^*, we have that $T' - \{x\} = T_1 - \{x\}$ and therefore $T' - X_m = T_1 - X_m = T_m$.

Informally, we now have a clear notion of where T_2 "wants" x to go, relative to T_m. There is a unique edge e in T_m such that grafting x on e will give the tree $T_2 - (X_m \setminus \{x\})$. If we assume that this is the "correct" position to add x, then it only remains to add the remaining labels of X_m back in a way that agrees with T_1 (we will describe how this can be done at the end of the proof). Unfortunately, grafting x onto the obvious choice e does not necessarily lead to a graph in P^*. This is due to the fact that T_2 can be "mistaken" about labels outside of X_m.

To address this, we have try grafting x on other edges of T_m. There are too many edges to try them all. We therefore need the following claim, which allows us to limit the number of edges to try.

Claim: *In $O(n)$ time, we can find $y \in V(T_m)$ and $Z \subseteq V(T_m), |Z| \leq 4$, such that:*

- *For any T' in P^*, $x \in \mathcal{L}_{T'}(y_{T'}) \setminus \bigcup_{z' \in Z} \mathcal{L}_{T'}(z'_{T'})$*
- $|\mathcal{L}_{T_m}(y) \setminus \bigcup_{z' \in Z} \mathcal{L}_{T_m}(z')| \leq 8(d + d')$

Informally, the claim identifies a node y and set of nodes Z in T_m, such that x should be added as a descendant of y but not of any node in Z, and the number of such positions is bounded. Algorithm 4 describes the formal procedure to produce y and Z. The proof of the claim takes up most of the remainder of our proof; the reader may wish to skip it on their first readthrough.

Algorithm 4 FPT algorithm to restrict possible locations of x given $(T_m, T_1, T_2, x, d, d')$

procedure LOCATION-RESTRICTION$(T_m, T_2, X_m, x, d, d')$
T_1, T_2 are two trees, T_m is a common subtree of T_1 and T_2 such that $T_m = T_2 - X_m$, x is a label that cannot be moved in T_2 (but must be moved in T_1), d is the maximum number of leaves we can remove in a tree, d' is the maximum number of leaves we can move in T_1. Output is a pair (y, Z) with $y \in V(T_m)$, $Z \subseteq V(T_m)$, such that we may assume x is a descendant of y but not a descendant of any $z' \in Z$, and the number of labels like this in T_m is $O(d)$. For this pseudocode, every tree T has a degree-1 root $r(T)$.
    Set $T'_m = T_2 - (X_m \setminus \{x\})$
    Set $z =$ lowest ancestor of x in T'_m such that $|\mathcal{L}_{T'_m}(z) \setminus x| \geq d + d'$, or return $(r(T'_m), \emptyset)$ if no such z exists.
    Set $y =$ lowest ancestor of z in T'_m such that $|\mathcal{L}_{T'_m}(y) \setminus \mathcal{L}_{T'_m}(z)| \geq d + d'$, or $r(T'_m)$ if no such ancestor exists.
    ▷ Find sets $Z = Z_1 \cup Z_2$ of nodes that cover all but a bounded number of the descendants of y, and such that we can rule out x being descended from any z' in Z.
    Let z_1, z_2 be the children of z such that x is descended from z_1 in T'_m
    Set $Z_1 = \{z'$ descended from $z_2 : |\mathcal{L}_{T'_m}(z')| \geq d + d'$ and $|\mathcal{L}_{T'_m}(z_2) \setminus \mathcal{L}_{T'_m}(z')| \geq d + d'$, and this does not hold for any ancestor of $z'\}$
    Let y_1, y_2 be the children of y such that x is descended from y_1 in T'_m
    Set $Z_2 = \{y'$ descended from $y_2 : |\mathcal{L}_{T'_m}(y')| \geq d + d'$ and $|\mathcal{L}_{T'_m}(y_2) \setminus \mathcal{L}_{T'_m}(y')| \geq d + d'$, and this does not hold for any ancestor of $y'\}$
    ▷ Note that $|Z_1| \leq 2, |Z_2| \leq 2$.
    Set $y^* =$ node of T_m for which y is the corresponding node in T'_m
    Set $Z = \{z^*$ in $T_m : z' \in Z_1 \cup Z_2$ is the node corresponding to z^* in $T_m\}$
    Return (y^*, Z)

Proof. Let $T'_m = T_2 - (X_m \setminus \{x\})$. Note that $T'_m - \{x\} = T_m$. We will use the presence of x in T'_m to identify the node y and set Z. (Technically, this means the nodes we find are nodes in T'_m rather than T_m. However, we note that apart the parent of x and x itself, neither of which will be added to $\{y\} \cup Z$, every node in T'_m is the node $v_{T'_m}$ corresponding to some node v in T_m. For the sake of clarity, we ignore the distinction and write v to mean $v_{T'_m}$ throughout this proof. The nodes in $\{y\} \cup Z$ should ultimately be replaced with the nodes in T_m to which they correspond.)

