Do Branch Lengths Help to Locate a Tree in a Phylogenetic Network?

Philippe Gambette[1] · **Leo van Iersel[2]** ·
Steven Kelk[3] · **Fabio Pardi[4]** · **Celine Scornavacca[5]**

Abstract Phylogenetic networks are increasingly used in evolutionary biology to represent the history of species that have undergone reticulate events such as horizontal gene transfer, hybrid speciation and recombination. One of the most fundamental questions that arise in this context is whether the evolution of a gene with one copy in all species can be explained by a given network. In mathematical terms, this is often translated in the following way: is a given phylogenetic tree contained in a given phylogenetic network? Recently this *tree containment* problem has been widely

✉ Fabio Pardi
pardi@lirmm.fr

Philippe Gambette
philippe.gambette@u-pem.fr

Leo van Iersel
l.j.j.v.iersel@gmail.com

Steven Kelk
steven.kelk@maastrichtuniversity.nl

Celine Scornavacca
celine.scornavacca@umontpellier.fr

[1] LIGM (UMR 8049), CNRS, ENPC, ESIEE Paris, UPEM, Université Paris-Est,
77454 Marne-la-Vallée, France

[2] Delft Institute of Applied Mathematics, Delft University of Technology, Postbus 5031,
2600 GA Delft, The Netherlands

[3] Department of Data Science and Knowledge Engineering (DKE), Maastricht University,
P.O. Box 616, 6200 MD Maastricht, The Netherlands

[4] Institut de Biologie Computationnelle (IBC) Laboratoire d'Informatique, de Robotique et de
Microélectronique de Montpellier (LIRMM), CNRS, Université de Montpellier, Montpellier,
France

[5] Institut de Biologie Computationnelle (IBC) Institut des Sciences de l'Evolution,
CC 064 Place Eugène Bataillon, Montpellier, France

investigated from a computational perspective, but most studies have only focused on the topology of the phylogenies, ignoring a piece of information that, in the case of phylogenetic trees, is routinely inferred by evolutionary analyses: branch lengths. These measure the amount of change (e.g., nucleotide substitutions) that has occurred along each branch of the phylogeny. Here, we study a number of versions of the tree containment problem that explicitly account for branch lengths. We show that, although length information has the potential to locate more precisely a tree within a network, the problem is computationally hard in its most general form. On a positive note, for a number of special cases of biological relevance, we provide algorithms that solve this problem efficiently. This includes the case of networks of limited complexity, for which it is possible to recover, among the trees contained by the network with the same topology as the input tree, the closest one in terms of branch lengths.

1 Introduction

The last few years have witnessed a growing appreciation of reticulate evolution—that is, cases where the history of a set of taxa (e.g., species, populations or genomes) cannot be accurately represented as a phylogenetic tree (Doolittle 1999; Bapteste et al. 2013), because of events causing inheritance from more than one ancestor. Classic examples of such reticulate events are hybrid speciation (Mallet 2007; Nolte and Tautz 2010; Abbott et al. 2013), horizontal gene transfer (Boto 2010; Hotopp 2011; Zhaxybayeva and Doolittle 2011) and recombination (Posada et al. 2002; Vuilleumier and Bonhoeffer 2015). Inferring the occurrence of these events in the past is a crucial step toward tackling major biological issues, for example, to understand recombinant aspects of viruses such as HIV (Rambaut et al. 2004), or characterizing the mosaic structure of plant genomes.

Reticulate evolution is naturally represented by *phylogenetic networks*—mathematically, simple generalizations of phylogenetic trees, where some nodes are allowed to have multiple direct ancestors (Huson et al. 2010; Morrison 2011). Currently, much of the mathematical and computational literature on this subject focuses solely on the topology of phylogenetic networks (Huson and Scornavacca 2011), namely not taking into account branch length information. This information—a measure of elapsed time, or of change that a species or gene has undergone along a branch—is usually estimated when inferring phylogenetic trees, and it may have a big impact on the study of reticulate evolution as well.

For example, in the literature investigating hybridization in the presence of incomplete lineage sorting, the branch lengths of a phylogenetic network are the key parameters to calculate the probability of observing a gene tree, and thus to determine the likelihood of the network (Meng and Kubatko 2009; Yu et al. 2012). Moreover, accurate estimates of branch lengths in the gene trees are known to improve the accuracy of the inferred network (Kubatko 2009; Yu et al. 2014). Similarly, for another large class of methods for network reconstruction, otherwise indistinguishable net-

work scenarios can become distinguishable, if branch lengths are taken into account (Pardi and Scornavacca 2015). The precise meaning of branch lengths is often context-dependent, ranging from expected number of substitutions per site, generally adopted in molecular phylogenetics, to a measure of the probability of coalescence, often adopted for smaller timescales where incomplete lineage sorting is common, to the amount of time elapsed. In the last case, we may expect the phylogeny (network or tree) to be ultrametric, that is to have all its leaves at the same distance from the root (Chan et al. 2006; Bordewich and Tokac 2016).

In this paper, we explore the impact of branch lengths on a fundamental question about phylogenetic networks: the *tree containment* problem. Informally (formal definitions will be given in the next section), this problem involves determining whether a given phylogenetic tree is contained, or *displayed*, by a given phylogenetic network, and in the positive case, locating this tree within the network. Biologically, this means understanding whether a gene—whose phylogenetic history is well-known—is consistent with a given phylogenetic network, and understanding from which ancestor the gene was inherited at each reticulate event. From a computational perspective, the tree containment problem lies at the foundation of the reconstruction of phylogenetic networks. In its classic version, where only topologies are considered, the problem is NP-hard (Kanj et al. 2008), but for some specific classes of networks it can be solved in polynomial time (van Iersel et al. 2010).

Intuitively, an advantage of considering branch lengths is that it should allow one to locate more precisely a gene history within a network, and, more generally, it should give more specific answers to the tree containment problem. For example, whereas a tree topology may be contained in multiple different locations inside a network (Cordue et al. 2014), this will happen much more rarely when branch lengths are taken into account (see, e.g., T_1 in Fig. 1). Similarly, some genes may only be detected to be inconsistent with a network when the branch lengths of their phylogenetic trees are considered (see, e.g., T_2 in Fig. 1). In practice, some uncertainty in the branch length estimates is to be expected, which implies that deciding whether a tree is contained in
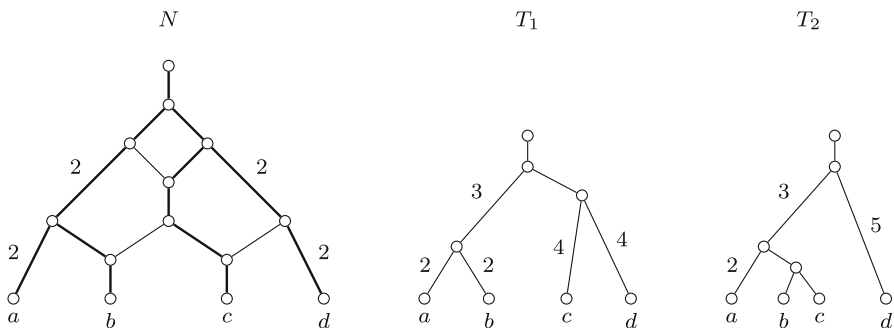
Fig. 1 Toy example on the impact of branch lengths on locating a tree within a network. If lengths are not taken into account, both T_1 and T_2 are displayed by N. Moreover, locating uniquely T_1 within N is not possible: there are 4 *switchings* (formally defined in the Preliminaries) of N for T_1, and 3 different ways to locate (*images* of) T_1 within N. If instead lengths are taken into account, the only image of T_1 within N is the one highlighted in bold, and T_2 is not displayed by N (in fact, the tree displayed by N isomorphic to T_2 has significantly different branch lengths). *Note*: branches with no label are assumed to have length 1

a network will depend on the confidence in these estimates (e.g., T_2 in Fig. 1 is only displayed by N if we allow its branch lengths to deviate by 2 or more units from their specified values).

While the possibility of having more meaningful answers to a computational problem is certainly an important advantage, another factor to consider is the complexity of calculating its solutions. It is known that adding constraints on branch lengths can lead to polynomial tractability of other problems in phylogenetics that would otherwise be NP-complete (Doyon et al. 2011). In this paper, we will show a number of results on the effect of taking into account branch lengths on the computational complexity of the tree containment problem. We first introduce the necessary mathematical preliminaries (Sec. 2), including a formal definition of the main problem that we consider TREE CONTAINMENT WITH BRANCH LENGTHS (TCBL), and of some variations of this problem accounting for the fact that branch lengths are usually only imprecise estimates of their true values (RELAXED-TCBL and CLOSEST-TCBL). We then show a number of hardness (negative) results for the most general versions of these problems (Sec. 3), followed by a number of positive results (Sec. 4). Specifically, a suite of polynomial time, pseudo-polynomial time and fixed-parameter tractable algorithms that solve the problems above for networks of limited complexity (measured by their level Choy et al. 2005; Jansson and Sung 2006; definition below) and containing no unnecessary complexity (no redundant blobs Iersel and Moulton 2014; also defined below).

