On cherry-picking and network containment

Janssen, Remie; Murakami, Yukihiro

DOI
10.1016/j.tcs.2020.12.031

Publication date
2021

Document Version
Final published version

Published in
Theoretical Computer Science

Theoretical Computer Science 856 (2021) 121–150

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

On cherry-picking and network containment ☆

Remie Janssen [1], Yukihiro Murakami *,[1]

Delft Institute of Applied Mathematics, Delft University of Technology, Van Mourik Broekmanweg 6, 2628 XE, Delft, the Netherlands

ARTICLE INFO

Article history:
Received 8 May 2020
Received in revised form 5 October 2020
Accepted 15 December 2020
Available online 4 January 2021
Communicated by T. Calamoneri

Keywords:
Phylogenetic networks
Network containment
Cherry-picking sequences
Tree containment
Linear-time algorithms
Cherry-picking networks

ABSTRACT

Phylogenetic networks are used to represent evolutionary scenarios in biology and linguistics. To find the most probable scenario, it may be necessary to compare candidate networks. In particular, one needs to distinguish different networks and determine whether one network is contained in another. In this paper, we introduce cherry-picking networks, a class of networks that can be reduced by a so-called cherry-picking sequence. We then show how to compare such networks using their sequences. We characterize reconstructible cherry-picking networks, which are the networks that are uniquely determined by the sequences that reduce them, making them distinguishable. Furthermore, we show that a cherry-picking network is contained in another cherry picking network if a sequence for the latter network reduces the former network, provided both networks can be reconstructed from their sequences in a similar way (i.e., they are in the same reconstructible class). Lastly, we show that the converse of the above statement holds for tree-child networks, thereby showing that Network Containment, the problem of checking whether a network is contained in another, can be solved by computing cherry picking sequences in linear time for tree-child networks.

1. Introduction

In the study of evolutionary histories of (biological) species and languages, directed graphs are used to represent different evolutionary scenarios. These graphs are most often assumed to be phylogenetic trees, that is, rooted trees whose leaves are labeled with the studied set of species. It is increasingly common to change this assumption and allow for (undirected) cycles in the scenarios as well. These directed graphs, called phylogenetic networks, then allow for the representation of reticulate evolutionary events such as hybridization and horizontal gene transfer [20,1].

Reconstruction of evolutionary histories is most often based on genetic data (i.e., DNA sequences). Although the evolutionary history of a set of species may be reticulate, small stretches of DNA (e.g., pieces of DNA coding for protein domains) still evolve mostly tree-like. The network representing the species' evolution must then contain the trees for such pieces of DNA. This leads to the following mathematical problem. For a given network N and a tree T on the same set of species, decide whether N contains T.

This problem, called Tree Containment, is NP-complete for general rooted phylogenetic networks [18]. Many have overcome this computational challenge by considering inputs of topologically restricted networks. It was shown initially that

☆ This is an extended version of the conference paper Janssen and Murakami [16].

* Corresponding author.
E-mail addresses: R.Janssen-2@tudelft.nl (R. Janssen), Y.Murakami@tudelft.nl (Y. Murakami).

[1] Research funded by the Netherlands Organization for Scientific Research (NWO), with the Vidi grant 639.072.602.

https://doi.org/10.1016/j.tcs.2020.12.031

Tree Containment can be solved in polynomial time for non-binary normal networks, binary tree-child networks, and binary level-k networks [13]. Stronger results have been proven for genetically stable networks (quadratic time: Gambette et al. [8]), binary nearly-stable networks (linear time: Gambette et al. [9]), and for binary tree-child networks (linear time: Gunawan [10], Gambette et al. [9]).

From a biological and a computational perspective, there is no reason to restrict to inputs of a tree and a network. Indeed, while small stretches of DNA evolve tree-like, it is possible for larger parts of the genome to evolve as a network. In such instances, it is of great interest to consider a more general version of Tree Containment introduced in [16] called Network Containment: For given networks N and N' on the same set of species, decide whether N contains N'.

Computationally, it is natural to wonder whether we can also solve Network Containment efficiently in all network classes where Tree Containment can be solved efficiently. This question was first posed in the conference paper [16], of which this paper is an extended version. In that conference paper, the focus was restricted to semi-binary tree-child networks and all proofs were omitted. In this paper, we do include all proofs, and we broaden the scope to include non-binary networks in the class of so-called cherry-picking networks, which contains the class of tree-child networks.

Like in our conference paper, we study the Network Containment problem in the light of *cherry-picking sequences*. These sequences were developed to tackle variations on the Hybridization problem, of finding a "simple" network that contains a given set of trees [12,5,19,3]. Two leaves of a tree form a *cherry* if they share a common parent—by successively 'picking' cherries (removing one of the leaves in a cherry) from the set of input trees, we obtain a sequence of cherries that ultimately reduce each input tree to a tree on a single leaf. This sequence of cherries then corresponds to some network that contains the set of all input trees.

As cherry-picking sequences were introduced to study Hybridization, actions of cherry-picking sequences were defined only on trees, and not on networks. In this paper, we fill this gap by defining the action of a cherry-picking sequence on a (non-binary) network. This leads to the introduction of the class of *cherry-picking networks*: networks that can be reduced to a single leaf by a cherry-picking sequence. We briefly mentioned this class in the discussion of the previous version of this paper [15]. The class for binary networks was also introduced independently by Erdős et al. [6] (where it was called the class of *orchard networks*). One of the main results of this paper, that cherry-picking networks may be reduced in any order (Theorem 1), was also shown by Erdős et al. [6]. The focus of their paper and this paper is rather different (they focus on reconstructing networks from so-called *ancestral profiles*), and therefore both papers have independently come across the same network class via different approaches.

We investigate the correspondence between cherry-picking networks and the sequences that reduce them, and ultimately show that within a particular *reconstructible* class, cherry-picking networks are characterized uniquely by their *smallest* cherry-picking sequences (Theorem 2). These reconstructible classes are based on several constructions used to obtain a network from a cherry-picking sequence. Although Linz and Semple [19] and Döcker et al. [5] mention how one can construct some network from a cherry-picking sequence, no further characterizations for the type of networks that can be obtained from such sequences have been investigated. Here, we reintroduce these constructions as a way to reverse the reduction. As multiple different networks can be reduced by the same cherry-picking sequence, this reversal cannot be unique. Hence, we investigate several constructions corresponding to the different reversals of the reductions. Each construction then leads to a reconstructible class of networks, for which we can prove relations between containment and reduction by cherry-picking sequences.

These relations depend on a careful definition of what it means for a network to be a *subnetwork* of and to be *contained* in another network. Roughly speaking, a network N' is a *subnetwork* of another network N if N' can be obtained from N by deleting reticulation edges and suppressing degree-2 vertices. A network N' is *contained* in another network N if N' can be obtained from N by deleting reticulation edges, suppressing degree-2 vertices, and by contracting edges. We show that within particular classes of cherry-picking networks, if a sequence for a network N reduces another network N', then N' is contained in N (Lemma 15). Unfortunately the converse does not hold (Theorem 4), unless the two input networks are tree-child.

It turns out that the class of tree-child networks is contained in the class of cherry-picking networks, as each tree-child network has a special type of cherry-picking sequence—a *tree-child sequence*—that reduces it. We examine how these sequences can be used to solve Network Containment for tree-child networks. In particular, within some cherry-picking network classes, we show that a tree-child sequence for a tree-child network N reduces another tree-child network N' if and only if N' is contained in N (Theorem 5). Following this, we provide a linear-time algorithm for Network Containment for inputs of tree-child networks (Algorithm 6).

Structure of the paper In Section 2, we recall all relevant definitions and outline how to construct networks from cherry-picking sequences. In Section 3, we investigate properties of cherry-picking networks, and show that the order in which cherries are picked does not matter (Theorem 1). Furthermore, we show that networks are unique up to a particular minimal cherry-picking sequence that reduces them, given an order on the set of species. In Section 4, we show that if a sequence for a network N reduces another network N', then N' is contained in N (Lemma 14). We also give a counter-example for why the converse does not hold (Theorem 4). In Section 5, we restrict our attention to tree-child networks. We show that a sequence for a tree-child network N reduces another network N' if and only if N' is contained in N (Theorem 5). In Section 6, we use this characterization in an algorithm for Network Containment for tree-child networks, and show that its running time is linear. We also show that, by defining an ordering on the leaves, it is possible to check whether two

cherry-picking networks are isomorphic in polynomial time (Theorem 8). In Section 7, we conclude with open problems and future directions for the use of cherry-picking strategies.

2. Preliminaries

Definition 1. A *phylogenetic network* N on a non-empty taxa set X is a directed acyclic multigraph with one *root* (the outdegree-1 source), a set $L(N)$ of *leaves* (indegree-1 sinks) bijectively labeled with X, and all other nodes are either *tree nodes* (indegree-1, outdegree at least 2) or *reticulations* (indegree at least 2, outdegree-1).

A phylogenetic network is *semi-binary* if each tree node has outdegree 2, and it is *binary* if it is semi-binary and each reticulation has indegree 2. A *phylogenetic tree* is a phylogenetic network with no reticulations.

Note that phylogenetic networks are generally defined as simple directed acyclic graphs to avoid parallel edges. We allow for parallel edges in some of our networks later on in the paper, so we define them as multigraphs.

In the rest of this paper, we drop the 'phylogenetic' term as each network in this paper is a phylogenetic network. To make the assumptions on the degrees of the network nodes clear, we always mention in the statement of a claim whether a network has to be binary, semi-binary, or there are no assumptions on the degrees. In the last case, we call the network *non-binary* even though it may be semi-binary or even binary. The following definition gives us a way of relating two networks that are not of the same nature.

Definition 2. Let N and N' be non-binary networks on X. Then N is a *refinement* of N' if N' can be obtained from N by contracting some edges.

The counterpart to contracting edges is *refining vertices* (the term *splitting vertices* is more common in graph theory, however, we use refining to stay in line with the refinement definition for networks). Let N be a network, and let w be a tree vertex of degree at least 4 in N. We *refine* w by replacing it by an edge uv where the parent of w in N is a parent of u in the new network, some of the outgoing edges of w in N are incident to u, and the rest of the outgoing edges of w are incident to v. Similarly, if r is a reticulation of degree at least 4 in N, we *refine* r by replacing it by an edge uv where the child of r in N is a child of v in the new network, some of the incoming edges of r in N are incident to u, and the rest of the incoming edges of r are incident to v. Observe that refining a tree vertex returns a network with one extra tree vertex; refining a reticulation returns a network with one extra reticulation. Contracting the newly introduced edge upon refining a vertex returns the original network.

An edge feeding into a reticulation is called a *reticulation edge*. Given an edge uv in N, we say that u is a *parent* of v and that v is a *child* of u. The node u is *above* v, or v is *below* u if there is a directed path from u to v in N. We also call u and v the *tail* and *head* of the edge uv, respectively. We say that u is a *lowest common ancestor (LCA)* of two vertices x and y if $u \notin \{x, y\}$, u is above x and y, and no other vertices below u has this property. Within trees, the LCA of any two vertices is unique; this property does not hold in networks.

We say that an edge is *binary* if the head of the edge is a degree 3 vertex. We call an edge uv an *rr-edge* if u and v are both reticulations, a *tr-edge* if u is a tree vertex and v a reticulation, an *rt-edge* if u is a reticulation and v a tree-vertex, and a *tt-edge* if u and v are both tree-vertices. We call a directed path $u_1u_2 \dots u_n$ an *rr-path* if u_i is a reticulation for all $i \in [n] = \{1, \dots, n\}$, an *rtr-path* if u_1 and u_n are reticulations and u_i is a tree vertex for at least one $2 \leq i \leq n-1$, a *trt-path* if u_1 and u_n are tree vertices and u_i is a reticulation for at least one $2 \leq i \leq n-1$, and a *tt-path* if u_i is a tree vertex for all $i \in [n]$.

A network N is *stack-free* if N contains no rr-edges. A network N is *tree-child* if it is stack-free and every tree node in N is a parent of a tree node or a leaf. This property inherently implies that every vertex in a tree-child network has a path to a leaf consisting only of tree-vertices. The *reticulation number* of a network is the total number of reticulation edges minus the total number of reticulations.

Finally, we introduce the idea of adding vertices to a network. Let x be a leaf in a network with a parent vertex p. *Adding a vertex q directly above x* is the action of deleting the edge px and adding two edges pq and qx. Upon adding a vertex to a network, the graph is no longer a network since there is a vertex of indegree-1 and outdegree-1. We note here that adding a vertex to a network is a technical operation that is always succeeded by another graph operation, in which we add an edge incident to the recently added indegree-1 and outdegree-1 vertex. While the intermediate graphs are sometimes not networks, this ensures that the resulting graph obtained from our graph operations is a network.

2.1. Cherry-picking sequences

In this subsection, we introduce cherry-picking sequences and their action on networks. This starts with definitions of specific structures within the networks called cherries and reticulated cherries. We define what it means to *reduce* such structures from networks, and show that reversing such reductions—called *adding* pairs to networks—can be done in many ways. We show that these additions can be applied to some sequence of ordered pairs of leaves to obtain a network. We impose conditions on the sequences to ensure that these additions are well-defined, and, in doing so, we formally define cherry-picking sequences and cherry-picking networks. See Fig. 2 for an illustration of the terms defined in this subsection.

2.1.1. Reducible pairs

Definition 3. Let (x, y) be an ordered pair of leaves in a non-binary network N, and let p_x, p_y denote the parents of x, y respectively. We call (x, y) a *cherry* if $p_x = p_y$, that is, if x and y share a common parent. We call (x, y) a *reticulated cherry* if p_x is a reticulation, p_y is a tree vertex, and p_y is a parent of p_x. If (x, y) is a cherry or a reticulated cherry in N, we say (x, y) is a *reducible pair*.

We may *reduce* cherries and reticulated cherries from a network to obtain a network of smaller size.

Definition 4. Let N be a network and let (x, y) be an ordered pair of leaves. *Reducing* (x, y) in N is the action of

- deleting x and suppressing degree-2 nodes in N if (x, y) is a cherry in N;
- deleting the reticulation edge between the parents of x and y and subsequently suppressing degree-2 nodes, if (x, y) is a reticulated cherry in N;
- doing nothing to N otherwise.

In all cases, the resulting network is denoted $N(x, y)$. We sometimes refer to this as *picking a cherry or pair* (x, y) *from* N. We say that an ordered pair (x, y) *affects* N if $N \neq N(x, y)$.

Given a network N and a sequence of ordered pairs S, we denote by NS the network obtained by repeatedly reducing N with each element of S in order. We say that S reduces N if NS is a network with a single leaf (for any leaf in N), a root, and no other vertices. In particular, we call these networks *single-leaf networks*. We say that a sequence of ordered pairs S *affects* N if every element of S affects N when applied successively.

2.1.2. Adding pairs to networks

As each reduction makes a simple change to a network, it is natural to attempt to reverse this change. Such reversals can be done by adding a leaf to obtain a new cherry in the network, or by adding a reticulation edge to create a new reticulated cherry. If the reduction involved the pair (x, y), then we call the reverse action *adding* (x, y) *to the network*. Since we allow for non-binary networks, it is possible to reduce reticulated cherries with a *multi-reticulation* (a reticulation with indegree at least 3). Because of this, there may not be a unique way to add the reticulation edge back: we have the option of choosing an existing reticulation vertex or a newly created reticulation vertex as the head of this reticulation edge.

A similar observation can be made for tree nodes. Just like multi-reticulations, reductions may pick cherries or reticulated cherries that contain *multifurcations* (tree nodes of outdegree more than 2). Here, we have the option of choosing an existing tree vertex or a newly created tree vertex as the tail of the inserted edge.

With this in mind, there are 6 ways of adding (x, y) to a network: 2 ways of adding cherries and 4 ways of adding reticulated cherries.

Definition 5. Let N be a non-binary network with a reducible pair (x, y). Let p_x and p_y denote the parents of x and y in $N(x, y)$, respectively (note that x and p_x may not be nodes in $N(x, y)$ if (x, y) is a cherry in N). Then we may *add* (x, y) *to* $N(x, y)$ to obtain N by using one of the following six *constructions* (see Fig. 1):

1. If x is not a leaf in $N(x, y)$ (i.e., if (x, y) is a cherry in N), then add a labeled node x, add a node q directly above y, and add an edge qx.

   (a) Do not contract any edges; or
   (b) If p_y is a tree node, then contract $p_y q$.

2. If x is a leaf in $N(x, y)$ (i.e., if (x, y) is a reticulated cherry in N), then add nodes p, q directly above x, y, respectively, and add an edge qp.

   (a) Do not contract any edges;
   (b) If p_x is a reticulation, then contract $p_x p$;
   (c) If p_y is a tree vertex, then contract $p_y q$; or
   (d) If p_x is a reticulation, contract $p_x p$; and, if p_y is a tree vertex, contract $p_y q$.

Since all tree vertices have indegree-1 and all reticulations have outdegree-1, there are no other ways of adding a reducible pair to a network other than the six ways mentioned above. Note that the constructions 1b, 2b, 2c, and 2d may only be used if the 'if' conditions are satisfied. Also note that the above actions are only well-defined if y is a leaf in $N(x, y)$. In the setting of Definition 5, this is not an issue: since we assume that (x, y) is a reducible pair of N, it is indeed the case that y is a leaf in $N(x, y)$.