We first identify two nodes z, y of T_m as follows:

- Let z be the least ancestor of x in T'_m such that $|\mathcal{L}_{T'_m}(z) \setminus \{x\}| \geq d+d'$. If no such x exists, then $\mathcal{X}(T'_m) \leq d + d'$ and we may return $y = r(T'_m), Z = \emptyset$.

- Let y be the least ancestor of z in T'_m such that $|\mathcal{L}_{T'_m}(y) \setminus \mathcal{L}_{T'_m}(z)| \geq d+d'$. If no such y exists, set $y = r(T'_m)$.

Using this definition, we will show that x must be a descendant of $y_{T'}$ for any $T' \in P^*$. We first describe a general tactic for restricting the position of x in T', as this tactic will be used a number of times.

Suppose that for some $T' \in P^*$ there is a set of $d+d'$ triplets in $confset(T', T_2)$ whose only common element is x. Then let $X' \subseteq \mathcal{X}$ be a set of labels such that $T' - X' = T^* - X'$ and $|X'| = d_{LR}(T', T^*) \leq d_{LR}(T_1, T^*) - 1 \leq d' - 1$. Note that $T_2 - (X' \cup X_2) = T^* - (X' \cup X_2) = T' - (X' \cup X_2)$, and therefore $(X' \cup X_2)$ is a hitting set for $confset(T', T_2)$. As $|X' \cup X_2| \leq d + d - 1$ and there are $d + d'$ triplets in $confset(T', T_2)$ whose only common element is x, it must be the case that $x \in X' \cup X_2$. As $x \notin X_2$, we must have $x \in X'$. But this implies that $T_1 - X' = T' - X' = T^* - X'$ and therefore $d_{LR}(T_1, T^*) \leq |X'| = d_{LR}(T', T^*) \leq d_{LR}(T_1, T^*) - 1$, a contradiction. Thus we may assume that such a set of triplets does not exist.

We now use this idea to show that $x \in \mathcal{L}_{T'}(y_{T'})$, for any $T' \in P^*$. Indeed, suppose $x \notin \mathcal{L}_{T'}(y_{T'})$. We may assume $y \neq r(T'_m)$ as otherwise $y_{T'} = r(T')$ by definition and so $\mathcal{L}_{T'}(y_{T'}) = \mathcal{X}(T')$. Then let $z_1, \dots, z_{d+d'}$ be $d + d'$ labels in $\mathcal{L}_{T'_m}(z) \setminus \{x\}$. Let $y_1, \dots, y_{d+d'}$ be $d + d'$ labels in $\mathcal{L}_{T'_m}(y) \setminus \mathcal{L}_{T'_m}(z)$. Observe that for each $i \in [d + d']$, T'_m (and therefore T_2) contains the triplet $(z_i x | y_i)$, but T' contains the triplet $(z_i y_i | x)$. Therefore $confset(T', T_2)$ contains $d + d'$ sets whose only common element is x. As this implies a contradiction, we must have $x \in \mathcal{L}_{T'}(y_{T'})$.

Note however that $|\mathcal{L}_{T'_m}(y)|$ maybe be very large. In order to provide a bounded range of possible positions for x, we still need to find a set Z of nodes such that $|\mathcal{L}_{T'_m}(y) \setminus \bigcup_{z' \in Z} \mathcal{L}_{T'_m}(z'))|$ is bounded, and such that we can show $x \notin \mathcal{L}_{T'}(z'_{T'})$ for any $z' \in Z$.

We now construct a set Z_1 of descendants of z as follows:

- Let z_1, z_2 be the children of z in T'_m such that x is descended from z_1.

- If $|\mathcal{L}_{T'_m}(z_2)| \leq 3(d + d')$ then set $Z_1 = \emptyset$.

- Otherwise, let Z_1 be the set of highest descendants z' of z_2, such that $|\mathcal{L}_{T'_m}(z')| \geq d + d'$ and $|\mathcal{L}_{T'_m}(z_2) \setminus \mathcal{L}_{T'_m}(z')| \geq d + d'$ (*i.e.* by highest descendant we mean such that z' has no ancestor z'' with the same properties).