2 Preliminaries

We define a *phylogenetic network* on X as a rooted directed acyclic graph with exactly one vertex of indegree 0 (the *root*), with no vertices with indegree and outdegree 1, and whose outdegree 0 vertices (the *leaves*) are bijectively labeled by the elements of X (the *taxa*). A *phylogenetic tree* is a phylogenetic network whose underlying undirected graph has no cycles. We consider phylogenetic networks (and thus trees) where each arc has an associated length. Formally, given an arc (u, v) of a phylogenetic network N, its *length* $\lambda_N(u, v)$ is a positive integer, i.e., strictly greater than zero. In this paper we will use the terms "arc lengths" and "branch lengths" interchangeably.

A phylogenetic tree or network is *binary* if all vertices have indegree 1 and outdegree 2 (*bifurcations*), indegree 2 and outdegree 1 (*reticulations*), indegree 0 and outdegree 1 (root) or indegree 1 and outdegree 0 (leaves). For example, all networks and trees in Fig. 1 are binary.

A *biconnected component* is a maximal connected subgraph that remains connected after removal of any one vertex. A *blob* of a phylogenetic network N is a biconnected component in which the undirected graph underpinning the biconnected component contains at least one cycle. Note that if a biconnected component of N is not a blob, then it is simply a cut arc (i.e., an arc whose removal disconnects N). The *level* of a binary phylogenetic network N is the maximum number of reticulations in any blob of N. An *outgoing arc* of a blob B is an arc (u, v) such that u is in B but v is not. An *incoming arc* (u, v) of B is such that v is in B but u is not. Note that a blob has at most one incoming arc. A blob is *redundant* if it has fewer than two outgoing arcs (i.e., one outgoing arc if the network is binary). As an example of these notions, the

network N in Fig. 1 contains only one blob, which has 4 outgoing arcs and is thus non-redundant. Because this blob has 3 reticulations, N is level-3.

We say that two phylogenetic trees T_1 and T_2 are *isomorphic*, or that they have the same *topology*, if there is a bijection between the nodes that is both edge-preserving and leaf label-preserving. (Note that arc lengths do not play a role here.)

Given a phylogenetic tree T and a phylogenetic network N whose leaves are labeled bijectively by the same set X, we say that *T is displayed by N taking into account lengths*, if T can be obtained from N in the following way:

- for each reticulation, remove all incoming arcs except one; the tree obtained after this process is called a *switching* of N;
- repeat as long as possible the following *dummy leaf deletions*: for each leaf not labeled by an element of X, delete it;
- repeat as long as possible the following *vertex smoothings*: for each vertex v with exactly one parent p and one child c, replace it with an arc from p to c, with $\lambda_N(p, c) = \lambda_N(p, v) + \lambda_N(v, c)$.

In the following, we sometimes only say that *T is displayed by N* (with no mention of lengths) to mean that arc lengths are disregarded, and only topological information is taken into account.

Note that N displays T taking into account lengths if and only if there exists a subtree T' of N with the same root as N such that T can be obtained by repeatedly applying vertex smoothings to T'. In this case, T' is said to be the *image* of T. There is a natural injection from the vertices of T to the vertices of T', so the definition of image extends naturally to any subgraph of T. In particular, the image of any arc in T is a path in N. Note that T can potentially have many images in N, but for a switching S of N, the image of T within S, if it exists, is unique. As an example of these notions, consider again Fig. 1, where N displays both T_1 and T_2, but only T_1 if lengths are taken into account. The part of N in bold is both a switching and an image of T_1 (as no dummy leaf deletions are necessary in this case).

Finally, is worth noting that, in this paper, if N displays T taking into account lengths, then the image of the root of T will always coincide with the root of N (no removal of vertices with indegree 0 and outdegree 1 is applied to obtain T). The biological justification for this is that trees and networks are normally rooted using an outgroup, which is sometimes omitted from the phylogeny; if arc lengths are taken into account, then the length of the path to the root of N in a tree displayed by N conveys the information regarding the distance from the outgroup. (See also Pardi and Scornavacca 2015 for a full discussion about this point.)

In this paper, we consider the following problem:

Problem 1 TREE CONTAINMENT WITH BRANCH LENGTHS (TCBL)

Input: A phylogenetic network N and a phylogenetic tree T on the same set X, and both with positive integer arc lengths.
Output: YES if T is displayed by N taking into account lengths, NO otherwise.

We also consider two variations of TCBL seeking trees displayed by N that are allowed to somehow deviate from the query tree, to account for uncertainty in the branch lengths of the input tree. The first of these two problems aims to determine the existence of a tree displayed by N, whose branch lengths fall within a specified (confidence) interval.

Problem 2 RELAXED-TCBL

Input: A phylogenetic network N with positive integer arc lengths, and a phylogenetic tree T, whose arcs are labeled by two positive integers $m_T(a)$ and $M_T(a)$, representing, respectively, the minimum and the maximum arc length. Both N and T are on the same set X.

Output: YES if and only if there exists a tree $\widetilde{T}$ displayed by N, isomorphic to T, and such that, for each arc a of T:

$$\lambda_{\widetilde{T}}(\tilde{a}) \in [m_T(a), M_T(a)],$$

where $\tilde{a}$ denotes the arc in $\widetilde{T}$ that corresponds to a in T.

The second variation of TCBL we consider here, seeks—among all trees displayed by the network, and that are isomorphic to the input tree T—one that is closest to T, in terms of the maximum difference between branch lengths. There are several other alternative choices for defining the "closest" tree to T, for example if distance is measured in terms of the average difference between branch lengths. Later on, we will see that our results on this problem also apply to many of these alternative formulations (see Theorem 7).

Problem 3 CLOSEST-TCBL

Input: A phylogenetic network N and a phylogenetic tree T on the same set X, and both with positive integer arc lengths.

Output: A tree $\widetilde{T}$ displayed by N, isomorphic to T, that minimizes

$$\max \left| \lambda_T(a) - \lambda_{\widetilde{T}}(\tilde{a}) \right|,$$

where the max is over any choice of an arc a in T, and $\tilde{a}$ denotes the arc in $\widetilde{T}$ that corresponds to a in T. If no tree isomorphic to T is displayed by N, then report FAIL.

Note that all problems in this paper involving positive integer arc lengths are equivalent to problems where arc lengths are positive rational numbers: it suffices to multiply those rational numbers by the least common denominator of the fractions corresponding to these numbers in order to obtain integers.

We conclude with some definitions concerning computational complexity. An NP-complete decision problem that includes numbers in the input may or may not permit a *pseudo-polynomial time* algorithm. This is an algorithm which runs in polynomial time if the numbers in the input are encoded in unary, rather than binary. Formally speaking such algorithms are not polynomial time, since unary encodings artificially inflate the size of the input. Nevertheless, a pseudo-polynomial time algorithm has the potential to run quickly if the numbers in the input are not too large. An NP-complete problem with numbers in the input is said to be *strongly* NP-complete if it remains NP-complete even under unary encodings of the numbers. Informally, such problems remain intractable even if the numbers in the input are small. An NP-complete problem is *weakly* NP-complete if it is NP-complete when the numbers are encoded in binary. Summarizing, if one shows that a weakly NP-complete problem also permits a pseudo-polynomial time algorithm, then (under standard complexity assumptions) this excludes strong NP-completeness. Similarly, demonstrating strong NP-completeness excludes (under standard complexity assumptions) the existence of a pseudo-polynomial time algorithm. We refer to Garey and Johnson (1979) for formal definitions.

On a slightly different note, an algorithm is said to be *fixed parameter tractable* (FPT) if it runs in time $O(f(k) \cdot poly(n))$ where n is the size of the input, k is some parameter of the input (in this article: the level of the network) and f is some computable function that depends only on k. An FPT algorithm for an NP-complete problem has the potential to run quickly even when n is large, as long as the parameter k is small, for example when f is a function of the form c^k, where c is a small constant greater than 1. We refer to Downey and Fellows (2013), Gramm et al. (2008) for more background on FPT algorithms.

3 Negative Results

3.1 Strong NP-Completeness

Theorem 1 *TCBL is strongly NP-complete, even when the phylogenetic tree T and the phylogenetic network N are binary.*

Proof We reduce to TCBL the following 3- PARTITION problem, which is strongly NP-complete (Garey and Johnson 1975):

Input: an integer Σ and a multiset S of $3m$ positive integers n_i in $]\Sigma/4, \Sigma/2[$ such that $m\Sigma = \sum\limits_{i \in [1...3m]} n_i$.