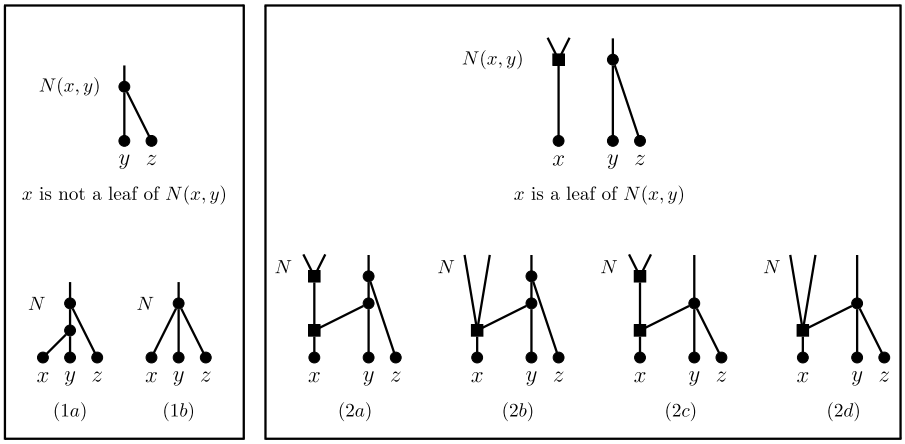

Fig. 1. The six different constructions for adding a reducible pair (x, y) to a network $N(x, y)$, as in Definition 5. The left and the right boxes show how cherries and reticulated cherries can be added, respectively: The top subgraphs show the part of the network $N(x, y)$ that will be changed by adding (x, y); the bottom subgraphs show the corresponding parts of N for the different constructions. There are more cases, for example when adding a cherry (left box) and the parent of y is a reticulation. In such cases, however, the constructions are the same for both 1a and 1b. Similarly, there are more cases for when adding a reticulated cherry. We only depict these examples, as they showcase each of the different constructions.

$$S = (2,1)(3,2)(3,4)(2,1)(1,4)$$

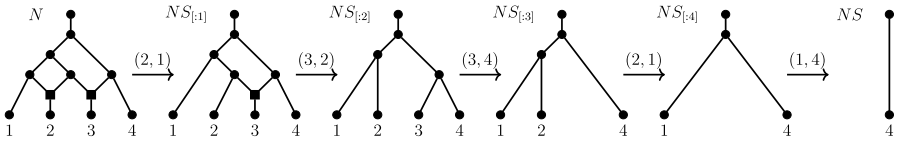

Fig. 2. A binary cherry-picking network N reduced to leaf 4 by the CPS S. The reduction is shown as a sequence of networks $NS_{[:i]}$ for $i = 0, 1, \ldots, 5$ from left to right, in which an element of S is applied to the network successively. This sequence is minimal for the network, as every element of the sequence reduces either a cherry or a reticulated cherry of the network. An example of a cherry $(3, 4)$ can be seen in the network $NS_{[:2]}$, and a reticulated cherry $(2, 1)$ can be seen in the network N. The reduction of both reducible pairs is carried out as in Definition 3. Observe that this sequence is not a tree-child sequence, as the element 2 appears as a first coordinate in S_1 and as a second coordinate in S_2. Constructing a network from a sequence S in the class $(1a, 2a)$ can be seen by moving through the six networks in reverse order (from right to left).

On the other hand, if we were to start with any sequence of ordered pairs and sought to construct a network by successively adding ordered pairs backwards through the sequence, the story would be a little different. That is, we may come across a case where, upon trying to add a reducible pair (x, y) to a network, y does not already exist in the network as a leaf. Let $S = S_1 S_2 \ldots S_{|S|} = (x_1, y_1)(x_2, y_2) \ldots (x_{|S|}, y_{|S|})$ be a sequence of ordered pairs. Starting with a network on a single leaf $y_{|S|}$, we may iteratively add S_i to the network for $i = |S|, |S| - 1, \ldots, 1$ (i.e., backwards through the sequence S), choosing a suitable construction for each ordered pair, to obtain some network. We call this *a network obtained from* S. Now, if y_i was not a leaf in the network when adding S_i, then such a construction would not be well-defined. Fortunately, we can fix this by imposing a simple condition on the sequences. This motivates the following definition.

Definition 6. A *cherry-picking sequence (CPS)* on a set X is a sequence of ordered pairs on distinct elements from X, such that the second coordinate of each ordered pair occurs as a first coordinate in some ordered pair in the rest of the sequence, or as the second coordinate of the last pair.

Returning to the example that we had before, we observe if S was a CPS, then y_i must already have been a leaf in the network when adding $S_i = (x_i, y_i)$. By definition of CPSs, y_i appears as a first coordinate in some ordered pair that appears after S_i, or y_i appears as the second coordinate of the final ordered pair, which implies that the network contains the leaf y_i in both cases when adding $S_i = (x_i, y_i)$. Therefore, this construction is well-defined, and we can always obtain a network from a CPS. This brings us to the definition of a cherry-picking network.

Definition 7. A network on X is a *cherry-picking network (CPN)* if it can be obtained from some CPS S. Equivalently, a CPN is a network that can be reduced by some CPS.

See Fig. 2 for an example of a CPN with a CPS that reduces it. See Fig. 3 for examples of networks that are not CPNs. In particular, single-leaf networks are also CPNs, since these can be reduced by the empty CPS. By definition, a CPN with at

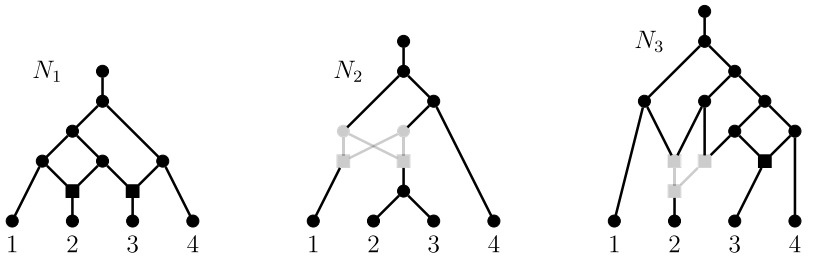

Fig. 3. A CPN N_1 and two non-CPN networks N_2 and N_3. We know from Fig. 2 that N_1 is a CPN. N_2 contains a cherry $(2, 3)$; upon reducing $(2, 3)$, the resulting network does not contain a reducible pair because the 'crown' structure (in gray) prevents cherries and also reticulated cherry. N_3 contains a reticulated cherry $(3, 4)$; upon reducing $(3, 4)$, the resulting network does not contain a reducible pair because the gray structure prevents 2 from being picked as a cherry or a reticulated cherry.

least two leaves contains either a cherry or a reticulated cherry. Intuitively, reducing these structures returns a network of smaller size that is a CPN; we may repeatedly reduce cherries and reticulated cherries until the network has been reduced.

A *subsequence* of a CPS refers to any sequence of ordered pairs that can be obtained by deleting some elements from the CPS. Note that a subsequence need not be a CPS. In what follows, we will often have to reduce a network by a subsequence of a CPS. These subsequences are most often the initial parts of the sequence, and hence we introduce notation for them. Let $S = (x_1, y_1)(x_2, y_2)\dots(x_n, y_n)$ be a CPS. For $i \in [n]$, we use the following notations to denote some subsequence of S. The ith ordered pair of S is $S_i = (x_i, y_i)$. The first i ordered pairs in S are denoted by $S_{[:i]} = (x_1, y_1)\dots(x_i, y_i)$. The subsequence of S without the first i ordered pairs is denoted by $S_{[i+1:]} = (x_{i+1}, y_{i+1})(x_{i+2}, y_{i+2})\dots(x_n, y_n)$. We let $S_{[:0]}$ denote the empty sequence.

A CPS S is *minimal* for a CPN N if S reduces N and each ordered pair S_i of S affects $NS_{[:i-1]}$ for all $i \in [|S|]$. In other words, NS is a network on a single leaf, and $NS_{[:i-1]} \neq NS_{[:i]}$ for all $i \in [|S|]$. We often write *a CPS of/for a network N* to refer to a minimal CPS for N.

A *partial CPS* S' of length i is a sequence of ordered pairs such that there exists a CPS S where $S_{[:i]} = S'$. If S and S' are partial CPSs and N is a non-binary network, then applying S and then S' is the same as appending S' to S, denoted SS', and applying the whole sequence. In notation, we write

$$(NS)S' = N(SS'),$$

and hence we denote this network without brackets as NSS'.

Observation 1. *Let N be a non-binary CPN that can be reduced by a CPS S. Then the network $NS_{[:i]}$ is a CPN for all $i = 1, \dots, |S|$.*

By choosing a suitable construction, we may obtain a CPN from any of its minimal CPSs.

Observation 2. *Every non-binary CPN can be obtained from a minimal CPS that reduces it.*

2.2. CPN classes

Using different combinations of the six constructions from Definition 5 can yield different CPNs from the same CPS. These differences could be due to the nature of the network vertex degrees (binary, semi-binary, non-binary) or due to their topological features (stack-free, tree-child). One way of categorizing these CPNs is to choose and stay consistent with one particular construction for adding cherries and reticulated cherries to networks. That is, we construct networks from CPSs with a chosen construction A to add cherries and a chosen construction B to add reticulated cherries.

There are two motivations to do so. Firstly, this categorizes CPNs into classes defined by their topological restrictions. We may specify classes of CPNs that contain only binary networks, those that contain only semi-binary networks, those without stacks, and many more. Secondly, and more importantly, we can introduce some notion of a correspondence between CPNs and minimal CPSs that reduce them. Within some CPN classes, it turns out that if a sequence is minimal for two networks, then the networks must be isomorphic.

Definition 8. Let A and B be a cherry construction and a reticulated cherry construction, respectively. We let (A, B) denote the class of all CPNs that can be obtained from CPSs by using the suitable constructions A or B.

Within the CPN class (A, B), we say that we use the (A, B)-*construction* to obtain CPNs from CPSs. We write $N \in (A, B)$ or say that N is an (A, B)-CPN to mean that N is a CPN in the CPN class (A, B).

Since there are two cherry constructions and four reticulated cherry constructions, there are in total eight CPN classes. For example, the CPN class $(1a, 2a)$ contains all binary CPNs (see Fig. 2 for an example of obtaining a $(1a, 2a)$-class CPN

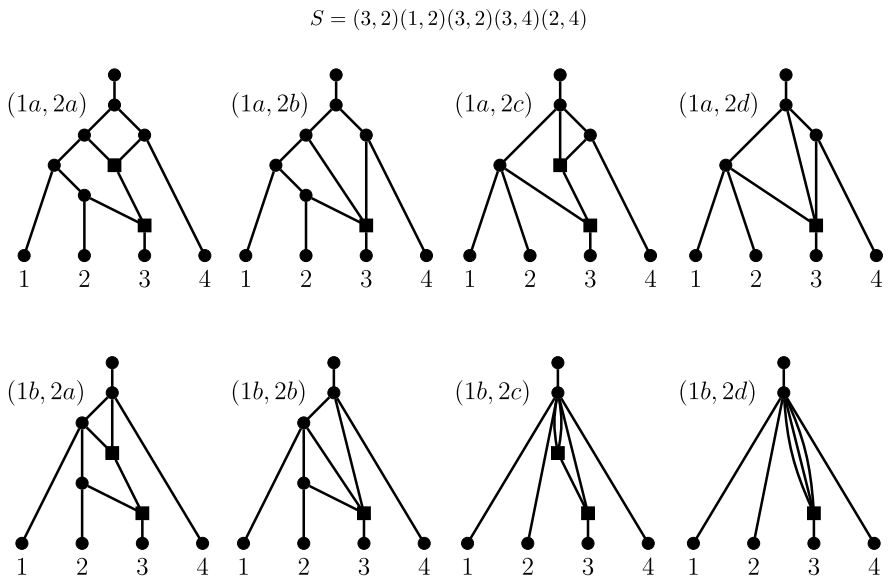

Fig. 4. A CPS S and the unique networks obtained from it within the eight CPN classes. Observe that the eight networks are distinct.

from a CPS). We note that it is possible obtain the same CPN from the same CPS within different CPN classes. Indeed, a CPS corresponding to a tree will give the same network in all the CPN classes that use the 1a cherry construction, and the same can be said for CPN classes that use the 1b cherry construction. On the other hand, there do exist CPSs that return distinct networks amongst the different CPN classes; an example of such a CPS is given in Fig. 4.

Suppose that we are given a CPN N within a CPN class (A, B), and let S be a minimal CPS that reduces N. To form some notion of correspondence between CPNs and the sequences that reduce them, we pose the following question: is it always the case that applying the (A, B) construction on S returns the network N? It turns out that this is true only for half of the CPN classes. We start by defining what it means for a CPN class to be *reconstructible*.

Definition 9. A CPN class $C = (A, B)$ is called *reconstructible* if for any two networks $N, N' \in C$ with a common minimal CPS, we have that N and N' are isomorphic.

Since the construction is fixed, each CPS gives rise to a unique network within each of the CPN classes. Then if two distinct networks N and N' have a common minimal CPS S, at most one of these networks, say N, can be constructed from the sequence. This means that although S is a minimal CPS of N', it cannot be used to construct the network N'. Indeed, there does exist some minimal CPS of N' which can be used to construct N'. Reconstructible CPN classes have the nice property that for a given CPN N, *any* minimal CPS for N can be used to construct N.

Lemma 1. *Let* $(A, B) \in \{(1a, 2a); (1a, 2b); (1b, 2c); (1b, 2d)\}$. *Let* N *be a CPN in the* (A, B)*-class, and let* (x, y) *be a reducible pair in* N. *Then adding* (x, y) *to* $N(x, y)$ *using the* (A, B) *construction results in* N.

Proof. Observe that these four classes are characterized by the following properties. The networks in $(1a, 2a)$ are binary; the networks in $(1a, 2b)$ do not contain rr-edges; the networks in $(1b, 2c)$ do not contain tt-edges; and the networks in $(1b, 2d)$ do not contain rr-edges nor tt-edges. Since N is a network of one of these classes, N must also have these properties. Furthermore, the network $N(x, y)$ also has these properties, since deleting edges and potentially suppressing vertices does not create new vertices, which may subdivide existing rr-edges or tt-edges. This means that upon inserting an edge to the network (as a result of adding a cherry or a reticulated cherry (x, y)), one should either do nothing; contract all rr-edges; contract all tt-edges; or contract all rr-edges and tt-edges, depending on which class of CPNs are being considered. Note that these contractions, should they occur, only involve vertices that have just been added as a result of adding the reducible pair, since $N(x, y)$ also has the properties. This is precisely what happens when we add (x, y) back to the network $N(x, y)$ using the respective constructions, and it follows immediately that the constructions defined in these CPN classes return the original network N. □

Lemma 1 states that four of the eight CPN classes have the property that adding back a reduced pair to the network returns the original network. By applying this lemma in the construction of a network from a CPS, we obtain the following corollary.

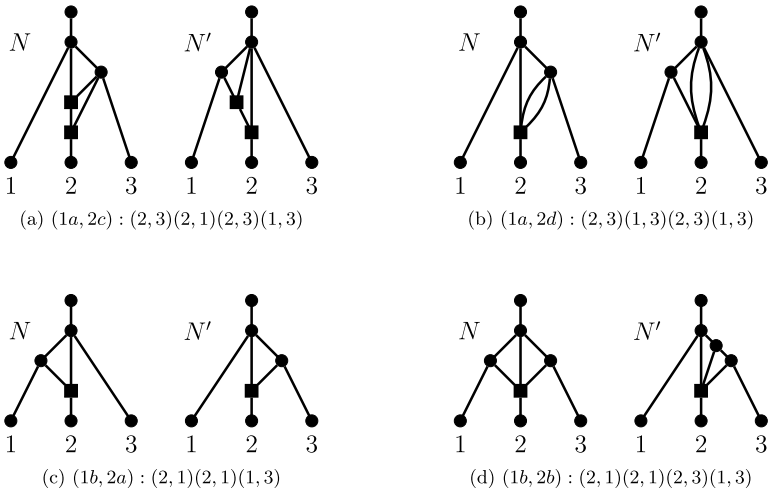

Fig. 5. Two distinct networks N and N' that can be reduced by the same minimal CPS for the $(1a, 2c), (1a, 2d), (1b, 2a), (1b, 2b)$-classes. The networks obtained by using the respective constructions are N. This means that given a network and a minimum sequence that reduces it, the sequence cannot always be used to construct the original network (let N' be the original network in these four cases).

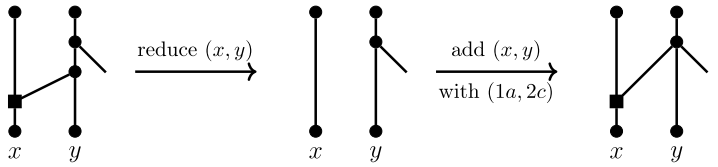

Fig. 6. Reducing a reticulated cherry (x, y) and adding it back using the construction $(1a, 2c)$ can return a different network.