Note that $|\mathcal{L}_{T'_m}(z_1)| \leq d+d'$ by our choice of z. It follows that if $|\mathcal{L}_{T'_m}(z_2)| \leq 3(d + d')$ then $|\mathcal{L}_{T'_m}(z)| \leq 4(d + d')$. If on the other hand $|\mathcal{L}_{T'_m}(z_2)| > 3(d + d')$ then Z_1 is non-empty. Indeed, let z' be a lowest descendant of z_2 with $|\mathcal{L}_{T'_m}(z')| \geq d+d'$, and observe that $|\mathcal{L}_{T'_m}(z')| \leq 2(d+d')$. Then either $z' \in Z_1$, or $|\mathcal{L}_{T'_m}(z_2) \setminus \mathcal{L}_{T'_m}(z')| \leq d+d'$, in which case $|\mathcal{L}_{T'_m}(z_2)| \leq |\mathcal{L}_{T'_m}(z_2) \setminus \mathcal{L}_{T'_m}(z')| + |\mathcal{L}_{T'_m}(z')| \leq d + d' + 2(d + d') = 3(d + d')$.

We also have that $|Z_1| \leq 2$. Indeed, let z'_1, z'_2, z'_3 be three distinct nodes in Z_1, and suppose without loss of generality that $(z'_1 z'_2|z'_3) \in tr(T'_m)$. Then setting $z' = \textsc{lca}_{T'_m}(z'_1, z'_2)$, we have that z' is an ancestor of z'_1 such that $|\mathcal{L}_{T'_m}(z')| \geq d+d'$ and $|\mathcal{L}_{T'_m}(z_2) \setminus \mathcal{L}_{T'_m}(z')| \geq |\mathcal{L}_{T'_m}(z'_3)| \geq d+d'$, a contradiction by minimality of z_1.

We have that $|\mathcal{L}_{T'_m}(z) \setminus \bigcup_{z' \in Z_1} \mathcal{L}_{T'_m}(z'))| \leq 4(d+d')$. Indeed, if $Z_1 = \emptyset$ then $|\mathcal{L}_{T'_m}(z)| \leq 4(d + d')$ as described above. Otherwise, let z' be an element of Z_1 and z_p its parent, z_s its sibling in T'_m. Clearly $|\mathcal{L}_{T'_m}(z_p)| \geq |\mathcal{L}_{T'_m}(z')| \geq d + d'$, and so as $z_p \notin Z_1$ we have $|\mathcal{L}_{T'_m}(z_2) \setminus \mathcal{L}_{T'_m}(z_p)| < d + d'$. If $|\mathcal{L}_{T'_m}(z_s)| \geq d + d'$ then $z_s \in Z_1$ (since $|\mathcal{L}_{T'_m}(z_2) \setminus \mathcal{L}_{T'_m}(z_s)| \geq |\mathcal{L}_{T'_m}(z')| \geq d + d'$), and so $|\mathcal{L}_{T'_m}(z) \setminus \bigcup_{z' \in Z_1} \mathcal{L}_{T'_m}(z'))| \leq |\mathcal{L}_{T'_m}(z_1)| + |\mathcal{L}_{T'_m}(z_2) \setminus \mathcal{L}_{T'_m}(z_p)| \leq 2(d + d')$. Otherwise, $|\mathcal{L}_{T'_m}(z) \setminus \bigcup_{z' \in Z_1} \mathcal{L}_{T'_m}(z'))| \leq |\mathcal{L}_{T'_m}(z_1)| + |\mathcal{L}_{T'_m}(z_2) \setminus \mathcal{L}_{T'_m}(z_p)| + |\mathcal{L}_{T'_m}(z_s)| \leq 3(d + d')$.

We have now shown that $|Z_1| \leq 2$ and that $|\mathcal{L}_{T'_m}(z) \setminus \bigcup_{z' \in Z_1} \mathcal{L}_{T'_m}(z'))| \leq 4(d + d')$. The final property of Z_1 we wish to show is that for any $z' \in Z_1$ and any $T' \in P$, $x \notin \mathcal{L}_{T'}(z'_{T'})$.

So suppose $x \in \mathcal{L}_{T'}(z'_{T'})$. Let $\hat{z}_1, \ldots, \hat{z}_{d+d'}$ be $d + d'$ labels in $\mathcal{L}_{T'_m}(z_2) \setminus L_{T'_m}(z')$. Also, z_1 and z_2 were already taken. Let $w_1, \ldots, w_{d+d'}$ be $d + d'$ labels in $\mathcal{L}_{T'_m}(z')$. Then for each $i \in [d+d']$, T'_m (and therefore T_2) contains the triplet $(\hat{z}_i w_i|x)$, but T' contains the triplet $(xw_i|\hat{z}_i)$. Therefore $confset(T', T_2)$ contains $d + d'$ sets whose only common element is x. As this implies a contradiction, we must have $x \notin \mathcal{L}_{T'}(z'_{T'})$.