Output: YES if S can be partitioned into m subsets of elements $S_1, S_2, \ldots, S_m$ each of size 3, such that the sums of the numbers in each subset are all equal; NO otherwise.

Let us consider a multiset S containing $3m$ positive integers n_i which have sum $m\Sigma$.

We build a phylogenetic tree T in the following way. We first build a directed path containing $m + 2$ vertices, whose arcs all have length 1. We call its initial vertex ρ,
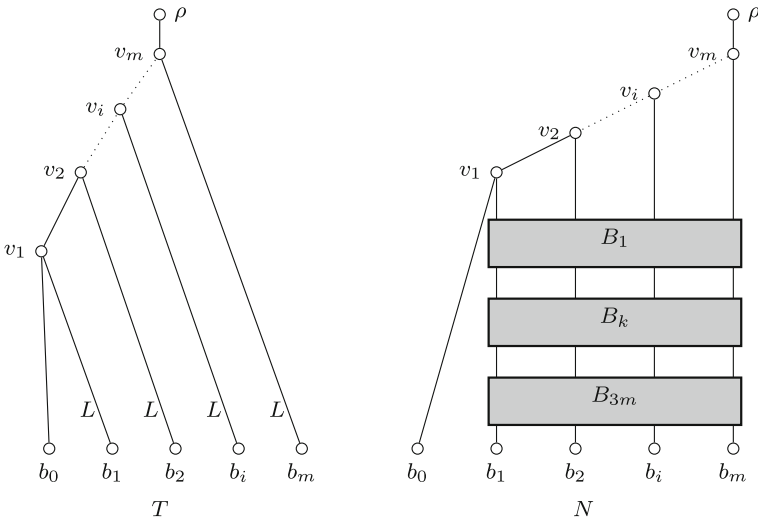
Fig. 2 The tree T and the network N used in the proof of Theorem 1. All arcs are directed downwards. The *dotted arcs* represent parts of the network which are not shown in details but which ensure connectivity. All arcs incident to leaves b_i of T, for $i \in [1 \dots m]$ have length $L = \Sigma + 6m^2 - 3m + 1$; and remaining arcs of T have length 1. All arcs of N have length 1, except in the $3m$ boxes B_k (see Fig. 3a for more details on the content of those $3m$ boxes B_k)

its final vertex b_0, and the ancestors of b_0, from the parent of b_0 to the child of ρ are called v_1 to v_m. Then, to each of the m vertices v_i for $i \in [1 \dots m]$ on this directed path, from bottom to top, we add an arc of length $L = \Sigma + 6m^2 - 3m + 1$ to a child, called b_i.

We now build a phylogenetic network N in the following way. We start by creating a copy of T but for each $i \in [1 \dots m]$ we remove the arc (v_i, b_i) and replace it by an arc of length 1 from v_i to a new vertex r_1^i (see Fig. 2). Then we create $3m$ subnetworks called B_k, for $k \in [1 \dots 3m]$, as described in Fig. 3. For ease of notation, we consider that vertex p_k^2 is also labeled p_k^1 and c_k^2 is also labeled c_k^1 for any $k \in [1 \dots 3m]$. Finally, we add arcs (b_k^i, r_{k+1}^i) of length 1 for each $k \in [1 \dots 3m - 1]$ and $i \in [1 \dots m]$ (to connect each B_k with B_{k+1}) and arcs of length 1 from b_{3m}^i to b_i for each $i \in [1 \dots m]$ to obtain N.

Suppose that S can be partitioned into m subsets of elements $S_1, S_2, \dots, S_m$ each of size 3, such that the sums of the numbers in each subset are all equal to Σ. We now prove that this implies that T and N constructed above constitute a positive instance of TCBL.

For each n_k, if it belongs to S_i then we remove from N all arcs (c_k^j, b_k^j) for $j \in [1 \dots m] - \{i\}$, as well as all arcs (r_k^j, p_k^j) for $j \in [1 \dots m] - \{i\} - \{1 \text{ if } i \neq 2\}$, the arc (r_k^i, b_k^i), and finally the arc (p_k^{i-1}, p_k^i) if $i \notin \{1, 2\}$. This way, we obtain a switching T' of N for T, shown in Fig. 3b.

In T', the only path from r_k^i to b_k^i goes through the arc (p_k^m, c_k^m) of length n_k, so the total length of this path is $2m - 2 + n_k$. For all other S_j, $j \in [1 \dots k] - \{i\}$, the only directed path from r_k^i to b_k^i is an arc of length $2m - 2$. Thanks to the arcs (b_k^j, r_{k+1}^j), for
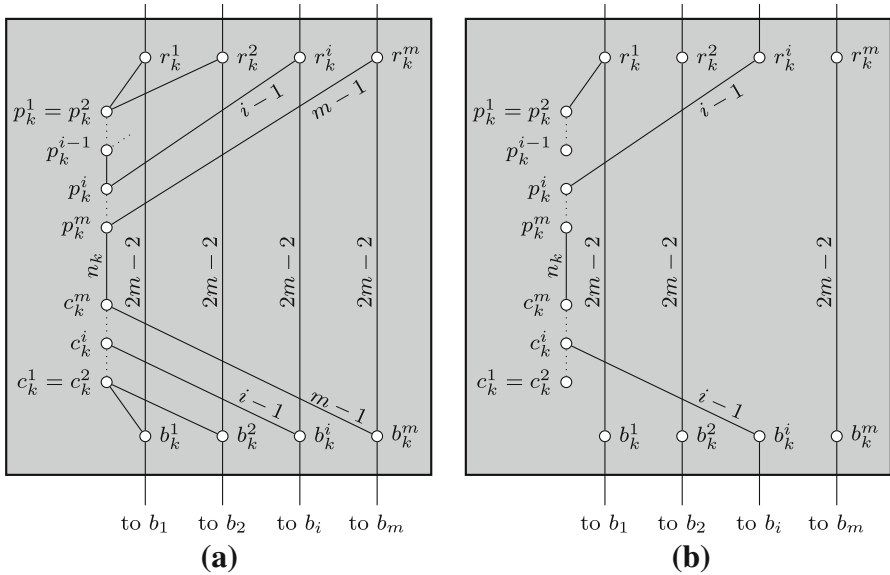
Fig. 3 The content of the box B_k (**a**) and a corresponding switching (**b**) of the network of Fig. 2. All arcs are directed downwards. The *dotted arcs* represent parts of the network which are not shown in details but which ensure connectivity. All arcs have length 1 except arcs (r_k^i, b_k^i) for $i \in [1 \ldots m]$ which have length $2m - 2$, arcs (r_k^i, p_k^i) and (c_k^i, b_k^i), for $i > 1$, which have length $i - 1$, and the arc (p_k^m, c_k^m) with length n_k

$j \in [1, m]$ a unique path can be found in T' from v_j to b_j. We can check that the lengths of the arcs of T leading to b_i with $i \in [1 \ldots m]$ are consistent with the lengths of these paths: the latter have all length $3m((2m-2)+1) + (\sum_{n_k \in S_i} n_k) + 1 = \Sigma + 6m^2 - 3m + 1$.

Furthermore, all other arcs of T (on the path from ρ to b_0) are also present in T' with the same configuration and length, meaning that, as we wished to prove, T is displayed by N taking into account lengths.

We now focus on the converse, supposing that the tree T is displayed by N taking into account lengths. We first note that any switching T' of N for T contains the vertices b_0, v_i for $i \in [1 \ldots m]$, ρ and the arcs between these vertices. Furthermore, T' also contains a path $P_i(T')$ from v_i to b_i, for each $i \in [1 \ldots m]$, of length L.

Claim 1: For any switching T' of N for T, for any $i \in [1 \ldots m]$ and $k \in [1 \ldots 3m]$, $r_k^i \in P_i(T')$ and $b_k^i \in P_i(T')$.

We prove it by induction on k. For $k = 1$, for all $i \in [1 \ldots m]$, vertex r_1^i has indegree 1 and its unique parent is contained in $P_i(T')$ so it is also contained in $P_i(T')$. As arc (p_1^m, c_1^m) belongs to all paths between p_1^i and c_1^j for $i, j \in [1 \ldots m]$, at most one of the paths $P_i(T')$ contains (p_1^m, c_1^m). If no such path exists then all paths $P_i(T')$ contain arc (r_1^i, b_1^i), so $b_1^i \in P_i(T')$. Otherwise, we denote by $P_{i_0}(T')$ the path containing (p_1^m, c_1^m). All other paths $P_i(T')$ for $i \in [1 \ldots m] - i_0$ contain arc (r_1^i, b_1^i),

so $b_1^i \in P_i(T')$. Because none of those paths contain $b_1^{i_0}$, we must have $b_1^{i_0} \in P_{i_0}(T')$. Therefore, for all $i \in [1 \ldots m]$, $b_1^i \in P_i(T')$.