Corollary 1. *Let* $(A, B) \in \{(1a, 2a); (1a, 2b); (1b, 2c); (1b, 2d)\}$. *Then* (A, B) *is reconstructible.*

To show that the above lemma and the corollary do not hold for the other four CPN classes, we present two networks with a common minimal CPS for each of the CPN classes in Fig. 5. Unlike their reconstructible counter-parts, the networks in these four classes can contain both tt-edges and rr-edges whilst also containing multifurcations and multi-reticulations. When constructing networks from CPSs, this allows for a mixture of choosing to contract some tt-edges and some rr-edges, but not all. This can make adding reticulated cherries problematic. Take the $(1a, 2c)$ class for example. Since there can exist tt-edges that are binary, we may, in particular, assume that a network in the class contains a reticulated cherry (x, y) where the parent of y is a head of a binary tt-edge e. But this means that upon reducing (x, y) and adding back the reticulated cherry using the 2c construction, we essentially contract this tt-edge, which returns a different network (see Fig. 6).

2.2.1. Refinement of constructed networks

The six constructions that were introduced in Definition 5 can be rephrased as follows. When adding (x, y) to a network, check if x is a leaf in the network. If x is not a leaf in the network, then add a labeled leaf x and an edge from the (newly added) parent of y to x (add a cherry). If x is a leaf in the network, then add an edge between the newly added parents of y and x (add a reticulated cherry). Decide whether or not to contract some of the edges incident to the parent of x and edges incident to the parent of y. This means that given some CPS S, the binary network N in the $(1a, 2a)$-class constructed from S is a refinement of all networks that can be constructed from S, using any combination of the constructions. This gives the following observation.

Lemma 2. *Given a non-binary CPN* N *and a minimal CPS* S *for* N, *there exists a binary refinement* N_b *of* N *such that* S *is a minimal CPS for* N_b.

Proof. The unique binary network N_b obtained by using construction $(1a, 2a)$ on S is a refinement of N, and S is a minimal CPS for N_b by definition of this network. □

Finally, the following lemma shows how general refinements of CPNs (not necessarily binary) are related to the CPNs.

Lemma 3. *Let* N_r *be a refinement of a non-binary network* N *that is a CPN. Then* N *is a CPN, and every minimal CPS of* N_r *is also a minimal CPS of* N.

Proof. We prove this statement by induction on $|N|$, the number of edges in N. For the base case take the single-leaf network. So suppose that for every network of size at most $|N|-1$, the claim is true.

Let S be a minimal CPS of N_r, and let $S_1 = (x, y)$ be the first element of S. Since N_r can be obtained from N by refining vertices, it must be the case that S_1 is also a reducible pair in N. Furthermore, if S_1 is a cherry in N_r then S_1 is also a cherry in N; if S_1 is a reticulated cherry in N_r then S_1 is also a reticulated cherry in N. Now it is easy to see that $N_r S_1$ is a refinement of NS_1 Note that $|NS_1| < |N|$ since every reduction reduces the size of the network. The network $N_r S_1$ is a CPN by Observation 1. By induction hypothesis, NS_1 is a CPN and every minimal CPS of $N_r S_1$ is also a minimal CPS of NS_1. Then in particular, $S_{[:2]}$ is a minimal CPS of NS_1. It follows then that S is a minimal CPS of N. □

Note that the converse of Lemma 3 does not hold in general. Consider the tree T on three leaves $\{x, y, z\}$ that all share a common parent (the claw graph with a root). Let T_r be a binary refinement of T in which x and y form a cherry. Then the CPS $(y, z)(x, z)$ is minimal for T but not for T_r.

3. Properties of cherry-picking networks

In this section, we investigate properties of cherry-picking networks. First, we continue where we left off in the previous section: we inspect the relation between CPSs and CPNs. This includes the reticulation number defined by a CPS, changes in the sets of reducible pairs that are ready for picking after picking a pair, and the order in which we can reduce a network. The last of these allows us to consider distinguishability of two CPNs by their CPSs. Then, we use this to investigate the relation between embedded networks of a CPN and its CPSs.

3.1. Why CPNs are nice: order does not matter

Lemma 4. *Let S be a minimal length sequence of ordered pairs of leaves that reduces a non-binary network N. Then S is a CPS. Furthermore, $|S| = n + r - 1$, where n and r denote the number of leaves and the reticulation number of N, respectively.*

Proof. Suppose for a contradiction that S is not a CPS. Then, there is an $i < |S|$ with $S_i = (x, y)$ such that y is not a first coordinate in any of the elements of $S_{[i+1:]}$ or the second coordinate of $S_{|S|}$. This means y cannot be a leaf in $NS_{[:i-1]}$ (if it were, then S would not reduce N). This implies $NS_{[:i-1]} = NS_{[:i]}$, and there is a shorter sequence $S_{[:i-1]}S_{[i+1:]}$ that reduces N, a contradiction. We conclude that S is a CPS.

We now prove the second part of the lemma. Let $S_i = (x, y)$. We first construct a binary network M from S using the $(1a, 2a)$ construction. Upon constructing $MS_{[:i-1]}$ from $MS_{[:i]}$, a new leaf x is added if x is not a leaf in $MS_{[:i]}$, and a reticulation is added otherwise. By Lemma 2, M is a binary refinement of N, and therefore N has the same leaf set and it has the same reticulation number as that of M. Since S is a minimal CPS for N, it follows that $|S| = n + r - 1$. □

Definition 10. Let N be a non-binary network. Denote with $\mathcal{C}_c(N)$ the set of cherries of N, and with $\mathcal{C}_r(N)$ the set of reticulated cherries of N. The set of all reducible pairs is denoted $\mathcal{C}(N) = \mathcal{C}_c(N) \cup \mathcal{C}_r(N)$.

The following lemma states that all new reducible pairs after picking a pair (x, y) must involve either x or y.

Lemma 5. *Let N be a non-binary network on a taxa set X, and let (x, y) be a reducible pair of N. Then we have the following inclusion:*

$$\mathcal{C}(N(x, y)) \setminus \mathcal{C}(N) \subseteq (\{x, y\} \times X) \cup (X \times \{x, y\}).$$

Proof. Note that the LHS of the containment relation represents the reducible pairs in $N(x, y)$ that were not present in N. Suppose, for contradiction, that this set contains a pair (z, w) not involving x or y. Then, this pair is not reducible in N, but it is in $N(x, y)$. Adding the pair (x, y) back into $N(x, y)$ may only subdivide the pendant edges leading to x and y. This implies that this action will not change the fact that z and w form a reducible pair. Therefore, (z, w) is a reducible pair in N as well, a contradiction. Hence, all new cherries and reticulated cherries of $N(x, y)$ involve x or y. □

We also have similar inclusions for looking at reducible pairs in the original network that are not reducible pairs in the new network. Roughly speaking, the following lemma states that reducing a network by the element (x, y) preserves the other reducible pairs.

Lemma 6. *Let N be a network on X, and (x, y) a reducible pair of N. Then, if N is non-binary, we have the inclusion $\mathcal{C}(N) \setminus \mathcal{C}(N(x, y)) \subseteq \{x\} \times X \cup X \times \{x\}$, and in particular*

$$\mathcal{C}_c(N) \setminus \mathcal{C}_c(N(x, y)) \subseteq \{x\} \times X \cup X \times \{x\},$$

and

$$\mathcal{C}_r(N) \setminus \mathcal{C}_r(N(x, y)) \subseteq \{x\} \times X.$$

If N is semi-binary, the inclusions can be sharpened to $\mathcal{C}(N) \setminus \mathcal{C}(N(x, y)) \subseteq \{(x, y), (y, x)\}$, with

$$\mathcal{C}_c(N) \setminus \mathcal{C}_c(N(x, y)) \subseteq \{(x, y), (y, x)\},$$

and

$$\mathcal{C}_r(N) \setminus \mathcal{C}_r(N(x, y)) \subseteq \{(x, y)\}.$$

Proof. Let N be a non-binary network and let (x, y) be a reducible pair of N. Let (z, w) be a reducible pair in N where $z \neq x$ and $w \neq x$. We claim that (z, w) must also be a reducible pair in $N(x, y)$. Suppose for a contradiction that it was not. Adding the pair (x, y) may only subdivide the pendant edges leading to x and y; this action will not change the fact that (z, w) does not form a reducible pair. But this means that (z, w) is not a reducible pair in N, which is a contradiction. Therefore, (z, w) must also be a reducible pair in $N(x, y)$. It follows that every reducible pair in N that is not a reducible pair in $N(x, y)$ must contain the element x. Note that since semi-binary networks are non-binary, this fact also holds for semi-binary networks.

If (x, y) is a cherry in N, then, since the network is non-binary, it is possible for x to form a cherry with another leaf, say z. Then (x, z) and (z, x) are both reducible pairs in N, while they are not reducible pairs in $N(x, y)$ since x is not a leaf in $N(x, y)$. On the other hand, if (x, y) is a reticulated cherry in N, then x may only appear as a first coordinate in a reducible pair of N. Therefore the inclusions for non-binary networks follow.

Suppose now that N is semi-binary. As stated in the first paragraph of this proof, every reducible pair in N that is not a reducible pair in $N(x, y)$ must contain the element x. If (x, y) is a cherry in N, then x may only be in a cherry (and reducible pair) with the leaf y. If (x, y) is a reticulated cherry in N, then the only reticulated cherry involving the parent p_y of y is (x, y). All other reticulated cherries that contain x do not involve p_y, as p_y is of outdegree-2: this implies all reticulated cherries in N involving x (as the first coordinate) is still a reducible pair in $N(x, y)$. The inclusions for the semi-binary networks then follow. □

We now start our investigation into the order in which pairs can be reduced. We start with a lemma that implies a cherry on two leaves x and y can be reduced either as (x, y) or as (y, x). Then we show that reducing an arbitrary pair in a CPN gives a new CPN.

Lemma 7. *Let S be a minimal CPS for a non-binary CPN N and suppose $S_i = (x, y)$ reduces a cherry when applying the sequence. Let z and w be distinct leaves (not necessarily different from x and y) that have a common parent, equal to the parent of x and y. Let S' be the sequence $S_{[i+1:]}$ where each occurrence of z is replaced by x. Then $S_{[:i-1]}(z, w)S'$ is a minimal CPS for N.*

Proof. Because (x, y) forms a cherry in $N' = NS_{[:i-1]}$, and x and y share their parents with z and w, the reduced network $N'(x, y)$ is equal to the network $N'(z, w)$ when z is replaced by x. Hence, if we switch the roles of x and z in the remaining part of the sequence, the result after reduction by both sequences is the same modulo the $x \leftrightarrow z$ replacement. □

Lemma 8. *Let N be a non-binary CPN that can be reduced by a minimal CPS $S = S_1, S_2, \ldots, S_{|S|}$ such that $S_2 \in \mathcal{C}(N)$. Then NS_2 is a CPN.*

Proof. Note that $S_1, S_2 \in \mathcal{C}(N)$ by assumption. We distinguish several cases and prove in every case that NS_2 is a CPN.

- **The leaves in S_1 and S_2 are the same.** Then either $S_1 = S_2$, or $S_1 = (x, y)$ and $S_2 = (y, x)$ for some pair of leaves x, y. In the first case $NS_2 = NS_1$, which is a CPN. In the second case, as (x, y) and (y, x) are both present in N, (x, y) must be a cherry in N. This means that $NS_1S_2 = NS_1$, and thus S is not a minimal CPS for N. This case is not possible.

Let $S_1 := (x, y)$.

- **The pairs S_1 and S_2 have exactly one leaf in common.**
  - **S_2=(x,z).** The common leaf x is below the reticulation common to the two reticulated cherries. Applying S_1 and S_2 in any order removes these two reticulation edges, so clearly $NS_1S_2 = NS_2S_1$. By Observation 1, NS_1S_2 is a CPN. This implies NS_2S_1 is a CPN and, therefore, that NS_2 is also a CPN.
  - **S_2=(z,x).** Observe first that (x, y) cannot form a reticulated cherry, as otherwise the first coordinate of every reducible pair that involves x is x, which contradicts our assumption that $S_2 = (z, x) \in \mathcal{C}(N)$. Therefore (x, y) must be a cherry. Then the network $NS_1 = N(x, y)$ does not have the leaf x, which implies that $S_2 = (z, x)$ is not a reducible pair of NS_1. This contradicts the fact that S was a minimal CPS for N, and therefore this case is not possible.
  - **S_2=(y,z).** The two possibilities for this case are either that x, y, z all share the same parent, or that (x, y) form a reticulated cherry in N and z shares a common parent with y. In the former case, NS_1S_2 is the CPN obtained by

deleting the leaves x and y and suppressing all degree-2 vertices. We obtain the same CPN by picking the cherries $S_2 = (y, z)$ and (x, z) in succession, that is, $NS_2(x, z) = NS_1S_2$. This implies that NS_2 is also a CPN. A similar argument can be done for the reticulated cherry case—it is easy to see that $NS_2(x, z) = NS_1S_2$.

- **S_2=(z, y).** This is the case where either y and z share a common parent, or (z, y) forms a reticulated cherry. In both of these cases, the leaf x could share a common parent with y, or (x, y) could be a reticulated cherry (there are in total 4 possible cases). In all cases, reducing N by S_1 first or by S_2 first has no real difference, and so $NS_1S_2 = NS_2S_1$. For the same reason as before, NS_2 is a CPN.

- **The pairs S_1 and S_2 have no leaf in common.** Then obviously, S_1 and S_2 independently remove edges in N, not influenced by the order of S_1 and S_2. Hence we get $NS_1S_2 = NS_2S_1$ and for the same reason as before, NS_2 is a CPN.

In all cases, we have concluded that NS_2 is a CPN, so the result follows. □

Lemma 9. *Let N be a non-binary network, and $(x, y) \in \mathcal{C}(N)$. Then, there exists a minimal CPS S for N such that $S_i = (x, y)$ or $S_i = (y, x)$ for some i, and (x, y) is reducible until that point, i.e., $(x, y) \in \mathcal{C}(NS_{[:j]})$ for all $j < i$.*

Proof. Let S be a minimal CPS for N. If S contains (x, y) or (y, x) as S_i, and (x, y) is a reducible pair in $NS_{[:j]}$ for all $j < i$, we are done, so assume that this is not the case. Let $i > 0$ be minimal such that $(x, y) \notin \mathcal{C}(NS_{[:i]})$. Then $S_i = (x, z)$ or $S_i = (y, z)$ for some $z \neq x, y$ by Lemma 6. Because $(x, y) \in \mathcal{C}(NS_{[:i-1]})$, $(x, y) \notin \mathcal{C}(NS_{[:i]})$, and the second element of S_i is z, (y, z) must form a cherry in $NS_{[:i-1]}$.

First, suppose that (x, y) forms a reticulated cherry in $NS_{[:i-1]}$, and that $S_i = (x, z)$. In that case, $NS_{[:i]} = NS_{[:i-1]}(x, y)$, so replacing S_i with (x, y) in S gives a new minimal CPS for N that contains (x, y). Next, suppose that (x, y) forms a reticulated cherry in $NS_{[:i-1]}$, and that $S_i = (y, z)$. Then, upon switching the roles of y and z, we have $NS_{[:i]} = NS_{[:i-1]}(z, y)$. Letting S' denote the sequence $S_{[i:]}$ where each occurrence of z is replaced by y, we obtain a minimal CPS $S^{new} = S_{[:i-1]}(z, y)S'$ for N. In this sequence, we have that the minimal value $k > 0$ for which $(x, y) \notin \mathcal{C}(N^{new}_{[:k]})$ satisfies $k > i$. We may repeat this until we enter the first case; such a process must terminate as the length of S is finite. On the other hand if (x, y) forms a cherry in $NS_{[:i-1]}$, then x, y and z share a common parent. Therefore, by Lemma 7, there is a minimal CPS for N that starts with $S_{[:i-1]}(x, y)$. □

Proposition 1. *Let N be a non-binary CPN with $c \in \mathcal{C}(N)$. Then Nc is a CPN. That is, there exists a CPS S such that cS is a CPS reducing N.*

Proof. Let $c = (x, y)$. By Lemma 9, there is a CPS S' for N that contains either (x, y) or (y, x), and (x, y) is reducible until that point in the sequence. If $S'_1 = (x, y)$, then set $S := S'_{[2:]}$ and we are done. Now suppose S'_1 is not equal to (x, y). Note that there must be a smallest $i \geq 1$ with $S'_i = (x, y)$ or $S'_i = (y, x)$.

- **Suppose S'_i=(x, y).** Recall that we have $(x, y) \in \mathcal{C}(NS'_{[:j]})$ for all $j < i$. Hence, by applying Lemma 8 i times, $N(x, y)$ is a CPN.
- **Suppose S'_i=(y, x).** Again, we have $(x, y) \in \mathcal{C}(NS'_{[:j]})$ for all $j < i$. Hence, $NS'_{[:i-1]}$ has both reducible pairs (x, y) and (y, x), and it must contain the cherry (x, y). By Lemma 7 there is a CPS of N starting with $S'_{[:i-1]}(x, y)$. Redefining S' as this sequence, we are in the previous case and thus $N(x, y)$ is a CPN.

We conclude that Nc is a CPN. □

The following theorem is a corollary of the previous proposition. It essentially states that a network can be cherry picked in any order.

Theorem 1. *Let N be a non-binary CPN, and S a partial CPS. If in each step of the reduction of N by S, the network is changed, then there exists a minimal CPS S' starting with S that reduces N.*

3.2. Distinguishability

By Theorem 1, any order of picking reducible pairs gives a minimal CPS for a CPN. This inherently implies that for a given CPN, there could be many CPSs that reduce it. However, given a class (A, B), every CPS uniquely constructs a CPN in that class by Lemma 2.

