New FPT Algorithms for Finding the Temporal Hybridization Number for Sets of Phylogenetic Trees

Borst, Sander; van Iersel, Leo; Jones, Mark; Kelk, Steven

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

New FPT Algorithms for Finding the Temporal Hybridization Number for Sets of Phylogenetic Trees

Sander Borst[1] · Leo van Iersel[2] · Mark Jones[1] · Steven Kelk[3]

Abstract

We study the problem of finding a temporal hybridization network containing at most k reticulations, for an input consisting of a set of phylogenetic trees. First, we introduce an FPT algorithm for the problem on an arbitrary set of m binary trees with n leaves each with a running time of $O(5^k \cdot n \cdot m)$. We also present the concept of *temporal distance*, which is a measure for how close a tree-child network is to being temporal. Then we introduce an algorithm for computing a tree-child network with temporal distance at most d and at most k reticulations in $O((8k)^d 5^k \cdot k \cdot n \cdot m)$ time. Lastly, we introduce an $O(6^k k! \cdot k \cdot n^2)$ time algorithm for computing a temporal hybridization network for a set of two nonbinary trees. We also provide an implementation of all algorithms and an experimental analysis on their performance.

Keywords Parameterized algorithms · Phylogenetic networks · Phylogenetic trees · Hybridization number

✉ Leo van Iersel
  L.J.J.vanIersel@tudelft.nl

  Sander Borst
  Sander.Borst@cwi.nl

  Mark Jones
  markelliotlloyd@gmail.com

  Steven Kelk
  steven.kelk@maastrichtuniversity.nl

[1] Centrum Wiskunde & Informatica (CWI), P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

[2] Delft Institute of Applied Mathematics, Delft University of Technology, Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands

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

 Springer

1 Introduction

Phylogenetics is the study of the evolutionary history of biological species. Traditionally such a history is represented by a phylogenetic tree. However, hybridization and horizontal gene transfer, both so-called *reticulation* events, can lead to multiple seemingly conflicting trees representing the evolution of different parts of the genome [1, 2]. Directed acyclic networks can be used to combine these trees into a more complete representation of the history [3]. Reticulations are represented by vertices with in-degree greater than one.

Therefore, an important problem is how to construct such a network based on a set of input trees that are known to represent the evolutionary history for different parts of the genome. The network should display all of these input trees. In general there are many solutions to this problem, but in accordance with the parsimony principle we are especially interested in the most simple solutions to the problem. These are the solutions with a minimal number of reticulations. Finding a network for which the number of reticulations, also called the *hybridization number*, is minimal now becomes an optimization problem. This problem is NP-complete, even for only two binary input trees [4]. The problem is fixed parameter tractable for an arbitrary set of non-binary input trees if either the number of trees or the out-degree in the trees is bounded by a constant [5]. For a set of two binary input trees an FPT algorithm with a reasonable running time exists [6]. For more than two input trees theoretical FPT algorithms and practical heuristic algorithms exist, but no FPT algorithm with a reasonable running time is known. That is why we are interested in slightly modifying the problem to make it easier to solve.

One way to do this is by restricting the solution space to the class of tree-child networks, in which each non-leaf vertex has at least one outgoing arc that does not enter a reticulation [7]. The minimum hybridization number over all tree-child networks that display the input trees is called the *tree-child hybridization number*. These networks can be characterized by so-called cherry picking sequences [8]. This characterization can be used to create a fixed parameter tractable algorithm for this restricted version of the problem for any number of binary input trees with time complexity $O((8k)^k \cdot poly(n, m))$ where k is the tree-child hybridization number, n is the size of leaves and m is the number of input trees [9].

The solution space can be reduced even further [10], leading to the problem of finding the *temporal hybridization number*. The extra constraints enforce that each species can be placed at a certain point in time such that evolution events take a positive amount of time and that reticulation events can only happen between species that live at the same time. For the problem of computing the temporal hybridization number a cherry picking characterization exists too and it can be used to develop a fixed parameter tractable algorithm for problems with two binary input trees with time complexity $O((7k)^k \cdot poly(n, m))$ where k is the temporal hybridization number, n is the number of leaves and m is the number of input trees [10]. In this paper we introduce a faster algorithm for solving this problem in $O(5^k \cdot n \cdot m)$ time using the cherry picking characterization. Moreover, this algorithm works for any number of binary input trees.

A disadvantage of the temporal restrictions is that in some cases no solution satisfying the restrictions exists. In fact determining whether such a solution exists is an NP-hard problem [11, 12]. Because of this our algorithm will not find a solution network for all problem instances. However we show that it is possible to find a network with a minimum number of non-temporal arcs, thereby finding a network that is 'as temporal as possible'. For that reason we also introduce an algorithm that also works for non-temporal instances. This algorithm is a combination of the algorithm for tree-child networks and the one for temporal networks introduced here.

In practical data sets, the trees for parts of the genome are often non-binary. This can be either due to simultaneous divergence events or, more commonly, due to uncertainty in the order of divergence events [13]. This means that many real-world datasets contain non-binary trees, so it is very useful to have algorithms that allow for non-binary input trees. While the general hybridization number problem is known to be FPT when either the number of trees or the out-degree of the trees is bounded by a constant [5], an FPT algorithm with a reasonable running time ($O(6^k k! \cdot poly(n))$) is only known for an input of two trees [14]. Until recently no such algorithm was known for the temporal hybridization number problem however. In this paper the first FPT algorithm for constructing optimal temporal networks based on two non-binary input trees with running time $O(6^k k! \cdot k \cdot n^2)$ is introduced.
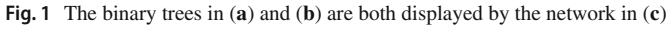
We implemented and tested all new algorithms [15].

The structure of the paper is as follows. First we introduce some common theory and notation in Sect. 2. In Sect. 3 we present a new algorithm for the temporal hybridization number of binary trees, prove its correctness and analyse the running time. In Sect. 4 we combine the algorithm from Sect. 3 with the algorithm from [9] to obtain an algorithm for constructing tree-child networks with a minimum number of non-temporal arcs. In Sect. 5 we present the algorithm for the temporal hybridization number for two non-binary trees. In Sect. 6 we conduct an experimental analysis of the algorithms.

2 Preliminaries

2.1 Trees

A *rooted binary phylogenetic X-tree* $\mathcal{T}$ is a rooted binary tree for which the leaf set is equal to X with $|X| = n$. Because we will mostly use rooted binary phylogenetic trees in this paper we will just refer to them as *trees*. Only in Sect. 5 trees that are not necessarily binary are mentioned, but we will explicitly call them non-binary trees.

Each of the leaves of a tree is an element of X. We will also refer to the set of leaves in $\mathcal{T}$ as $\mathcal{L}(\mathcal{T})$. For a tree $\mathcal{T}$ and a set of leaves A with the notation $\mathcal{T} \backslash A$ we refer to the tree obtained by removing all leaves that are in A from $\mathcal{T}$ and repeatedly contracting all vertices with both in- and out-degree one. Observe that $(\mathcal{T} \backslash \{x\}) \backslash \{y\} = \mathcal{T} \backslash \{x, y\} = (\mathcal{T} \backslash \{y\}) \backslash \{x\}$. We will often use T to refer to a set of m trees $\mathcal{T}_1, \ldots, \mathcal{T}_m$. We will write $T \backslash A$ for $\{\mathcal{T}_1 \backslash A, \ldots, \mathcal{T}_m \backslash A\}$ and $\mathcal{L}(T) = \cup_{i=1}^{m} \mathcal{L}(\mathcal{T}_i)$.

Fig. 1 The binary trees in (**a**) and (**b**) are both displayed by the network in (**c**)

2.2 Temporal Networks

A *network* on X is a rooted acyclic directed graph satisfying:

1. The root ρ has in-degree 0 and an out-degree not equal to 1.
2. The *leaves* are the nodes with out-degree zero. The set of leaves is X.
3. The remaining vertices are *tree vertices* or *hybridization vertices*

   (a) A tree vertex has in-degree 1 and out-degree at least 2.
   (b) A hybridization vertex (also called *reticulation*) has out-degree 1 and in-degree at least 2.

We will call the arcs ending in a hybridization vertex *hybridization arcs*. All other arcs are *tree arcs*. A network is a *tree-child* network if every tree vertex has at least one outgoing tree arc.

We say that a network $\mathcal{N}$ on X displays a set of trees T on X' with $X' \subseteq X$ if every tree in T can be obtained by removing edges and vertices and contracting vertices with both in-degree 1 and out-degree 1. For a set of leaves A we define $\mathcal{N} \backslash A$ to be the network obtained from $\mathcal{N}$ by removing all leaves in A and afterwards removing all nodes with out-degree zero and contracting all nodes with both in- and out-degree one.

For a tree-child network $\mathcal{N}$, the *hybridization number* $r(\mathcal{N})$ is defined as

$$r(\mathcal{N}) = \sum_{v \neq \rho} (d^-(v) - 1).$$

where $d^-(v)$ is the in-degree of a vertex v and ρ is the root of $\mathcal{N}$.

A tree-child network $\mathcal{N}$ with set of vertices V is *temporal* if there exists a map $t : V \to \mathbb{R}^+$, called a temporal labelling, such that for all $u, v \in V$ we have $t(u) = t(v)$ when (u, v) is a hybridization arc and $t(u) < t(v)$ when (u, v) is a tree arc. In Fig. 2 both a temporal and a non-temporal network are shown.
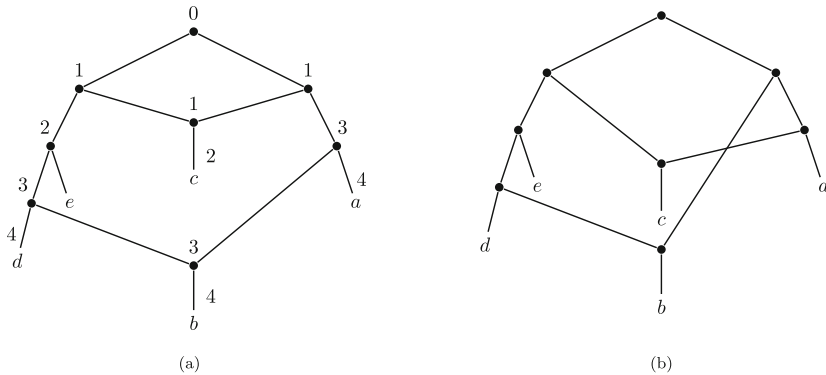
Fig. 2 a A temporal labeling is shown in the network above, asserting that the network is temporal. **b** No temporal labeling exists for this network. Therefore the network is not temporal
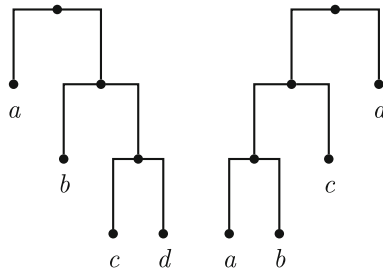

Fig. 3 No temporal network that displays these trees exists

For a set of trees T we define the minimum temporal-hybridization number as

$$h_t(T) = \min\{r(\mathcal{N}) : \mathcal{N} \text{ is a temporal network that displays } T\}$$

This definition leads to the following decision problem.

Temporal hybridization

   Instance: A set of trees T and an integer k

   Question: Is $h_t(T) \leq k$?

   Note that there are sets of trees such that no temporal network exists that displays them. In Fig. 3 an example is given. For such a set T we have $h_t(T) = \infty$.

2.3 Cherry Picking Sequences

Temporal networks can now be characterized by so-called cherry-picking sequences [10]. A *cherry* is the set of children of a tree vertex that only has leaves as children. So for binary trees a cherry is a pair of leaves. We will write $(a, b) \in \mathcal{T}$ if $\{a, b\}$ is a *cherry* of $\mathcal{T}$ and $(a, b) \in T$ if there is a $\mathcal{T} \in T$ with $(a, b) \in \mathcal{T}$. First we introduce some notation to make it easier to speak about cherries.

Definition 2.1 For a set of binary trees T on with the same leaf set define $H(T)$ to be the set of leaves that is in a cherry in every tree.

If two leaves are in a cherry together we call them *neighbors*. We also introduce notation to speak about the *neighbors* of a given leaf:

Definition 2.2 Define $N_T(x) = \{y \in \mathcal{X} : (y, x) \in \mathcal{T}\}$. For a set of trees T define $N_T(x) = \cup_{\mathcal{T} \in T} N_\mathcal{T}(x)$.

Definition 2.3 For a set of binary trees T containing a leaf x define $w_T(x) = |N_T(x)| - 1$. We will also call this the *weight* of x in T.

Using this theory, we can now give the definition of cherry picking sequences.

Definition 2.4 A sequence of leaves $s = (s_1, s_2, \ldots, s_n)$ is a *cherry picking sequence* (CPS) for a set of binary trees T with the same leaf set if it contains all leaves of T exactly once and if for all $i \in [n-1]$ we have $s_i \in H(T \backslash \{s_1, \ldots, s_{i-1}\})$. The weight $w_T(s_1, \ldots s_n)$ of the sequence is defined as $w_T(s) = \sum_{i=1}^{n-1} w_{T \backslash \{s_1, \ldots, s_{i-1}\}}(s_i)$.

Example 2.5 For the two trees in Fig. 1, ($\boldsymbol{b}$, e, $\boldsymbol{c}$, d, a) is a minimum weight cherry-picking sequence of weight 2. Leaves b and c (indicated in bold) have weight 1 and the rest of the leaves have weight 0 in the sequence.

For a cherry picking sequence s with $s_i = x$ we say that x is *picked* in s at index i.

Theorem 2.6 *([10,Theorem 1, Theorem 2]) Let T be a set of trees on $\mathcal{X}$. There exists a temporal network $\mathcal{N}$ that displays T with $h_t(\mathcal{N}) = k$ if and only if there exists a cherry-picking sequence s for T with $w_T(s) = k$.*

This has been proven in [10,Theorem 1, Theorem 2]. The proof works by constructing a cherry picking sequence from a temporal network and vice versa. Here, we only repeat the construction to aid the reader, and refer to [10] for the proof of correctness.

The construction of cherry picking sequence s from a temporal network $\mathcal{N}$ with temporal labeling t works in the following way: For $i = 1$ choose s_i to be a leaf x of $\mathcal{N}$ such that $t(p_x)$ is maximal where p_x is the parent of x in $\mathcal{N}$. Then increase i by one and again choose s_i to be a leaf x of $\mathcal{N} \backslash \{s_1, \ldots, s_{i-1}\}$ that maximizes $t(p_x)$ where p_x is the parent of x in $\mathcal{N} \backslash \{s_1, \ldots, s_{i-1}\}$. In [10,Theorem 1, Theorem 2] it is shown that now s is a cherry picking sequence with $w_T(s) = r(\mathcal{N})$.

The construction of a temporal network $\mathcal{N}$ from a cherry picking s is somewhat more technical: for cherry picking sequence $s_1, \ldots, s_t$, define $\mathcal{N}_n$ to be the tree, only consisting of a root and leaf s_n Now obtain $\mathcal{N}_i$ from $\mathcal{N}_{i+1}$ by adding node s_i and a new node p_{s_i}, adding edge (p_{s_i}, s_i) subdividing (p_x, x) for every $x \in N_{T \backslash \{s_1, \ldots, s_{i-1}\}}(s_i)$ with node q_x and adding an edge (q_x, p_{s_i}) and finally suppressing all nodes with in- and out-degree one. Then $\mathcal{N} = \mathcal{N}_1$ displays T and $r(\mathcal{N}) = w_T(s)$.