Supposing vertices r_{k-1}^i and b_{k-1}^i belong to $P_i(T')$ for all $i \in [1 \ldots m]$, we can reproduce the proof above by replacing "1" by "k" each time we refer to b_1^i, c_1^i, p_1^i and r_1^i for any $i \in [1 \ldots m]$, in order to deduce that r_k^i and b_k^i belong to $P_i(T')$.

Claim 2: For any switching T' of N for T, for any $k \in [1 \ldots 3m]$, one of the paths $P_i(T')$ contains arc (p_k^m, c_k^m).

First, using Claim 1, we can consider each portion of the path $P_i(T')$ from r_k^i to b_k^i in T', and note that this portion has length $2m - 2 + n_k$ if $P_i(T')$ contains arc (p_k^m, c_k^m), or length $2m - 2$ otherwise.

Therefore, supposing by contradiction that there exists at least one $k_0 \in [1 \ldots 3m]$ such that none of the paths $P_i(T')$ contain arc $(p_{k_0}^m, c_{k_0}^m)$, then the cumulative length L_{k_0} of the portions of all paths $P_i(T')$ between $r_{k_0}^i$ and $b_{k_0}^i$, for $i \in [1 \ldots m]$, is $m(2m - 2)$. Therefore, summing the lengths of all these portions and the ones of arcs (b_k^i, r_{k+1}^i) between them as well as the ones of the arcs (v_i, r_1^i) and (b_{3m}^i, b_i) for any $i \in [1 \ldots m]$, the sum L' of the lengths of all paths $P_i(T')$ for $i \in [1 \ldots m]$ is at most $m + 3m(L_{k_0} + m) + (\sum_{k \in [1 \ldots 3m]} n_k) - n_{k_0} = m(6m^2 - 3m + 1 + \Sigma) - n_{k_0} = mL - n_{k_0}$. However, the sum L_T of the lengths of all arcs (v_i, b_i) of T is equal to mL so $L' < L_T$, meaning that T is not displayed by N taking into account lengths: contradiction.

Claim 3: for any switching T' of N for T, for any $i \in [1 \ldots m]$, there are exactly 3 arcs of the form (p_k^m, c_k^m) contained in $P_i(T')$.

We suppose by contradiction that there exists $i \in [1 \ldots m]$, and k_1, k_2, k_3 and $k_4 \in [1 \ldots 3m]$ such that $(p_{k_1}^m, c_{k_1}^m)$, $(p_{k_2}^m, c_{k_2}^m)$, $(p_{k_3}^m, c_{k_3}^m)$ and $(p_{k_4}^m, c_{k_4}^m)$ are contained in $P_i(T')$. Then, this path has length at least $n_{k_1} + n_{k_2} + n_{k_3} + n_{k_4} + 3m(2m - 2) + 3m + 1 > \Sigma + 3m(2m - 1) + 1$ because $n_i > \Sigma/4$ for all $i \in [1 \ldots 3m]$. So T' contains a path from v_i to b_i which is strictly longer than the arc from v_i to b_i in T, so T is not displayed by T', nor in N: contradiction.

Now, we suppose by contradiction that there exists $i \in [1 \ldots m]$ such that P_i contains at most 2 arcs of the form (p_k^m, c_k^m). Then, according to Claim 2, each of the the remaining $3m - 2$ arcs of the form (p_k^m, c_k^m) must be contained by one of the remaining $m - 1$ paths P_j for $j \in [1 \ldots m] - \{i\}$. So at least one of those paths must contain strictly more than 3 such arcs, which contradicts the previous paragraph: contradiction.

Finally, for any switching T' of N for T, the fact that T is displayed by N taking into account lengths, implies that the length of each arc (v_i, b_i) of T, $\Sigma + 6m^2 - 3m + 1$, equals the length of each path $P_i(T')$. Claim 2 and 3 imply that the arcs of the form (p_k^m, c_k^m) are partitioned into the paths $P_i(T')$, with each $P_i(T')$ containing exactly 3 such arcs. Denoting by n_{k_i}, $n_{k_i'}$ and $n_{k_i''}$ the length of such arcs, we obtain that the length of $P_i(T')$ equals $n_{k_i} + n_{k_i'} + n_{k_i''} + 6m^2 - 3m + 1$, therefore $n_{k_i} + n_{k_i'} + n_{k_i''} = \Sigma$, which implies that S can be partitioned into m subsets of elements $S_i = \{n_{k_i}, n_{k_i'}, n_{k_i''}\}$, such that the sums of the numbers in each subset S_i are all equal to Σ.

Finally, it is easy to see that the problem is in NP: a switching T' of the input network N is a polynomial size certificate of the fact that the input tree T is contained
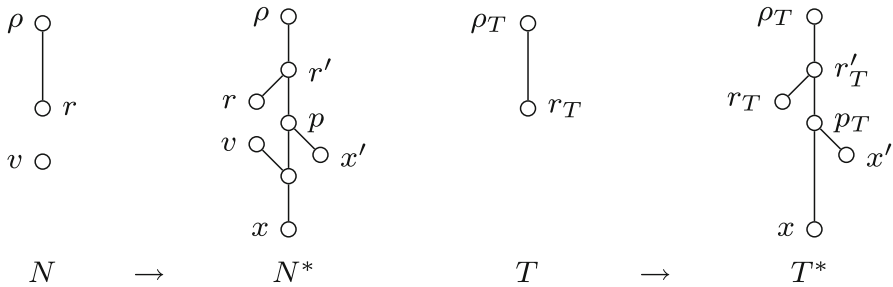
Fig. 4 How our slightly modified HANGLEAVES(v) modifies N and T. Vertices ρ and ρ_T are the roots of N and T, respectively. All arcs have length 1, except (r', r) of N^* which has the same length as (ρ, r) of N, (r'_T, r_T) of T^* which has the same length as (ρ_T, r_T) of T and (p_T, x) which has length 2

in N. We can check in polynomial time that T can be obtained from T' by applying dummy leaf deletions and vertex smoothings until possible, and checking that the obtained tree is isomorphic with T.   □

We note that Theorem 1 can be extended to binary tree-sibling (Cardona et al. 2008) time-consistent (Baroni et al. 2006) networks, by multiplying by 2 all arc lengths of the network constructed in the proof (in order to keep integer arc lengths even if those arcs are subdivided, which happens at most once), and using a gadget shown in Fig. 4, adapted from Fig. 4 of van Iersel et al. (2010) with arcs of length 1, and the operations described in the proof of Theorem 3 of the same article.

Corollary 1 RELAXED-TCBL *is strongly NP-complete, and* CLOSEST-TCBL *is strongly NP-hard.*

Proof TCBL can be easily reduced to both problems. Indeed, any instance of TCBL corresponds to an instance of RELAXED-TCBL with $m_T(a) = M_T(a) := \lambda_T(a)$ for each arc of T. Additionally, TCBL can be reduced to CLOSEST-TCBL by checking whether there exists a solution $\widetilde{T}$ with $\max |\lambda_T(a) - \lambda_{\widetilde{T}}(\tilde{a})| = 0$.   □

3.2 Weak NP-Completeness for Level-2 Networks

The strong NP-completeness result above does not imply anything about the hardness of TCBL on networks of bounded level. Unfortunately, TCBL is hard even for low-level networks, as we now show.

Theorem 2 *TCBL is weakly NP-complete for level-2 binary networks.*

Proof First, recall that TCBL is in NP (Theorem 1). To prove the theorem, we will reduce from the SUBSET SUM problem: given a multiset of positive integers $I = \{n_1, \ldots, n_k\}$ and a positive integer s, is there a non-empty subset of I whose sum is s? The SUBSET SUM problem is known to be weakly NP-complete.