We now define a set Z_2 of descendants of y:

- If $y = r(T'_m)$, set $Z_2 = \emptyset$.
- Otherwise, let y_1, y_2 be the children of y in T'_m such that z is descended from y_1.
- If $|\mathcal{L}_{T'_m}(y_2)| \leq 3(d + d')$ then set $Z_2 = \emptyset$.
- Otherwise, let Z_2 be the set of highest descendants y' of y_2, such that $|\mathcal{L}_{T'_m}(y')| \geq d + d'$ and $|\mathcal{L}_{T'_m}(y_2) \setminus \mathcal{L}_{T'_m}(y')| \geq d + d'$ (*i.e.* such that y' has no ancestor y'' with the same properties).

In a similar way to the proofs for Z_1, we can show that $|Z_2| \leq 2$, that $|(\mathcal{L}_{T'_m}(y) \setminus \mathcal{L}_{T'_m}(z)) \setminus \bigcup_{y' \in Z_2} \mathcal{L}_{T'_m}(y'))| \leq 4(d+d')$, and that $x \notin \mathcal{L}_{T'}(y'_{T'})$ for any $y' \in Z_2$ and any $T' \in P^*$.

Note that $|\mathcal{L}_{T'_m}(y_1) \setminus \mathcal{L}_{T'_m}(z)| \leq d+d'$ by our choice of y. It follows that if $|\mathcal{L}_{T'_m}(y_2)| \leq 3(d+d')$ then $|\mathcal{L}_{T'_m}(y) \setminus \mathcal{L}_{T'_m}(z)| \leq 4(d+d')$. If on the other hand $|\mathcal{L}_{T'_m}(y_2)| > 3(d+d')$, then Z_2 is non-empty. Indeed, let y' be a lowest descendant of y_2 with $|\mathcal{L}_{T'_m}(y')| \geq d+d'$, and observe that $|\mathcal{L}_{T'_m}(y')| \leq 2(d+d')$. Then either $y' \in Z_2$, or $|\mathcal{L}_{T'_m}(y_2) \setminus \mathcal{L}_{T'_m}(y')| \leq d+d'$, in which case $|\mathcal{L}_{T'_m}(y_2)| \leq |\mathcal{L}_{T'_m}(y_2) \setminus \mathcal{L}_{T'_m}(y')| + |\mathcal{L}_{T'_m}(y')| \leq d+d'+2(d+d') = 3(d+d')$.

We also have that $|Z_2| \leq 2$. Indeed, let y'_1, y'_2, y'_3 be three distinct nodes in Z_2, and suppose without loss of generality that $(y'_1 y'_2 | y'_3) \in tr(T'_m)$. Then setting $y' = \text{LCA}_{T'_m}(y'_1, y'_2)$, we have that y' is an ancestor of y'_1 such that $|\mathcal{L}_{T'_m}(y')| \geq d+d'$ and $|\mathcal{L}_{T'_m}(y_2) \setminus \mathcal{L}_{T'_m}(y')| \geq |\mathcal{L}_{T'_m}(y'_3)| \geq d+d'$, a contradiction by minimality of y_1.

We have that $|(\mathcal{L}_{T'_m}(y) \setminus \mathcal{L}_{T'_m}(z)) \setminus \bigcup_{y' \in Z_2} \mathcal{L}_{T'_m}(y'))| \leq 4(d+d')$. Indeed, if $y = r(T'_m)$ then by construction $|\mathcal{L}_{T'_m}(\hat{y}) \setminus \mathcal{L}_{T'_m}(z)| < d+d'$ for any ancestor $\hat{y}$ of z (noting that otherwise there would be no reason to set y as $r(T'_m)$ rather than the child of $r(T'_m)$), and so in particular $|\mathcal{L}_{T'_m}(y) \setminus \mathcal{L}_{T'_m}(z)| < d+d'$. If $y \neq r(T'_m)$ and $Z_2 = \emptyset$ then $|\mathcal{L}_{T'_m}(y) \setminus \mathcal{L}_{T'_m}(z)| \leq 4(d+d')$ as described above. Otherwise, let y' be an element of Z_2 and y_p its parent, y_s its sibling in T'_m. Clearly $|\mathcal{L}_{T'_m}(y_p)| \geq |\mathcal{L}_{T'_m}(y')| \geq d+d'$, and so as $y_p \notin Z_2$ we have $|\mathcal{L}_{T'_m}(y_2) \setminus \mathcal{L}_{T'_m}(y_p)| < d+d'$. If $|\mathcal{L}_{T'_m}(y_s)| \geq d+d'$ then $y_s \in Z_2$ (since $|\mathcal{L}_{T'_m}(y_2) \setminus \mathcal{L}_{T'_m}(y_s)| \geq |\mathcal{L}_{T'_m}(y')| \geq d+d'$), and so $|(\mathcal{L}_{T'_m}(y) \setminus \mathcal{L}_{T'_m}(z)) \setminus \bigcup_{y' \in Z_2} \mathcal{L}_{T'_m}(y'))| \leq |\mathcal{L}_{T'_m}(y_1) \setminus \mathcal{L}_{T'_m}(z)| + |\mathcal{L}_{T'_m}(y_2) \setminus \mathcal{L}_{T'_m}(y_p)| \leq 2(d+d')$. Otherwise, $|(\mathcal{L}_{T'_m}(y) \setminus \mathcal{L}_{T'_m}(z)) \setminus \bigcup_{y' \in Z_2} \mathcal{L}_{T'_m}(y'))| \leq |\mathcal{L}_{T'_m}(y_1) \setminus \mathcal{L}_{T'_m}(z)| + |\mathcal{L}_{T'_m}(y_2) \setminus \mathcal{L}_{T'_m}(y_p)| + |\mathcal{L}_{T'_m}(y_s)| \leq 3(d+d')$.