Remark 1. Within a CPN class, exactly one CPN can be constructed for each CPS. On the other hand, a CPN can have more than one minimal CPS that reduces it.

While this remark holds true for all eight of the CPN classes, only the classes that are reconstructible are interesting to examine. The aim of this subsection is to set up some distinguishability notion of CPNs using their minimal CPSs. That is,

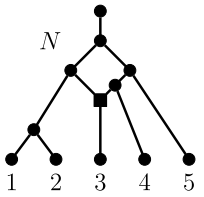

Fig. 7. The smallest CPS for this network is $(1,2)(3,2)(3,4)(4,5)(2,5)$. Initially we have the choice of picking either $(1,2)$, $(2,1)$, or $(3,4)$. For the smallest CPS we pick the smallest reducible pair $(1,2)$.

we would like to encode each CPN by delegating one of its minimal CPSs to be its representative, such that the sequence can be used to reconstruct the CPN. Since there could be more than one CPN that can be reduced by the same minimal CPS within CPN classes that are not reconstructible, it makes no sense to consider these classes. Therefore, we define a distinguishability notion only for the classes that are reconstructible.

Within a reconstructible CPN class, each network can have many minimal CPSs that reduce it by Remark 1. To choose a representative from these minimal CPSs, we introduce an ordering on the CPSs. Doing so allows us to prescribe a unique *smallest* CPS to each CPN. So let us take an arbitrary ordering on the leaves, and let us define a lexicographical ordering on the reducible pairs as follows. We say that $(a,b) < (c,d)$ if and only if $a < c$ or if $a = c$ and $b < d$. We naturally extend this ordering to minimal CPSs. Let S and S' be CPSs such that $|S| \neq |S'|$. If $|S| < |S'|$, then $S < S'$—this ensures the smallest CPS is minimal. Now suppose $|S| = |S'|$ and let i be the smallest index such that $S_i \neq S'_i$. If no such i exists, then $S = S'$; otherwise, $S < S'$ if and only if $S_i < S'_i$.

By Theorem 1, we may pick a CPN in any order. We define a *smallest* CPS as one that is obtained by picking the smallest reducible pair at each iteration (see Fig. 7). Such a sequence is naturally a minimal CPS. By the following theorem, distinguishing two CPNs of the same reconstructible class comes down to finding their smallest CPS and checking whether these are the same.

Theorem 2. *Suppose we are given an ordering on the taxa set X. Every CPN on X has a unique smallest CPS. In particular within a reconstructible CPN class, these CPSs can be used to reconstruct the CPN. Every CPS can be used to construct a unique CPN within each of the eight CPN classes.*

Proof. Let N be a CPN on X. Since we have a total ordering on the cherries of N, we have that if there exists a smallest CPS then it is unique. Furthermore, we know that a smallest CPS exists: simply pick a smallest cherry at every iteration. Therefore every CPN on X has a unique smallest CPS. Within reconstructible CPN classes, no two networks have the same minimal CPSs. It then follows that a smallest CPS for a network can be used to construct said network.

By Remark 1, we have that every CPS gives rise to a unique CPN. □

The following corollary is a direct consequence of Theorem 2.

Corollary 2. *Suppose we are given an ordering on the taxa set X. Within a reconstructible CPN class, two CPNs on X are isomorphic if and only if they have the same smallest CPS.*

This leads to a polynomial-time algorithm for checking whether two CPNs of a reconstructible CPN class on the same set of taxa are isomorphic, which we describe in Section 6.

4. Reduction and containment

In this section, we prove that, within reconstructible CPN classes, the reduction of a network by a CPS for another network implies 'containment' of the former in the latter. We also show that the converse does not always hold: containment of a network N' in another network N does not imply that there exists a minimal CPS of N that reduces N'.

4.1. Reduction implies containment

We first formally define what it means for a network to contain another network.

Definition 11. Let N be a non-binary network on the set of taxa X. A non-binary network N' on $X' \subseteq X$ is a *subnetwork* of N if N' can be obtained from N by deleting reticulation edges, and then *cleaning up w.r.t. X'*, i.e., applying the following changes until a network on X' is obtained:

- removing outdegree-0 nodes not labeled by X', together with their incoming edges;
- suppressing all degree-2 nodes.

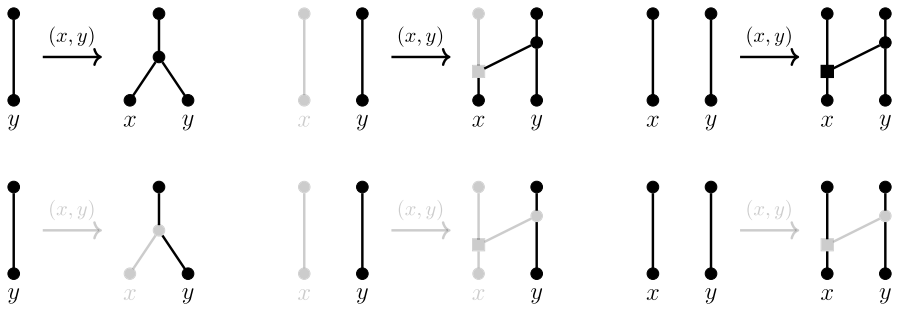

Fig. 8. The visualization of the three cases in Lemma 12 (each column corresponds to a case). The six different cases for adding a pair (x, y) to a network N, and possibly adding (x, y) to a subnetwork N' of N. N is indicated by the black and gray nodes and edges; N' is indicated by only the black nodes and edges. Only the parts of the networks that will be affected by the addition of (x, y) is shown here. The part to the left of each arrow represents the networks before the addition of (x, y); the part to the right of each arrow represents the networks after the addition of (x, y). A black arrow indicates that (x, y) is added also to N'; a gray arrow indicates that (x, y) will not be added to N'. The embedding of the black network into the black and gray network show that in any of the six cases, regardless of whether (x, y) is added to N' or not, the resulting network is still a subnetwork of the other.

Equivalently, N' is a subnetwork of N if there is an embedding of N' in N: an injective map of the nodes of N' to a subset of the nodes of N, and of the edges of N' to edge disjoint paths of N, such that the mapping of the edges respects the mapping of the nodes. A non-binary network N *contains* another non-binary network N' if some refinement N'_b of N' is a subnetwork of N, i.e., if N' can be obtained from a subnetwork of N by contracting edges.

The difference between being a subnetwork of a network and being contained in a network can be seen in Fig. 8. As seen above, a subnetwork of a network can be defined by deleting reticulation edges and cleaning up, but also with embeddings. The equivalence of these two definitions has been shown for when the two networks are binary and on the same leaf-sets (Lemma 1 of [21]). It is easy to extend this equivalence to non-binary networks on different leaf-sets (in which one leaf-set is a subset of the other).

Lemma 10. *Let N and N' be non-binary networks on X and $X' \subseteq X$. N' can be embedded into N if and only if N' can be obtained from N by deleting a set of reticulation edges and then cleaning up w.r.t. X'.*

Proof. First suppose there is an embedding of N' into N. Note that in this embedding, we may assume that the root of N' is mapped to the root of N. The image of this map (a subgraph of N) is a subdivision of N'. Let R be the set of reticulation edges of N not used in the embedding. We will show that N' can be obtained from N by removing R and cleaning up w.r.t. X'. To show this, we prove that the edges that are removed in the clean-up are exactly the edges not used by the embedding of N' into N.

Let M be the network obtained from N by removing R and cleaning up w.r.t. X'. First note that no edges used by the embedding are removed in the process of removing R and cleaning up: indeed, for each such edge, there is a path to a leaf of X' using only edges of the embedding, which cannot be removed by cleaning up. Now suppose M has an edge that is not used in the embedding of N' into N. Consider a lowest such edge xy.

Node y cannot be a leaf of N, because all leaves of N are in the embedding of N' into N; or they are removed in the clean-up because they are not in X', in which case they cannot be part of M.

Now suppose that y is a tree node of N. It is impossible for an outgoing edge of y to be in the embedding, because the root of the embedding is the root of N. Hence, the outgoing edges of y are not in the embedding. At least one of these outgoing edges of y is in M, because, otherwise, y would have been deleted by cleaning up outdegree-0 nodes. Hence, at least one outgoing edge of y is in M but not in the embedding of N' into N, contradicting the assumption that xy is a lowest such edge.

Lastly, suppose that y is a reticulation. If none of the other incoming edges of the reticulation are in the embedding, it follows, similarly to the previous case, that the outgoing edge of y is in M but not in the embedding. This contradicts the assumption that xy is a lowest such edge. Hence, at least one incoming edge of y is used by the embedding. This implies xy is an element of R, and has been deleted, contradicting the assumption that xy is an edge of M.

For the other direction, suppose N' can be obtained from N by removing a set of reticulation edges R and cleaning up. By reversing the operations used to clean up, we get an embedding of N' into N. Indeed, this only introduces reticulation edges not used by the embedding, and subdivides edges. When subdividing an edge used by the embedding, adapt the embedding accordingly, by mapping the edge of N' to the resulting path. □

In what follows, in settings where we consider whether N' is a subnetwork of/contained in N, we informally refer to N as the larger network and N' as the smaller network. Note that the notions of a network being a subnetwork and a network being contained are not always synonymous. If N' is a subnetwork of N, then N' is contained in N. However, if N' is contained in N, then it does not immediately follow that N' is a subnetwork of N. They are synonymous when the

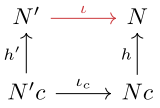

Fig. 9. The commutative diagram for the networks N, N', Nc, and $N'c$ as in the setting of Lemma 12. Given the maps h, h', and ι_c, we wish to produce the map ι.

smaller network (i.e., N') is binary. The following lemma shows that for any non-binary network (not necessarily a CPN), the network obtained by reducing a pair is a subnetwork of the original network.

Lemma 11. *Let N be a non-binary network, and c a pair of leaves. Then Nc is a subnetwork of N.*

Proof. For the embedding of Nc into N, note that there is a natural map of the nodes of Nc to the nodes of N. Each edge of N, corresponds naturally to an edge of Nc, or is part of a path (of two edges) if an endpoint got suppressed. These mappings form an embedding of Nc into N, as the mapping of the nodes is respected, and no edge of Nc is part of more than one corresponding path of N. □

To show the relation between subsequences and containment, we first focus on the binary CPN class, $(1a, 2a)$, for which the definitions of subnetwork and containment are synonymous.

4.1.1. Subnetworks

Intuitively, when a CPS S for a binary network N also reduces another binary network N', the embedding of the network N' in N can be found as follows. Reconstruct the network N from S, and annotate the edges used by N' in the process. Let S' denote the CPS of ordered pairs in S that is used in the reduction of N'. Let $S_i = (x, y)$ be an ordered pair that appears in S'. Then in $NS_{[:i]}$, label the paths $p_y y$ and $p_y x$ as 'used'. Upon reconstructing N, the embedding of N' into N can be seen as the subnetwork of N which uses all labeled edges.

Lemma 12. *Let N and N' be binary networks, and $c = (x, y)$ a reducible pair in N and N'. If $N'c$ is a subnetwork of Nc, then N' is a subnetwork of N.*

Proof. First note that there are three cases: neither Nc nor $N'c$ contains x; only Nc contains x; or both Nc and $N'c$ contain x. To find an embedding ι of N' into N, we extend the embedding ι_c of $N'c$ into Nc in each of these cases. We denote the natural embeddings of $N'c$ into N' and of Nc into N by h' and h (see Fig. 9). We now split into three cases (see Fig. 8).

- **The leaf x is in neither Nc nor $N'c$.** Let p and p' be the parent of y in N and N', and g and g' the grandparent of y in N and N'. An embedding ι can be constructed from ι_c by setting $\iota(e) = h(\iota_c(h'^{-1}e))$ for all edges of N' other than $g'p'$, $p'y$, and $p'x$—h' maps each edge to a path of length one, except for the edge incident to y. Then, we map the remaining edges by setting $\iota(g'p') = h(\iota(h'^{-1}(g'))y)) \setminus \{py\}$, $\iota(p'x) = px$, and $\iota(p'y) = py$—the node $h'^{-1}(g')$ is well defined because the embedding h' is injective on the nodes, and there are exactly two more nodes in N' than in $N'c$, which cannot be the image of any node as the edge incident to x is not part of the embedding. This mapping gives an embedding, because the corresponding node mapping is still injective, no edge is in more than one image-path of an edge, and the endpoint relations are respected.
- **The leaf x is only in Nc.** This case is almost the same as the previous case, except that $\iota(p'x) = p_y p_x x$. The only edge of N that may be in the ι-image of more than one edge, is $p_x x$. However, the only edge in Nc that maps to $p_x x$ is the edge incident to x, and this edge is not in the image of ι_c. Hence, as the image of each edge other than $g'p'$, $p'y$, and $p'x$ is defined by $\iota(e) = h(\iota_c(h'^{-1}e))$, no other edge of N' is mapped to a path of N that contains $p_x x$. Furthermore, the endpoint relations are still respected, so the map ι is an embedding of N' into N.
- **The leaf x is in both Nc and $N'c$.** Let p_x and p'_x be the parent of x, p_y and p'_y the parent of y, $g_x \neq p_y$ and $g'_x \neq p'_y$ the other grandparent of x, and g_y and g'_y the grandparent of y in N and N'. An embedding ι can be constructed from ι_c by setting $\iota(e) = h(\iota_c(h'^{-1}e))$ for all edges of N' other than $g'_y p'_y$, $p'_y y$, $p'_y p'_x$, $g'_x p'_x$ and $p'_x x$—h' maps each edge to a path of length one, except for the edges incident to x and y. Then, we map the remaining edges by setting $\iota(g'_y p'_y) = h(\iota(h'^{-1}(g'_y))y)) \setminus \{py\}$, $\iota(p'_y y) = p_y y$, $\iota(p'_y p'_x) = p_y p_x$, $\iota(g'_x p'_x) = h(\iota(h'^{-1}(g'_x))x)) \setminus \{p_x x\}$, and $\iota(p'_x x) = p_x x$—the nodes $h'^{-1}(g'_x)$ and $h'^{-1}(g'_y)$ are well defined because the embedding h' is injective on the nodes, and there are exactly two more nodes in N' than in $N'c$, which cannot be the image of any node as the reticulation edge $p'_y p'_x$ is not part of the embedding. This mapping gives an embedding, because the corresponding node mapping is still injective, no edge is in more than one image-path of an edge, and the endpoint relations are respected. □

Lemma 13. *Let N and N' be binary CPNs on taxa set X and $X' \subseteq X$ respectively, and suppose that a CPS S reduces both N and N', such that all elements of S that affect N' also affect N. Then N' is a subnetwork of N (see Fig. 10).*

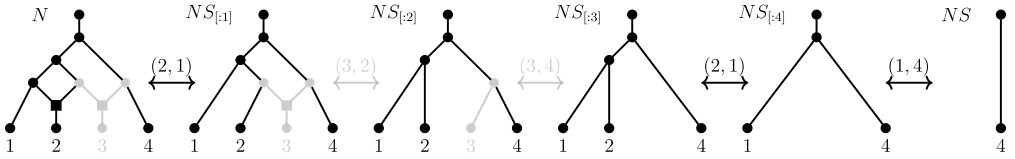

Fig. 10. A visualization of Lemma 13. The binary CPN N from Fig. 2 (gray and black), together with one of its minimal CPSs S. The subnetwork of N (black) is also reduced by S, and the embedding can be constructed by building both networks simultaneously and keeping track of the edges added by the pairs that change the subnetwork (black pairs and arrows).

Proof. Let S' denote the CPS of ordered pairs in S such that every ordered pair in S' is used in the reduction of N'. Due to this, we have that for every $i \in \{1, \ldots, |S'|\}$ there exists a $\pi(i) \in \{1, \ldots, |S|\}$ such that $S'_i = S_{\pi(i)}$. By definition of S', $\pi : \{1, \ldots, |S'|\} \to \{1, \ldots, |S|\}$ is a strictly increasing function (i.e., if $i < j$ then $\pi(i) < \pi(j)$). Recall that $S'_{[:i]}$ denotes the sequence obtained by taking the first i ordered pairs from S'. We show by induction on i, for $i = |S'|, \ldots, 0$, that the CPN $N'S'_{[:i]}$ is a subnetwork of the CPN $NS_{[:\pi(i)]}$, where $N'S'_{[:0]} = N'$, and $\pi(0) = \pi(1) - 1$.

For the base case, we prove the claim for $i = |S'|$. Let $S'_{|S'|} = (x, y)$. Then $N'S'$ is the tree with one leaf y. Since (x, y) affects $NS_{[:\pi(|S'|)-1]}$, the network $NS_{[:\pi(|S'|)]}$ must still contain y. As there is a path from the root to any leaf in a phylogenetic network, $N'S' = N'S'_{[:|S'|]}$ is a subnetwork of $NS_{[:\pi(|S'|)]}$.

So now assume $0 \leq i < |S'|$ and suppose we have proven that $N'S'_{[:j]}$ is a subnetwork of $NS_{[:\pi(j)]}$ for every $j > i$. As $i \geq 0$, there is an element $S'_{i+1} = S_{\pi(i+1)}$ in each sequence, and $N'S'_{[:i+1]}$ is a subnetwork of $NS_{[:\pi(i+1)]}$ by the induction hypothesis. By Lemma 12, $N'S'_{[:i]}$ is a subnetwork of $NS_{[:\pi(i+1)-1]}$ because $S'_{i+1} = S_{\pi(i+1)}$ acts on both networks. Applying Lemma 11 to $N'S'_{[:i]}$, $NS_{[:j]}$ and $NS_{[:j+1]}$ for all $j = \pi(i+1) - 1, \ldots, \pi(i)$, we get that $N'S'_{[:i]}$ is a subnetwork of $NS_{[:j]}$ for all $j = \pi(i), \ldots, \pi(i+1) - 1$. Hence, in particular, $N'S'_{[:i]}$ is a subnetwork of $NS_{[:\pi(i)]}$.