Now, we show how to construct an instance of the TCBL problem with the required characteristics, for each instance of the SUBSET SUM problem. This can be done by
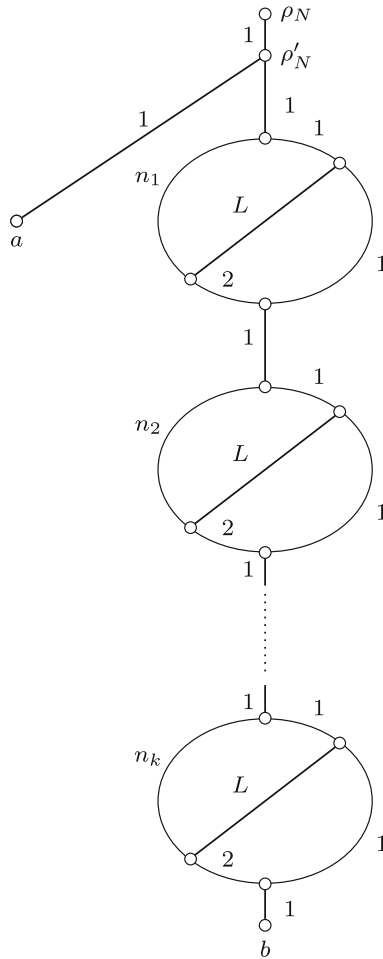
**Fig. 5** The network used in the proof of Theorem 2

defining the tree T and the network N as follows. The tree T is defined as the rooted tree on two leaves labeled a and b, parent ρ' and root ρ, and arcs (ρ, ρ'), (ρ', a) and (ρ', b), respectively, of length 1, 1 and $s + 3k + 1$. The network N is the network on the two leaves labeled a and b shown in Fig. 5, where $L > s + 3k + 1$. Then, it is easy to see that a positive instance of the TCBL problem gives a positive instance of the SUBSET SUM problem through the previous transformation, and vice versa. This is true because no switching S of N giving rise to T will ever contain the arcs with length L. Thus, the paths in S going through the blob containing the arc with length n_i can have either length 2 or $2 + n_i$. Now, any path from ρ'_N to the leaf labeled b has to go through all blobs, and through all arcs connecting these blobs. The sum of the lengths of the arcs on this path but outside the blobs is $k + 1$. Thus, there exists a path from ρ'_N to b with length $s + 3k + 1$ if and only if there is a non-empty subset of $I = \{n_1, \ldots, n_k\}$ whose sum is s.

As to the weakness of this NP-completeness result, we refer to Sect. 4.2, where we give a pseudo-polynomial algorithm for TCBL on any binary network of bounded level. □

4 Positive Results

4.1 TCBL is FPT in the Level of the Network When No Blob is Redundant

Note that in the weak NP-completeness result from Sect. 3.2 the blobs have only one outgoing arc each—that is, they are redundant. If we require that every blob has at least two outgoing arcs, then dynamic programming becomes possible, and the problem becomes much easier. The high-level reason for this as follows. Because blobs in the network N have at least two outgoing arcs, the image of any tree displayed by N will branch at least once inside each blob. This means that for each arc (u', v') of T, if the image of u' lies inside a blob B, then the image of v' either lies (i) also inside B or (ii) in one of the biconnected components C_i immediately underneath B. This last observation holds with or without arc lengths, but when taking lengths into account it has an extra significance. Indeed, suppose N displays T taking lengths into account, and S is a switching of N that induces the image of T inside N. Let (u', v') be an arc of T. If, within S, the image of u' lies inside a blob B and the image of v' lies inside a biconnected component C_i immediately underneath B, then the image of the arc (u', v')—a path—is naturally partitioned into 3 parts. Namely, a subpath inside B (starting at the image of u'), followed by an outgoing arc of B, followed by a subpath inside C_i (terminating at the image of v'). See Fig. 6 for an illustration. Within S, the
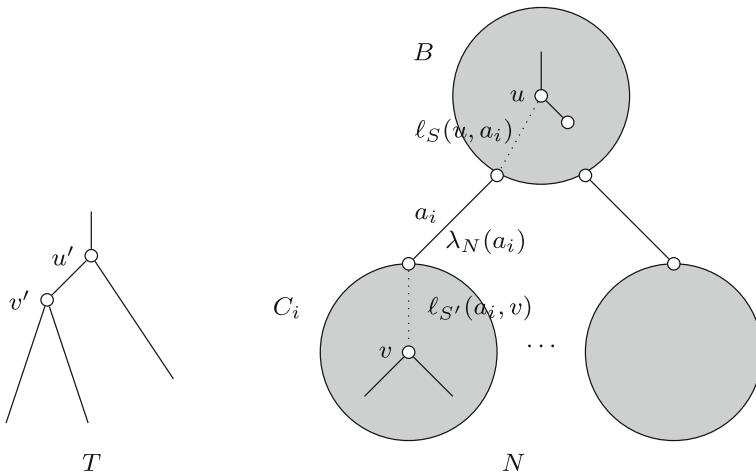
Fig. 6 Illustration of the idea at the basis of Algorithm 1. If a network N displays a tree T and the image u of u' (for an arc (u', v') of T) lies inside a blob B of N, then—assuming every blob of the network has at least two outgoing arcs—the image v of v' will either lie inside B, or inside a blob C_i that is immediately beneath B. In the latter case the image of (u', v') can be naturally partitioned into three parts, as shown. This is the foundation for the dynamic programming approach used in Theorem 3 and later in Theorems 5 and 6

lengths of these 3 parts must sum to $\lambda_T(u', v')$. The dynamic programming algorithm described below, in which we process the blobs in a bottom-up fashion, makes heavy use of this insight.

Theorem 3 *Let N be a level-k binary network and T be a rooted binary tree, both on X. If no blob of N is redundant, then TCBL can be solved in time $O(k \cdot 2^k \cdot n)$ using $O(k \cdot 2^k \cdot n)$ space, where $n = |X|$.*

Proof Firstly, note that networks can have nodes that are not inside blobs (i.e., tree-like regions). To unify the analysis, it is helpful to also regard such a node u (including when u is a taxon) as a blob with 0 reticulations: the definition of incoming and outgoing arcs extends without difficulty. Specifically, in this case, they will simply be the arcs incoming to and outgoing from u. We regard such blobs as having exactly one switching.

Next, it is easy to see that the blobs of N can themselves be organized as a rooted tree, known as the *blobbed-tree* (Gambette et al. 2009; Gusfield 2014). In other words, the parent-child relation between blobs is well-defined, and unique. The idea is to process the blobs in bottom-up, post-order fashion. Hence, if a blob B has blob children $C_1, C_2, \ldots$ (underneath outgoing arcs $a_1, a_2 \ldots$) we first process $C_1, C_2, \ldots$ and then B. Our goal is to identify some switching of B which can legitimately be merged with one switching each from $C_1, C_2, \ldots$. We initialize the dynamic programming by, for each blob B that is a taxon, recording that it has exactly one switching whose root-path has length 0. (The definition and meaning of root-path will be given in due course).

For each blob B that is not a taxon, we will loop through the (at most) 2^k ways to switch the reticulations within B. Some of these candidate switchings can be immediately discarded on topological grounds, i.e., such a switching of B induces bifurcations that are not present in T. Some other candidate switchings S can be discarded on the basis of the lengths of their internal paths, that is, the paths $u \to v$ entirely contained within S coinciding with the image of some arc (u', v') in T. Clearly the path $u \to v$ must have the same length as (u', v').

Finally, we need to check whether the candidate switching S can be combined with switchings from $C_1, C_2, \ldots$ such that arc lengths are correctly taken into account. This proceeds as follows. Observe that, for each outgoing arc a_i of B, a_i lies on the image of an arc (u', v') of T. This arc of T is uniquely defined. Let u be the image of u' in B, and let $\ell_S(u, a_i)$ be the total length of the path (in S) from u to the tail of a_i. The image of v' will lie somewhere inside C_i. For a switching S' of C_i, let v be the image of v' within S', and let $\ell_{S'}(a_i, v)$ be the total length of the path (in S') from the head of a_i to v. (See Fig. 6).

If we wish to combine S' with S, then we have to require $\lambda_T(u', v') = \ell_S(u, a_i) + \lambda_N(a_i) + \ell_{S'}(a_i, v)$. To know whether such an S' exists, B can ask C_i the question: "do you have a candidate switching S' such that $\ell_{S'}(a_i, v) = \lambda_T(u', v') - \ell_S(u, a_i) - \lambda_N(a_i)$?" This will be true if and only if C_i has a candidate switching S' such that the *root-path* in S'—defined as the path from the root of C_i to the first branching node of S'—has length exactly $\lambda_T(u', v') - \ell_S(u, a_i) - \lambda_N(a_i)$. (We consider a node of S' to be a branching node if it is the image of some node of T.) B queries all its children $C_1, C_2, \ldots$ in this way. If all the C_i answer affirmatively, then we store S,

together with the length of its root-path, as a candidate switching of B, otherwise we discard S.

This process is repeated until we have finished processing the highest blob B of N. The answer to TCBL is YES, if and only if this highest blob B has stored at least one candidate switching. Pseudocode formalizing the description above is provided in Algorithm 1.