The theorem implies that the weight of a minimum weight CPS is equal to the temporal hybridization number of the trees. Because finding an optimal temporal reticulation network for a set of trees is an NP-hard problem [11], this implies that finding a minimum weight CPS is an NP-hard problem.
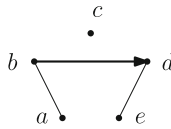
Fig. 4 An example showing the neighbour relation for the trees in Fig. 1, together with a constraint (b, d). Two elements $x, y \in X$ are depicted as adjacent if $x \in N_T(y)$ i.e. if x and y appear in a cherry together. An arc from x to y indicates the presence of a constraint (x, y)

Definition 2.7 We call two sets of trees T and T' *equivalent* if there exists a bijection from $\mathcal{L}(T)$ to $\mathcal{L}(T')$ such that applying it to the leaves of T maps T to T'. We call them equivalent because they have the same structure and consequently the same (temporal-) hybridization number, however the biological interpretation can be different.

3 Algorithm for Constructing Temporal Networks from Binary Trees

Finding a cherry picking sequence comes down to deciding in which order to pick the leaves. Our algorithm relies on the observation that this order does not always matter. Intuitively the observation is that the order of two leaves in a cherry picking sequence only matters if they appear in a cherry together somewhere during the execution of the sequence. Therefore the algorithm keeps track of the pairs of leaves for which the order of picking matters. We will make this more precise in the remainder of this section. The algorithm now works by branching on the choice of which element of a pair to pick first. These choices are stored in a so-called constraint set. Each call to the algorithm branches into subcalls with more constraints added to the constraint set. As soon as it is known that a certain leaf has to be picked before all of its neighbors and is in a cherry in all of the trees, the leaf can be picked.

Definition 3.1 Let $C \subseteq \mathcal{L}(T) \times \mathcal{L}(T)$. We call C a *constraint set* on T if every pair $(a, b) \in C$ is a cherry in T. A cherry picking sequence $s = (s_1, \ldots, s_k)$ of T *satisfies* C if for all $(a, b) \in C$, we have $s_i = a$ and $(a, b) \in T'$ and $w_{T'}(a) > 0$ with $T' = T \backslash \{s_1, \ldots, s_{i-1}\}$ for some i.

Intuitively, a cherry picking sequence satisfies a constraint set if for every pair (a, b) in the set a is picked with positive weight and (a, b) is a cherry just before picking a. This implies that a occurs in the cherry picking sequence before b.

We now prove a series of results about what sets of constraints are valid, which will then be used to guide our algorithm.

Observation 3.2 *Let s be a cherry picking sequence for T and $w_T(x) > 0$ and $a, b \in N_T(x)$. Then s satisfies one of the following constraint sets:*
$$\{(a, x)\}, \{(b, x)\}, \{(x, a), (x, b)\}.$$

Proof Let i be the lowest index such that $s_i \in \{x, a, b\}$. If $s_i = x$, then $(x, a) \in T \backslash \{s_1, \ldots, s_{i-1}\}$ and $(x, b) \in T \backslash \{s_1, \ldots, s_{i-1}\}$, so s satisfies $\{(x, a), (x, b)\}$. If $s_i = a$, then there is a $\mathcal{T} \in T \backslash \{s_1, \ldots, s_{i-1}\}$ with $(x, b) \in \mathcal{T}$, so $(a, x) \notin \mathcal{T}$, which implies that $w_{T \backslash \{s_1, \ldots, s_{i-1}\}}(s_i) > 0$, so s satisfies $\{(a, x)\}$. Similarly if $s_i = b$ then s satisfies $\{(b, x)\}$. $\qquad \square$
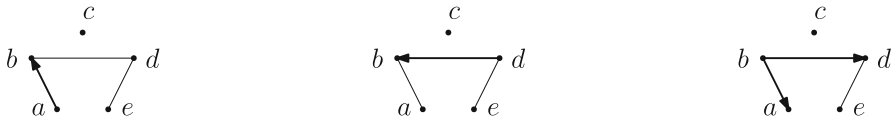
Fig. 5 Illustration of Example 3.3, showing the possible constraint sets on a, b, d implied by Observation 3.2

Fig. 6 Illustration of Example 3.5, showing the two possible constraints on d and e implied by Observation 3.4, in the case that there already exists a constraint $(d, b) \in C$ and thus $d \in \pi_1(C)$

Example 3.3 The trees in Fig. 1a, b contain the cherries (a, b) and (d, b). So by Observation 3.2 every cherry picking sequence for these trees satisfies one of the constraint sets $\{(a, b)\}$, $\{(d, b)\}$, $\{(b, a), (b, d)\}$. For example, (**b**, d, **c**, e, a) is a cherry picking sequence of weight 2 for these trees. This sequence satisfies the constraint set $\{(b, a), (b, d)\}$. See Figure 5.

This observation implies that the problem can be reduced to three subproblems, corresponding to either appending $\{(a, x)\}$, $\{(b, x)\}$ or $\{(x, a), (x, b)\}$ to C. As we will see, this is used by the algorithm. It is possible to implement an algorithm using only this rule, but the running time of the algorithm can be improved by using a second rule that branches into only two subproblems when it is applicable. The rule relies on the following observation. Here we will write $\pi_i(C) = \{c_i : (c_1, c_2) \in C\}$.

Observation 3.4 *If C is satisfied by s then for all $x \in \pi_1(C)$ and $y \in N_T(x)$ we have that either $C \cup \{(y, x)\}$ or $C \cup \{(x, y)\}$ is also satisfied by s.*

Proof If $x \in \pi_1(C)$ then C contains a pair (x, a). If $a = y$ it is trivial that s satisfies $C \cup \{(x, y)\} = C$. Otherwise Observation 3.2 implies that s satisfies one of the constraint sets $\{(a, x)\}$, $\{(y, x)\}$, $\{(x, a), (x, y)\}$. Because s satisfies $\{(x, a)\}$, s can not satisfy $\{(a, x)\}$. So s will satisfy either $\{(y, x)\}$ or $\{(x, a), (x, y)\}$. □

Using this observation we can let the algorithm branch into two paths by either adding (x, y) or (y, x) to the constraint set C if $x \in \pi_1(C)$.

Example 3.5 Consider again the situation in Example 3.3. Suppose we guess that the solution satisfies the constraint set $\{(d, b)\}$. Then we have $d \in \pi_1(C)$. Hence, we are in the situation of Observation 3.4 and we can conclude that either (d, e) or (e, d) can be added to the constraint set C. See Fig. 6.

We define $G(T, C)$ to be the set of cherries for which there is no constraint in C, so $G(T, C) = \{(x, y) : (x, y) \in T \wedge (x, y), (y, x) \notin C\}$. Observe that $(x, y) \in G(T, C)$ is equivalent with $(y, x) \in G(T, C)$.

Before proving the next result about constraints, we need the following lemma. It states that if we have a set of trees, a leaf that is in a cherry in all of the trees and a

corresponding cherry picking sequence then the following holds: for every element in a cherry picking sequence, we can either move it to the front of the sequence without affecting the weight of the sequence or there is a neighbor of this element that occurs earlier in the sequence.

Lemma 3.6 *Let $(s_1, s_2, \ldots)$ be a cherry picking sequence for a set of trees T that satisfies constraint set C. Let $x \in H(T)$. Then at least one of the following statements is true:*

(1) There exists an i such that $s_i = x$ and $s' = (s_i, s_1, \ldots, s_{i-1}, s_{i+1}, \ldots)$ is a cherry picking sequence for T satisfying C and $w(s) = w(s')$.
(2) If $s_i = x$ then there exists a j such that $s_j \in N_T(x)$ and $j < i$.

Proof Let r be the smallest number such that $s_r \in N_T(x) \cup \{x\}$. In case $s_r \neq x$ it follows directly that condition (2) holds for $j = r$. For $s_r = x$ we will prove that condition (1) holds with $i = r$. The key idea is that, because s_i is not in a cherry with any of $s_1, \ldots, s_{i-1}$, removing s_i first will not have any effect on the cherries involving $s_1, \ldots, s_{i-1}$.

More formally, take an arbitrary tree $\mathcal{T} \in T$. Now take arbitrary j, k with $s'_j = s_k$. Now we claim that for an arbitrary z we have $(s'_j, z) \in \mathcal{T} \backslash \{s'_1, \ldots, s'_{j-1}\}$ if and only if $(s_k, z) \in \mathcal{T} \backslash \{s_1, \ldots, s_{k-1}\}$.

For $s'_j = s'_1 = s_i = s_k$ this is true because none of the elements $s_1, \ldots, s_{i-1}$ are in $N_T(s_i)$ so for each z we have $(s'_1, z) \in \mathcal{T}$ if and only if $(s_i, z) \in \mathcal{T} \backslash \{s_1, \ldots, s_{i-1}\}$.

For k with $k < i$ we have $s'_{j+1} = s_j$. Because $s_i \notin N_T(s_j)$ we have that $(s_j, z) \in \mathcal{T} \backslash \{s'_1, \ldots, s'_j\} = \{s_1, \ldots, s_{j-1}, s_i\}$ if and only if $(s_j, z) \in \mathcal{T} \backslash \{s_1, \ldots, s_{j-1}\}$.

For $k > i$ we have $j = k$ and also $\mathcal{T} \backslash \{s'_1, \ldots s'_{j-1}\} = \mathcal{T} \backslash \{s_1, \ldots s_{j-1}\}$ because $\{s_1, \ldots s_{j-1}\} = \{s'_1, \ldots s'_{j-1}\}$. It directly follows that $(s'_j, z) \in \mathcal{T} \backslash \{s'_1, \ldots s'_{j-1}\}$ if and only if $(s_j, z) \in \mathcal{T} \backslash \{s_1, \ldots s_{j-1}\}$.

Now because we know that for each k we have $s_k \in H(T \backslash \{s_1, \ldots, s_{k-1}\})$ and $s_k = s'_j$ is in exactly the same cherries in $T \backslash \{s_1, \ldots, s_{k-1}\}$ as in $T \backslash \{s'_1, \ldots, s'_{j-1}\}$, we know that $s'_j \in H(T \backslash \{s'_1, \ldots, s'_{j-1}\})$, that $w_{T \backslash \{s'_1, \ldots, s'_{j-1}\}}(s'_j) = w_{T \backslash \{s_1, \ldots, s_{k-1}\}}(s_k)$ and that s' satisfies C. This implies that s' is a CPS with $w_T(s) = w_T(s')$. □

As soon as we know that a leaf in $H(T)$ has to be picked before all its neighbors we can pick it, as stated by the following lemma.

Lemma 3.7 *Suppose $x \in H(T)$ and constraint set C is satisfied by cherry picking sequence s of T, with $\{(x, n) : n \in N_T(x)\} \subseteq C$. Then there is a cherry picking sequence s' with $s'_1 = x$ and $w(s') = w(s)$.*

Proof This follows from Lemma [3.6], because statement (2) can not be true because for every j with $s_j \in N_T(x)$ we have $(x, s_j) \in C$ and therefore $i < j$ for $s_i = x$. So statement (1) has to hold which yields a sequence s' with $w(s) = w(s')$ and $s'_1 = x$. □

The following lemma shows that we can also safely remove all leaves that are in a cherry with the same leaf in every tree.

Lemma 3.8 *Let s be a cherry picking sequence for T satisfying constraint set C with $x \notin \pi_1(C)$ and $x \notin \pi_2(C)$. If $x \in H(T)$ and $w_T(x) = 0$, then there is a cherry picking sequence s' with $s'_1 = x$ and $w(s') = w(s)$ satisfying C.*

Proof Because $w_T(x) = 0$ we have $N_T(x) = \{y\}$. Then from Lemma 3.6 it follows that a sequence s' exists such that either $s'' = (x)|s'$ or $s'' = (y)|s'$ is a cherry picking sequence for T and $w_T(s'') = w(s)$ and s'' satisfies C. However, because the position of x and y in the trees are equivalent (i.e. swapping x and y does not change T) both are true. □

We are almost ready to describe our algorithm. There is one final piece to introduce first: the measure $P(C)$. This is a measure on a set of constraints C, which will be used to provide a termination condition for our algorithm. We show below that $P(C)$ provides a lower bound on the weight of any cherry picking sequence satisfying C, and so if during any recursive call to the algorithm $P(C)$ is greater than the desired weight, we may stop that call.

Definition 3.9 Let $\psi = \frac{\log(2)}{\log(5)} \simeq 0.4307$. Let $P(C) = \psi \cdot |C| + (1 - 2\psi)|\pi_1(C)|$.

Lemma 3.10 *If cherry picking sequence s for T satisfies C, then $w_T(s) \geq P(C)$.*

Proof For $x = s_i$ with $i < n$ we prove that for $C_x := \{(a, b) : (a, b) \in C \wedge a = x\}$ we have $w_{T \backslash \{s_1, \ldots, s_{i-1}\}}(x) \geq P(C_x)$. If $|C_x| = 0$, then $P(C_x) = 0$ and the inequality is trivial. If $|C_x| = 1$, then there is some $(x, b) \in C$, which implies that $w_{T \backslash \{s_1, \ldots, s_{i-1}\}}(x) > 0$, so $w_{T \backslash \{s_1, \ldots, s_{i-1}\}}(x) \geq |\pi_1(C_x)| = 1 \geq P(C)$. Otherwise if $|C_x| \geq 2$, then $w_{T \backslash \{s_1, \ldots, s_{i-1}\}}(x) = N_T(x) - 1 \geq |C_x| - 1 = \psi \cdot |C_x| - 1 + (1 - \psi)|C_x| \geq \psi \cdot |C_x| - 1 + 2(1 - \psi) = \psi \cdot |C_x| + (1 - 2\psi) = P(C_x)$. Now the result follows because $w_T(s) = \sum_{i=1}^{n-1} w_{T \backslash \{s_1, \ldots, s_{i-1}\}}(s_i) \geq \sum_{i=1}^{n-1} P(C_{s_i}) = P(C)$. □

We now present our algorithm, which we split into two parts. The main algorithm is `CherryPicking`, a recursive algorithm which takes as input parameters a set of trees T, a desired weight k and a set of constraints C, and returns a non-empty set of cherry picking sequences for T of weight at most k satisfying C, if they exist.

The second part is the procedure `Pick`. In this procedure zero-weight cherries and cherries for which all neighbors are contained in the constraint set are greedily removed from the trees.

3.1 Proof of Correctness

In this section a proof of correctness will be given. First some properties of the auxiliary procedure `Pick` are proven.