We have now shown that $|Z_2| \leq 2$ and $|(\mathcal{L}_{T'_m}(y) \setminus \mathcal{L}_{T'_m}(z)) \setminus \bigcup_{y' \in Z_2} \mathcal{L}_{T'_m}(y'))| \leq 4(d+d')$. The final property of Z_2 we wish to show is that for any $y' \in Z_2$ and any $T' \in P^*$, we have that $x \notin \mathcal{L}_{T'}(y'_{T'})$.

So suppose $x \in \mathcal{L}_{T'}(y'_{T'})$. Let $\hat{y}_1, \dots, \hat{y}_{d+d'}$ be $d+d'$ labels in $\mathcal{L}_{T'_m}(y_2) \setminus L_{T'_m}(y')$. Let $w_1, \dots, w_{d+d'}$ be $d+d'$ labels in $\mathcal{L}_{T'_m}(y')$. Then for each $i \in [d+d']$, T'_m (and therefore T_2) contains the triplet $(\hat{y}_i w_i | x)$, but T' contains the triplet $(x w_i | \hat{y}_i)$. Therefore $confset(T', T_2)$ contains $d+d'$ sets whose only common element is x. As this implies a contradiction, we must have $x \notin \mathcal{L}_{T'}(y'_{T'})$.

Now that Z_1 and Z_2 have been constructed, let $Z = Z_1 \cup Z_2$. Note that $|Z| \leq 4$. Algorithm 4 describes the construction of y and Z formally (see Figure 5).

We have shown above that for any $T' \in P^*$, x is descended from $y_{T'}$ in T' and not from $z'_{T'}$ for any $z' \in Z$, and so $x \in \mathcal{L}_{T'}(y_{T'}) \setminus \bigcup_{z' \in Z} \mathcal{L}_{T'}(z'_{T'})$. As $|\mathcal{L}_{T'_m}(z) \setminus \bigcup_{z' \in Z_1} \mathcal{L}_{T'_m}(z'))| \leq 4(d+d')$ and $|(\mathcal{L}_{T'_m}(y) \setminus \mathcal{L}_{T'_m}(z)) \setminus \bigcup_{y' \in Z_2} \mathcal{L}_{T'_m}(y'))| \leq 4(d+d')$, we have $|\mathcal{L}_{T'_m}(y) \setminus \bigcup_{z' \in Z} \mathcal{L}_{T'_m}(z'))| \leq 8(d+d')$.

To analyze the complexity, note that we can calculate the value of $|\mathcal{L}_{T'_m}(u)|$ for all u in $O(n)$ time using a depth-first search approach, together with the fact that $|\mathcal{L}_{T'_m}(u)| = |\mathcal{L}_{T'_m}(u_1)| + |\mathcal{L}_{T'_m}(u_2)|$ for any node u with children u_1, u_2. Then we can find z in $O(n)$ time, and once we have found z we can find y, and thence z_1, z_2, y_1, y_2, in $O(n)$ time. Similarly, once these nodes are found we can

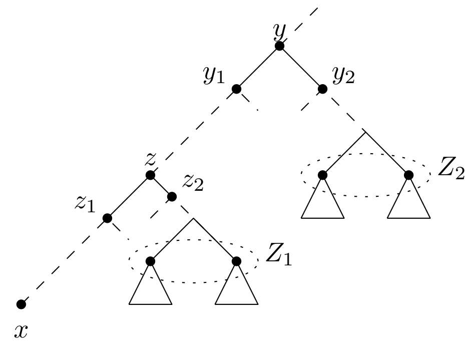

Figure 5: Construction of y and $Z = Z_1 \cup Z_2$, see Algorithm 4. Dashed edges represent parts of the tree that were omitted. Triangles represent parts of the tree that may contain more than $d + d'$ leaves.