Therefore, $N'S'_{[:i]}$ is a subnetwork of $NS_{[:\pi(i)]}$ for all $i \in \{0, \ldots, |S'|\}$. In particular, $N' = N'S'_{[:0]}$ is a subnetwork of $N = NS_{[:0]}$. □

Note that Lemma 13 was only shown for the class of binary CPNs ($(1a, 2a)$ CPN class). This result generalizes to non-binary CPNs if we consider the notion of containment instead of subnetwork.

4.1.2. Containment results

Recall that a network N' is contained in another network N if there exists a refinement of N' that is a subnetwork of N. We first show a result that follows immediately from Lemma 13 given that the larger network is binary.

Theorem 3. *Let N be a binary CPN, and N' a non-binary CPN on X and $X' \subseteq X$, respectively. If a minimal CPS S for N also reduces N', then N' is contained in N.*

Proof. Let S' be the subsequence of S consisting of the elements that affect N', and let N'_b be the unique binary network corresponding to S'. Then, by Lemma 2, N'_b is a binary refinement of N' with minimal CPS S'. As N is the binary network corresponding to S, N'_b is a subnetwork of N by Lemma 13. Hence, N' is contained in N. □

Unfortunately for two general non-binary CPNs, an analogue of Lemma 13 is not possible to obtain. For example, the two networks of the CPN class $(1a, 2c)$ in Fig. 5 are not contained in one another, yet there is a common CPS that reduces both networks. Therefore, we need to use the more relaxed notion of containment, rather than subnetwork.

Lemma 14. *Let C be a reconstructible class of CPNs. Let N and N' be networks in C on taxa set X and $X' \subseteq X$, respectively. Then N' is contained in N if and only if there exist binary refinements N'_b of N' and N_b of N such that N'_b is a subnetwork of N_b.*

Proof. See Fig. 11 for a visualization of the proof. Suppose first that N' is contained in N. Then there exists a refinement N'_f of N' that is a subnetwork of N. There exists an embedding ι of N'_f into N.

Observe that every reticulation vertex in N'_f is mapped to a reticulation vertex in N of the same or higher degree. Let r' be a reticulation vertex in N'_f, such that $\iota(r') = r$ for some reticulation vertex r in N. Let $e'_1, \ldots, e'_a$ denote all incoming edges of r', and let $e_1, \ldots, e_b$ denote all incoming edges of r where $a \leq b$, such that the paths $\iota(e'_i)$ contains the edge e_i for all $i \in [a]$. Resolve r' as a single path of reticulation vertices $r'_1 \cdots r'_{a-1}$ such that the edge e'_1 is incident to r'_1, and the edges e'_{i+1} are incident to r'_i for all $i \in [a-1]$. Resolve r as a single path of reticulation vertices $r_1 \cdots r_{b-1}$, such that the edge e_1 is incident to r_1, and the edges e_{i+1} are incident to r_i for all $i \in [b-1]$. We now show that N'_f with r' refined in this manner is a subnetwork of N with r refined in this manner. We do this by altering the mapping ι as follows. The edges e'_i are still mapped to the same paths containing the edges e_i for $i \in [a]$. The reticulation edges $r'_i r'_{i+1}$ are mapped

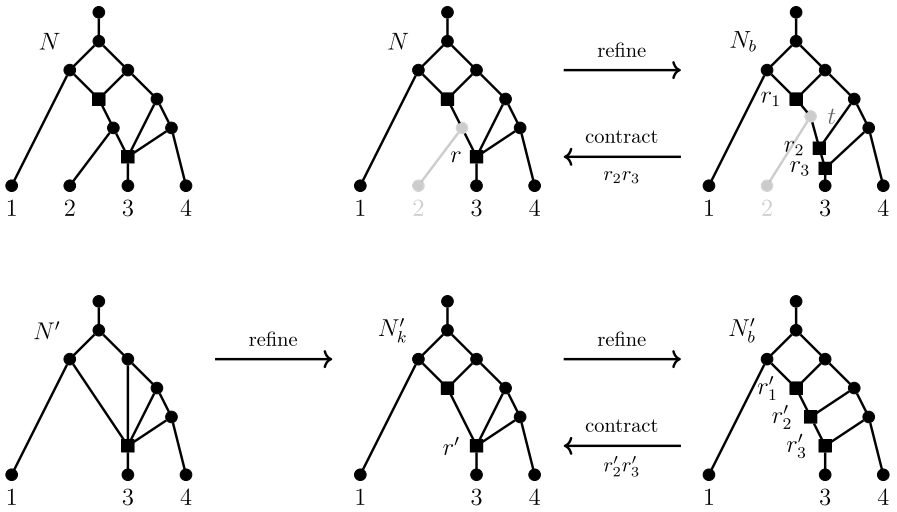

Fig. 11. Visualization of proof of Lemma 14. N and N' are CPNs of the $(1a, 2b)$ class on different sets of taxa where N' is contained in N. In proving the "only if" direction (supposing that N' is contained in N), we use the fact that N'_k, a refinement of N' is a subnetwork of N. The embedding of N'_k into N can be seen in the top middle figure, by looking only at vertices and edges in black. In this embedding, r' is mapped to r. We obtain a binary resolution N_b of N by refining r. We refine r' to obtain N'_b, a binary resolution of N, which is a subnetwork of N_b. For the other direction (supposing that a binary resolution N'_b of N' is a subnetwork of a binary resolution N_b of N), notice that an embedding of N'_b into N_b maps the edges $r'_1r'_2$ and $r'_2r'_3$ into r_1r_2 and r_2r_3, respectively. Observe that r_1r_2 is an rtr-path (with the tree vertex t) and r_2r_3 is an rr-path; because of this, we contract the edge $r'_2r'_3$ but not $r'_1r'_2$ to obtain the refinement N'_k. Upon contracting all rr-edges of N_b, which is only the edge r_2r_3 in this case, we obtain N, and we can see that N'_k is a subnetwork of N.

to r_ir_{i+1} for $i \in [a-2]$. Let c denote the child of r' in N'_f. The edge $r'_{a-1}c$ is mapped to the path from r_{a-1} to $\iota(c)$. All other mappings remain unchanged.

A similar observation can be made for tree vertices. Now, observe that binary refinements of these CPNs are obtained by either resolving all multifurcations, all multi-reticulations, or both (depending on C). Then we may apply the above to all multifurcations/multi-reticulations as needed to obtain binary refinements N'_b of N' and N_b of N such that N'_b is a subnetwork of N_b.

Now suppose that there exist binary refinements N'_b of N' and N_b of N such that N'_b is a subnetwork of N_b. We note that N can be obtained from N_b by contracting all tt-edges, all rr-edges, or both depending on if N is a CPN of class $(1a, 2b)$, $(1b, 2c)$, or $(1b, 2d)$. Now, if $N_b = N$, then we are done. So suppose that N is not a binary network.

The plan is to contract the edges of N_b to obtain N. In the process, we choose to either contract or not contract 'corresponding' edges in N'_b, ensuring at each step that the resulting network is a subnetwork of some refinement of the main network. We start by looking at contracting the rr-edges of N_b. We claim that contracting all rr-edges in N'_b that do not map to an rtr-path gives a refinement of N' that is a subnetwork of N.

Let e be an edge in N'_b. If e is an rr-edge, then it is mapped to some path P_e in N_b. Since reticulation vertices are mapped to reticulation vertices in the embedding, the path P_e is an rr-path or an rtr-path. If P_e is an rr-path, then contract all edges of P_e in N_b. Contract e in N'_b. It is easy to see that by mapping the reticulation vertex obtained by contracting e to the reticulation vertex obtained by contracting P_e, we may extend the embedding of N'_b into N_b in a natural manner, thereby showing that the newly obtained network is a subnetwork of the other. Now suppose that P_e was an rtr-path. Then we contract all rr-edges contained in P_e. Observe that N'_b is still embedded in this newly obtained graph, since we may extend the original embedding by simply changing the mapping of the edge e to the newly contracted rtr-path. On the other hand, suppose that e was not an rr-edge. If it is mapped to a path containing rr-edges in N_b, then we may simply contract the rr-edges of the path, and update the embedding by changing the mapping of the edge to the newly contracted path. Finally suppose that there exists an rr-edge in N_b that is not used in the embedding. Then contracting such an edge has no effect on the embedding.

Similarly, contracting all tt-edges in N'_b that do not map to a path containing a trt-path gives a refinement of N' that is a subnetwork of N.

We now obtain a sequence of networks $N'_b = N'_0, N'_1, \ldots, N'_k$ and $N_b = N_0, N_1, \ldots, N_k = N$ such that N'_i is a subnetwork of N_i for all $i \in [k]$, and N'_i is obtained from N'_{i-1} by contracting an rr-edge that does not map to an rtr-path in N_{i-1} (or by contracting a tt-edge that does not map to a trt-path in N_{i-1}, depending on the CPN class) for $i \in [k]$. The networks N_i are obtained by contracting the edges of the rr-path or the rtr-path (or the tt-path or the trt-path) as outlined in the previous paragraph. Note also that N'_i is a refinement of N'. Then we have that N'_k, which is a refinement of N', is a subnetwork of N. Therefore N' is contained in N. □

Finally, we present a lemma analogous to Lemma 13 for two CPNs within the same reconstructible class.

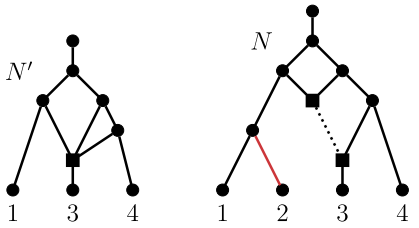

Fig. 12. An illustration of Lemma 15. The tree-child network N' is contained in the tree-child network N, which can be seen by deleting leaf 2 and contracting the dotted edge in N. The TCS $(2,1)(3,4)(3,1)(3,4)(1,4)$ for N reduces N' as well. However, N' is clearly not a subnetwork of N.

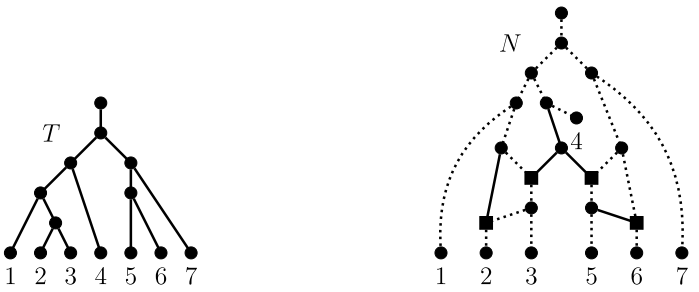

Fig. 13. The tree T is a subnetwork of a CPN N, and one embedding of T into N is shown (dotted edges). There exists no minimal CPS of N that reduces T to a single leaf.

Lemma 15. *Let $\mathcal{C}$ be a reconstructible class of CPNs. Let N and N' be CPNs in $\mathcal{C}$ on taxa set X and $X' \subseteq X$ respectively, and suppose that a minimal CPS S for N also reduces N'. Then N' is contained in N.*

Proof. Let S' be the subsequence of S consisting of the elements that affect N', and let N_b and N'_b be the unique binary CPNs obtained from S and S' using the $(1a, 2a)$ construction. Then, by Lemma 2, N_b is a binary refinement of N with minimal CPS S, and N'_b is a binary refinement of N' with minimal CPS S'. By Lemma 13, N'_b is a subnetwork of N_b. By Lemma 14, N' is contained in N. □

Note that Lemma 15 does not hold if we used subnetwork instead of containment (see Fig. 12). The converse of Lemmas 13 and 15 do not hold for general CPNs, and we show this in the following subsection.

4.2. Containment does not always imply reduction

Following subsection 4.1, we give an example of a CPN N and a tree T on the same leaf-sets, for which T is a subnetwork of N, such that there exists no minimal CPS for N that reduces T to a single leaf. This example is shown in Fig. 13.

Theorem 4. *There exists a binary CPN N that contains a tree T, such that no minimal CPS for N reduces T.*

Proof. We refer by N and T to the network and the tree of Fig. 13.

Note first that $\mathcal{C}(N) = \{(2,3),(6,5)\}$. So initially, we are required to pick one of the reticulated cherries $(2,3)$ or $(6,5)$.

Picking $(2,3)$ first reduces both T and N, and in the next step we have the option of picking one of the reticulated cherries $(3,2)$ or $(6,5)$. Picking $(3,2)$ does not affect $T(2,3)$, and reduces the reticulated cherry $(3,2)$ in $N(2,3)$. In $N(2,3)(3,2)$, 3 is no longer a child of a reticulation. Let u denote the LCA of 1 and 3. The path from u to 3 contains the parent of 4. This implies that one of 3 or 4 is picked by the time we can pick leaf 1, which ultimately means that T is not reduced to a leaf with any CPS starting with $(2,3)(3,2)$. So we pick $(6,5)$.

Picking $(6,5)$ first reduces both T and N, and in the next step we have the option of picking one of the reticulated cherries $(2,3)$ or $(5,6)$. Picking $(5,6)$ does not affect $T(6,5)$, and reduces the reticulated cherry $(5,6)$ in $N(6,5)$. Let u denote the LCA of 5 and 7 in $N(6,5)(5,6)$. Since u is the child of the root, the pair $(5,7)$ can only appear in the CPS as the final cherry. But $(5,7)$ is a cherry in the tree $T(6,5)(5,6)$, which still contains other leaves. This implies that a minimal CPS for N starting with $(6,5)(5,6)$ cannot reduce T to a single leaf.

Thus, the CPS must start with either $(2,3)(6,5)$ or $(6,5)(2,3)$. Note that $T(2,3)(6,5) = T(6,5)(2,3)$ and $N(2,3)(6,5) = N(6,5)(2,3)$ and so the order in which we pick the two reticulated cherries do not matter. In both cases, we have the choice of picking one of the reticulated cherries $(3,2)$ or $(5,6)$. However, doing so results in a CPS that does not reduce T to a single leaf, by the argument above. Since every CPS of N starts with these cherries, we have that T is not reduced to a single leaf for any CPS of N. □

Since this particular network is semi-binary stack-free, it serves as an example for when containment does not imply reduction for the $(1a, 2a)$ and the $(1a, 2b)$ classes. For the other two reconstructible classes, the claim may still hold true. We speculate that similar results follow, which we discuss in Section 7.

In what follows, we show that if we restrict our scope to the class of tree-child networks, then the converse does hold. That is, for those networks, containment implies reduction.

5. Tree-child sequences

Recall that a condition was imposed on a sequence of ordered pairs to define CPSs as those that can be used to construct networks. We show here that imposing an additional condition on the CPSs can ensure the constructed network to be tree-child. Recall that a network is tree-child if every non-leaf vertex has a child that is a tree vertex or a leaf.

In this section, we assume that we work in CPN classes that use the $2b$ or the $2d$ reticulated cherry construction, to ensure that the network constructed from the sequences is stack-free. Outside of stacks, the only forbidden structure of tree-child networks are tree vertices whose children are all reticulations. To ensure these structures do not appear in the constructed networks, we impose a condition on our sequences, that the first coordinate of each pair does not appear as a second coordinate of another pair in the remainder of the sequence. We will refer to this condition as the *tree-child condition*.

Definition 12. A CPS is a *tree-child sequence (TCS)* if every leaf appearing as the first coordinate does not appear as a second coordinate in the rest of the sequence.

Lemma 16. *Let S be a TCS. Then each of the networks obtained by choosing construction $2b$ or $2d$ for each reticulated cherry is tree-child.*

Proof. By construction, the networks obtained from S are CPNs and they are stack-free. It remains to show that each tree vertex has at least one child that is a tree vertex or a leaf.

Let $S_i = (x, y)$ be a reticulated cherry, and consider the network N after having just added S_i. In N, the tree vertex parent p_y of y currently has at least one leaf child y. Suppose that all its other children were reticulations. For this vertex p_y to have all reticulation children in the fully constructed network, we require some reticulation vertex to be inserted between p_y and y, which can only happen if we add some ordered pair $S_j = (y, z)$ where $j < i$. However, this would mean that y appears as a first coordinate of some pair S_j and also as a second coordinate of some pair S_i later on in the sequence, which contradicts our tree-child condition.

Now, if $S_i = (x, y)$ was a cherry, then the parent p of x cannot be a parent of only reticulations after adding more pairs to the network. Indeed, this would imply that we have added some reducible pair (y, z) later on to the network—and hence it would appear earlier in the sequence—which again contradicts our tree-child condition.