Observation 3.11 *Suppose `Pick`(T', k', C') returns (T, k, C, p).*

1. *There are no $x \in H(T)$ with $w_T(x) = 0$.*
2. *There are no $x \in H(T)$ with $\{(x_i, n) : n \in N_{T^{(i-1)}}(x_i)\} \subseteq C$.*

Lemma 3.12 *(Correctness of `Pick`) Suppose `Pick`(T', k', C') returns (T, k, C, p).*

1. *If a cherry picking sequence s of weight at most k for T that satisfies C exists then a cherry picking sequence s' of weight at most k' for T' that satisfies C' exists.*
2. *If s is a cherry picking sequence of weight at most k for T that satisfies C then $p|s$ is a cherry picking sequence for T' of weight at most k' and satisfying C'.*

Algorithm 1

```
1: procedure CHERRYPICKING(T, k, C)
2:    if k − P(C) < 0 then
3:        return ∅
4:    end if
5:    T′, k′, C′, p ←PICK(T, k, C)
6:    if |L(T′)| = 1 then
7:        return {p}
8:    else if π₁(C′) ⊄ L(T′) then
9:        return ∅
10:   else if k′ − P(C′) ≤ 0 then
11:       return ∅
12:   end if
13:
14:   R ← ∅
15:   if ∃(x, y) ∈ G(T′, C′) : w_T(x) > 0 ∧ x ∈ π₁(C′) then
16:       R ← R∪ CHERRYPICKING(T′,k′,C′ ∪ {(x, y)})
17:       R ← R∪ CHERRYPICKING(T′,k′,C′ ∪ {(y, x)})
18:   else if ∃(x, a) ∈ G(T′, C′) : w_T′(x) > 0 ∧ x ∉ π₂(C′) then
19:       Choose b ≠ a such that (x, b) ∈ G(T′, C′)
20:       R ← R∪ CHERRYPICKING(T′,k′,C′ ∪ {(a, x)})
21:       R ← R∪ CHERRYPICKING(T′,k′,C′ ∪ {(b, x)})
22:       R ← R∪ CHERRYPICKING(T′,k′,C′ ∪ {(x, a), (x, b)})
23:   end if
24:   return {p|r : r ∈ R}
25: end procedure
```

Algorithm 2

```
1: procedure PICK(T′, k′, C′)
2:    (T^(0), k₁, C₁) ← (T′, k′, C′)
3:    p^(0) ← ()
4:    i ← 1
5:    while ∃xᵢ ∈ H(T^(i−1)) : w_{T^(i−1)}(x) = 0 ∨ {(xᵢ, n) : n ∈ N_{T^(i−1)}(xᵢ)} ⊆ C_{i−1} do
6:        p^(i) ← p^(i−1)|(xᵢ)
7:        kᵢ ← k_{i−1} − w_{T^(i−1)}(xᵢ)
8:        T^(i) ← T^(i−1)\{xᵢ}
9:        Cᵢ ← {(a, b) ∈ C_{i−1} : a ≠ xᵢ}
10:       i ← i + 1
11:   end while
12:   return T^(i−1), k_{i−1}, C_{i−1}, p^(i−1)
13: end procedure
```

Proof We will prove the first claim for $(T, k, C, p) = (T^{(i)}, k_i, C_i, p^{(i)})$ for all i defined in Pick. We will prove this with induction on i. For $i = 1$ this is obvious because $T^{(1)} = T$, $p^{(1)} = ()$, $C_1 = C$ and $k_1 = k$.

Now assume the claim is true for $i = i'$. Now there are two cases to consider:

– If we have $\{(x_{i'}, n) : n \in N_T(x_{i'})\} \subseteq C_{i'}$ we know from Lemma 3.7 that if a cherry picking sequence s satisfying C_i exists then also a cherry picking sequence $(x)|s'$ that satisfies C' exists with $w(p|(x)|s') = w(p|s)$. Note that this implies that s' is a cherry picking sequence for $T^{(i+1)} = T'\setminus\{x\}$, that $C_{i+1} = c \in C' : x \notin \{c_1, c_2\}$

is satisfied by s_{i+1} and that $w(s_{i+1}) = w(s_i) - w_{T^{(i)}}(x_i) = k_i - w_{T^{(i)}}(x)$. So this proves the statement for $i = i' + 1$.

– Otherwise we have $w_{T^{(i')}}(x) = 0$ and $x \notin \pi_1(C)$ and $x \notin \pi_2(C)$. Then the statement for $i = i' + 1$ follows directly from Lemma 3.8.

Let j be the maximal value such that x_j is defined in a given invocation of `Pick`. We will prove the second claim for $(T, k, C, p) = (T^{(i)}, k_i, C_i, p^{(i)})$ for all $i = 0, \ldots, j$ with induction on i. For $i = 0$ this is trivial. Now assume the claim is true for $i = i'$ and assume s is a cherry picking sequence for $T^{(i'+1)}$ of weight at most $k_{i'+1}$ that satisfies $C_{i'+1}$. Then if $x_{i'}$ is defined, it will be in $H(T^{(i')})$, so $s' = (x_{i'})|s$ is a cherry picking sequence for $T^{(i')}$. Because $w_{T^{i'}}(x_{i'}) = k_{i'} - k_{i'+1}$, s' will have weight at most $k_{i'}$. We can write $C_{i'} = C_x \cup C_{-x}$ where $C_x = \{(a, b) : (a, b) \in C_{i'} \wedge a = x\}$ and $C_{-x} = C_{i'} \backslash C_x$. Note that s satisfies $C_{i'+1} = C_{-x}$, so $s' = (x_{i'})|s$ also satisfies $C_{i'+1}$. Because for every $(a, b) \in C_x$, also $(a, b) \in T^{i'}$, s' also satisfies C_x, so s' satisfies $C_{i'}$. Now it follows from the induction hypothesis that $p^{i'+1}|s = p^{i'}|s'$ is a cherry picking sequence for T' of weight at most k' and satisfying C'. $\qquad\square$

Note that on line 19 of Algorithm 1 an element $b \neq a$ with $(x, b) \in G(T', C')$ is chosen. The following lemma states that such an element does indeed exist.

Lemma 3.13 *When the algorithm executes line 19 there exist an element $b \neq a$ with $(x, b) \in G(T', C')$.*

Proof Because $w_{T'}(x) > 0$, there is at least a $b \neq a$ such that $b \in N_{T'}(x) \backslash \{x\}$. Because $x \notin \pi_2(C')$ we have $(b, x) \notin C'$. If $(x, b) \in C'$ then $x \in \pi_1(C')$, but then x satisfies the if-statement on line 15 and it would not have gotten to this line. Therefore $(x, b) \notin C'$ and so $(x, b) \in G(T', C')$. $\qquad\square$

The proof of correctness of Algorithm 1 will be given in two parts. First, in Lemma 3.14 we show that for any feasible problem instance the algorithm will return a sequence. Second, in Lemma 3.15 we show that every sequence that the algorithm returns is a valid cherry picking sequence for the problem instance.

Lemma 3.14 *When a cherry picking sequence of weight at most k that satisfies C exists, then a call to `CherryPicking`(T, k, C) from Algorithm 1 returns a non-empty set.*

Proof Let $W(k, u)$ be the claim that if a cherry picking sequence s of weight at most k exists that satisfies constraint set C with $n^2 - |C| \leq u$, then calling `CherryPicking`(T, k, C) will return a non-empty set. We will prove this claim with induction on k and $n^2 - |C|$.

For the base case $k = 0$ if a cherry picking sequence of weight k exists, by Observation 3.11.1 we will have $|\mathcal{L}(T')| = 1$. In this case a sequence is returned on line 7.

Note that we can never have a constraint set C with $|C| > n^2$ because $C \subseteq \mathcal{L}(T)^2$. Therefore $W(k, -1)$ is true for all k.

Now suppose $W(k, n^2 - |C|)$ is true for all cases where $0 \leq k < k_b$ and all cases where $k = k_b$ and $n^2 - |C| \leq u$. We consider the case where a cherry picking sequence

s of weight at most $k = k_b + 1$ exists for T that satisfies C and $n^2 - |C| \leq u + 1$. Lemma 3.10 implies that $k - P(C) \geq 0$, so the condition of the if-statement on line 2 will not be satisfied.

From Lemma 3.12 it follows that a CPS s' of weight at most k' exists for T' that satisfies C'. From the way the Pick works it follows that either $k' < k$ or $n^2 - C' = n^2 - C$. If $|\mathcal{L}(T')| = 1$ then $\{()\}$ is returned and we have proven $W(k_b + 1, u + 1)$ to be true for this case. Because s' satisfies C', we know that $\pi_1(C) \subseteq \mathcal{L}(T')$. We know there is a $y \in N_{T'}(s'_1)$ with $(s'_1, y) \notin C'$, because otherwise s'_1 would be picked by Pick. Also s' satisfies $C' \cup \{(s'_1, y)\}$, which implies that $k \geq P(C' \cup \{(s'_1, y)\}) > P(C')$, so the condition of the if-statement on line 10 will not be satisfied.

Note that we have $(s'_1, x) \in G(T', C')$, $w_{T'}(s'_1) > 0$ and $s'_1 \notin \pi_2(C')$.

This implies that either the body of the if-statement on line 15 or the body of the else-if-statement on line 18 will be executed.

Suppose the former is true. By Observation 3.4 we know that s satisfies $C' \cup \{(x, y)\}$ or $C' \cup \{(y, x)\}$. Because $(x, y) \in G(T', C')$ we know $|C' \cup \{x, y\}| = |C' \cup \{y, x\}| = |C'| + 1$ and therefore $n^2 - |C' \cup \{x, y\}| = n^2 - |C' \cup \{y, x\}| \leq u$. So by our induction hypothesis we know that at least one of the two subcalls will return a sequence, so the main call to the function will also return a sequence.

If instead the body of the else-if-statement on line 18 is executed we know by Observation 3.2 that at least one of the constraint sets $C'_1 = C \cup \{(a, x)\}$, $C'_2 = C \cup \{(b, x)\}$ and $C'_3 = C \cup \{(x, a), (x, b)\}$ is satisfied by s. Note that $|C'_3| \geq |C'_2| = |C'_1| \geq |C'| + 1$, so $n^2 - |C'_3| \leq n^2 - |C'_2| = n^2 - |C'_1| \leq u$. By the induction hypothesis it now follows that at least one of the three subcalls will return a sequence, so the main call to the function will also return a sequence. So for both cases we have proven $W(k_b + 1, u + 1)$ to be true. □

Lemma 3.15 *Every element in the set returned by* CherryPicking(T, k, C) *from Algorithm 1 is a cherry picking sequence for T of weight at most k that satisfies C.*

Proof Consider a certain call to CherryPicking(T, k, C). Assume that the lemma holds for all subcalls to CherryPicking. We claim that during the execution every element that is in R is a partial cherry picking sequence for T' of weight at most k' that satisfies C'. This is true because R starts as an empty set, so the claim is still true at that point. At each point in the function where sequences are added to R, these sequences are elements returned by CherryPicking(T', k', C'') with $C' \subseteq C''$. By our assumption we know that all of these elements are cherry picking sequences for T' of weight at most k' and satisfy C''. The latter implies that every element also satisfies C' because $C' \subseteq C''$. The procedure now returns $\{p|r : r \in R\}$ and from Lemma 3.12 it follows that all elements of this set are cherry picking sequences for T of weight at most k and satisfying C. □

3.2 Runtime Analysis

The key idea behind our runtime analysis is that at each recursive call in Algorithm 3, the measure $k - P(C)$ is decreased by a certain amount, and this leads to a bound on the number of times Algorithm 1 is called. It is straightforward to get a bound of

$O(9^k)$. Indeed, it can be shown that for $k < |C|/2$ no feasible solution exists, and so the algorithm could stop whenever $2k - |C| < 0$. One call to the algorithm results in at most 3 subcalls, and in each subcall $|C|$ increases by at least one. Then the total number of subcalls to Algorithm 1 would be bounded by $O(3^{2k}) = O(9^k)$. By more careful analysis, and using the lower bound of $P(C)$ on the weight of a sequence satisfying C, we are able to improve this bound to $O(5^k)$.

We will now state some lemmas that are needed for the runtime analysis of the algorithm. We first show that the measure $k - P(C)$ will never increase at any point in the algorithm. The only time this may happen is during `Pick`, as the values of k and C are not otherwise changed, except at the point of a recursive call where constraints are added to C (which cannot increase $P(C)$). Thus we first show that `Pick` cannot cause $k - P(C)$ to increase.

Lemma 3.16 *Let* $(s, T', k', C') = $ `Pick`(T, k, C) *from Algorithm* 1. *Then* $k' - P(C') \leq k - P(C)$.

Proof We will prove with induction that for the variables k_i and C_i defined in the function body, we have $k_i - P(C_i) \leq k - P(C)$ for all i, from which the result follows. Note that for $i = 0$ this is trivial. Now suppose the inequality holds for i. If $w_{T^{(i)}}(x_{i+1}) = 0$, then we have $C_{i+1} = C_i$, so:

$$k_{i+1} - P(C_{i+1}) = k_i - w_{T^{(i)}}(x_i) - P(C_i)$$
$$\leq k_i - P(C_i)$$

Otherwise we must have, $\{(x_{i+1}, n) : n \in N_{T^{(i)}}(x_{i+1})\} \subseteq C_i$, so:

$$k_{i+1} - P(C_{i+1}) = (k_i - w_{T^{(i)}}(x_i)) - (P(C_i) - (w_{T^{(i)}}(x_i) + 1) \cdot \psi - (1 - 2\psi))$$
$$= k_i - P(C_i) - (w_{T^{(i)}}(x_i) - 1)(1 - \psi)$$
$$\leq k_i - P(C_i)$$
$$\leq k - P(C)$$

The next lemma will be used later to show that a recursive call to `CherryPicking` always increases $k - P(C)$ by a certain amount.

Lemma 3.17 *For a and b on line 19 of Algorithm* 1 *it holds that* $a \notin \pi_1(C')$ *and* $b \notin \pi_1(C')$.

Proof Suppose $a \in \pi_1(C')$. Then $(a, z) \in C'$ for some $z \in N_{T'}(x)$. If $w_{T'}(a) > 0$ then a satisfies the conditions in the if-statement on line 15, so line 19 would not be executed. If $w_{T'}(a) = 0$ then we must have $|N_{T'}(a)\setminus\{a\}| = 1$, so $N_{T'}(a)\setminus\{a\} = \{x\}$, which implies that $z = x$. But $(a, x) \notin C'$ because $(x, a) \in G(T', C')$, which contradicts that $(a, z) \in C'$. So $a \notin \pi_1(C')$. Because of symmetry, the same argument holds for b. □

We now give the main runtime proof.

Lemma 3.18 `CherryPicking` *from Algorithm* 1 *has a time complexity of* $O(5^k \cdot nm)$.

Proof The non-recursive part of CherryPicking(T,k,C) can be implemented to run in $O(n \cdot m)$ time by constructing $H(T^{(i)})$ from $H(T^{(i-1)})$ in each step. To prove the lemma, we will bound the number of calls that are made to CherryPicking.

Consider the tree of calls to CherryPicking that are made in a single run of CherryPicking(T,k,C). Each node in this tree corresponds to a call to CherryPicking and the children of this node correspond to its recursive calls. Let $t(k, C)$ be the number of leaves of this tree. Observe that every internal node has at least two children. This implies that the number of nodes in the tree is upper bounded by $2t(k, C)$.

We will now show using induction that $t(k, C) \leq 5^{k-P(C)+1}$, thereby proving the lemma. For $-1 \leq k - P(C) \leq 0$ the claim follows from the fact that the function will return on either line 3 or line 11 and therefore will not do any recursive calls.

Now assume the claim holds for $-1 \leq k - P(C) \leq w$. Consider an instance with $k - P(C) \leq w + \psi$. Note that $k' - P(C') \leq k - P(C)$ (Lemma 3.16). If the function CherryPicking does any recursive calls then it either executes the body of the if-clause on line 15, or the body of the else-if clause on line 18.