find the members of Z in $O(n)$ time. $\square$ $\square$

Using the claim, we may now construct a set P' of $O \leq 16(d+d')+8$ trees on $\mathcal{X} \setminus (X_m \setminus \{x\})$, such for any $T' \in P^*$, P' contains the tree $T'' = T' - (X_m \setminus \{x\})$. Indeed, let F be the set of arcs uv in T_m that exist on a path from y to a node in $(\mathcal{L}_{T_m}(y) \setminus \bigcup_{z \in Z} \mathcal{L}_{T_m}(z')) \cup Z$. As $|\mathcal{L}_{T_m}(y) \setminus \bigcup_{z' \in Z} \mathcal{L}_{T_m}(z')| \leq 8(d+d')$, $|Z| \leq 4$ and T_m is a binary tree, we have $|F| \leq 16(d+d')+8$. For each $e \in F$, let T_e be the tree obtained from T_m by grafting x onto the arc e . Let $P' = \{T_e : e \in F\}$.

Let T' be a tree in P^* and consider $T'' = T' - (X_m \setminus \{x\})$. Note that $T'' - \{x\} = T' - X_m = T_m$. Therefore $u_{T''}$ is well-defined for every node $u \in V(T_m)$, and every node in T'' is equal to $u_{T''}$ for some $u \in V(T_m)$, except for x and its parent in T''. So let w be the parent of x in T'', $u_{T''}$ the parent of w in T'', and $v_{T''}$ the child of w in T'' that is not x. Observe that T'' can be obtained from T_m by grafting x onto the arc uv. Then it is enough to show that $uv \in F$.

To see that $uv \in F$, first note that for each $z' \in \{y\} \cup Z$, $z'_{T''}$ is well-defined and $(z'_{T''})_{T'} = z'_{T'}$ (see Lemma 9). Then as x is descended from $(y_{T''})_{T'} = y_{T'}$ in T', x is descended from $y_{T''}$ in T'' (Lemma 8). Similarly, as x is not descended from $(z'_{T''})_{T'} = z'_{T'}$ in T' for any $z' \in Z$, x is not descended from $z_{T''}$ in T''. Thus $x \in \mathcal{L}_{T''}(y_{T''}) \setminus \bigcup_{z' \in Z} \mathcal{L}_{T''}(z'_{T''})$. It follows that $u_{T''}$ is a descendant of $y_{T''}$ in T'' (note that $y_{T''} \neq w$, as w is not the least common ancestor of any set of labels in $\mathcal{X}(T_m)$). Also, $v_{T''}$ is not a descendant of $z_{T''}$ for any $z' \in Z$, unless $v_{T''} \in \bigcup_{z \in Z} z_{T''}$, as otherwise x would be a descendant of such a $z_{T''}$. Thus, $v_{T''}$ is either a member or an ancestor of $(\mathcal{L}_{T''}(y_{T''}) \setminus \bigcup_{z' \in Z} \mathcal{L}_{T''}(z'_{T''}))) \cup \bigcup_{z' \in Z} z'_{T''}$. It follows using Lemma 8 that u is a descendant in T_m of y, and v is an ancestor of $(\mathcal{L}_{T_m}(y) \setminus \bigcup_{z' \in Z} \mathcal{L}_{T_m}(z'))) \cup \bigcup_{z' \in Z} z'$. Then $uv \in F$, as required.

Now that we have constructed our set P', it remains to find, for each $T_e \in P'$, every tree T' on $\mathcal{X}$ such that $T' - (X_m \setminus \{x\}) = T_e$ and $T' - \{x\} = T_1 - \{x\}$.

This will give us our set P, as for every $T' \in P^*$, $T' - (X_m \setminus \{x\})$ is a tree T_e in P', and $T' - \{x\} = T_1 - \{x\}$.

Let $e = uv$, where $u, v \in V(T_m)$, and let T_{1e} be the subtree of $T_1 - \{x\}$ whose root is v, and has as its label set v together with all labels in $X_m \setminus \{x\}$ descended from u but not v. Then we have to try every way of adding x into this tree. If T_{1e} contains t labels from X_m, then there are $2t-1$ places to try adding x. Therefore P will have at most $2|X_m| \leq 2(d+d')$ additional trees compared to P', and so $|P| \leq 18(d+d')+8$. Algorithm 5 gives the full procedure to construct P.