We now analyze the running time and storage requirements. For step 1, observe that the blobbed-tree can easily be constructed once all the biconnected components of the undirected, underlying graph of N have been identified. The biconnected components can be found in linear time (in the size of the graph) using the well-known algorithm of Hopcroft and Tarjan (see, e.g., Cormen et al. 2001). Because every blob has at least two outgoing arcs, N will have $O(kn)$ vertices and arcs, (see, e.g., Lemma 4.5 in van Iersel (2009) and discussion thereafter) so the time to construct the blobbed-tree is at most $O(kn)$. Moreover, N has $O(n)$ blobs, meaning that the blobbed-tree has $O(n)$ nodes and that step 2 can be completed in $O(n)$ time by checking whether T and the blobbed-tree are compatible. (The compatibility of two trees can be tested in linear time Warnow 1994). Each blob has at most 2^k switchings, and each switch can be encoded in k bits. If we simply keep all the switchings in memory (which can be useful for constructing an actual switching of N, whenever the answer to TCBL is YES) then at most $O(k \cdot 2^k \cdot n)$ space is required.

For time complexity, each blob B loops through at most 2^k switchings, and for each switching S it is necessary to check the topological legitimacy of S [step 3(a)], that internal paths of the switching have the correct lengths [step 3(b)], and subsequently to make exactly one query to each of its child blobs C_i [step 3(c)]. We shall return to steps 3(a) and 3(b) in due course. It is helpful to count queries from the perspective of the blob that is queried. In the entire course of the algorithm, a blob will be queried at most 2^k times. Recalling that the number of blobs is $O(n)$, in total at most $O(2^k \cdot n)$ queries will be made, so the total time devoted to queries is $O(q \cdot 2^k \cdot n)$, where q is the time to answer each query. Recall that a query consists of checking whether a blob has a switching whose root-path has a given length. Each blob needs to store at most 2^k switchings. By storing these switchings (ranked by the lengths of their root-paths) in a balanced look-up structure (e.g., red-black trees) an incoming query can be answered in logarithmic time in the number of stored switchings, that is, in time $\log 2^k = k$. Hence, the total time spent on queries is $O(k \cdot 2^k \cdot n)$.

For steps 3(a) and 3(b) we require amortized analysis. Let $d^+(B)$ denote the number of outgoing arcs from blob B. The blob B can be viewed in isolation as a rooted phylogenetic network with $d^+(B)$ "taxa", so inside B there are $O(k \cdot d^+(B))$ vertices and arcs (van Iersel 2009). Therefore, the time to convert a switching S from B into a tree T' on $d^+(B)$ "taxa" (via dummy leaf deletions and vertex smoothings) is at most $O(k \cdot d^+(B))$. The topology and internal arc lengths of T' can be checked against those of the corresponding part of T in $O(d^+(B))$ time (Warnow 1994). Hence, the total time spent on steps 3(a) and 3(b) is

$$\sum_B O(2^k \cdot k \cdot d^+(B)), \tag{1}$$

where the sum ranges over all blobs. Note that $\sum_B d^+(B)$ is $O(n)$ because there are $O(n)$ blobs and each outgoing arc enters exactly one blob. Hence, the expression (1) is $O(k \cdot 2^k \cdot n)$, matching the time bound for the queries. Hence, the overall running time of the algorithm is $O(k \cdot 2^k \cdot n)$. □

Algorithm 1 FPT algorithm for TCBL on networks with no redundant blobs

1. Decompose N into blobs and construct the blobbed-tree T_N, whose nodes are the blobs in N and whose arcs are the arcs external to the blobs.
2. Check that T_N is compatible with the input tree T (in fact check that T_N can be obtained from T via arc contractions). If this is not the case, then terminate with a NO. Otherwise each vertex B in T_N can be obtained as the contraction of a subtree $T(B)$ of T, and each arc a in the blobbed-tree T_N originates from an arc a' in T. Store references to the a' and the $T(B)$.
3. **for each** blob B, in bottom-up order:
    for each switching S of B:
        (a) check that S is topologically compatible with $T(B)$;
        (b) check that each arc of $T(B)$ is as long as its image in S;
        (c) **for each** blob C_i that is a child of B, via the arc a_i:
            – check that C_i has stored a switching S' whose root-path has the appropriate length. Specifically, we require $\ell_{S'}(a_i, v) = \lambda_T(u', v') - \ell_S(u, a_i) - \lambda_N(a_i)$, where (u', v') is the arc of T on whose image a_i lies (i.e., a_i'), and u and v are the uniquely defined images of u' and v' in S and S', respectively.
        (d) if none of the checks above failed, store S along with the length of its root-path;
        (e) if no switching is stored for B, then terminate with NO, as no tree displayed by N satisfies the requirements.
4. If the algorithm gets this far, then it returns YES and the image in N of T can be obtained by combining a switching S stored for the root blob, to the switchings S' found for its child blobs, recursively.

4.2 Pseudo-Polynomial Solution of TCBL on Any Network of Bounded Level

Redundant blobs are problematic for TCBL because when they appear "in series" (as in Fig. 5) they give rise to an exponential explosion of paths that can be the images of an arc a in T, and, as we saw, checking the existence of a path of the appropriate length $\lambda_T(a)$ is at least as hard as SUBSET SUM. Just like for SUBSET SUM, however, a pseudo-polynomial time solution is possible, as we now show.

Theorem 4 *Let N be a level-k binary network with b blobs, and let T be a rooted binary tree on the same set of n taxa. TCBL can be solved in time $O(k \cdot b \cdot L + 2^k \cdot n \cdot L)$ using $O(k \cdot n \cdot L)$ space, where L is an upper bound on arc lengths in T.*

Proof The algorithm we now describe is based on the following two observations (we use here the same notational conventions as in Algorithm 1). First, if T is indeed displayed by N, the image $u \to v$ of any of its arcs (u', v') will either be entirely contained in one blob, or u and v will be in two different blobs, which can only be separated by redundant blobs. Second, it only makes sense to store a switching S' of a blob C_i, if $\ell_{S'}(a_i, v) < \lambda_T(u', v')$, i.e., if its root-leaf path is shorter than the corresponding arc in T, meaning that we only need to store $O(L)$ switchings per blob.

Accordingly, we modify Algorithm 1 as follows:

> *Step 2a:* Check that T_N is compatible with the input tree T in the following way: replace any chain of redundant blobs $M_1, M_2, \ldots, M_h$ in T_N with a single arc from the parent of M_1 to the child of M_h, and then check that the resulting blobbed-tree T'_N can be obtained from T via arc contractions. If this is not the case then terminate with a NO. Otherwise for each arc a and vertex B in T'_N, define and store a' and $T(B)$ as before.
>
> *Step 2b:* For each arc a in T'_N, calculate a set of lengths $L(a)$ as follows. If a originates from a chain of redundant blobs $M_1, M_2, \ldots, M_h$, then $L(a)$ is obtained by calculating the lengths of all paths in N starting with the incoming arc of M_1 and ending with the (unique) outgoing arc of M_h. Only keep the lengths that are smaller than $\lambda_T(a')$. For the remaining arcs in T'_N, simply set $L(a) := \{\lambda_N(a)\}$.

The algorithm only visits non-redundant blobs, performing a bottom-up traversal of T'_N, and doing the same as Algorithm 1 except for:

> *Step 3c:* for each blob C_i that is a child of B in T'_N :
>   − check the existence of an $\ell \in L(a_i)$ and a switching S' stored for C_i that satisfy:
>
> $$\ell_{S'}(a_i, v) = \lambda_T(u', v') - \ell_S(u, a_i) - \ell. \qquad (2)$$

To complete the description of the algorithm resulting from these changes, we assume that the switchings for a (non-redundant) blob B are stored in an array S_B indexed by the root-path length of the switching. If two or more switchings of a blob have the same root-path length ℓ, we only keep one of them in $S_B[\ell]$. Because for C_i we only store the switchings whose root-path length is less than $\lambda_T(a'_i)$, the S_B arrays have size $O(L)$.

As for step 2b above, the computation of $L(a)$ for an arc in T'_N corresponding to a chain of redundant blobs can be implemented in a number of ways. Here, we assume that the vertices in $M_1, M_2, \ldots, M_h$ are visited following a topological ordering, and that, for each visited vertex v, we fill a boolean array P_v of length $\lambda_T(a')$, where $P_v[\ell]$ is true if and only if there exists a path of length ℓ from the tail of the arc incoming M_1 to v. Once P_{v_h} for the head v_h of the arc outgoing M_h has been filled, $L(a)$ will then be equal to the set of indices ℓ for which $P_{v_h}[\ell]$ is true.

We are now ready to analyze the complexity of this algorithm. We start with the space complexity. First note that every redundant blob of level k in a binary network must have exactly $2k$ vertices (as the number of bifurcations must equal the number of reticulations). Because each redundant blob has $O(k)$ vertices, and each P_v array is stored in $O(L)$ space, step 2b requires $O(kL)$ space to process each redundant blob. Because every time a new redundant blob M_{i+1} is processed, the P_v arrays for the

vertices in M_i can be deleted, step 2b only requires $O(kL)$ space in total. This however is dominated by the space required to store $O(L)$ switchings for each non-redundant blob. Since there are $O(n)$ non-redundant blobs in N and each switching requires $O(k)$ bits to be represented, the space complexity of the algorithm is $O(k \cdot n \cdot L)$.