If the former is true, then the function does 2 recursive calls. The number of leaves in the call-tree of CherryPicking(T, k, C) will now be the sum of the number of leaves in the call-trees of the recursive calls. Each recursive call to the function CherryPicking(T', k', C'') is done with a constraint set C'' for which $|C''| = |C'|+1$. Therefore for both subproblems $P(C'') \geq P(C') + \psi$ and also $k' - P(C'') \leq k' - P(C') - \psi \leq k - P(C) - \psi \leq w$. By our induction hypothesis we have $t(k', C'') \leq 5^{k'-P(C'')+1} \leq 5^{k-P(C)-\psi+1}$ Hence:

$$t(k, C) \leq 2 \cdot 5^{k-P(C)-\psi+1} = 5^{\psi} 5^{k-P(C)-\psi+1} = 5^{k-P(C)+1}.$$

If instead the body of the else-if statement on line 18 is executed then 3 recursive subcalls are made. Consider the first subcall CherryPicking(T', k', C''). We have $C'' = C' \cup \{(a, x)\}$. Because $(x, a) \in G(T', C')$ we have $(a, x) \notin C'$. Therefore $|C''| = |C'| + 1$. By Lemma 3.17 we know that $a \notin \pi_1(C')$, but we have $a \in \pi_1(C')$, so $|\pi_1(C'')| = |\pi_1(C')| + 1$. Therefore $P(C'') = P(C') + 1 - \psi$, so $k' - P(C'') = k' - P(C') - 1 + \psi \leq k - P(C) - 1 + \psi < k - P(C) - \psi \leq w$. By our induction hypothesis we now know that $t(k', C'')$ is bounded by $5^{k'-P(C'')+1} \leq 5^{k-P(C)-\psi}$. By symmetry the same holds for the second subcall.

For the third subcall CherryPicking(T', k', C''), because $(x, a), (x, b) \in G(T', C')$ we have $|C''| = |C'| + 2$, and because $x \notin \pi_1(C')$ we have $|\pi_1(C'')| = |\pi_1(C')| + 1$. So we know that $P(C'') = P(C') + 2\psi + (1 - 2\psi) = P(C') + 1$ and $k' - P(C'') + 1 \leq k - P(C)$. Therefore $t(k', C'') \leq 5^{k-P(C)}$.

Summing the leaves of these subtrees, we see:

$$t(k, C) \leq 2 \cdot 5^{k-P(C)-\psi} + 5^{k-P(C)} = 5^{k-P(C)} + 5^{k-P(C)} \leq 5^{k-P(C)+1}$$

This proves the induction claim for $k - P(C) \leq w + \psi$. □

Theorem 3.19 *CherryPicking(T, k, C) from Algorithm 1 returns a cherry picking sequence of weight at most k that satisfies C if and only if such a sequence exists. The algorithm terminates in $O(5^k \cdot n \cdot m)$ time.*
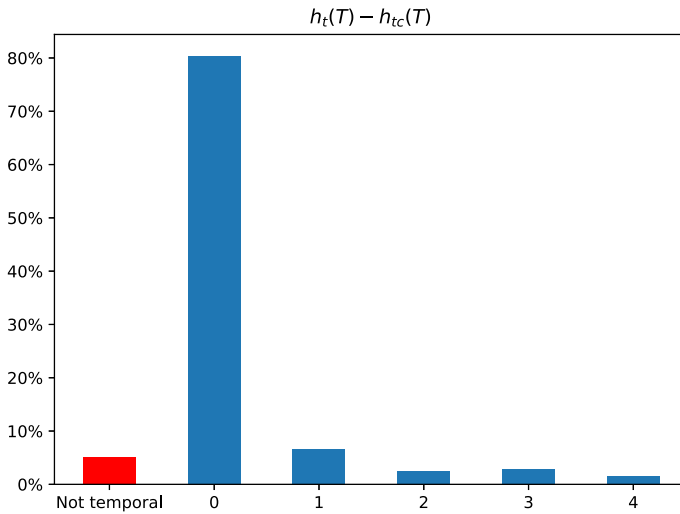
Fig. 7 The difference between the tree-child reticulation number and the temporal reticulation number on the dataset generated in [9]. If no temporal network exists, the instance is shown under 'Not temporal'. Instances for which it could not be decided if they were temporal within 10 minutes (2.6% of the instances), are excluded

Proof This follows directly from Lemmas 3.15, 3.14 and 3.18. □

4 Constructing Non-temporal Tree-Child Networks from Binary Trees

For every set of trees there exists a tree-child network that displays the trees. However there are sets of trees for which no temporal network displaying the trees exist, so we can not always find such a network. As shown in Fig. 7, approximately 5 percent of the instances used in [9] do not admit a temporal solution.

In this section we introduce theory that makes it possible to quantify how close a network is to being temporal. We can then pose the problem of finding the 'most' temporal network that displays a set of trees.

Definition 4.1 For a tree-child network with vertices V we call a function $t : V \to \mathbb{R}^+$ a semi-temporal labeling if:

1. For every tree arc (u, v) we have $t(u) < t(v)$.
2. For every hybridization vertex v we have $t(v) = \min\{t(u) : (u, v) \in E\}$.

Note that every network has a semi-temporal labeling.

Definition 4.2 For a tree-child network $\mathcal{N}$ with a semi-temporal labeling t, define $d(\mathcal{N}, t)$ to be the number of hybridization arcs (u, v) with $t(u) \neq t(v)$. We call these arcs non-temporal arcs.

Definition 4.3 For a tree-child network $\mathcal{N}$ define

$$d(\mathcal{N}) = \min\{d(\mathcal{N}, t) : t \text{ is a semi-temporal labeling of } \mathcal{N}\}$$

Call this number the *temporal distance* of $\mathcal{N}$. Note that this number is finite for every network, because there always exist semi-temporal labelings.

The temporal distance is a way to quantify how close a network is to being temporal. The networks with temporal distance zero are the temporal networks. We can now state a more general version of the decision problem.

Semi-temporal hybridization

Instance: A set of m trees T with n leaves and integers k, p.
Question: Does there exist a tree-child network $\mathcal{N}$ with $r(\mathcal{N}) \leq k$ and $d(\mathcal{N}) \leq p$?

There are other, possibly more biologically meaningful ways to define such a temporal distance. The reason for defining the temporal distance in this particular way is that an algorithm for solving the corresponding decision problem exists. For further research it could be interesting to explore if other definitions of temporal distance are more useful and whether the corresponding decision problems could be solved using similar techniques.

Van Iersel et al. presented an algorithm to solve the following decision problem in $O((8k)^k \cdot \text{poly}(m, n))$ time.

Tree-child hybridization

Instance: A set of m trees T with n leaves and integer k.
Question: Does there exist a tree-child network $\mathcal{N}$ with $r(\mathcal{N}) \leq k$?

Notice that for $p = k$ SEMI- TEMPORAL HYBRIDIZATION is equivalent to TREE-CHILD HYBRIDIZATION and for $p = 0$ it is equivalent to TEMPORAL HYBRIDIZATION. The algorithm for TREE- CHILD HYBRIDIZATION uses a characterization by Linz and Semple [8] using *tree-child sequences*, that we will describe in the next section. We describe a new algorithm that can be used to decide SEMI- TEMPORAL HYBRIDIZATION. This algorithm is a combination of the algorithms for TREE- CHILD HYBRIDIZATION and TEMPORAL HYBRIDIZATION.

4.1 Tree-Child Sequences

First we will define the *generalized cherry picking sequence* (generalized CPS), which is called a cherry picking sequence in [9]. We call it generalized cherry picking sequence because it is a generalization of the cherry picking sequence we defined in Definition 2.4.

Definition 4.4 A *generalized CPS* on X is a sequence

$$s = ((x_1, y_1), \ldots, (x_r, y_r), (x_{r+1}, -), \ldots, (x_t, -))$$

with $\{x_1, x_2, \ldots, x_t, y_1, \ldots, y_r\} \subseteq X$. A generalized CPS is *full* if $t > r$ and $\{x_1, \ldots, x_t\} = X$.

For a tree $\mathcal{T}$ on $X' \subseteq X$ the sequence s defines a sequence of trees $(\mathcal{T}^{(0)}, \dots, \mathcal{T}^{(r)})$ as follows:

– $\mathcal{T}^{(0)} = \mathcal{T}$.
– If $(x_j, y_j) \in \mathcal{T}^{(j-1)}$, then $\mathcal{T}^{(j)} = \mathcal{T}^{(j-1)} \setminus \{x_j\}$. Otherwise $\mathcal{T}^{(j)} = \mathcal{T}^{(j-1)}$.

We will refer to $\mathcal{T}^{(r)}$ as $\mathcal{T}(s)$, the tree obtained by *applying* sequence s to $\mathcal{T}$.

A full generalized CPS on X is a *generalized CPS* for a tree $\mathcal{T}$ if the tree $\mathcal{T}(s)$ contains just one leaf and that leaf is in $\{x_{r+1}, \dots, x_t\}$. If such a sequence is a generalized CPS for all trees $\mathcal{T} \in T$, then we say it is a generalized CPS for T. The *weight* of the sequence is then defined to be $w_T(s) = |s| - |X|$.

A generalized CPS is a *tree-child* sequence if $|s| \leq r + 1$ and $y_j \neq x_i$ for all $1 \leq i < j \leq |s|$. If for such a *tree-child* sequence $|s| = r$, then s is also called a *tree-child* sequence prefix.

It has been proven that a tree-child network displaying a set of trees T with $r(\mathcal{N}) = k$ exists if and only if a tree-child sequence s with $w(s) = k$ exists [8]. The network can be efficiently computed from the corresponding sequence. The algorithm presented by Van Iersel et al. works by searching for such a sequence.

We will show that it is possible to combine their algorithm with the algorithm presented in Sect. 4. This yields an algorithm that decides SEMI- TEMPORAL HYBRIDIZATION in $O(5^k (8k)^p \cdot k \cdot n \cdot m)$ time.

Definition 4.5 Let $s = ((x_1, y_1), \dots, (x_t, -))$ be a full generalized CPS. An element (x_i, y_i) is a *non-temporal* element when there are $j, k \in [t]$ with $i < j < k \leq t$ and $x_j \neq x_i$ and $x_k = x_i$.

Definition 4.6 For a sequence s we define $d(s)$ to be the number of non-temporal elements in s.

Lemma 4.7 *Let s be a full tree-child sequence s for T. Then there exists a network $\mathcal{N}$ with semi-temporal labeling t such that $r(\mathcal{N}) \leq w_T(s)$ and $d(\mathcal{N}, t) \leq d(s)$.*

The full proof of Lemma 4.7 is given in the appendix. We construct a tree-child network $\mathcal{N}$ from s in a similar way to [8,Proof of Theorem 2.2], working backwards through the sequence. At each stage when a pair (x, y) is processed, we adjust the network to ensure there is an arc from the parent of y to the parent of x. Our contribution is to also maintain a semi-temporal labeling t on $\mathcal{N}$. This can be done in such a way that for each pair (x, y), at most one new non-temporal arc is created, and only if (x, y) is a non-temporal element of s. This ensures that $d(\mathcal{N}, t) \leq d(s)$.

Lemma 4.8 *For a tree-child network $\mathcal{N}$ there exists a full tree-child sequence s with $d(s) \leq d(\mathcal{N})$ and $w_T(s) \leq r(\mathcal{N})$.*

The full proof of Lemma 4.8 is given in the appendix. We construct the sequence in a similar way to [8,Lemma 3.4]. The key idea is that at any point the network will contain some pair of leaves x, y that either form a *cherry* (where x and y share a parent) or a *reticulated cherry* (where the parent of x is a reticulation, with an incoming edge from the parent of y). We process such a pair by appending (x, y) to s, deleting an edge from $\mathcal{N}$, and simplifying the resulting network. By being careful about the

order in which we process reticulated cherries, we can ensure that we only add a non-temporal element to s when we delete a non-temporal arc from $\mathcal{N}$. This ensures that $d(s) \leq d(\mathcal{N}, t)$.

Observation 4.9 *A tree-child sequence s can not contain both (a, b) and (b, a).*

Observation 4.10 *If a tree-child sequence s has a subsequence s' that is a generalized cherry picking sequence for T, then s is also a generalized cherry picking sequence for T.*

Lemma 4.11 *If $s = ((x_1, y_1), \ldots, (x_{r+1}, -))$ is a generalized CPS for T and there is a z such that $y_i \neq z$ for all i. Then $(T\backslash\{z\})(s) = T(s)$ and therefore s is also a generalized CPS for $T\backslash\{z\}$.*

Proof Suppose this is not true. Because $T(s)$ consists of a tree with only one leaf x_{r+1}, this implies that $\mathcal{L}((T\backslash\{z\})(s)) \nsubseteq \mathcal{L}(T(s))$. Let i be the smallest index for which we have that $\mathcal{L}((T\backslash\{z\})((x_1, y_1), \ldots, (x_i, y_i))) \nsubseteq \mathcal{L}(T((x_1, y_1), \ldots, (x_i, y_i))\backslash\{z\})$.

This implies that $x_i \in \mathcal{L}((T\backslash\{z\})((x_1, y_1), \ldots, (x_i, y_i)))$ but $x_i \notin \mathcal{L}(T((x_1, y_1), \ldots, (x_i, y_i))\backslash\{z\})$, so $(x_i, y_i) \notin (T\backslash\{z\})((x_1, y_1), \ldots, (x_{i-1}, y_{i-1}))$, but $(x_i, y_i) \in T((x_1, y_1), \ldots, (x_{i-1}, y_{i-1}))\backslash\{z\}$. Let p be the lowest vertex that is an ancestor of both x_i and y_i in the tree $(T\backslash\{z\})((x_1, y_1), \ldots, (x_{i-1}, y_{i-1}))$. Because x_i and y_i do not form a cherry in this tree, there is another leaf q that is reachable from p. Because $q \in \mathcal{L}(T((x_1, y_1), \ldots, (x_{i-1}, y_{i-1}))\backslash\{z\})$, q is also reachable from the lowest common ancestor p' in $T((x_1, y_1), \ldots, (x_{i-1}, y_{i-1}))\backslash\{z\}$, contradicting the fact that (x_i, y_i) is a cherry in this tree. $\qquad\square$

4.2 Constraint Sets

The new algorithm also uses constraint sets. However, because the algorithm searches for a generalized cherry picking sequence, we need to define what it means for such a sequence to satisfy a constraint set.

Definition 4.12 A generalized cherry picking sequence $s = ((x_1, y_1), \ldots, (x_k, y_k))$ for T satisfies constraint set C if for every $(a, b) \in C$ there is an i with $(x_i, y_i) = (a, b)$ and there is a tree $T \in T$ for which $(a, b) \notin T(s_1, \ldots, s_{i-1})$.

In Definition 2.1 the function $H(T)$ was defined for sets of binary trees with the same leaves. After applying a tree-child sequence not all trees will necessarily have the same leaves. Because of this, we generalize the definition of $H(T)$ to sets of binary trees.

Definition 4.13 For a set of binary trees T, let

$$H(T) = \{x \in \mathcal{L}(T) : \forall T \in T \text{ if } x \in T \text{ then } x \text{ is in a cherry in } T\}.$$

Lemma 4.14 *If $s = ((x_1, y_1), \ldots, (x_{r+1}, -))$ is a tree-child sequence for T and $(a, b) \in T$, then there is an i such that $(x_i, y_i) = (a, b)$ or $(x_i, y_i) = (b, a)$.*

Proof Let $T \in \mathcal{T}$ be a tree in T containing cherry (a, b). Because s fully reduces T, $\mathcal{T}(s)$ consists of only the leaf x_{r+1}. So a or b has to be removed from $\mathcal{T}$ by applying s. Without loss of generality we can assume a is removed first. This can only happen if there is an i with $(x_i, y_i) = (a, b)$. □

Now we prove that if there are two cherries (a, z) and (b, z) in T, then we can branch on three possible additions to the constraint set, just like we did for cherry picking sequences.