Algorithm 5 FPT algorithm to find candidate trees for (T_1, T_i, x)

---

procedure CANDIDATE-TREES(T_1, T_2, x, d, d')
T_1, T_2 are two trees, x is a label that cannot be moved in T_2 (but must be moved in T_1), d is the maximum number of leaves we can remove in a tree, d' is the maximum number of leaves we can move in T_1. For this pseudocode, every tree T has a degree-1 root $r(T)$.
    Find X'_m such that $|X'_m| \leq d' + d - 1$ and $(T_1 - \{x\}) - X'_m = (T_2 - \{x\}) - X'_m$
    Set $X_m = X'_m \cup \{x\}$
    Set $T_m = T_1 - X_m$
    Set $(y, Z) =$ LOCATION-RESTRICTION$(T_m, T_2, X_m, x, d, d')$ ▷ y, Z are nodes in T_m such that roughly speaking, we may assume x must become a descendant of y but not of any $z' \in Z$.
    Set $U = \{u \in V(T_m) : u \in Z$ or u is a leaf descended from y but not from any $z' \in Z\}$
    Set $F = \{uv \in E(T_m) : uv$ is on a path from y to $U\}$ ▷ F is the set of edges we could graft x onto.
    Set $P = \emptyset$ ▷ Given F we now begin constructing P.
    Set $T'_1 = T_1 - \{x\}$
    for $e = uv \in F$ with u the parent of v **do** ▷ Try grafting x on e
        Set $u_{T'_1} =$ the node in T'_1 corresponding to u
        Set $v_{T'_1} =$ the node in T'_1 corresponding to v
        Set $X_e =$ set of labels l in $X_m \setminus \{x\}$ for which l has an ancestor v' in T'_1 with v' descended from $u_{T'_1}$, $v_{T'_1}$ descended from v'
            ▷ X_e is the set of leaves of T_1 for which we have to subdivide e.
        Set $U = v_{T'_1} \cup X_e$
        Set $E_e = \{u'v' \in E(T'_1) : u'v'$ is on a path from $u_{T'_1}$ to $U\}$
        for $u'v' \in E_e$ **do**
            Constuct T' from T'_1 by grafting x on $u'v'$
            Set $P = P \cup \{T'\}$
    Return P

To analyze the complexity, recall that we find X_m, and therefore construct T_m and T'_m, in $O(n \log n)$ time. As shown above, we can find the node y and set Z in $O(n)$ time. Given y and Z, the set of arcs F can be found in $O(n)$

time using a depth-first search approach. For each $e \in F$ it takes $O(n)$ time to construct T_e, and so the construction of P' takes $O(|F|n) = O((16(d+d')+8)n)$ time. Finally, the construction of of P from P' takes $O(|P|n) = O((18(d+d')+8)n)$ time. Putting it all together, we have that the construction of P takes $O(n(\log n + 18(d+d') + 8))$ time. □ □

We will call the set of trees P described in Lemma 10 the set of *candidate trees* for (T_1, T_2, x).

We are finally ready to give the proof of Theorem 6

Theorem 6 *(restated).* `AST-LR-d` *can be solved in time $O(c^d d^{3d}(n^3 + tn\log n))$, where c is a constant not depending on d or n.*

Proof. The outline for our algorithm is as follows. We employ a branch-and-bound algorithm, in which at each step we attempt to modify the input tree T_1 to become close to a solution. We keep track of an integer d', representing the maximum length of an LPR sequence between T_1 and a solution. Initially set $d' = d$. At each step, if $d_{LR}(T_1, T_i) \leq d$ for each $T_i \in \mathcal{T}$ then T_1 is a solutioon, and we are done. Otherwise, there must exist sime T_i for which $d_{LR}(T_1, T_i) \geq d + d'$. In this case, we calculate the $(d + d')$ disagreement kernel S between T_1 and T_i (using the procedure of Lemma 7), and for each $x \in S$, attempt to construct a set P of trees as in Lemma 10. For each $T' \in P$, we try replacing T_1 with T', reducing d' by 1, and repeating the procedure. Algorithm 6 describes the full procedure formally.

We claim that Algorithm 6 is a correct algorithm for `AST-LR-d`, and runs in time $O(c^d d^{3d}(n^2 + tn\log n))$, for some constant c not depending on n or d.

First notice that if, in a leaf node of the branch tree created by Algorithm 6, a tree T^* is returned, this occurs at line 3 in which case it has been verified that T^* is indeed a solution. As an internal node of the branch tree returns a tree if and only if a child recursive call also returns a tree (the for loop on line 9), this shows that when the algorithm outputs a tree T^*, it is indeed a solution.

We next show that if a solution exists, then Algorithm 6 will return one. Suppose that $\mathcal{T}$ admits a solution, and let T^* be a solution that minimizes $d_1 := d_{LR}(T_1, T^*)$, with $d_1 \leq d'$. We show that one leaf of the branch tree created by the algorithm returns T^* (and thus the root of the branch tree also returns a solution, albeit not necessarily T^*). This is done by proving that in one of the recursive calls made to MASTRL-DISTANCE on line 13, the tree T' obtained from T_1 satisfies $d_{LR}(T', T^*) = d_1 - 1$. By applying this argument inductively, this shows that the algorithm will find T^* at some node of depth d_1 in the branch tree of the algorithm.