We now analyze the time complexity. Checking the compatibility of the blobbed-tree and T (step 2a) can be done in time $O(n + b)$, as this is the size of T'_N before replacing the chains of redundant blobs. The computation of the arrays P_v (step 2b) involves $O(L)$ operations per arc in $M_1, M_2, \ldots, M_h$. Because there are $O(b)$ redundant blobs, and because each of them contains $O(k)$ arcs, calculating all the P_v arrays requires time $O(k \cdot b \cdot L)$.

The other runtime-demanding operations are the queries in step 3c. These involve asking, for each $\ell \in L(a_i)$, whether C_i has a switching whose root-path has the length in Eq. (2). Each of these queries can be answered in constant time by checking whether $S_{C_i}[\lambda_T(u', v') - \ell_S(u, a_i) - \ell]$ is filled or not. Because every non-redundant blob C_i will be queried at most $2^k \cdot L(a_i)$ times, and because there are $O(n)$ non-redundant blobs, the total time devoted to these queries is $O(2^k \cdot n \cdot L)$. The remaining steps require the same time complexities as in Theorem 3. By adding up all these runtimes we obtain a total time complexity of $O(k \cdot b \cdot L + 2^k \cdot n \cdot L)$.   □

4.3 CLOSEST-TCBL and RELAXED-TCBL are FPT in the Level of the Network When No Blob is Redundant

We now show that Algorithm 1 can be adapted to solve the "noisy" variations of TCBL that we have introduced in the Preliminaries section.

Theorem 5 *Let N be a level-k binary network and T be a rooted binary tree, both on the same set of n taxa. The arcs of N are labeled by positive integer lengths, and the arcs of T are labeled by a minimum and a maximum positive integer length. If no blob of N is redundant, then RELAXED-TCBL can be solved in $O(k \cdot 2^k \cdot n)$ time and space.*

Proof We modify Algorithm 1 to allow some flexibility whenever a check on lengths is made: instead of testing for equality between arc lengths in the tree and the path lengths observed in the partial switching under consideration, we now check that the path length belongs to the input interval. Specifically, we modify two steps in Algorithm 1 as follows:

Step 3b: check that every arc (u', v') of $T(B)$, whose image is an internal path $u \to v$ of S, is such that $\ell_S(u, v) \in [m_T(u', v'), M_T(u', v')]$.
Step 3c: check that, among the switchings stored for C_i, there exists at least one switching S' whose root-path $\ell_{S'}(a_i, v)$ has a length in the appropriate interval. Specifically, using the same notation as in Algorithm 1 check that:

$$\ell_S(u, a_i) + \lambda_N(a_i) + \ell_{S'}(a_i, v) \quad \in \quad [m_T(a'_i), M_T(a'_i)],$$

that is:

$$m_T(a_i') - \ell_S(u, a_i) - \lambda_N(a_i) \leq \ell_{S'}(a_i, v) \leq M_T(a_i') - \ell_S(u, a_i) - \lambda_N(a_i).$$

(3)

We can use the same data structures used by Algorithm 1, so the space complexity remains $O(k \cdot 2^k \cdot n)$. As for time complexity, the only relevant difference is in step 3c: instead of querying about the existence of a switching with a definite root-path length, we now query about the existence of a switching whose root-path length falls within an interval (see Eq. (3)). In a balanced look-up structure, this query can be answered again in time $O(\log 2^k) = O(k)$. In conclusion the time complexity remains the same as that in Theorem 3, that is $O(k \cdot 2^k \cdot n)$. □

Theorem 6 *Let N be a level-k binary network and T be a rooted binary tree, both with positive integer arc lengths and on the same set of n taxa. If no blob of N is redundant, then* CLOSEST-TCBL *can be solved in time $O(2^{2k} \cdot n)$ using $O(k \cdot 2^k \cdot n)$ space.*

Proof As we shall see, we modify Algorithm 1 by removing all checks on arc lengths, and by keeping references to those switchings that may become part of an optimal solution in the end: any topologically viable switching S of a blob B is stored along with a reference, for each child blob C_i, to the switching S' that must be combined with S. Moreover, we compute recursively μ_S, which we define as follows:

$$\mu_S = \max \left| \lambda_T(a) - \lambda_{\widetilde{T}}(\tilde{a}) \right|,$$

where $\widetilde{T}$ is the subtree displayed by N obtained by (recursively) combining S to the switchings stored for its child blobs, and then applying dummy leaf deletions and vertex smoothings. The max is calculated over any arc $\tilde{a}$ in $\widetilde{T}$ and its corresponding arc a in T, excluding the root arc $\tilde{a}_r$ of $\widetilde{T}$ from this computation. This is because the length of the path above S, which must be combined with $\tilde{a}_r$, is unknown when S is defined.

In more detail, we modify Algorithm 1 as follows:

Step 3b: no check is made on the lengths of the internal paths of S; instead initialize μ_S as follows:

$$\mu_S := \max \left| \lambda_T(u', v') - \ell_S(u, v) \right|,$$

where the max is over all arcs (u', v') in $T(B)$, and u, v are the images of u', v' in S, respectively. Trivially, if B is just a vertex in N, the max above is over an empty set, meaning that μ_S can be initialized to any sufficiently small value (e.g., 0).

Step 3c: **for each** blob C_i that is a child of B:
    − among the switchings stored for C_i, seek the switching S' minimizing

$$\max \left\{ \mu_{S'}, \left| \ell_S(u, a_i) + \lambda_N(a_i) + \ell_{S'}(a_i, v) - \lambda_T(u', v') \right| \right\}$$

(4)

 — set μ_S to max $\{\mu_S,$ value of (4)for $S'\}$

Step 3d: store S along with μ_S, with the length of its root-path, and with references to the child switchings S' minimizing (4)

Step 4: seek the switching S stored for the root blob that has minimum μ_S, and combine it recursively to the switchings S' found for its child blobs. In the end, a switching $\widetilde{S}$ for the entire network N is obtained, which can be used to construct $\widetilde{T}$.

The correctness of the algorithm presented above is based on the following observation, allowing our dynamic programming solution of the problem:

Observation. Let B be a blob of N, and C_i be one of its child blobs. If a switching S of B is part of an optimal solution to CLOSEST-TCBL, then we can assume that S must be combined with a switching S' of C_i that minimizes Eq. (4). This means that even if there exists an optimal solution in which S is combined with S'', a non-minimal switching of C_i with respect to Eq. (4), then we can replace S'' with S' and the solution we obtain will still be optimal.

Once again, space complexity is $O(k \cdot 2^k \cdot n)$, as the only additional objects to store are the references to the child switchings of S, and the value of μ_S for each the $O(2^k \cdot n)$ stored switchings. As for time complexity, each query within step 3c now involves scanning the entire set of $O(2^k)$ switchings stored for C_i, thus taking time $O(2^k)$—whereas the previous algorithms only required $O(k)$ time. Since there are again $O(2^k \cdot n)$ queries to make, the running time is now $O(2^{2k} \cdot n)$. □

We conclude this section by noting that, if we reformulate CLOSEST-TCBL replacing the max with a sum, and taking any positive power of the absolute value $\left|\lambda_T(a) - \lambda_{\widetilde{T}}(\tilde{a})\right|$ in the objective function, then the resulting problem can still be solved in a way analogous to that described above.

Theorem 7 *Consider the class of minimization problems obtained from* CLOSEST-TCBL *by replacing its objective function with*

$$\biguplus_a \left|\lambda_T(a) - \lambda_{\widetilde{T}}(\tilde{a})\right|^d, \tag{5}$$

with $\biguplus$ representing either max *or* $\sum$, *and with* $d > 0$.

If no blob of N is redundant, then any of these problems can be solved in time $O(2^{2k} \cdot n)$ using $O(k \cdot 2^k \cdot n)$ space, where n is the number of taxa in N and T, and k is the level of N.

Proof In the proof of Theorem 6, replace every occurrence of $|\dots|$ with $|\dots|^d$, and—if $\biguplus$ represents a sum—replace every occurrence of max with $\sum$. □

It is worth pointing out that the algorithm described in the proofs above requires storing *all* switchings of a blob that are topologically compatible with the input tree. This is unlike the algorithms shown before, where a number of checks on arc lengths (quite stringent ones in the case of the algorithm for TCBL) ensure that, on realistic instances, the number of switchings stored for a blob with k reticulations will be much smaller than 2^k.