Lemma 4.15 *Let s be a tree-child sequence for T and $a, b \in N_T(z)$ with $a \neq b$. Then s satisfies one of the following constraint sets:*
$$\{(a, z)\}, \{(b, z)\}, \{(z, a), (z, b)\}.$$

Proof From Lemma 4.14 it follows that either (a, z) or (z, a) is in s and that either (b, z) or (z, b) is in s. Now let $s_i = (x_i, y_i)$ be the element of these that appears first in s. Now we have three cases:

1. If $x_i = a$, then $s_i = (a, z)$. Let $\mathcal{T} \in \mathcal{T}$ be the tree in which (b, z) is a cherry. Now $(b, z) \in \mathcal{T}(s_1, \ldots, s_{i-1})$, so $(a, z) \in \mathcal{T}(s_1, \ldots, s_{i-1})$. Because $(s_{i+1}, \ldots, s_{r+1})$ is a tree-child sequence for $\mathcal{T}(s_1, \ldots, s_i)$, this implies that there is some $j > i$ with $x_j = a$. Consequently $\{(a, z)\}$ is satisfied by s.
2. If $x_i = b$, then the same argument as in (1) can be applied to show that $\{(b, z)\}$ is satisfied by s.
3. If $x_i = z$, then we either have $y_i = a$ or $y_i = b$. Without loss of generality we can assume $y_i = a$. We still have $(b, z) \in T(s_1, \ldots, s_{i-1})$, which implies that there is some $j > i$ with $(x_j, y_j) = (b, z)$ or $(x_j, y_j) = (z, b)$. Because $j > i$ and s is tree-child, we know that $y_j \neq z$. So $(x_j, y_j) = (z, b)$, and consequently $\{(z, a), (z, b)\}$ is satisfied by s.

□

We also prove that if $a \in \pi_1(C)$ and $(a, b) \in T$, then we only need to do two recursive calls.

Lemma 4.16 *Let s be a tree-child sequence for T that satisfies constraint set C and $a, b \in N_T(z)$ with $(z, b) \in C$. Then s satisfies one of the following constraint sets:* $\{(a, z)\}, \{(z, a)\}.$

Proof From Lemma 4.15 it follows that s satisfies one of the constraint sets $\{(a, z)\}$, $\{(b, z)\}$ and $\{(z, a), (z, b)\}$. However, because s satisfies C and $(z, b) \in C$, from Observation 4.9 it follows that (b, z) does not appear in s. Therefore s has to satisfy either $\{(a, z)\}$ or $\{(z, a), (z, b)\}$. If s satisfies $\{(z, a), (z, b)\}$, then it also satisfies $\{(z, a)\}$. □

Next, we prove an analogue to Lemma 3.10.

Lemma 4.17 *If a tree-child sequence $s = ((x_1, y_1), \ldots, (x_r, x_y), (x_{r+1}, -))$ for T satisfies constraint set C, then $w_T(s) \geq P(C)$.*

Proof For $z \in \mathcal{L}(T) \setminus \{x_{r+1}\}$, let $C_z := \{(a, b) : (a, b) \in C \wedge a = z\}$ and let $S_z := \{(x_i, y_i) : i \leq r \wedge x_i = z\}$. We show that we have $|S_z| - 1 \geq P(C_x)$. If $|C_z| = 0$,

then $P(C_z) = 0$ and the inequality is trivial. If $|C_z| = 1$, then from the definition of constraint sets it follows that $|S_z| \geq 2$, so $|S_z| - 1 \geq 1 \geq P(C_z)$. Otherwise if $|C_z| \geq 2$, then because $C_z \subseteq S_z$, $|S_z| - 1 \geq |C_z| - 1 = \psi \cdot |C_z| - 1 + (1 - \psi)|C_z| \geq |C_z| - 1 + 2(1 - \psi) = |C_z| + (1 - 2\psi) = P(C_z)$. Now the result follows because $w_T(s) = |s| - |\mathcal{L}(T)| = \sum_{z \in \mathcal{L}(T) \setminus \{x_{r+1}\}}(|S_z| - 1) \geq \sum_{z \in \mathcal{L}(T) \setminus \{x_{r+1}\}} P(C_z) = P(C)$. $\qquad \square$

Next we prove that if a leaf z is in $H(T)$ and appears in s with all of its neighbors, then we can move all elements containing z to the start of the sequence.

Lemma 4.18 *If $s = ((x_1, y_1), \ldots, (x_{r+1}, -))$ is a tree-child sequence for T, $z \in H(T)$ and I is a set of indices such that $\{y_i : i \in I\} = N_T(z)$ and $x_i = z$ for all $i \in I$. Then the sequence s' obtained by first adding the elements from s with an index in I and then adding elements (x, y) of s for which $x \neq z$ is a tree-child sequence for T. We have $d(s') \leq d(s)$.*

Proof We can write $s' = ((x_1', y_1'), \ldots, (x_{r+1}, -)) = s^a | s^b$ where s^a consists of the elements $\{s_i : i \in I\}$ and s^b is s with the elements at indices in I removed. First we prove that s' is a tree-child sequence. Suppose that s' is not a tree-child sequence. Then there are i, j with $i < j$ such that $x_i' = y_j'$. Note that we can not have that $y_j' = z$, because of how we constructed s'. This implies that both indices i and j are in s^b, implying that s^b is not tree-child. But because s^b is a subsequence of s this implies that s is not tree-child, which contradicts the conditions from the lemma. So s' is tree-child.

We now prove that s' fully reduces T. Because $T(s^a) = T \setminus \{z\}$ from Lemma 4.11 it follows that $s^a | s$ is a generalized CPS for T. Because $z \notin \mathcal{L}(T(s^a))$, $T(s^a | s) = T(s^a | s^b)$. So s' is a generalized CPS for T.

Finally since for every non-temporal element in s' the corresponding element in s is also non-temporal, we conclude that $d(s') \leq d(s)$. $\qquad \square$

4.3 Trivial Cherries

We will call a pair (a, b) a *trivial cherry* if there is a $\mathcal{T} \in T$ with $a \in \mathcal{L}(\mathcal{T})$ and for every tree $\mathcal{T} \in T$ that contains a, we have $(a, b) \in \mathcal{T}$. They are called trivial cherries because they can be picked without limiting the possibilities for the rest of the sequence, as stated in the following lemma.

Lemma 4.19 *If $s = ((x_1, y_1), \ldots, (x_{r+1}, -))$ is a tree-child sequence for T of minimum length and (a, b) is a trivial cherry in T, then there is an i such that $(x_i, y_i) = (a, b)$ or $(x_i, y_i) = (b, a)$. Also, there exists a tree-child sequence s' for T with $|s| = |s'|$, $d(s') \leq d(s)$ and $s_1' = (a, b)$.*

Proof This follows from Lemma 4.18. $\qquad \square$

The following lemma was proven in [9,Lemma 11].

Lemma 4.20 *Let $s^a | s^b$ be a tree-child sequence for T with weight k. If $T(s^a)$ contains no trivial cherries, then the number of unique cherries is at most $4k$.*

Algorithm 3

1: **procedure** SEMITEMPORALCHERRYPICKING($T, k, k^\star, p, C$)
2:   **if** $k - P(C) < 0$ **then**
3:     **return** $\emptyset$
4:   **end if**
5:   $T', k', C', f \leftarrow$ PICK(T, k, C)
6:   **if** $|\mathcal{L}(T')| = 1$ **then**
7:     **return** $\{f\}$
8:   **else if** $k' - P(C') \leq 0 \vee \pi_1(C') \nsubseteq \mathcal{L}(T')$ **then**
9:     **return** $\emptyset$
10: **end if**
11:
12: $R \leftarrow \emptyset$
13: **if** $\exists(x, y) \in T : w_T(x) > 0 \wedge x \in \pi_1(C')$ **then**
14:    $R \leftarrow R \cup$ SEMITEMPORALCHERRYPICKING($T', k', k^\star, p, C' \cup \{(x, y)\}$)
15:    $R \leftarrow R \cup$ SEMITEMPORALCHERRYPICKING($T', k', k^\star, p, C' \cup \{(y, x)\}$)
16: **else if** $\exists(x, a) \in G(T', C') : w_{T'}(x) > 0 \wedge x \notin \pi_2(C')$ **then**
17:    Choose $b \neq a$ such that $(x, b) \in G(T', C')$
18:    $R \leftarrow R \cup$ SEMITEMPORALCHERRYPICKING($T', k', k^\star, p, C' \cup \{(a, x)\}$)
19:    $R \leftarrow R \cup$ SEMITEMPORALCHERRYPICKING($T', k', k^\star, p, C' \cup \{(b, x)\}$)
20:    $R \leftarrow R \cup$ SEMITEMPORALCHERRYPICKING($T', k', k^\star, p, C' \cup \{(x, a), (x, b)\}$)
21: **else if** $p > 0$ **then**
22:    $P \leftarrow \{(x, y) \in T' : y \in \mathcal{T}' \; \forall \mathcal{T}' \in T' \wedge x \notin \pi_2(C)\}$
23:    **if** $|P| > 8k^\star$ **then**
24:      **return** $\emptyset$
25:    **end if**
26:    **for** $(x, y) \in P$ **do**
27:      $C'' \leftarrow C \backslash \{(x, y)\}$
28:      **if** $|\{(x, z) \in C\}| = 1$ **then**
29:        $C'' \leftarrow C'' \backslash \{(x, z) \in C\}$
30:      **end if**
31:      $R \leftarrow R \cup \{(x, y)|r : r \in$ SEMITEMPORALCHERRYPICKING($T'((x, y)), k' - 1, k^\star, p - 1, C''$)\}.
32:    **end for**
33: **end if**
34: **return** $\{f|r : r \in R\}$
35: **end procedure**

Algorithm 4

procedure PICK(T', k', C')
  $(T^{(1)}, k_1, C_1) \leftarrow (T', k', C')$
  $p^{(1)} \leftarrow ()$
  $i \leftarrow 1$
  while $\exists x_i \in H(T^{(i)}) : \big(w_{T^{(i)}}(x_i) = 0 \vee \{(x_i, n) : n \in N_{T^{(i)}}(x_i)\} \subseteq C_i\big) \wedge (\forall y \in N_{T^{(i)}} \forall \mathcal{T} \in T : y \in \mathcal{T})$ **do**
    $(n_1, \ldots, n_t) \leftarrow N_T(x_i)$
    $p^{(i+1)} \leftarrow p^{(i)}|((x_i, n_1), \ldots, (x_i, n_t))$
    $k_{i+1} \leftarrow k_i - w_{T^{(i)}}(x_i)$
    $T^{(i+1)} \leftarrow T^{(i)}$
    $C_{i+1} \leftarrow \{c \in C_i : x \notin \{c_1, c_2\}\}$
    $i \leftarrow i + 1$
  end while
  return $T^{(i)}, k_i, C_i, p_i$
end procedure

Lemma 4.21 *If* $((x_1, y_1), \ldots, (x_2, y_2), (x_{r+1}, -), , (x_t, -))$ *is a full tree child-sequence of minimal length for* T *satisfying* C *and* $H(T)\backslash\pi_2(C) = \emptyset$, *then* (x_1, y_1) *is a non-temporal element.*

Proof First observe that $x_1 \notin \pi_2(C)$ because the sequence satisfies C. Suppose (x_1, y_1) is a temporal element. This implies that there is an i such that for all $j < i$ we have $x_j = x_1$ and $x_k \neq x_1$ for all $k \geq i$. This implies that for every $\mathcal{T} \in T$ there is a $j < i$ such that x_1 is not in $\mathcal{T}((x_j, y_j))$. Consequently (x_j, y_j) is a cherry in $\mathcal{T}$. Because this holds for every tree $\mathcal{T} \in T$ we must have $x_1 \in H(T)\backslash\pi_2(C)$, contradicting the assumption that $H(T)\backslash\pi_2(C) = \emptyset$. □

4.4 The Algorithm

We now present our algorithm for SEMI- TEMPORAL HYBRIDIZATION. As with TREE-CHILD HYBRIDIZATION, we split the algorithm into two parts: `SemiTemporalCherryPicking`(Algorithm 3) is the main recursive procedure, and `Pick`(Algorithm 4) is the auxiliary procedure.

The key idea is that we try to follow the procedure for temporal sequences as much as possible. Algorithm 3 only differs from Algorithm 1 in the case where neither of the recursion conditions of Algorithm 1 apply, but there are still cherries to be processed. In this case, we can show that there are no trivial cherries, and hence Lemma 4.20 applies. Then we may assume there are at most $4k^*$ unique cherries, where k^* is the original value of k that we started with. In this case, we branch on adding (x, y) or (y, x) to the sequence, for any x and y that form a cherry. Any such pair will necessarily be a non-temporal element, and so we decrease p by 1 in this case.

Lemma 4.22 *(Correctness of* `Pick`*) Suppose* `Pick`(T', k', C') *in Algorithm 4 returns* (T, k, C, p). *Then a tree-child sequence* s *of weight at most* k *for* T, *that satisfies* C, *exists if and only if a tree-child sequence* s' *of weight at most* k' *for* T', *that satisfies* C', *exists. In this case* $p|s$ *is a tree-child sequence for* T' *of weight at most* k' *and satisfying* C'.

The proof for this lemma is the same as for Lemma 3.12, but uses Lemma 4.18

Lemma 4.23 *Let* $s^\star$ *be a tree-child sequence prefix,* $T^\star$ *a set of trees with the same leaves and define* $T := T^\star(s^\star)$. *Suppose* $k, p \in \mathbf{N}$ *and* $C \in \mathcal{L}(T)^2$. *When a generalized cherry picking sequence* s *exists and such that* $s^\star|s$ *is a tree-child sequence for* $T^\star$, *that satisfies* C *with* $w_{T^\star}(s^\star|s) \leq k^\star$, *and such that* $d(s) \leq p$, `SemiTemporalCherryPicking`$(T, k, k^\star, p, C)$ *from Algorithm 3 returns a non-empty set.*

A full proof of this lemma is given in the appendix.

Lemma 4.24 *Let* $s^\star$ *be a tree-child sequence prefix,* $T^\star$ *a set of trees with the same leaves and let* $T := T^\star(s^\star)$. *Let* $k, p \in \mathbf{N}$ *and* $C \in \mathcal{L}(T)^2$. *If a call to* `SemiTemporalCherryPicking`$(T, k, k^\star, p, C)$ *returns a set* S, *then for every* $s \in S$, *the sequence* $s' = s^\star|s$ *is a tree-child sequence for* $T^\star$ *with* $d(s) \leq p$ *and* $w(s) \leq k$.

The proof of this lemma is similar to the proof of Lemma 3.14 using Lemma 4.22.

Lemma 4.25 *Algorithm 3 has a running time of $O(5^k \cdot (8k)^p \cdot k \cdot n \cdot m)$.*

Proof This can be proven by combining the proofs from Lemmas 3.18 and 4.20. ☐

Theorem 4.26 `SemiTemporalCherryPicking`$(T, k, k, p, \emptyset)$ *from Algorithm 3 returns a nonempty set of tree-child cherry picking sequences for T, of weight at most k, with at most p non-temporal elements, if and only if such a sequence exists. The algorithm terminates in $O(5^k \cdot (8k)^p \cdot k \cdot n \cdot m)$ time.*

Proof This follows directly from Lemmas 4.24, 4.23 and 4.25. ☐

5 Constructing Temporal Networks from Two Non-binary Trees

The algorithms described in the previous sections only work when all input trees are binary. In this section we introduce the first algorithm for constructing a minimum temporal hybridization number for a set of two non-binary input trees. The algorithm is based on [14] and has time complexity $O(6^k k! \cdot k \cdot n^2)$.