First notice that since d_{LR} is a metric, for each $T_i \in \mathcal{T}$, $d_{LR}(T_1, T_i) \leq d_{LR}(T_1, T^*) + d_{LR}(T^*, T_i) \leq d' + d$, and so the algorithm will not return $FALSE$ on line 5.

If T_1 isn't a solution, then there is a tree of $\mathcal{T}$, say T_2 w.l.o.g., such that $d_{LR}(T_1, T_2) > d$. Notice that in this case, all the conditions of Lemma 6 are satisfied, *i.e.* $d_{LR}(T_1, T_2) > d$, and there are sets $X_1, X_2 \subseteq \mathcal{X}$ both of size at most d such that $T_1 - X_1 = T^* - X_1$, $T_2 - X_2 = T^* - X_2$. Thus there is a minimal disagreement X between T_1 and T_2, $|X| \leq d' + d$, and $x \in X$ such that $x \in X_1 \setminus X_2$. By Lemma 3, there is an LPR sequence $L = (x_1, \ldots, x_k)$ turning T_1 into T^*, where $\{x_1, \ldots, x_k\} = X_1$. As $x \in X_1$, by Lemma 4, the leaves appearing in L can be reordered, and we may assume that $x = x_1$. Finally by Lemma 10, if T' satisfies $d_{LR}(T', T^*) \leq d_1 - 1$ and T' can be obtained from T_1 by an LPR move on x, then $T' \in P$. As we are making one recursive call to MASTRL-DISTANCEfor each tree in P, this proves that one such call replaces T_1 by T' such that $d_{LR}(T', T^*) = d_1 - 1$.

As for the complexity, recall from Lemma 7 that the $(d + d')$-disagreement kernel S computed in line 8 contains at most $8d^2$ labels.Therefore when Algorithm 6 enters the 'for' loop of line 9, it branches into at most $8d^2$ cases, one for each $x \in S$. Within each of these cases, the algorithm enters at most $|P|$ recursive calls, each of which decrements d'. As $|P| \leq 18(d+d')+8 \leq 36d+8$ by Lemma 10, a single call of the algorithm splits into at most $8d^2(36d+8) = O(d^3)$, each of which decrements d'. Therefore, the branching tree created by the algorithm has degree at most cd^3 (for some constant c) and depth at most d, and so $O(c^d d^{3d})$ cases are considered.

As $d_{LR}(T_1, T_i)$ can be calculated in $O(n \log n)$ time for each T_i, a single call of lines 2-5 of the algorithm takes $O(tn \log n)$ time. A single call of lines 6-8 takes $O(n^2)$ time by Lemma 10. Thus the total time for all calls of lines 2-8 is $O(c^d d^{3d} n(n^2 + t \log n)$. Each call of line 10 occurs just before a recursive call to the algorithm, as so line 10 is called at most $O(c^d d^{3d})$ times. A single call of line 10 takes $O(n(\log n + 18(d+d')+8)) = O(n(\log n + 36d))$ time by Lemma 10, and so the total time for all calls of line 10 is $O(c^d d^{3d} n(\log n + 36d))$. Thus in total, we have that the running time of the algorithm is $O(c^d d^{3d}(n^2 + n(t \log n + 36d))$. As we may assume $d \leq n$, this simplifies to $O(c^d d^{3d}(n^2 + tn \log n))$. □ □

Algorithm 6 FPT algorithm for parameter d.

procedure MASTRL−DISTANCE($\mathcal{T} = (T_1, T_2, \ldots, T_t), d, d'$)
$\mathcal{T}$ is the set of input trees (represented as a sequence to distinguish T_1 from the other trees), d is the maximum number of leaves we can remove in a tree, d' is the maximum number of leaves we can move in T_1, which should be initially set to d.
    if $d_{LR}(T_1, T_i) \leq d$ for each $T_i \in \mathcal{T}$ **then**
        Return T_1
    else if there is some $T_i \in \mathcal{T}$ such that $d_{LR}(T_1, T_i) > d' + d$ **then**
        Return FALSE #handles the $d' \leq 0$ case
    else ▷ here we 'guess' a leaf prune-and-regraft move on T_1
        Choose $T_i \in \mathcal{T}$ such that $d_{LR}(T_1, T_i) > d$
        Set $S =$ DISAGREEMENT-KERNEL($d + d', T_1, T_i$)
        for $x \in S$ **do** ▷ we are 'guessing' that x should go where T_i wants it.
            Set $P =$ CANDIDATE-TREES(T_1, T_i, x, d, d')
            $T^* = FALSE$
            for $T \in P$ **do**
                $T' =$ MASTRL−DISTANCE($(T, T_2, \ldots, T_t), d, d' - 1$)
                if T' is not $FALSE$, let $T^* := T'$
            Return T^*