Moreover, again unlike the previous algorithms, the queries at step 3c involve considering *all* switchings stored for a child blob C_i, which is what causes the factor 2^{2k} in the runtime complexity. We note that, for certain objective functions, it might be possible to make this faster (with some algorithmic effort), but in order to achieve the generality necessary for Theorem 7, we have opted for the simple algorithm described above.

5 Discussion

In this paper, we have considered the problem of determining whether a tree is displayed by a phylogenetic network, when branch lengths are available. We have shown that, if the network is permitted to have redundant blobs (i.e., non-trivial biconnected components with only one outgoing arc), then the problem becomes hard when at least one of the following two conditions hold: (1) the level of the network is unbounded (Theorem 1), (2) branch lengths are potentially long (Theorem 2). If neither condition holds (i.e., branch lengths are short and level is bounded) then—even when redundancy is allowed—the problem becomes tractable (Theorem 4). We note that phylogenetic networks with redundant blobs are unlikely to be encountered in practice, as their reconstructability from real data is doubtful (Iersel and Moulton 2014; Huber et al. 2015; Pardi and Scornavacca 2015). This is relevant because, if redundant blobs are not permitted, the problem becomes fixed-parameter tractable in the level of the network (Theorem 3) *irrespective* of how long the branches are.

Building on our result on networks with no redundant blobs, we have then shown how the proposed strategy can be extended to solve a number of variants of the problem accounting for uncertainty in branch lengths. This includes the case where an interval of possible lengths is provided for each branch of the input tree (Theorem 5), and the case where we want to find—among all trees displayed by the network with the same topology as the input tree T—one that is closest to T, according to a number of measures of discrepancy between branch length assignments (Theorems 6 and 7).

The fixed-parameter algorithms we present here have runtimes and storage requirements that grow exponentially in the level of the network. However, in the case of storage, this is a worst-case scenario: in practice, this will depend on the number of "viable" switchings stored for each blob, that is, the switchings that pass all checks on topology and branch lengths. In the case of the algorithm for TCBL (Theorem 3), where strict equalities between arc lengths in T and path lengths in N must be verified, we can expect it to be very rare that multiple switchings will be stored for one blob. Similarly, in the case of the algorithm for RELAXED-TCBL (Theorem 5), when the input intervals are sufficiently small, we can expect the number of stored switchings to be limited. In some particular cases, it might even be possible to find the few viable switchings for a blob, without having to consider all $O(2^k)$ switchings, thus removing this factor from the runtime complexity as well.

The algorithm for TCBL (Algorithm 1) provides a good example of the effect of taking into account branch lengths in the tree containment problems: if all checks on branch lengths are removed, what is left is an algorithm that solves the classic (topology-only) tree containment problem, and also provides all ways to locate the

input tree in the network (for each blob, it can produce a list of possible images of the corresponding part of of the input tree). This algorithm may run a little faster than Algorithm 1 (as no queries to child blobs are necessary). However, for a small computational overhead, including branch lengths allows to locate more precisely the displayed trees, and provides more strict answers to the tree containment problem.

References

Abbott R, Albach D, Ansell S, Arntzen J, Baird S, Bierne N, Boughman J, Brelsford A, Buerkle C, Buggs R et al (2013) Hybridization and speciation. J Evolut Biol 26(2):229–246

Bapteste E, van Iersel L, Janke A, Kelchner S, Kelk S, McInerney JO, Morrison DA, Nakhleh L, Steel M, Stougie L et al (2013) Networks: expanding evolutionary thinking. Trends Genet 29(8):439–441

Baroni M, Semple C, Steel M (2006) Hybrids in real time. Syst Biol 55(1):46–56

Bordewich M, Tokac N (2016) An algorithm for reconstructing ultrametric tree-child networks from inter-taxa distances. Discrete Appl Math. doi:10.1016/j.dam.2016.05.011

Boto L (2010) Horizontal gene transfer in evolution: facts and challenges. Proc R Soc B Biol Sci 277(1683):819–827

Cardona G, Llabrés M, Rosselló F, Valiente G (2008) A distance metric for a class of tree-sibling phylogenetic networks. Bioinformatics 24(13):1481–1488

Chan HL, Jansson J, Lam TW, Yiu SM (2006) Reconstructing an ultrametric galled phylogenetic network from a distance matrix. J Bioinform Comput Biol 4(04):807–832

Choy C, Jansson J, Sadakane K, Sung WK (2005) Computing the maximum agreement of phylogenetic networks. Theor Comput Sci 335(1):93–107

Cordue P, Linz S, Semple C (2014) Phylogenetic networks that display a tree twice. Bull Math Biol 76(10):2664–2679

Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms. MIT Press, Cambridge

Doolittle WF (1999) Phylogenetic classification and the universal tree. Science 284:2124–2128

Downey RG, Fellows MR (2013) Fundamentals of parameterized complexity, vol 4. Springer, Berlin

Doyon JP, Scornavacca C, Gorbunov KY, Szöllösi GJ, Ranwez V, Berry V (2011) An efficient algorithm for gene/species trees parsimonious reconciliation with losses, duplications, and transfers. In: Proceedings of the eighth RECOMB comparative genomics satellite workshop (RECOMB-CG'10), LNCS, vol 6398, pp 93–108. Springer

Gambette P, Berry V, Paul C (2009) The structure of level-k phylogenetic networks. In: CPM09, LNCS, vol 5577, pp 289–300. Springer

Garey MR, Johnson DS (1975) Complexity results for multiprocessor scheduling under resource constraints. SIAM J Comput 4(4):397–411

Garey MR, Johnson DS (1979) Computers and intractability. W. H. Freeman and Co. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences

Gramm J, Nickelsen A, Tantau T (2008) Fixed-parameter algorithms in phylogenetics. Comput J 51(1):79–101

Gusfield D (2014) ReCombinatorics: the algorithmics of ancestral recombination graphs and explicit phylogenetic networks. MIT Press, Cambridge

Hotopp JCD (2011) Horizontal gene transfer between bacteria and animals. Trends Genet 27(4):157–163

Huber KT, van Iersel L, Moulton V, Wu T (2015) How much information is needed to infer reticulate evolutionary histories? Syst Biol 64(1):102–111

Huson DH, Rupp R, Scornavacca C (2010) Phylogenetic networks: concepts, algorithms and applications. Cambridge University Press, Cambridge

Huson DH, Scornavacca C (2011) A survey of combinatorial methods for phylogenetic networks. Genome Biol Evol 3:23–35

Jansson J, Sung WK (2006) Inferring a level-1 phylogenetic network from a dense set of rooted triplets. Theor Comput Sci 363(1):60–68

Kanj IA, Nakhleh L, Than C, Xia G (2008) Seeing the trees and their branches in the network is hard. Theor Comput Sci 401(1):153–164

Kubatko LS (2009) Identifying hybridization events in the presence of coalescence via model selection. Syst Biol 58(5):478–488

Mallet J (2007) Hybrid speciation. Nature 446(7133):279–283

Meng C, Kubatko LS (2009) Detecting hybrid speciation in the presence of incomplete lineage sorting using gene tree incongruence: a model. Theor Popul Biol 75(1):35–45

Morrison DA (2011) Introduction to Phylogenetic Networks. RJR Productions

Nolte AW, Tautz D (2010) Understanding the onset of hybrid speciation. Trends Genet 26(2):54–58

Pardi F, Scornavacca C (2015) Reconstructible phylogenetic networks: do not distinguish the indistinguishable. PLoS Comput Biol 11(4):e1004135

Posada D, Crandall KA, Holmes EC (2002) Recombination in evolutionary genomics. Annu Rev Genet 36(1):75–97

Rambaut A, Posada D, Crandall K, Holmes E (2004) The causes and consequences of HIV evolution. Nat Rev Genet 5(1):52–61

van Iersel L (2009) Algorithms, haplotypes and phylogenetic networks. Ph.D. thesis, Eindhoven University of Technology

van Iersel L, Moulton V (2014) Trinets encode tree-child and level-2 phylogenetic networks. J Math Biol 68(7):1707–1729

van Iersel L, Semple C, Steel M (2010) Locating a tree in a phylogenetic network. Inf Process Lett 110(23):1037–1043

Vuilleumier S, Bonhoeffer S (2015) Contribution of recombination to the evolutionary history of HIV. Curr Opin HIV AIDS 10(2):84–89

Warnow TJ (1994) Tree compatibility and inferring evolutionary history. J Algorithms 16(3):388–407

Yu Y, Degnan JH, Nakhleh L (2012) The probability of a gene tree topology within a phylogenetic network with applications to hybridization detection. PLoS Genet 8(4):e1002660

Yu Y, Dong J, Liu KJ, Nakhleh L (2014) Maximum likelihood inference of reticulate evolutionary histories. PNAS 111(46):16448–16453

Zhaxybayeva O, Doolittle WF (2011) Lateral gene transfer. Curr Biol 21(7):R242–R246