Hence all tree vertices of a network obtained from a TCS have at least one child that is a tree vertex or a leaf. Therefore such a network is tree-child. □

Tree-child networks (TCNs) always contain a reducible pair, and after reducing one of these, we obtain a new tree-child network (Lemma 4.1 of [2]). Naturally, this implies that TCNs are CPNs. As we have seen in the previous section, given a CPN that contains a tree on the same set of taxa, there may not exist a minimal CPS of the network that reduces the tree (Theorem 4). In this section, we make the switch from CPSs to TCSs, and show that this is no longer an issue for TCSs. In fact, we prove a stronger result: within a reconstructible CPN class, a TCN contains another TCN on the same taxa if and only if every minimal TCS of the first TCN reduces the second TCN. If, in addition, the first TCN is binary, then if *any* minimal TCS for the larger network reduces the smaller network, then the smaller network is a subnetwork of the larger network. We also show that—similar to Proposition 1 for CPNs—we may pick reducible pairs in any order for TCNs. We start by showing that every TCN has a minimal TCS. This was shown implicitly for semi-binary TCNs in the proof of Lemma 3.4 of [19]; however we include it here for completeness and to generalize for non-binary TCNs.

Lemma 17. *Let N be a non-binary TCN. Then there is a TCS for N.*

Proof. Let N be a network on X. For a partial TCS S, denote by $F(S) \subseteq X$ the set of first elements of pairs of S. Because N is tree-child, for each node v, there is a path to a leaf $t(v)$ in which all internal nodes are tree nodes, and $t(v) = t(c)$ for some child c of v if v is not a leaf node.

Assume S is a partial TCS such that $t(v)$ is still below v via a tree-path for all nodes v of NS, and for each $l \in F(S)$, the nodes v for which $t(v) = l$, are l, and possibly the parent of l if it is a reticulation. Suppose N is not fully reduced by S, we show that we can find a longer sequence S' starting with S to which the conditions also apply.

Let p be a lowest tree node in NS, then each node below p is either a leaf, or a reticulation which is directly above a leaf. The leaf $t(p)$ is directly below p (as there is a tree-path from p to $t(p)$) and $t(p) \notin F(S)$ by the assumption on S. By reducing all cherries and reticulated cherries involving p using $t(p)$ as second element, we obtain a new partial tree-child

sequence S'. In essence, this reduction deletes all outgoing edges of p except for $pt(p)$, suppresses all degree-2 vertices (which includes p), and deletes any isolated vertices. We claim that for all nodes v of NS', $t(v)$ is still below v by a tree-path. Indeed, deleting all but one outgoing edge of p only affects the tree-paths that were leading to p. So for all vertices in NS that were not ancestors of p, the claim holds. On the other hand, for all ancestors g of p in NS with $t(g) = t(p)$, there is still a tree-path from g to $t(g)$ in NS'. So our claim holds. Furthermore, each leaf $l \in F(S')$ is either directly below a reticulation, or directly below a tree node v with $t(v) \neq l$, as in NS, l was below a reticulation below p; or $l \in F(S)$ and l was already below v, so $t(v) \neq l$ in NS by assumption.

Starting with the empty partial TCS, we can repeat this process until NS has no more tree nodes. When this is the case, we have found a TCS S for N. □

5.1. Subnetwork/containment implies reduction

As in the previous sections, we will try to keep the results as general as possible. We first show results on reduction and subnetworks.

Lemma 18. *Let N be a non-binary tree-child network, N' a tree-child subnetwork of N with the same leaf set, and S a TCS. Then $N'S$ is a subnetwork of NS.*

Proof. We prove this fact inductively on the length of S. If S is empty, then $N'S = N'$ and $NS = N$, so $N'S$ is a subnetwork of NS.

Now suppose that for any TCS S' of length at most j, $N'S'$ is a subnetwork of NS'. We prove that for any TCS S of length $j+1$, $N'S$ is a subnetwork of NS. Let us denote $S = S'(x, y)$. Note that S' is of length at most j. Hence, by the induction hypothesis, $N'S'$ is a subnetwork of NS'. We consider the following cases:

- ***$N'S'$ has only one of x and y.*** Because $S = S'(x, y)$ is a TCS, y is not the first coordinate in any element of S'. Hence, $N'S'$ must still contain y, and x must have been deleted from N' by applying S'. This means the edge of NS' deleted by applying (x, y) is not used by the embedding of $N'S'$ into NS', and $N'S = N'S'$ can still be embedded in NS.
- ***$N'S'$ has both x and y.*** There are a few cases we must consider, depending on whether there are reducible pairs (x, y) in $N'S'$ and NS'.
  - ***NS' has a cherry (x, y).*** As $N'S'$ also contains both leaves x and y, $N'S'$ also has the cherry (x, y), which is mapped to the corresponding cherry in NS' by the embedding. The reduction of (x, y) in both networks removes the pendant edge leading to x in both networks, not changing the embedding otherwise.
  - ***NS' has a reticulated cherry (x, y).***
    - **The edge $p_y p_x$ is used by the embedding of $N'S'$ into NS'.** First note that $N'S'$ must have either a cherry or a reticulated cherry (x, y): if the edge $p_y p_x$ is used by the embedding, then the only way to reach x and y in NS', is by using the edges $p_x x$ and $p_y y$, making (x, y) either a cherry or a reticulated cherry in $N'S'$. Now applying (x, y) to $N'S'$ deletes the edge using $p_y p_x$ of NS' in the embedding. Hence, upon deleting both these edges by applying (x, y) to both networks, we may naturally extend the embeddings.
    - **Otherwise.** $N'S'$ can be embedded in NS' without the edge $p_y p_x$. Hence, $N'S'$ is a subnetwork of NS' after the removal of this edge $p_y p_x$ (i.e., the network NS). As $N'S$ is a subnetwork of $N'S'$ and $N'S'$ is a subnetwork of NS, $N'S$ is a subnetwork of NS.
  - **Otherwise.** The network NS' contains neither a cherry, nor a reticulated cherry on x and y. This means $NS = NS'$. As $N'S$ is a subnetwork of $N'S'$, and $N'S'$ is a subnetwork of NS', $N'S$ is a subnetwork of $NS (= NS')$. □

The following corollary follows immediately, as subnetworks of single-leaf networks are single-leaf networks.

Corollary 3. *Let N and N' be non-binary tree-child networks on the same leaf set, with N' a subnetwork of N. If a TCS S reduces N, then S also reduces N'.*

Observe that in the setting of Corollary 3, the two networks are reduced to the same single-leaf network, with the same leaf label.

Theorem 5. *Let C be a reconstructible class of CPNs. Let N and N' be tree-child networks in C on the same leaf set. Then N' is contained in N if and only if* any *TCS of N reduces N'.*

Proof. Follows immediately from Corollary 3, Lemma 13, and Lemma 15. □

Note that in Theorem 5, it is necessary for the two networks to be contained in the same reconstructible class of CPNs. Consider the networks in the (1a, 2d) and the (1b, 2d) classes in Fig. 4, which are both tree-child and can be reduced by the same tree-child sequence. The latter network is not contained in the former.

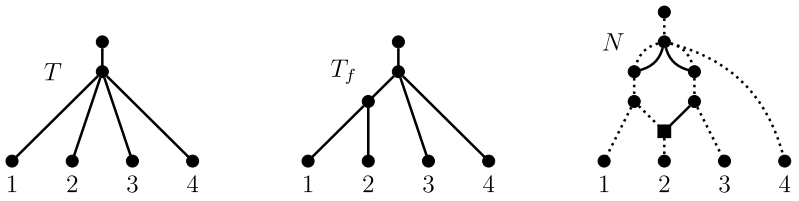

Fig. 14. The tree T is contained in the tree-child network N using the refinement T_f, but it is not a subnetwork of N The embedding of T_f into N is shown by dotted edges. In particular, T and N belong to the $(1b, 2c)$ and the $(1b, 2d)$ classes.

For semi-binary TCNs, we show that the notions of containment and subnetwork are equivalent.

Lemma 19. *Let N, N' be both semi-binary TCNs on the same leaf-set. Then N contains N' if and only if N' is a subnetwork of N.*

Proof. One direction is clear by definition of containment and subnetwork, so suppose that N' is contained in N. Then there exists some refinement N'_f of N' that is a subnetwork of N (i.e., there exists an embedding of N'_f into N). If $N'_f = N'$, then we are done, so we may assume that there exists an rr-edge e of N'_f that is mapped to an rtr-path P_e of N in the embedding. This implies that the tree-vertices on this path P_e are not used in the embedding; in particular, this means that any outgoing edge of a tree-vertex that is not in P_e is not used in the embedding. Let t denote the lowest tree-vertex on P_e. The child of t on P_e is a reticulation. By the tree-child property, t must have another child that is either a tree vertex or a leaf. Such a vertex is not on P_e, and in particular, this implies that there exists a tt-path from t to some leaf l, such that the path is not on P_e. So this tt-path is not used in the embedding. But every path from the root to the leaf l uses this tt-path. It follows that l does not appear as a leaf in the network N'_f and therefore in N'. This is a contradiction as N and N' have the same leaf-sets. Thus reticulations in N' need not be refined to obtain N'_f. However, since all tree vertices in N' are binary, all refinements of N' arise from refining its reticulation vertices. Then $N'_f = N'$, and therefore N' is a subnetwork of N. □

This lemma does not hold in general, for all other network classes for which the smaller network is not binary (see Fig. 14). Furthermore the lemma does not hold for when the larger network N is not a TCN.

Theorem 6. *Let N and N' be semi-binary TCNs on the same leaf set. Then N' is a subnetwork of N if and only if any TCS of N reduces N'.*

Proof. Follows immediately from Theorem 5 and Lemma 19 □

Note that we could assume N' is non-binary, but to be a subnetwork of a semi-binary network, it has to be semi-binary as well, because N and N' are both TCNs on the same leaf set. Note that Theorem 6 is no longer true if we allow for the networks to have different leaf-sets. Let N' be the cherry on leaf-set $\{1, 2\}$, and let N be the balanced tree on leaf-set $\{1, 2, 3, 4\}$ with cherries $(1, 3)$ and $(2, 4)$. Then both networks are semi-binary tree-child, and N' is a subnetwork of N. However, the TCS $(1, 3)(2, 4)(3, 4)$ of N does not reduce N'.

5.2. Order does not matter in TCSs

Theorem 1 states that we may pick cherries from a CPN in any order and still obtain a minimal CPS. In this section, we show that this also holds for TCSs.

Lemma 20. *Let N be a non-binary TCN, S a partial TCS and c an ordered pair of leaves such that cS is also a partial TCS. Then NcS is a subnetwork of NS.*

Proof. We consider the networks $NcS_{[:j]}$ and $NS_{[:j]}$ for $j \geq 0$. We prove, using induction on j, that $NcS_{[:j]}$ is a subnetwork of $NS_{[:j]}$. Obviously, this is true for $j = 0$, as Nc is a subnetwork of N.

Write $S_{j+1} = (x, y)$ for the $j+1$-th element in the sequence. First note that if x is not a leaf of $NcS_{[:j]}$, then $NcS_{[:j]}$ is still a subnetwork of $NS_{[:j]}(x, y)$, as the embedding of $NcS_{[:j]}$ in $NS_{[:j]}$ does not use the removed edge that leads only to x. Therefore $NcS_{[:j+1]} = NcS_{[:j]}$ is a subnetwork of $NS_{[:j+1]} = NS_{[:j]}(x, y)$.

Now assume x is a leaf of $NcS_{[:j]}$. If reducing $NS_{[:j]}$ with $S_{j+1} = (x, y)$ does not remove an edge e used by the embedding of $NcS_{[:j]}$, then $NcS_{[:j+1]}$ is still a subnetwork of $NS_{[:j+1]}$.

So, we now assume reducing $S_{j+1} = (x, y)$ in $NS_{[:j]}$ removes an edge e used by the embedding of $NcS_{[:j]}$. Suppose for a contradiction that $NcS_{[:j+1]}$ is not a subnetwork of $NS_{[:j+1]}$. For this to be true, the edge of $NcS_{[:j]}$ mapped to the

path containing e must not be removed by reducing S_{j+1}. This can only happen if $NcS_{[:j]}$ does not contain the reducible pair (x, y), whereas $NS_{[:j]}$ does have it. The reducible pair (x, y) in $NS_{[:j]}$ must be a reticulated cherry, as the only other option is that it is a cherry, but then, $NcS_{[:j]}$ also contains this cherry, as x and y are both part of $NcS_{[:j]}$ and $NcS_{[:j]}$ is displayed by $NS_{[:j]}$. Hence, $NcS_{[:j]}$ does not contain a cherry nor a reticulated cherry (x, y), and $NS_{[:j]}$ contains a reticulated cherry (x, y). As $NcS_{[:j]}$ is a subnetwork of $NS_{[:j]}$ whose embedding uses the reticulation edge of the reticulated cherry (x, y), the leaf y must have been deleted from $NcS_{[:j]}$ already. This means y must be the first element of a pair in the partial TCS $cS_{[:j]}$. However, cS is a partial TCS with y as the second coordinate of S_j, so we have a contradiction. We conclude that if the reduction removes an edge from $NS_{[:j]}$, it also removes the corresponding edge (i.e., the edge of $NcS_{[:j]}$ that was mapped to the path of $NS_{[:j]}$ containing the edge) from $NcS_{[:j]}$.

We conclude that $NcS_{[:j+1]}$ is a subnetwork of $NS_{[:j+1]}$. □

Lemma 21. *Let S and S' be TCSs such that S' is a subsequence of S. Then there exists a sequence of TCSs $S = \Sigma^0, \ldots, \Sigma^{|S|-|S'|} = S'$ such that $|\Sigma^i| = |\Sigma^{i-1}| - 1$ for $i \in [|S| - |S'|]$.*

Proof. Let S'' denote the sequence (not necessarily a CPS) obtained by taking the elements of S which do not occur in S' (in order). For $i \in [|S''|]$, let $f(i)$ denote the position where $\Sigma^{i-1}_{f(i)} = S''_i$, which is the element in the sequence that will be deleted to obtain Σ^{i+1}. We claim that if Σ^i is a TCS, then we can obtain a TCS Σ^{i+1} by removing the element $S''_i = \Sigma^{i-1}_{f(i)}$. Upon repeating this for all i, we obtain the sequence of TCSs that we are after.

To show this, suppose for a contradiction that Σ^i is a TCS, but removing S''_i from Σ^i results in a sequence Σ^{i+1} that is not a TCS. Note first that the 'tree-child property' of the sequence is retained when deleting elements from TCSs: indeed, every leaf appearing as a first coordinate still does not appear as a second coordinate in the rest of the sequence. So we must have that Σ^{i+1} is not a CPS, that is, there exists a leaf that appears as a second coordinate, but not as a first coordinate in the remaining sequence. Then the CPS property is violated for an element which occurs in $\Sigma^i_{[:f(i)-1]}$. Note that $\Sigma^i_{[:f(i)-1]}$ forms the first $f(i) - 1$ elements of S', since we defined S'' as the elements of S which do not occur in S' in order. Furthermore, note that at each step we do not add elements to the sequence. Then $\Sigma^{|S|-|S'|} = S'$ cannot be a CPS, let alone a TCS, which contradicts our assumption that S' was a TCS. Therefore Σ^{i+1} must be a TCS and the lemma follows. □

Corollary 4. *Let N be a non-binary TCN with S and S' TCSs such that S' is a subsequence of S. Then NS is a subnetwork of NS'.*

Proof. By Lemma 21, we only have to prove this for S' of length one less than S, so suppose $S = S'_{[:i]}sS'_{[i+1:]}$, where the first or the last part may be empty.

We consider $NS = NS'_{[:i]}sS'_{[i+1:]}$, by applying the three parts of the sequence separately. First we note that by writing $N' := NS'_{[:i]}$, we have $NS = N'sS'_{[i+1:]}$ and $NS' = N'S'_{[i+1:]}$. Because S is a TCS, the sequence $sS'_{[i+1:]}$ is in particular also a TCS. Hence, by Lemma 20, NS is a subnetwork of NS'. □

The following proposition says that we can find a TCS for a TCN by picking an arbitrary reducible pair, reducing it, and repeating this process.

Proposition 2. *Let N be a semi-binary TCN and S a partial TCS with every element affecting N. Then there exists a minimal TCS for N starting with S.*

Proof. We prove this using induction on the length l of S. If l is 0, then, as each TCN has a minimal TCS, there is a TCS $S' = SS'$ for N, which starts with S. Now suppose for any partial TCS S of length $l < L$ affecting N in every step, there is a minimal TCS SS' of N. We prove that the same holds for any such sequence of length L.

Let $S(x, y)$ be such a sequence of length L (where (x, y) is the last element of this sequence). Because each element of $S(x, y)$ affects N, we know in particular that $(x, y) \in \mathcal{C}(NS)$. By the induction hypothesis, there is a TCS SS' for N starting with S. The part S' of this sequence must contain an element (x, y) or (y, x) as N is semi-binary (Lemma 6). Let S'_i be the first occurrence of such an element.

Each of the intermediate networks $NSS'_{[:j]}$ for $j < i$ has the reducible pair (x, y). This means the only pairs involving x that affect these networks have x as first coordinate, or are equal to (y, x). As S'_i is the first occurrence of (x, y) or (y, x) in S', all S'_j with $j < i$ cannot have x as second coordinate. This means that $S(x, y)S'$ is a TCS, and it reduces N by Corollary 4.