We say that a binary tree $\mathcal{T}'$ is a refinement of a non-binary tree $\mathcal{T}$ when $\mathcal{T}$ can be obtained from $\mathcal{T}'$ by contracting some of the edges. Now we say that a network $\mathcal{N}$ displays a non-binary tree $\mathcal{T}$ if there exists a binary refinement $\mathcal{T}'$ of $\mathcal{T}$ such that $\mathcal{N}$ displays $\mathcal{T}'$. Now the hybridization number $h_t(T)$ can be defined for a set of non-binary trees T like in the binary case.

Definition 5.1 For a set of non-binary trees we let the set of neighbors $N_T(x)$ be defined as $N_T(x) := \{y \in \mathcal{L}(T) \backslash \{x\} : \exists \mathcal{T} \in T : \{x, y\}$ is a subset of some cherry in $\mathcal{T}\}$.

Definition 5.2 A set $S \subseteq N_T(x)$ is a *neighbor cover* for x in T if $S \cap N_{\mathcal{T}}(x) \neq \emptyset$ for all $\mathcal{T} \in T$.

Definition 5.3 For a set of non-binary trees T, define $w_T(x)$ as the minimum size of a neighbor cover of x in T minus 1.

Note that computing the minimum size of a neighbor cover comes down to finding a minimum hitting set, which is an NP-hard problem in general. However if $|T|$ is constant, the problem can be solved in polynomial time. Note that for binary trees this definition is equivalent to the definition given in Definition 2.3.

Next Definition 2.1 is generalized to non-binary trees.

Definition 5.4 For a set of non-binary trees T with the same leaf set define $H(T) = \{x \in \mathcal{L}(T) : \forall \mathcal{T} \in T : N_{\mathcal{T}}(x) \neq \emptyset\}$.

The non-binary analogue of Definition 2.4 is given by the following lemma.

Definition 5.5 For a set of non-binary trees T with $n = \mathcal{L}(T)$, let $s = (s_1, \ldots, s_{n-1})$ be a sequence of leaves. Let $T_0 = T$ and $T_i = T_{i-1} \backslash \{s_1, \ldots, s_i\}$. The sequence s is a *cherry picking sequence* if for all i, $s_i \in H(T \backslash \{s_1, \ldots, s_{i-1}\})$. Define the *weight* of the sequence as $w_T(s) = \sum_{i=1}^{n-1} w_{T_{i-1}}(s_i)$.

Lemma 5.6 *A temporal network $\mathcal{N}$ that displays a set of nonbinary trees T with reticulation number $r(\mathcal{N}) = k$ exists if and only if a cherry picking sequence of weight at most k exists.*

Proof Note that this is a generalization of Lemma 2.6 to the case of non-binary input trees and the proof is essentially the same. A cherry picking sequence with weight k can be constructed from a temporal network with reticulation number k in the same way as in the proof of Lemma 2.6.

The construction of a temporal network $\mathcal{N}$ from a cherry picking s is also very similar to the binary case: for cherry picking sequence $s_1, \ldots, s_t$, define $\mathcal{N}_{t+1}$ to be the network, only consisting of a root, the only leaf of $T \backslash \{s_1, \ldots, s_t\}$ and an edge between the two. For each i let S_i be a minimal neighbor cover of s_i in $T \backslash \{s_1, \ldots, s_{i-1}\}$. Now obtain $\mathcal{N}_i$ from $\mathcal{N}_{i+1}$ by adding node s_i, subdividing (p_x, x) for every $x \in S_i$ with node q_x and adding an edge (q_x, s_i) and finally suppressing all nodes with in- and out-degree one. It can be shown that $r(\mathcal{N}) = w_T(s)$. $\quad\square$

Lemma 5.7 *If s is a cherry picking sequence for T and for $x \in H(T)$ we have $w_T(x) = 0$ then there is a cherry picking sequence s' for T with $w_T(s') = w_T(s)$ and $s_1' = x$.*

Proof We have $N_T(x) = \{y\}$. Now let z be the element of $\{x, y\}$ that appears in s first with $s_i = z$. Now $s' = (s_i, s_1, \ldots, s_{i-1}, s_{i+1}, \ldots)$ is a cherry picking sequence for T with $w_T(s') = w_T(s)$. If $z = x$, then this proves the lemma. Otherwise we note that by swapping x and y in T, the trees stay the same. So we can also swap x and y in s' without affecting the weight. Now $s' = x$, which proves the lemma. $\quad\square$

The algorithm relies on some theory from [14], that we will introduce first.

For a vertex v of $\mathcal{T}$ we say that all vertices reachable by v form a pendant subtree. For a pendant subtree S we define $\mathcal{L}(S)$ to be the set of the leaves of S. Now we define

$$Cl(\mathcal{T}) = \{\mathcal{L}(S) : S \text{ is a pendant subtree of } \mathcal{T}\}.$$

We call this the set of *clusters* of $\mathcal{T}$. Then we define $Cl(T) = \bigcup_{\mathcal{T} \in T} Cl(\mathcal{T})$. Call a cluster C with $|C| = 1$ *trivial*. Now we call a nontrivial cluster $C \in Cl(T)$ a *minimal cluster* if there is no $C' \in Cl(T)$ with C' nontrivial and $C' \subsetneq C$.

In a cherry picking sequence s we say that at index i the cherry (s_i, y) is *reduced* if there is a $\mathcal{T} \in T$ such that $N_{\mathcal{T} \backslash \{s_1, \ldots, s_{i-1}\}}(s_i) = \{y\}$.

Lemma 5.8 *Let T be a set of trees with $|T| = 2$ such that T contains no leaf x with $w_T(x) = 0$. Let s be a cherry picking sequence for T. Then there is a minimal cluster C in T and a cherry picking sequence $s' = (s_1', \ldots)$ for T with $s_i' \in C$ for $i = 1, \ldots, |C| - 1$ and $w_T(s') \le w_T(s)$.*

Proof Let p be the first index at which a cherry is reduced in s. Let (a, b) be one of the cherries that is reduced at index p. Now there will be a cherry in T that contains both a and b. Let C be one of the minimum clusters that is contained in this cherry. Let x be the element of C that occurs last in s. Then let $c_1, \ldots, c_t$ be the elements from $C \backslash \{x\}$ ordered by their index in s. We claim that for any permutation σ of $[t]$

we have $s' = (c_{\sigma(1)}, \ldots, c_{\sigma(t)}) | (s \backslash (C \backslash \{x\}))$ is a cherry picking sequence for T and $w_T(s') \leq w_T(s)$.

Let i be the index of the last element of $C \backslash \{x\}$ in s. Suppose that s' is not a CPS for T. Let j be the smallest index for which $s'_j \notin H(T \backslash \{s'_1, \ldots, s'_{j-1}\})$.

Let $\mathcal{T} \in T$ be such that s'_j is not in a cherry in $\mathcal{T} \backslash \{s'_1, \ldots, s'_{j-1}\}$. Choose k such that $s_k = s'_j$. Now there are three cases:

- Suppose $j > i$, then $k = j$ and $\{s_1, \ldots, s_{k-1}\} = \{s'_1, \ldots, s'_{j-1}\}$. This implies that $s'_j \in H(T \backslash \{s'_1, \ldots, s'_{j-1}\})$, which contradicts our assumption.
- Otherwise, suppose $s'_j \in \{c_1, \ldots, c_t\}$. Then $j \leq t$. Now s_k has to be in a cherry in $\mathcal{T} \backslash \{s_1, \ldots, s_{k-1}\}$. Because no cherries are reduced before index i in s this means that s'_j is in a cherry in $\mathcal{T}$. Because no cherries are reduced in s' before index t, this implies that the same cherry is still in $\mathcal{T} \backslash \{s'_1, \ldots, s'_{j-1}\}$, which contradicts our assumption.
- Otherwise we must have $j \leq i$. Because no cherries are reduced before index i in s this means that s'_j is in a cherry Q in $\mathcal{T}$. If this cherry contains a leaf y with $s'_w = y$ for $w > j$, then s'_j is still in a cherry in $\mathcal{T} \backslash \{s'_1, \ldots, s'_{j-1}\}$, contradicting our assumption, so this can not be true. However, that implies that the neighbors of s_k in $\mathcal{T} \backslash \{s_1, \ldots, s_{k-1}\}$ are all elements of $\{c_1, \ldots, c_t\}$. Let v be the second largest number such that c_v is one of these neighbors. Let q be the index of c_v in s. Now cherry Q will be reduced by s at index $\max(q, j) < i$, which contradicts the fact that C is contained in a cherry of T that is reduced first by s.

To prove that $w_T(s') \leq w_T(s)$, we will prove that for $s_j = s'_k$ we have

$$w_{T \backslash \{s_1, \ldots, s_{j-1}\}}(s_j) \geq w_{T \backslash \{s'_1, \ldots, s'_{k-1}\}}(s'_k).$$

Note that for $j \geq i$ this is trivial, so assume $j < i$. If $s_j \in C \backslash \{x\}$, then $w_{T \backslash \{s_1, \ldots, s_{j-1}\}}(s_j) \geq w_T(s_j)$, because no cherries are reduced before i, which implies that no new elements are added to cherries before i. For the same reason we must have $s_j \in H(T)$. Because there are no $x \in H(T)$ with $w_T(x) = 0$. Because $|T| = 2$, we have $w_T(x) \leq 1$ for all x, hence $w_T(s_j) = 1$. So $w_{T \backslash \{s'_1, \ldots, s'_{k-1}\}}(s'_k) \leq w_{T \backslash \{s_1, \ldots, s_{j-1}\}}(s_j) = 1$. □

5.1 Bounding the Number of Minimal Clusters

By Lemma 5.8, in the construction of a cherry picking sequence, we can restrict ourselves to only appending elements from minimal clusters. We use the following theory from [14] to bound the number of minimal clusters.

Definition 5.9 Define the relation $x \xrightarrow{T} y$ for leaves x and y of T if every nontrivial cluster $C \in Cl(T)$ also contains y.

Observation 5.10 ([14,Observation 2]) If T contains no zero-weight leaves, the relation $\xrightarrow{T}$ defines a partial ordering on $\mathcal{L}(T)$.

Now call $x \in \mathcal{L}(T)$ a *terminal* if there is no $y \neq x$ with $x \xrightarrow{T} y$. Now we will first show that all minimal clusters contain a terminal. Then a bound on the number of terminals gives a bound on the number of minimal clusters.

Lemma 5.11 *If T contains no zero-weight leaves, every minimal cluster contains a terminal.*

Proof Let C be a minimal cluster of T. Let x be an element of C that is maximal in C with respect to the partial ordering '$\xrightarrow{T}$' (if we say that $x \xrightarrow{T} y$ means that y is 'greater than or equal to' x). Now suppose that x is not a terminal. Then there is an y such that $x \xrightarrow{T} y$. However then $y \in C$, but this contradicts the fact that x is a maximal element in C with respect to '$\xrightarrow{T}$'. Because this is a contradiction, x has to be a terminal. □

Lemma 5.12 *Let T be a set of trees with $h_t(T) \geq 1$ containing no zero-weight leaves. Let $\mathcal{N}$ be a network that displays T. Then T contains at most $2r(\mathcal{N})$ terminals that are not directly below a reticulation node.*

Proof We reformulate the proof from [14,Lemma 3]. We use the fact that for each terminal one of the following conditions holds: the parent p_x of x in N is a reticulation (condition 1) or a reticulation is reachable in a directed tree-path from the parent p_x of x (condition 2). This is always true because if neither of the conditions holds then another leaf y is reachable from p_x, implying that $x \xrightarrow{T} y$, which contradicts that x is a terminal.

Let R be the set of reticulation nodes in $\mathcal{N}$ and let W be the set of terminals in T that are not directly beneath a reticulation. We describe a mapping $F : W \to R$ such that each reticulation r is mapped to at most $d^-(r)$ times. Note that for each $x \in W$ condition 2 holds. For these elements let $F(x) = y$ where y is a reticulation reachable from $p(x)$ by a tree-path. Note that there can not be a path from $p(x)$ to y containing only tree arcs when $x \neq y$ are both in $H(T)$ because then $x \to y$ which contradicts that x is a terminal. It follows that each reticulation r can be mapped to at most $d^-(r)$ times: at most once per incoming edge. Then for the set of terminals Ω we have $|\Omega| \leq \sum_{r \in R} d^-(r) \leq \sum_{r \in R}(1 + (d^-(r) - 1)) \leq |R| + k \leq 2k$. □

Lemma 5.13 *Let T be a set of nonbinary trees such that $h_t(T) \geq 1$. Suppose that T contains no zero-weight leaves. Then any set S of terminals in T with $|S| \geq 2h_t(T)+1$ contains at least one element $x \in H(T)$ such that there is a cherry picking sequence s for T with $w_T(s) = h_t(T)$ and $s_1 = x$.*

Proof Let $\mathcal{N}$ be a temporal network that displays T such that $r(\mathcal{N}) = h_t(T)$ with corresponding cherry picking sequence s. From the Lemma 5.12 it follows that at most $r(\mathcal{N})$ terminals exist in T that are not directly below a reticulation. So there is an $x \in S$ that is directly below a reticulation.

Now let T' be the set of all binary trees displayed by $\mathcal{N}$. Note that s is a cherry picking sequence for T'. Let i be such that $s_i = x$. Because x is directly below a reticulation in $\mathcal{N}$, for all $j < i$ we have $s_j \notin N_{T'}(x)$, which implies by Lemma 3.6 that $s' = (s_i, s_1, \ldots, s_{i-1}, s_{i+1}, \ldots)$ is a cherry picking sequence for T' with $w_{T'}(s') = w_{T'}(s) = r(\mathcal{N}) = h_t(T)$. Now $w_T(s') \leq w_{T'}(s') = h_t(T)$, so $w_T(s') = h_t(T)$. □

Algorithm 5

```
 1: procedure CHERRYPICKING(T, k)
 2:     s ← ()
 3:     while ∃x ∈ H(T) : w_T(x) = 0 do
 4:         T ← T\{x}
 5:         s ← s|(x)
 6:     end while
 7:
 8:     if |L(T)| = 1 then
 9:         return {s}
10:     else if k = 0 then
11:         return ∅
12:     end if
13:
14:     S ← set of terminals in T
15:     if |S| > 2k then
16:         S' ← subset of S of size 2k + 1
17:         for x ∈ S' ∩ H(T) do
18:             R ← R ∪ {(x) | x : x ∈ CHERRYPICKING(T\{x}, k − 1 )}
19:         end for
20:     else
21:         for q ∈ S do
22:             D ← set of minimum clusters that contain q
23:             if ∃y, z : D = {{q, y}, {q, z}} then
24:                 for x ∈ {q, y, z} ∩ H(T) do
25:                     R ← R ∪ {(y) | x : x ∈ CHERRYPICKING(T\{y}, k − 1)}
26:                 end for
27:             else
28:                 for C ∈ D do
29:                     for x ∈ C : C\{x} ⊆ H(T) do
30:                         (c_1, ..., c_t) ← C\{x}
31:                         R ← R ∪ {(c_1, ..., c_t) | x : x ∈ CHERRYPICKING(T\{c_1, ..., c_t}, k − t)}
32:                     end for
33:                 end for
34:             end if
35:         end for
36:     end if
37:     return {s|x : x ∈ R}
38: end procedure
```

5.2 Run-Time Analysis

Lemma 5.14 *The running time of* `CherryPicking`(T, k) *from Algorithm 5 is* $O(6^k k! \cdot k \cdot n^2)$ *if T is a set consisting of two nonbinary trees.*

Proof Let $f(n)$ be an upper bound for the running time of the non-recursive part of the function. We claim that the maximum running time $t(n, k)$ for running the algorithm on trees with n leaves and parameter k is bounded by $6^k k! k f(n)$.

For $k = 0$ it is clear that this claim holds. Now we will prove that it holds for any call, by assuming that the bound holds for all subcalls.

If $|S| > 2k$, then the algorithm branches into $2k + 1$ subcalls. The total running time can then be bounded by

$$(2k + 1)t(n, k - 1) + f(n) \leq (2k + 1)6^{k-1}(k - 1)!(k - 1)f(n) + f(n)$$
$$\leq 6^k(k)!(k)f(n).$$

If the condition of the if-statement on line 23 is true, then for that q the function does 3 subcalls with k reduced by one. So the recursive part of the total running time for this q is bounded by

$$3t(n, k - 1) \leq 3 \cdot 6^{k-1}(k - 1)!(k - 1)f(n) \leq 6^k(k!)(k)f(n).$$

If the condition on line 23 holds then there is at most one $d \in D$ with $|d| \leq 2$. Using this information we can bound the total running time of the subcalls that are done for q in the else clause by

$$\sum_{d \in D} |d|t(k - |d| + 1) \leq \sum_{d \in D} |d|6^{k-|d|+1}(k - |d| + 1)!(k - |d| + 1)f(n)$$
$$\leq \sum_{d \in D} |d|6^{k-|d|+1}(k - |d| + 1)!(k - |d| + 1)f(n)$$
$$\leq (k - 1)!(k - 1)f(n) \sum_{d \in D} |d|6^{k-|d|+1} \tag{1}$$
$$\leq (k - 1)!(k - 1)f(n)(2 \cdot 6^{k-1} + 3 \cdot 6^{k-2}) \tag{2}$$
$$= (k - 1)!(k - 1)f(n)2^{k-1}(9 \cdot 3^{k-2}) $$
$$= (k - 1)!(k - 1)f(n)2^{k-1}3^k. \tag{3}$$

Note that (2) follows from the fact that $x \mapsto x6^{k-x+1}$ is a decreasing function for $x \in [1, \infty)$. So for each q the running time of the subcalls is bounded by $(k - 1)!(k - 1)f(n)2^{k-1}3^k$. Now the total running time is bounded by

$$f(n) + (k - 1)!(k - 1)f(n)2^{k-1}3^k|S| \tag{4}$$
$$\leq f(n) + (k - 1)!(k - 1)f(n)2^{k-1}3^k2k \tag{5}$$
$$\leq f(n) + k!(k - 1)f(n)6^k \tag{6}$$
$$\leq 6^k k!kf(n) \tag{7}$$

Because the non-recursive part of the function can be implemented to run in $O(n^2)$ time the total running time of the function is $O(6^k k! \cdot k \cdot n^2)$. □

Lemma 5.15 *Let T be a set of non-binary trees. If $h_t(T) \leq k$, then* `CherryPicking` *(T, k) from Algorithm 5 returns a cherry picking sequence for T of weight at most k.*

Proof First we will prove with induction on k that if $h_t(T) \leq k$ then a sequence is returned.

For $k = 0$ it is true because if $h_t(T) = 0$, as long as $\mathcal{L}(T) > 1$ then $|H(T)| > 0$ and all elements of $H(T)$ will have zero weight, so they are removed on line 4. After that $\mathcal{L}(T) = 1$ so an empty sequence will be returned, which proves that the claim is true for $k = 0$.

Now assume that the claim holds for for $k < k'$ and assume that $h_t(T) \leq k'$. Now we will prove that a sequence is returned by $\texttt{CherryPicking}(T, k)$ in this case. After removing an element x with weight zero on line 4 we still have $h_t(T) \leq k'$ (Lemma 5.7). If $|\mathcal{L}(T)| = 1$, an empty sequence is returned. If this is not the case then $0 < h_t(T) \leq k$, so the else if is not executed.

If $|S| > 2k$ then from Lemma 5.13 it follows that for $S' \subseteq S$ with $|S'| = 2k + 1$ there is at least one $x \in S'$ such that $h_t(T\backslash\{x\}) \leq k' - 1$. Now from the induction hypothesis it follows that $\texttt{CherryPicking}(T\backslash\{x\}, k')$ returns at least one sequence, which implies that R is not empty. Because of that the main call will return at least one sequence, which proves that the claim holds for $k = k'$.

The only thing left to prove is that every returned sequence is a cherry picking sequence for T. This follows from the fact that only elements from $H(T)$ are appended to s and that R consists of cherry picking sequences for $T\backslash\{s_1, \ldots, s_t\}$. $\qquad\square$

6 Experimental Results

We developed implementations of Algorithms 1, 5 and 3, which are freely available [15]. To analyse the performance of the algorithms we made use of dataset generated in [9] for experiments with an algorithm for construction of tree-child networks with a minimal hybridization number.

6.1 Algorithm 1

In Fig. 8 the running time of Algorithm 1 on the dataset from [9] is shown. The results are consistent with the bound on the running time that was proven in Sect. 3. Also, the algorithm is able to compute solutions for relatively high values of k, indicating that the algorithm performs well in practice.

The authors of [9] also provide an implementation of their algorithm for tree-child algorithms. The implementation contains several optimizations to improve the running time. One of them is an operation called cluster reduction [16]. The implementation is also multi-threaded. In Fig. 9 we provide a comparison of the running times of the tree-child algorithm with Algorithm 1. In this comparison we let both implementations use a single thread, because our implementation of the algorithm for computing the hybridization number does not support multithreading. The implementation could however be modified to solve different subproblems in different threads which will probably also result in a significant speed-up. For Algorithm 1 we see that the difference in time complexity between the $O((8k)^k)$ algorithm and the $O(5^k)$ algorithm is also observable in practice.
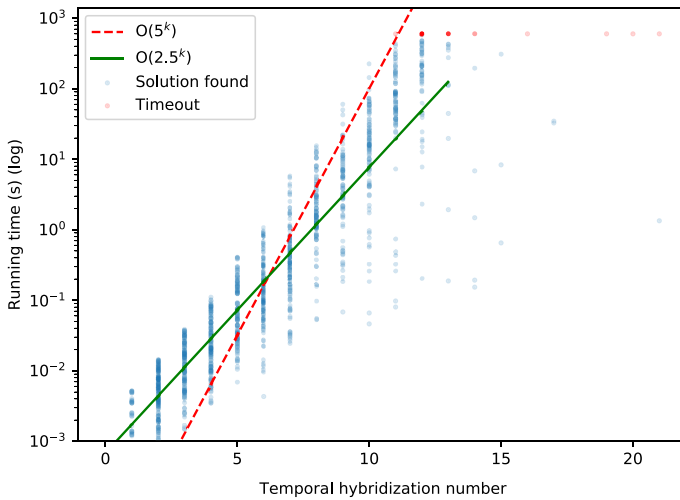
Fig. 8 The running time of Algorithm 1 on problem shown relative to the corresponding temporal hybridization number. A timeout of 10 min was used. Instances for which the algorithm timed out are shown in red at the value of k where they timed out. On the log scale the exponential relation is clearly visible. However fitting an exponential function on the data yields a $O(2.5^k)$ function for temporal hybridization number k, while the worst-case bound that we proved is $O(5^k)$
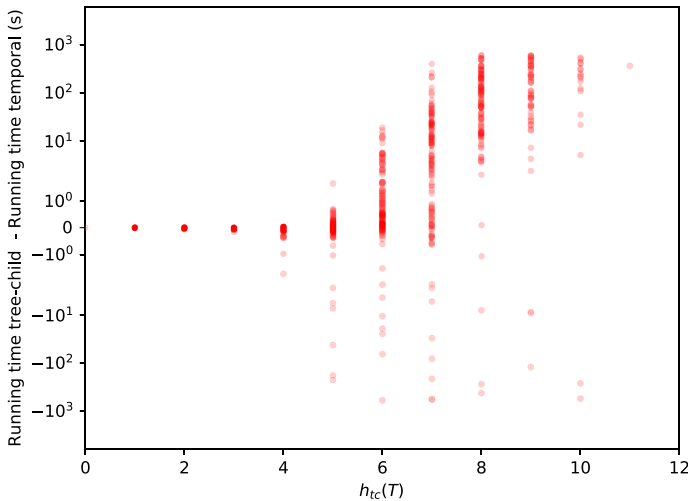

Fig. 9 Difference between the running time of Algorithm 1 and the algorithm for tree-child networks from [9]

6.2 Algorithm 5

We used the software from [9] to generate random binary problem instances and afterwards randomly contracted edges in the trees to obtain non-binary problem instances. We used this dataset to test the running time of Algorithm 5. The results are shown in
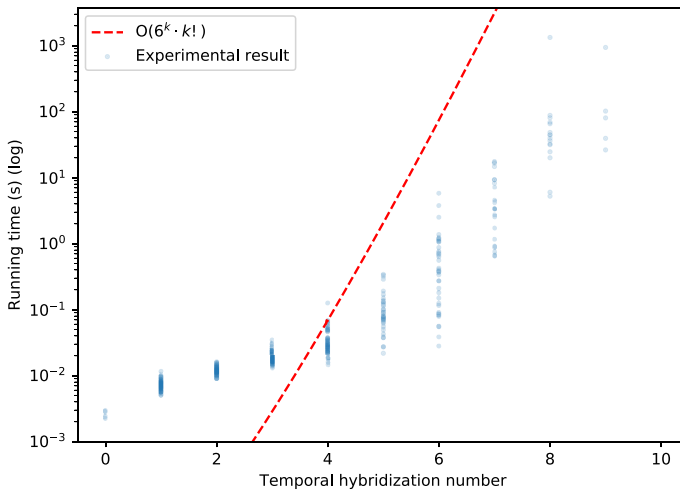
Fig. 10 Running time of Algorithm 5 on a generated set of instances consisting of trees with average out-degree 2.5 relative to the temporal hybridization number. A timeout of 10 min was used
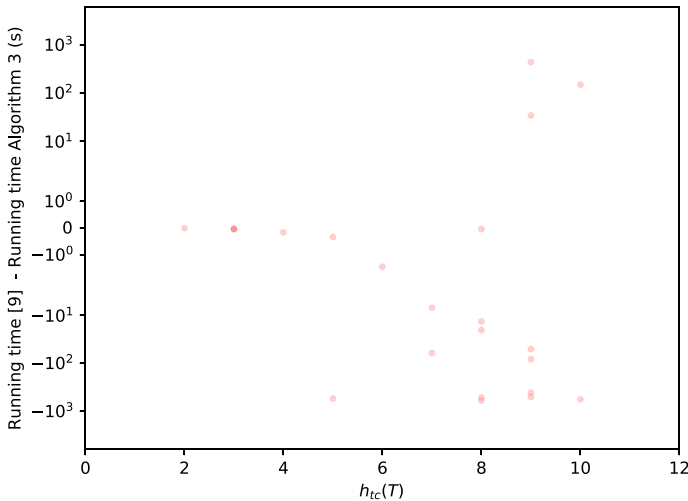

Fig. 11 Difference between the running time of Algorithm 3 and the algorithm for constructing tree-child networks from [9] on all non-temporal instances in the dataset from [9]

Fig. 10. We see that the algorithm is usable in practice and has a reasonable running time.

6.3 Algorithm 3

Algorithm 3 was tested on all non-temporal instances in the dataset from [9]. In Fig. 11 the running time of Algorithm 3 is compared to that of the algorithm from [9]. The data show that the algorithm from [9] is often faster than Algorithm 3. However, there

are also instances for which Algorithm 3 is much faster. Hence, in practice it can be worthwhile to run this algorithm on instances that cannot be solved by the algorithm from [9] in a reasonable time. It should also be noted that we only tested the algorithms on a relatively small dataset.

7 Discussion

Algorithm 1, the algorithm for constructing minimum temporal hybridization networks, has a significantly better running time than the algorithms that were known before. The results from the implementation show that the algorithm also works well in practice. However this implementation could still be improved, for example by making use of parallelization.

While we also present an algorithm that finds optimal temporal networks for non-binary trees, the running time of this algorithm is significantly worse and, moreover, it only works for pairs of trees. An open question is whether this could be improved to a running time of $O(c^k \cdot poly(n))$ for some $c \in \mathbb{R}$, perhaps using techniques similar to our approach for binary trees. Another important open problem is whether Temporal Hybridization is FPT for a set of more than two non-binary input trees.

In Sect. 4 a metric is provided to quantify how close a hybridization network is to being temporal. However, other, possibly more biologically meaningful, metrics could also be used for this purpose. An open problem is whether an FPT algorithm exists that solves the decision problem associated with these metrics.

A Omitted Proofs

Lemma 4.7. Let s be a full tree-child sequence s for T. Then there exists a network $\mathcal{N}$ with semi-temporal labeling t such that $r(\mathcal{N}) \leq w_T(s)$ and $d(\mathcal{N}, t) \leq d(s)$.

Proof This can be proven by constructing a tree-child network from the tree-child sequence as described in [8,Proof of Theorem 2.2]. We will show that a semi-temporal labeling satisfying our constraints exists for the resulting network. We will write

$$s = ((x_1, y_1), \ldots, (x_r, y_r), (x_{r+1}, -))$$

Now we merge all consecutive elements $(x_i, y_i), (x_{i+1}, y_{i+1}), \ldots, (x_{i+j}, y_{i+j})$ for which $x_i = x_{i+1} = \cdots = x_{i+j}$ into one element $(x_i, \{y_i, y_{i+1}, \ldots, y_{i+j}\})$ and call

the resulting sequence s'. Call an element of this sequence *temporal* if all corresponding elements in s are temporal. Call it *non-temporal* if all corresponding elements in s are non-temporal. Observe that it can not happen that some of the corresponding elements are temporal while some are non-temporal. The following algorithm now constructs a suitable network and temporal labeling:

1. Let $\mathcal{N}_{r+1}$ be the network consisting of root ρ, vertex x_{r+1} and edge (ρ, x_{r+1}). Set $i := r$. Define $t_{r+1} : \mathcal{L}(T) \to \{-\infty\} \cup \mathbb{N}$. Set $t(\rho) := 0$ and $t_{r+1}(x_{r+1}) := 1$ and for all other $z \in \mathcal{L}(x)$ set $t_{r+1}(z) = -\infty$.
2. If $i = 0$, contract all edges with in- and out-degree 1 in $\mathcal{N}_1$ and return the resulting network together with t_1.
3. Set $t_i := t_{i+1}$.
4. For the element $s'_i = (x, Y)$ do the following:

   (a) If s'_i is a temporal element then $x \notin \mathcal{L}(\mathcal{N}_{i+1})$. In this case let $\mathcal{N}_i$ be the network obtained from $\mathcal{N}_{i+1}$ by adding vertex x, vertex p_x, edge (p_x, x), subdividing edge (p_y, y) by vertex v_y and adding edge (v_y, p_x) for all $y \in Y$. Set $t_i(p_x) := -\infty$.
   (b) Otherwise s'_i is a non-temporal element and $x \in \mathcal{L}(\mathcal{N}_{i+1})$. In this case let $\mathcal{N}_i$ be the network obtained from $\mathcal{N}_{i+1}$ by subdividing (p_y, y) for all $y \in Y$ with a new vertex v_y and adding the edge (v_y, p_x).

5. Set $\tau = \max\{\max_{y \in Y} t_i(p_y), t_i(p_x) - 1\}$. For all $y \in Y$ set $t_i(v_y) := \tau + 1$ and $t_i(y) := \tau + 2$. If s'_i is a temporal element set $t_i(p_x) := \tau + 1$ and $t_i(x) := \tau + 2$.
6. Decrease i by one. Go to step 2.