Note that each element of $S(x, y)$ affects N. Since we know SS' is a minimal TCS for N, and because $S(x, y)S'$ contains one extra element, there is an element of S' that reduces nothing in N in the sequence $S(x, y)S'$. Removing this element gives a minimal TCS for N starting with $S(x, y)$. □

Proposition 3. *Let N be a non-binary TCN and S a partial TCS that affects N. Then there exists a minimal TCS for N starting with S.*

Proof. Let N' denote the refinement of N obtained by applying the $(1a, 2b)$ construction on NS for the sequence S. Observe that N' is a TCN since NS is tree-child. Because of this, every multifurcation of N' has a child that is a leaf or a tree vertex,

which we call t. By refining each multifurcation as a path in which the lowest vertex is the parent of t, we obtain a semi-binary refinement N'_{sb} of N' that is tree-child. In particular, the tree-vertices of the reducible pairs in N'_{sb} that are reduced by S have outdegree-2, because of the construction we have used to obtain N'. The way in which we have obtained N'_{sb} ensures that S is a partial TCS that affects N'_{sb}. By Proposition 2, there exists a minimal TCS S' for N'_{sb} starting with S. Since N'_{sb} is a refinement of N, it follows then that S' must also reduce N. □

6. Computational aspects of containment problems

Tree Containment is a well studied problem, where one asks whether a tree is contained in a given network (with a common set of taxa). In this section, we look at the more general problem Network Containment, where the aim is to determine whether a network is contained in another network. We restrict our attention to the problem where the input networks are both semi-binary, tree-child networks with the same leaf set. We will give an algorithm for this problem that runs in linear time. We also show that it is possible to check in linear time whether two CPNs are isomorphic.

Network Containment
Instance: Two networks N and N' on the same leaf-set.
Question: Does N contain N'?

6.1. Tree-child network containment

In this section, we give the linear time algorithm (Algorithm 6) for Network Containment. We use a few small subroutines (Algorithms 1, 2, 3, 4, 5). The idea of the algorithm follows from Theorem 5 and Proposition 2. For two TCNs N and N', find a minimal TCS of N by picking reducible pairs in any order. See if this TCS reduces N' to a network on a single leaf: if it does, then N contains N'; otherwise, N does not contain N'. Assume in this subsection that every network is semi-binary stack-free, unless stated otherwise.

Within semi-binary networks, tree vertices are of outdegree-2. This means that each leaf appears as a second coordinate in at most one reducible pair in a network. Algorithm 1 finds such a reducible pair, if it exists, for a given leaf, in constant time.

Algorithm 1: FindRP2nd(N, x).

```
Data: A semi-binary network N and a leaf x
Result: The singleton set containing the reducible pair of N having x as a second coordinate if it exists, ∅ otherwise
Let p be the parent of x;
if p is a tree node then
    let c(p) be the child of p that is not x;
    if c(p) is a leaf then
        return {(c(p), x)};
    if c(p) is a reticulation and the child c(c(p)) of c(p) is a leaf then
        return {(c(c(p)), x)};
    end
end
return ∅;
```

Lemma 22. *Let N be semi-binary, and x a leaf of N. If a reducible pair with x as the second element of the pair exists, then Algorithm 1 finds this pair. Otherwise, it returns the empty set. The algorithm runs in constant time.*

Proof. Each leaf is a second coordinate of at most one reducible pair, and this pair is found by taking the unique path down from the parent of this leaf. This algorithm runs in constant time: using in- and out-adjacency lists, we can check in constant time whether a node is a tree node, a reticulation node, or a leaf, and the out-list of each node has size at most 2 (as tree nodes have outdegree 2, reticulations outdegree 1, and leaves outdegree 0). □

Algorithm 2 on the other hand finds all reticulated cherries that contain a given leaf as the first coordinate of the reducible pair. The running time for this algorithm depends on the indegree of the parent of the given leaf, as this gives the maximum possible number of such reticulated cherries.

Lemma 23. *Let x be a leaf in a semi-binary network N and let p_x denote the parent of x. Let I denote the indegree of p_x. Algorithm 2 finds the set of all reticulated cherries with the reticulation on the leaf x in $O(I)$ time.*

Proof. We simply check that the parent of x is a reticulation, and that (x, y) is a reticulated cherry if a grandparent of x is the parent of y. The for loop iterates at most I times. The steps within the for loop run in constant time, since we may use the in- and out-adjacency lists as stated in the proof of Lemma 22. Therefore, Algorithm 2 runs in $O(I)$ time. □

Algorithm 2: FindRC1st(N, x).

Data: A semi-binary network N and a leaf x
Result: The set $\mathcal{C}_r = \{(l, k) \in \mathcal{C}(N) : l = x\}$ of reticulated cherries in N having x as first coordinate

Let p be the parent of x;
Set $\mathcal{C}_r = \emptyset$;
if *p is a reticulation* **then**
  for *every parent g of p* **do**
    let $c(g)$ be the other child of g;
    if *$c(g)$ is a leaf* **then**
      $\mathcal{C}_r = \mathcal{C}_r \cup \{(x, c(g))\}$
  end
end
return $\mathcal{C}_r$;

Algorithm 3: ReducePair$(N, (x, y))$.

Data: A non-binary network N and a pair (x, y)
Result: The network $N(x, y)$

if *(x, y) is a cherry in N* **then**
  Let p be the parent of x and y;
  Remove edge px from N;
  Suppress p (if necessary) and remove x in N;
if *(x, y) is a reticulated cherry in N* **then**
  Let p_x be the parent of x and p_y the parent of y;
  Remove edge $p_y p_x$ from N;
  Suppress p_y and p_x (if necessary) in N;
end
return N;

Lemma 24. *Algorithm 3 reduces a given reducible pair in a non-binary network N. The algorithm runs in constant time.*

Proof. The algorithm takes exactly the steps which define a reduction of a pair in a network, so the algorithm is correct. Then for the running time: comparing the unique parents of the leaves, we can check whether a pair constitutes a cherry or a reticulated cherry in constant time. Indeed, if a pair (x, y) forms a cherry, then the parents of x and y are the same; and if (x, y) forms a reticulated cherry, then the unique other child of the parent of y is the parent of x. The removal of the edge with subsequent suppression takes at most constant time. Hence, the subroutine runs in constant time. □

Algorithm 4: FindTCS(N).

Data: A semi-binary TCN N
Result: A minimal TCS S for N

Set $\mathcal{C} = \emptyset$;
for $x \in L(N)$ **do**
  $\mathcal{C} \cup$FindRP2nd(N, x);
end
Let S be an empty sequence;
while $\mathcal{C} \neq \emptyset$ **do**
  Choose $(x, y) \in \mathcal{C}$;
  Set $S = S(x, y)$;
  $N' =$ ReducePair$(N, (x, y))$;
  if *(x, y) is a cherry in N* **then**
    $\mathcal{C} = \mathcal{C} \setminus \{(x, y), (y, x)\} \cup$FindRP2nd$(N', y) \cup$FindRC1st$(N', y)$;
  if *(x, y) is a reticulated cherry in N* **then**
    $\mathcal{C} = \mathcal{C} \setminus \{(x, y)\} \cup$FindRP2nd$(N', y) \cup$FindRC1st$(N', y)$;
  end
  $N = N'$;
end
return S;

Recall that TCSs have the additional rule, on top of that of the CPSs, that any leaf appearing as a first coordinate of a pair in the sequence must not appear as a second coordinate of a pair later on in the sequence. Therefore we may have a cherry (x, y) in the network that can only be picked as (x, y) and not (y, x). We define a notion of when a reducible pair is *forbidden* before proving the correctness of Algorithm 4.

Definition 13. Given a partial TCS S, a reducible pair (x, y) is *forbidden* after S if y has appeared as the first coordinate of a pair in S.

Lemma 25. *Let N be a semi-binary TCN on taxa set X and reticulation number r. Algorithm 4 finds a minimal TCS for N in $O(|X|+r)$ time.*[2]

Proof. The first for loop finds all reducible pairs of N, as they all have a unique second coordinate, and each leaf is a second coordinate in at most one pair in a semi-binary network. Now by Proposition 2, each choice of non-forbidden pair gives a partial TCS for N. Hence, the algorithm may choose any non-forbidden pair and reduce it, so suppose that we pick (x, y). Regardless of whether (x, y) is a cherry or a reticulated cherry, the reduction takes constant time as ReducePair$(N, (x, y))$ runs in constant time. If (x, y) is a cherry, then we remove the pairs $(x, y), (y, x)$ from our list of pairs $\mathcal{C}$. All other pairs of $\mathcal{C}$ are still reducible in $N(x, y)$ by Lemma 6, and they are not forbidden because x is not a leaf of $N(x, y)$, so clearly there is no reducible pair with x as second coordinate. Noting that x is no longer a leaf in $N(x, y)$, the only new cherries in $N(x, y)$ must involve y. Since y is not a forbidden leaf, any cherry pair involving y is not forbidden: therefore we update our list $\mathcal{C}$ by appending both FindRP2nd(N, y) and FindRC1st(N, y).

On the other hand if (x, y) is a reticulated cherry, then the new cherry pairs in $N(x, y)$ must involve x or y, so by removing the pair (x, y) from the current list $\mathcal{C}$ and then adding the new pairs involving x and y, we get the updated list of pairs $\mathcal{C}$ for the TCN $N(x, y)$; we ensure that this updated list contains only non-forbidden pairs.

Observe that all reducible pairs of $N(x, y)$ that involve x are already contained in the set $\mathcal{C}$. Indeed, if x is involved in a cherry or a reticulated cherry (x, z), then (x, z) must have formed a reticulated cherry in N, which would have been found in the previous iteration of the while loop or in the initial search for all reducible pairs with second coordinate z. Note that x cannot be involved in a reticulated cherry where its parent p_x is a tree node, as this would imply that p_x was the parent of two reticulations in N, contradicting the tree-child property of N. Furthermore, since x has just been picked as a first element of some reducible pair, all pairs involving x as a second coordinate are now forbidden. Therefore, all possible new cherry pairs not in $\mathcal{C}$ that appear as a result of reducing (x, y) from N must involve the leaf y. These pairs are made up of those pairs contained in FindRP2nd(N, y) or in FindRC1st(N, y). The only potential problem that we may face is when $(y, x) \in$FindRC1st(N, y). However this is impossible; indeed, if $N(x, y)$ contained a reticulated cherry (y, x), then it is immediately clear that N was not a tree-child network. The grandparent of x that is not the parent of y is a tree node parent of two reticulations in N. Therefore appending the cherries from the two sets FindRP2nd(N, y) and FindRC1st(N, y) to $\mathcal{C}$ ensures that there are no forbidden cherry pairs in the updated list $\mathcal{C}$. Furthermore, updating $\mathcal{C}$ in this way ensures that we only look for each reducible pair once.

In the above cases, we must also make sure that the list $\mathcal{C}$ is non-empty as long as the network has not been reduced to a single leaf. But this is immediate by Proposition 2, and since we add all non-forbidden newly possible reducible pairs to the list. Therefore, Algorithm 4 is correct; repeating the while loop will eventually completely reduce the network, and give a TCS. In particular, since all ordered pairs in the sequence reduce a cherry or a reticulated cherry, the output TCS is minimal.

We now compute the running time of the algorithm. Let $|X|$ and r denote the number of leaves and the reticulation number of the network, respectively. Each call of FindRP2nd takes constant time, so the for loop takes $O(|X|)$ time. Within the while loop, the algorithms ReducePair, FindRP2nd, and FindRC1st are called $|X|+r-1$ times since the length of a minimal TCS is of length $|X|+r-1$ by Lemma 4. Each call of ReducePair and FindRP2nd runs in constant time. Since every reducible pair is found at most once, Lines 5-7 of FindRC1st are called at most r times in total. This means that although FindRC1st will be called $|X|+r-1$ times, the part of the algorithm that is dependent on the indegree of the reticulation vertex will only run at most r times. It follows then that the while loop runs in time at most $O(|X|+r)$, and therefore that Algorithm 4 runs in $O(|X|+r)$ time. □

Algorithm 5: CPSReducesNetwork(N, S).

```
   Data: A CPN N and a CPS S
   Result: Yes if S reduces N, No otherwise
1  for i = 1, ..., |S| do
2  |  N = ReducePair(N, S_i)
3  end
4  if N is a network on a single leaf then
5  |  return Yes;
6  end
7  return No;
```

Lemma 26. *Let N be a CPS on taxa set X and reticulation number r, and let S be a CPS of length $|S|$. Algorithm 5 correctly checks whether S reduces N and runs in $O(|S|)$ time if N is semi-binary.*

[2] [22] provided an improvement to our old version of Algorithm 4 in lines 11 and 13, when the newly obtained network is scanned for new reducible pairs. The algorithm presented here is the improved version. Before, the running time of the algorithm depended on the maximum indegree of the vertices in the network; now, the running time only depends on the number of leaves and the number of reticulation vertices.

Proof. The correctness of the algorithm follows straight from the definition of reducing a pair from CPNs. To compute the running time, note that we iterate over the for loop $|S|$ times. The step within the for loop, Algorithm 3, runs in constant time for semi-binary networks. Hence, it is clear that the algorithm runs in $O(|S|)$ time. □

Algorithm 6: TCNContains(N, N').

Data: Two semi-binary TCNs N and N' on the same set of taxa
Result: Yes if N contains N', No otherwise
1 Set $S =$FindTCS(N);
2 **return** CPSReducesNetwork(N', S);

Theorem 7. *Given two semi-binary TCNs N and N' on taxa set X where the reticulation number of N is r, it can be decided in time $O(|X| + r)$ whether N' is a subnetwork of N.*

Proof. We prove the correctness of Algorithm 6 and show that it runs in linear time. We use Algorithms 4 and 5. First we find a TCS for N in $O(|X| + r)$ time with Algorithm 4. Then, again in $O(|X| + r)$ time, we see if this TCS reduces N' using Algorithm 5. If this TCS does indeed reduce N', then N' is a subnetwork of N by Theorem 5. Hence, combining the two algorithms, the second algorithm outputs yes precisely if N' is a subnetwork of N. Furthermore, the algorithm runs in linear time as the combination of the two algorithms runs in $O(|X| + r)$ time. □

A python implementation of Algorithm 6 was presented and tested in [22]. The algorithm was tested again on different data sets in [16]. In both analyses, it was concluded that the algorithm runs in linear time in practice as well, thereby showing that Network Containment can be solved for semi-binary tree-child networks in linear time in practice. The results show that even networks with over a thousand leaves can be dealt with within a second.

Two networks are isomorphic if both networks are subnetworks of the other. Therefore, Theorem 7 has the following corollary regarding network Isomorphism, which asks whether two given networks are isomorphic. The problem for tree-child networks was previously shown to be solvable in $O(|X|^2)$ time [4]. We present the first linear-time algorithm for checking whether two tree-child networks are isomorphic.

Corollary 5. *Given two semi-binary TCNs N and N' on taxa set X where the reticulation number of N is r, it can be decided in $O(|X| + r)$ time whether N is isomorphic to N'.*

6.2. Isomorphism results for CPNs

We mentioned in Section 3.2 that we can distinguish CPNs within reconstructible classes by comparing their smallest CPSs. Indeed, by altering Algorithm 4, we can find the smallest CPS for a given CPN in polynomial time. The change is simple: at the start of the while loop, simply choose the smallest cherry instead of choosing any arbitrary cherry. Furthermore, every time we update the cherry list $\mathcal{C}$, if we have just reduced a reticulated cherry (x, y) from N, then we add another set FindRP2nd(N, x).

There is another thing to take into consideration. Tree-child networks are defined to be stack-free; binary CPNs are not necessarily stack-free. Because of this, upon reducing a reticulated cherry (x, y) from a binary CPN, it is possible that a new reticulated cherry with x as the first coordinate is in the reduced network. This means that when updating the list of reducible pairs ($\mathcal{C}$ in the algorithms), we must also append elements in which x occurs as the first element, by calling the algorithm FindRC1st(N, x). Observe that we do not need to do this if the CPN is stack-free, due to the same argument as stated in the proof of Lemma 25.

We start by introducing an algorithm that finds a smallest CPS for a semi-binary stack-free CPN, and then show that simply changing one of the lines in the algorithm allows for an input of binary CPNs. A slightly more involved change allows for an input of non-binary reconstructible CPNS.

Lemma 27. *Let N be a semi-binary stack-free CPN on taxa set X with reticulation number r. Algorithm 7 finds a smallest CPS for N in $O((|X| + r)|X| \log(|X|))$ time, if an ordering is given on X.*

Proof. Initially, we find the full set of reducible pairs by noting that a leaf appears as a second coordinate in a reducible pair at most once (the first for loop). Within the while loop, we always pick a smallest reducible pair. After picking a reducible pair (x, y), we update the list of all possible reducible pairs by adhering to Lemma 5: all new reducible pairs must contain one of the leaves x or y. As was the case for Algorithm 4, there is no need to include the set of reducible pairs FindRC1st(N, x) as these pairs would have been found in one of the previous iterations. This ensures that all possible reducible pairs are put into the set $\mathcal{C}$. Therefore, the algorithm iteratively picks the smallest reducible pair from a list of all possible reducible pairs—hence, it returns a smallest CPS of the input CPN (minimal in particular, since all the elements of the sequence reduce a cherry or a reticulated cherry in the CPN).