Note that the construction of the network is equivalent to the one described in [8,Proof of Theorem 2.2], where it is also proven that the resulting network is a tree-child network that is fully reduced by s. The only thing we have to prove is that t is a semi-temporal labeling of $\mathcal{N}$ with $d(\mathcal{N}, t) \leq d(s)$.

We will prove with induction on i that t_i is a semi-temporal labeling for $\mathcal{N}_i$. For N_n it is clear that this is true. Consider an arbitrary edge (u, v) in $\mathcal{N}_i$. If the edge was also in $\mathcal{N}_{i+1}$, then $t_{i+1}(u) = t_i(u)$ and $t_{i+1}(v) = t_i(v)$, so the edge satisfies the conditions for being semi-temporal.

Now we will go through all newly introduced edges in $\mathcal{N}_i$ and show that they satisfy the conditions for being semi-temporal.

 – In Item 4a edges (p_y, v_y), (v_y, y), and (v_y, p_x) are created for all $y \in Y$ and (p_x, x) is created. Because $t_i(v_y) = \tau + 1 > t_i(p_y)$ and $t_i(y) = \tau + 2 > \tau + 1 = t_i(v_y)$ the first two edges are semi-temporal. Because in this case s'_i is a temporal element $t_i(p_x) := \tau + 1$ and $t_i(x) := \tau + 2$ will be set in step 5, so (p_x, x) is semi-temporal. Consequently $t_i(v_y) = \tau + 1 = t_i(p_x)$, so (v_y, x) is also semi-temporal.
 – In Item 4b edges (p_y, v_y), (v_y, y), (v_y, p_x) for all $y \in Y$.
   Note that before these edges are added we already have $t_i(w) = t_i(p_x)$ for some parent x of p_x. From step 5 it follows that $t_i(v_y) = \tau + 1 > t_i(p_y)$, that $p(y) = \tau + 2 > \tau + 1$ and that $t(p_x) \leq \tau + 1 = t(v_y)$. Therefore all of the created edges are semi-temporal. We also have $t_i(w) = t_i(p_x) < t_i(x)$, so these edges remain semi-temporal.

Note that the only place where non-temporal reticulation edges can be introduced is in Item 4b in the creation of edges (v_y, p_x) for all $y \in Y$. This only happens for non-temporal items s_i' and for each of this item at most $|Y|$ non-temporal reticulation edges are created, so $d(\mathcal{N}_i, t_i) \leq d(\mathcal{N}_{i+1}, t_{i+1}) + |Y|$. Because a non-temporal element (x, Y) in s' corresponds to $|Y|$ non-temporal elements in s, this implies that $d(\mathcal{N}, t) \leq d(s)$. □

Lemma 4.8. For a tree-child network $\mathcal{N}$ there exists a full tree-child sequence s with $d(s) \leq d(\mathcal{N})$ and $w_T(s) \leq r(\mathcal{N})$.

Proof We provide a way of constructing a tree-child sequence s from a tree-child network $\mathcal{N}$ with semi-temporal labeling t such that $d(s) = d(\mathcal{N})$. We do this by modifying the proof from [8,Lemma 3.4]. Let ρ denote the root of $\mathcal{N}$ and let $v_1, \ldots, v_r$ denote the reticulations in the network. Let $\ell_\rho, \ell_1, \ldots, \ell_r$ denote the leaves at the end of the paths $P_\rho, P_1, \ldots, P_r$ starting at $v_1, \ldots, v_r$ respectively and consisting of only tree arcs.

We will call a set $\{x, y\}$ with parents p_x and p_y in a given network a *cherry* if $p_x = p_y$. We will call it a *reticulated cherry* if p_x and p_y are joined by a reticulation edge (p_y, p_x). In this case we call x the *reticulation leaf* of the cherry. We call such a reticulated cherry *temporal* if $t(p_y) = t(p_x)$, otherwise we call it *non-temporal*.

Start off with an empty sequence σ_0. Set $\mathcal{N}_0 := \mathcal{N}$ and $i := 1$.

1. If $\mathcal{N}_{i-1}$ consists of a single vertex x then set $\sigma_i := \sigma_{i-1}|((x, -))$ and return σ_i.
2. If there is a cherry $\{x, y\}$ in $\mathcal{N}_{i-1}$, then
   (a) If one of $\{x, y\}$, say x, is an element of $\{\ell_1, \ldots, \ell_r\}$ and v_j is not a reticulation in $\mathcal{N}_{i-1}$ set $x_i := x$ and $y_i := y$.
   (b) Otherwise let $\{x_i, y_i\} := \{x, y\}$ such that $x_i \notin \{\ell_p, \ell_1, \ldots, \ell_r\}$.
   (c) Set $\sigma_i = ((x_i, y_i))|\sigma_{i-1}$. Let $\mathcal{N}_i$ be the tree-child network obtained from $\mathcal{N}_{i-1}$ by deleting x_i.
   (d) Go to step 5.
3. Else, if there is a non-temporal reticulated cherry $\{x, y\}$ in $\mathcal{N}_{i-1}$ with x the reticulation leaf then set $\sigma_i = \sigma_{i-1}|((x_i, y_i))$. Let $\mathcal{N}_i$ be the tree-child network obtained from $\mathcal{N}_{i-1}$ by deleting the edge (y_i, x_i) and suppressing vertices of both in-degree and out-degree one.
4. Else, there has to be a temporal reticulated cherry $\{x, y\}$ in $\mathcal{N}_{i-1}$ with x the reticulation leaf. Let $q_1, \ldots, q_t$ be the set of leaves that x is in a reticulation cherry with in $\mathcal{N}_{i-1}$. Set $\sigma_i = \sigma_{i-1}|((x, q_1), \ldots, (x, q_t))$. Let $\mathcal{N}_i$ be the tree-child network obtained from $\mathcal{N}_{i-1}$ by deleting vertex x and suppressing vertices of both in-degree and out-degree one.
5. Increase i and go to step 1.

The proof that this yields a full tree-child sequence s for $\mathcal{N}$ with $w_T(s) \leq r(\mathcal{N})$ can be found in [8,Lemma 3.4], so we will omit it here. Note that non-temporal elements can only be added to s in step 3 and each time this happens a non-temporal arc is removed from the network. Consequently the resulting tree-child sequence can not contain more non-temporal elements than the number of non-temporal arcs in $\mathcal{N}$. It follows that $d(s) \leq d(\mathcal{N})$. □

Lemma 4.23. Let $s^\star$ be a tree-child sequence prefix, $T^\star$ a set of trees with the same leaves and define $T := T^\star(s^\star)$. Suppose $k, p \in \mathbf{N}$ and $C \in \mathcal{L}(T)^2$. When a generalized cherry picking sequence s exists and such that $s^\star|s$ is a tree-child sequence for $T^\star$, that satisfies C with $w_{T^\star}(s^\star|s) \leq k^\star$, and such that $d(s) \leq p$, SemiTemporalCherryPicking($T, k, k^\star, p, C$) from Algorithm 3 returns a non-empty set.

Proof Let $W(k, u)$ be the claim that if a tree-child sequence s for T of weight at most k exists that satisfies constraint set C with $n^2 - |C| \leq u$ and $d(s) \leq p$, such that $s^\star|s$ is a tree-child sequence of weight at most $k^\star$, then the algorithm will return a non-empty set. We will prove this claim with induction on k and $n^2 - |C|$.

For the base case $k = 0$, if a generalized cherry picking sequence of weight k exists we must have that all cherries in T are trivial cherries. Therefore $|\mathcal{L}(T')| = 1$, and a non-empty set is returned.

Note that we can never have a constraint set C with $|C| > n^2$ because $C \subseteq \mathcal{L}(T)^2$. Therefore $W(k, -1)$ is true for all k.

Now suppose $W(k, n^2 - |C|)$ is true for all cases where $0 \leq k < k_b$ and all cases where $k = k_b$ and $n^2 - |C| \leq u$. We consider the case where a sequence s with $d(s) \leq p$ of weight at most $k = k_b + 1$ exists for T that satisfies C and $n^2 - |C| \leq u_b + 1$ such that $s^\star|s$ is a tree-child sequence for $T^\star$ with $w_{T^\star}(s^\star|s) \leq k^\star$. Now we will prove that a non-empty set is returned by the algorithm.

Lemma 4.17 implies that $k - P(C) \geq 0$, so the if-statement on line 2 will not be satisfied. From Lemma 4.22 it follows that a tree-child cherry picking sequence s' for T' of weight at most k' exists for T' that satisfies C'. From the way Pick works it follows that either $k' < k$ or $n^2 - C' = n^2 - C$. If $|\mathcal{L}(T')| = 1$ then $\{()\}$ is returned and we have proven $W(k_b + 1, u + 1)$ to be true for this case. Otherwise s' is not empty, so $k' - P(C') \geq w_T(s') > 0$. Because s' satisfies C', $\pi_1(C') \subseteq \mathcal{L}(T')$. So the condition on line 8 is not satisfied.

Now we are left with three cases:

1. If there is a pair $(x, y) \in G(T', C')$ with $w_T(x) > 0 \wedge x \in \pi_1(C')$, then from Lemma 4.16 it follows that s satisfies either $C' \cup \{(x, y)\}$ or $C' \cup \{(y, x)\}$. From our induction hypothesis it now follows that either SemiTemporalCherryPic king($T', k', p, C' \cup \{(x, y)\}$) or SemiTemporalCherryPicking($T', k', p, C' \cup \{(y, x)\}$) will return a non-empty set. Therefore R will not be empty, so a non-empty set will be returned.

2. Otherwise, if there is a pair $(x, a) \in G(T', C')$ with $w_{T'}(x) > 0 \wedge x \notin \pi_2(C')$, there is a $b \neq a$ with $(x, b) \in G(T', C')$, for the same reasons as in Lemma 3.13 for the temporal case. Now from Lemma 4.15 it follows that s satisfies $C' \cup \{(a, x)\}$, $C' \cup \{(b, x)\}$ or $C' \cup \{(x, a), (x, b)\}$. From our induction hypothesis it now follows that the corresponding subcall will return a non-empty set. Therefore R will not be empty, so a non-empty set will be returned.

3. Because the conditions in both the if and the else-if statement are not satisfied it follows that $H(T) \backslash \pi_2(C)$ is empty. Indeed, any $x \in H(T)$ must have $w_{T'}(x) > 0$, as otherwise it would have been removed by Pick, and for any $x \in H(T) \backslash \pi_2(C)$ there exists at least one y with $(x, y) \in G(T', C)$, as otherwise $\{(x, n) : n \in N_{T'}(x)\} \subseteq C'$ and again x would be removed by Pick. Then either case 1 (if

$x \in \pi_1(C))$ or case 2 (otherwise) would apply.

Now from Lemma 4.21 it follows that s_1 has to be a non-temporal element. Observe that for $s'_1 = (x, y)$ we must have $x \notin \pi_2(C)$, because otherwise s' has to contains some element (z, x), but such an element can not appear after an element (x, y), because the sequence is a tree-child sequence. Also y has to be in all trees in T', because otherwise s contains an element (y, z), which contradicts the assumption that $s|s'$ is tree-child. So now we have shown that $s'_1 \in P$. Each element of P is a cherry in T'. Lemma 4.20 implies that there are at most $4k^\star$ unique cherries in T'. Therefore it follows that $|P| \le 8k^\star$. Because $s'_1 \in P$, there R is not empty because the result of SemiTemporalCherryPicking$(T'(s'_1), k' - 1, p - 1, C'')$ is added to R, which by our induction hypothesis is a non-empty set. □

References

1. Mallet, J., Besansky, N., Hahn, M.W.: How reticulated are species? BioEssays **38**(2), 140–149 (2016). https://doi.org/10.1002/bies.201500149
2. Soucy, S.M., Huang, J., Gogarten, J.P.: Horizontal gene transfer: building the web of life. Nat Rev Genet **16**(8), 472–482 (2015). https://doi.org/10.1038/nrg3962
3. Bapteste, E., van Iersel, L., Janke, A., Kelchner, S., Kelk, S., McInerney, J.O., Morrison, D.A., Nakhleh, L., Steel, M., Stougie, L., Whitfield, J.: Networks: expanding evolutionary thinking. Trends Genet. **29**(8), 439–441 (2013). https://doi.org/10.1016/j.tig.2013.05.007
4. Bordewich, M., Semple, C.: Computing the minimum number of hybridization events for a consistent evolutionary history. Discrete Appl. Math. **155**(8), 914–928 (2007). https://doi.org/10.1016/j.dam.2006.08.008
5. van Iersel, L., Kelk, S., Scornavacca, C.: Kernelizations for the hybridization number problem on multiple nonbinary trees. J. Comput. Syst. Sci. **82**(6), 1075–1089 (2016). https://doi.org/10.1016/j.jcss.2016.03.006
6. Bordewich, M., Semple, C.: Computing the hybridization number of two phylogenetic trees is fixed-parameter tractable. IEEE/ACM Trans. Comput. Biol. Bioinformatics **4**(3), 458–466 (2007). https://doi.org/10.1109/tcbb.2007.1019
7. Cardona, G., Rossello, F., Valiente, G.: Comparison of tree-child phylogenetic networks. arXiv:0708.3499 [cs, q-bio] (2007). http://arxiv.org/abs/0708.3499. ArXiv: 0708.3499
8. Linz, S., Semple, C.: Attaching leaves and picking cherries to characterise the hybridisation number for a set of phylogenies. Adv. Appl. Math. **105**, 102–129 (2019). https://doi.org/10.1016/j.aam.2019.01.004
9. van Iersel, L., Janssen, R., Jones, M., Murakami, Y., Zeh, N.: A practical fixed-parameter algorithm for constructing tree-child networks from multiple binary trees. arXiv:1907.08474 [cs, math, q-bio] (2019). http://arxiv.org/abs/1907.08474. ArXiv: 1907.08474
10. Humphries, P.J., Linz, S., Semple, C.: Cherry picking: a characterization of the temporal hybridization number for a set of phylogenies. Bull. Math. Biol. **75**(10), 1879–1890 (2013). https://doi.org/10.1007/s11538-013-9874-x
11. Humphries, P.J., Linz, S., Semple, C.: On the complexity of computing the temporal hybridization number for two phylogenies. Discrete Appl. Math. **161**(7–8), 871–880 (2013). https://doi.org/10.1016/j.dam.2012.11.022
12. Döcker, J., van Iersel, L., Kelk, S., Linz, S.: Deciding the existence of a cherry-picking sequence is hard on two trees. Discrete Appl. Math. **260**, 131–143 (2019). https://doi.org/10.1016/j.dam.2019.01.031
13. Linz, S., Semple, C.: Hybridization in nonbinary trees. IEEE/ACM Trans. Comput. Biol. Bioinformatics **6**(1), 30–45 (2009). https://doi.org/10.1109/TCBB.2008.86
14. Piovesan, T., Kelk, S.M.: A simple fixed parameter tractable algorithm for computing the hybridization number of two (not necessarily binary) trees. IEEE/ACM Trans. Comput. Biol. Bioinformatics **10**(1), 18–25 (2013). https://doi.org/10.1109/TCBB.2012.134

15. Borst, S.: Temporal hybridization number algorithm implementations (2020). https://doi.org/10.5281/zenodo.3601812. https://github.com/mathcals/temporal_hybridization_number
16. Linz, S., Semple, C.: A cluster reduction for computing the subtree distance between phylogenies. Ann. Combin. **15**(3), 465–484 (2011). https://doi.org/10.1007/s00026-011-0108-3

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.