Algorithm 7: FindCPS(N).

```
   Data: A semi-binary stack-free CPN N, an ordering on the leaf set of N
   Result: A smallest CPS S for N
 1 Set C = ∅ ;
 2 for x ∈ L(N) do
 3 |  C∪FindRP2nd(N, x);
 4 end
 5 Let S be an empty sequence;
 6 while C ≠ ∅ do
 7 |  Choose (x, y) ∈ C such that (x, y) is smallest;
 8 |  Set S = S(x, y);
 9 |  N' = ReducePair(N, (x, y));
10 |  if (x, y) is a cherry in N then
11 |  |  C = C \ {(x, y), (y, x)}∪FindRP2nd(N', y)∪FindRC1st(N', y);
12 |  if (x, y) is a reticulated cherry in N then
13 |  |  C = C \ {(x, y)}∪FindRP2nd(N', x)∪FindRP2nd(N', y)∪FindRC1st(N', y);
14 |  end
15 end
16 return S;
```

To compute the running time, observe first that the for loop takes $O(|X|)$ time. Within the while loop, we first choose a reducible pair that is smallest. This takes $O(|\mathcal{C}|\log(|\mathcal{C}|))$ time by running some sorting algorithm, and since any leaf can appear as a second coordinate of at most one reducible pair, we have that $|\mathcal{C}| = O(|X|)$. Therefore, choosing a smallest reducible pair takes at most $O(|X|\log(|X|))$ time. By the same argument used in the proof of Lemma 25, we have that all other steps within the while loop take $O(|X|+r)$ time. Therefore the algorithm runs in $O((|X|+r)|X|\log(|X|))$ time. □

For the algorithm to work on binary CPNs, simply exchange Line 13 of Algorithm 7 by the following:

$$\mathcal{C} = \mathcal{C} \setminus \{(x,y)\} \cup \textsc{FindRP2nd}(N',x) \cup \textsc{FindRC1st}(N',x) \cup \textsc{FindRP2nd}(N',y) \cup \textsc{FindRC1st}(N',y);$$

Lemma 28. *Let N be a binary CPN on taxa set X with reticulation number r. Algorithm 7 with the modification to Line 13 finds a smallest CPS for N in $O((|X|+r)|X|\log(|X|))$ time, if an ordering is given on X.*

Proof. The correctness of the algorithm follows from the proof of Lemma 27, with the addition that upon reducing the network by a pair (x,y), the newly introduced reducible pairs may have x as a first coordinate (which appear due to stacks). We resolve this by appending FindRC1st(N',x) to the list of possible reducible pairs (Line 13).

For the running time analysis, observe first that the for loop runs in $O(|X|)$ time. The while loop is iterated $O(|X|+r)$ times, as a minimal CPS for a network on $|X|$ leaves and reticulation number r has length $|X|+r-1$ by Lemma 4. As shown in the proof of Lemma 27, the running time of finding the smallest reducible pair from $\mathcal{C}$ takes $O(|X|\log(|X|))$ time. All other steps within the while loop take constant time: in particular, FindRC1st takes constant time as the indegree of the reticulation vertices is bounded in a binary network. Hence, the while loop runs in $O((|X|+r)|X|\log(|X|))$ time, so the algorithm runs in $O((|X|+r)|X|\log(|X|))$ time. □

In case of non-binary reconstructible CPNs, the operations require a slight care. This precaution stems from two reasons. Firstly, tree vertices are now allowed to have outdegree greater than 2. This means that every leaf no longer appears at most once as a second coordinate of a reducible pair: they can be a second coordinate of a pair multiple times. Therefore Algorithm FindRP2nd may now return a set containing more than one element. This poses a potential problem in the original Algorithm 7, since upon reducing the network, we update the list of reducible pairs by adding either FindRP2nd(N',x) or FindRP2nd(N',y) (or both). Indeed, this could mean that we could obtain the same reducible pairs multiple times if there exist cherries or reticulated cherries with a multifurcation. To avoid this, we only invoke these updates whenever the outdegree of the parent of x and y (in either the original or the reduced network) is 2. This helps avoid double or triple counting of the same reducible pairs. Secondly, as seen in Lemma 6 reducible pairs of the original network may not be reducible pairs in a reduced network, although the pair itself was not reduced. For example if a network contains cherries (x,y) and (x,z), then upon reducing by the pair (x,y), the new network no longer contains the cherry (x,z), since x is no longer a leaf in the network. Due to this, we must replace the lines 10 – 13 of Algorithm 7 by the following.

Lemma 29. *Let N be a non-binary reconstructible CPN on taxa set X with reticulation number r. Algorithm 7 with the modification to Lines 10 – 13 finds a smallest CPS for N in $O((|X|+r)|X|^2\log(|X|^2))$ time, if an ordering is given on X.*

Proof. The correctness of the algorithm follows from the proof of Lemma 27, with the changes outlined as above. As before, the first for loop finds all reducible pairs within the network. Upon reducing the smallest reducible pair, we update the list

```
10 if (x, y) is a cherry in N then
11     for l ∈ L(N) do
12         if (x, l) is a cherry in N then
13             C = C \ {(x, l), (l, x)};
14     end
15     C = C ∪ FindRC1st(N′, y);
16     Let p_y denote the parent of y in N;
17     if outdegree(p_y) = 2 then
18         C = C FindRP2nd(N′, y);
19 if (x, y) is a reticulated cherry in N then
20     for l ∈ L(N) do
21         if (y, l) is a cherry in N then
22             C = C \ {(x, l)};
23     end
24     C = C \ {(x, y)} ∪ FindRC1st(N′, y);
25     Let p_x denote the parent of x in N;
26     Let p_y denote the parent of y in N;
27     if indegree(p_x) = 2 then
28         C = C ∪ FindRC1st(N′, x) ∪ FindRP2nd(N′, x);
29     if outdegree(p_y) = 2 then
30         C = C ∪ FindRP2nd(N′, y);
31 end
```

of reducible pairs by using the above replacement of the algorithm. Suppose that we have just reduced by a reticulated cherry (x, y). Let N denote the original network and $N' = N(x, y)$, as in the algorithm. As mentioned before, there may be reducible pairs in N that are no longer reducible pairs in N'. These are all reticulated cherries of the form (x, z), for which the reticulation edge was between the parent of y and the parent of x. Then, y and z must have formed a cherry in N—therefore we check for all such cherries (line 20), and remove the corresponding reducible pairs from the list. When updating the list, all new reducible pairs involve either the leaf x or y. To avoid seeking for pairs that are already contained in the list, we only invoke the FindRP2nd algorithm whenever new pairs arise in N', that is, whenever the reticulation parent of x is of indegree-2 or when the tree vertex parent of y is of outdegree-2 in N. This holds similarly for updating the list upon reducing cherries. Therefore, we find each reducible pair exactly once in the algorithm.

For the running time, observe that the for loop takes at most $O(|X|^2)$ time, as there can be at most $\binom{|X|}{2}$ possible reducible pairs in a network. Therefore $|\mathcal{C}| = O(|X|^2)$. The while loop is iterated $O(|X| + r)$ times, as a minimal CPS for a network on $|X|$ leaves and reticulation number r has length $|X| + r - 1$ by Lemma 4. Each call of ReducePair runs in constant time. The running time for finding the smallest reducible pair from $|\mathcal{C}|$ takes $O(|X|^2 \log(|X|^2))$ time. Upon reducing (x, y) from a network N, we enter a for loop that checks whether there are reducible pairs in the list that are no longer reducible pairs in the network. This step takes $O(|X|)$ time. Since each reducible pair is found at most once, the part of the algorithms FindRP2nd and FindRC1st that has a dependency on the outdegree and the indegree, respectively will only run at most $O(\binom{|X|}{2}) = O(|X|^2)$ times. Therefore the steps within the while loop run in time at most $O(|X|^2 \log(|X|^2)$ time, and therefore Algorithm 7 with the proposed changes runs in $O((|X| + r)|X|^2 \log(|X|^2))$ time. □

Theorem 8. *Suppose we are given an ordering on the taxa set X. Let N and N' be two networks with r reticulations within the same reconstructible class. Then it can be decided in $O((|X| + r)|X|^2 \log(|X|^2)$ time whether they are isomorphic. In particular if the classes are $(1a, 2a)$ (binary) or $1a, 2b)$ (semi-binary stack-free), then this can be decided in $O((|X| + r)|X| \log(|X|)$ time.*

Proof. Use Algorithm 7 twice to find the smallest CPSs for the two CPNs. This can be done in $O((|X| + r)|X|^2 \log(|X|^2))$ time (in $O((|X| + r)|X|^2 \log(|X|^2)$ for the classes $(1a, 2a)$ (binary) or $1a, 2b)$ (semi-binary stack-free)). By Corollary 2, these CPSs are the same if and only if the CPNs are isomorphic. □

7. Discussion

In this paper, we have looked at the correspondence between cherry-picking sequences and containment and subnetwork relations. For this purpose, we have extended the definition of cherry reductions to networks. This leads to the introduction of the class of cherry-picking networks (consisting of all networks that can be reduced by a CPS), and of constructions that produce a CPN from a CPS. All networks obtainable from a given construction were then said to be in the same reconstructible class.

We have shown that, within reconstructible classes, a network is uniquely determined by a smallest sequence that reduces it (given an ordering on the leaves). This correspondence can be used to check whether two CPNs in the same reconstructible class are isomorphic in polynomial time. Moreover, we showed that if a sequence for a network reduces another network, then the second network is contained in the first network. For tree-child networks, we further showed that the converse was also true, i.e., that a tree-child network contained in another tree-child network will be reduced by

Table 1
Main results of the paper relating network containment and CPSs. The two networks N and N' are assumed to have the same leaf-sets. The N and N' columns show the assumptions on both networks, where TC stands for tree-child. The direction column shows whether containment of N' in N implies ($\Longrightarrow$), is implied by ($\Longleftarrow$), or is equivalent to ($\Longleftrightarrow$) reduction of N' by any CPS for N. The subnetwork column contains a ✓ when N' can be assumed to be a subnetwork instead of a contained network of N. For results in which only one direction is presented, we have counter-examples showing that the converse does not hold.

Result	N	N'	Direction	Subnetwork
Lemma 13	Binary		$\Longleftarrow$	✓
Theorem 3	Binary	Non-binary	$\Longleftarrow$	
Lemma 15	Same reconstructible class		$\Longleftarrow$	
Corollary 3	Non-binary TC		$\Longrightarrow$	✓
Theorem 5	Same reconstructible TC class		$\Longleftrightarrow$	
Theorem 6	Semi-binary TC		$\Longleftrightarrow$	✓

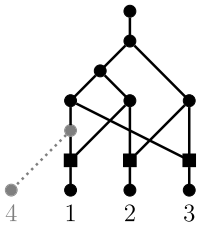

Fig. 15. A tree-based network on leaves $\{1, 2, 3\}$ that is not a CPN, since it contains a crown (the subgraph induced by the parents and the grandparents of the leaves $1, 2, 3$) without the gray vertex). The network on $\{1, 2, 3, 4\}$ is a CPN, since it provides a reticulated cherry $(1, 4)$, an 'escape' for the crown.

any tree-child sequence of the second network, given that the networks are in the same reconstructible class. A summary of all results that relate reduction of a network and subnetwork/containment can be found in Table 1.

An apparent limitation of the CPS approach is that they cannot be used to characterize every phylogenetic network, but only CPNs. The reason for this stems from the inability to pick a *double-reticulated cherry*. Two leaves x and y form a double-reticulated cherry if both parents p_x, p_y of x, y respectively are reticulations, and p_x and p_y share a common parent. As seen in other literature, the double-reticulated cherry poses a challenge when trying to correctly add back a once removed reticulation edge [2,21]. A CPS may contain a double-reticulated cherry, as long as there is an 'escape' at one of the sides of the double-reticulated cherry: a reticulated cherry which can be reduced, and whose reduction turns the double-reticulated cherry into a regular reticulated cherry. With no escapes however, there is currently no way of dealing with double-reticulated cherries.

One way of resolving this issue would be to consider leaf attachments as done by Linz and Semple [19]. Given any phylogenetic network, add a minimal number of leaves to make it a CPN. Recall that every reconstructible CPN has a unique smallest CPS (Theorem 2). Is it possible to extend this result to general networks by attaching leaves? If so, will the placement of the leaves affect the uniqueness of its smallest CPS? Furthermore, what is the minimal number of additional leaves to turn a general network into a CPN? To answer these questions, one could study the forbidden structures of a CPN; currently, very little is known about these.

Obviously, crowns—undirected cycles within a network whose bipartition into tree nodes and reticulation nodes makes it a bipartite subgraph—cannot be contained in CPNs. Aside from this, we are not aware of other forbidden structures of CPNs. In any case, we may gauge where the class of CPNs resides with respect to other prominent network classes. The class of CPNs does not contain the class of tree-based networks [7], since tree-based networks may contain crowns (see Fig. 15). Conversely, each binary CPN is tree-based [11], and [14] contains a proof that this holds in the non-binary case as well, but the latter has not been reviewed yet. Furthermore, as there exists a CPS (in fact, a TCS) for every TCN, we conclude that class of CPNs strictly contains the class of TCNs. Therefore, in the binary case, but possibly also in the non-binary case, the class of CPNs is intermediate between tree-child networks and tree-based networks.

We briefly comment on the correspondence between containment and reduction, and pose conjectures for remaining open questions. Let N and N' be CPNs in the same reconstructible class on the same leaf-sets. If a CPS S of N reduces N', then N' is contained in N (Lemma 15). We have shown that the converse of this does not hold for the reconstructible classes $(1a, 2a)$ and $(1a, 2b)$ (Theorem 4). For the reconstructible classes $(1b, 2c)$ and $(1b, 2d)$, the problem remains open. Therefore we conjecture the following.

Conjecture 1. *There exist CPNs N and N' on the same leaf-set in the $(1b, 2c)$ class where N' is contained in N, but no CPS of N reduces N'.*

Conjecture 2. *Let N and N' be CPNs on the same leaf-set in the $(1b, 2d)$ class, where N' is contained in N. Then there exists a CPS of N that reduces N'.*

Upon restricting N and N' to be tree-child, we have shown that the converse holds for all reconstructible classes: if N' and N are both in the same reconstructible class and N' is contained in N, then there exists a TCS of N that reduces N'. We believe that this still holds when N and N' are general non-binary tree-child networks, by requiring a stronger condition. Note that it is easy to find an example of two non-binary tree-child networks that are not contained in one another, such that a TCS for one network reduces the other (see Fig. 5 d).

Conjecture 3. *Let N and N' be non-binary tree-child networks on the same leaf set. Then N' is contained in N if and only if* all *TCSs of N reduce N'.*

In Section 1 we mentioned how cherry-picking sequences originated as a tool to tackle the problem of finding a 'simple' network that contains a given set of trees. This problem, called Hybridization, and our results on cherry-picking sequences—in particular Lemma 13—provide insight into how we may solve generalizations of this problem, where we have inputs consisting of networks [17]. Indeed, if we find a CPS that reduces the input set of networks, then the network corresponding to this CPS contains the input set. However, it is not quite clear when this gives an optimal CPN with minimal reticulation number. For example, take the network N and the tree T in Fig. 13. If a CPS reduces the tree and the network, the corresponding CPN must have at least one more reticulation than the optimal network, which is clearly the input network N. This means that given an input of networks, it may not be possible to find a CPS that corresponds to the CPN with minimal reticulation number that contains all inputs. It would be interesting to see if this problem also occurs when the input consists solely of trees. That is, given an input of trees, can we always find the CPS which corresponds to a CPN with minimal reticulation number containing all these trees? This problem aside, the problem of finding a minimal CPS gives an upper bound for Hybridization, and a lower bound for TC cherry-picking, where TC cherry-picking asks to find a minimal TCS which reduces all trees in the input set. Hence, it might also be of merit to see what exactly happens when only trees are considered as the input: does the minimal CPS give useful information about any of these problems?

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We would like to thank Leo van Iersel for providing feedback on multiple versions of our paper. We would like to thank Robbert Huijsman for implementing and testing our pseudocode in Python.

References

[1] E. Bapteste, L. van Iersel, A. Janke, S. Kelchner, S. Kelk, J.O. McInerney, D.A. Morrison, L. Nakhleh, M. Steel, L. Stougie, J. Whitfield, Networks: expanding evolutionary thinking, Trends Genet. 29 (8) (2013) 439–441.
[2] M. Bordewich, C. Semple, Determining phylogenetic networks from inter-taxa distances, J. Math. Biol. 73 (2) (2016) 283–303.
[3] S. Borst, L. van Iersel, M. Jones, S. Kelk, New FPT algorithms for finding the temporal hybridization number for sets of phylogenetic trees, preprint, arXiv:2007.13615, 2020.
[4] G. Cardona, F. Rossello, G. Valiente, Comparison of tree-child phylogenetic networks, IEEE/ACM Trans. Comput. Biol. Bioinform. 6 (4) (2009) 552–569.
[5] J. Döcker, L. van Iersel, S. Kelk, S. Linz, Deciding the existence of a cherry-picking sequence is hard on two trees, Discrete Appl. Math. 260 (2019) 131–143.
[6] P.L. Erdős, C. Semple, M. Steel, A class of phylogenetic networks reconstructable from ancestral profiles, Math. Biosci. 313 (2019) 33–40.
[7] A.R. Francis, M. Steel, Which phylogenetic networks are merely trees with additional arcs?, Syst. Biol. 64 (5) (2015) 768–777.
[8] P. Gambette, A.D. Gunawan, A. Labarre, S. Vialette, L. Zhang, Solving the tree containment problem for genetically stable networks in quadratic time, in: International Workshop on Combinatorial Algorithms, Springer, 2015, pp. 197–208.
[9] P. Gambette, A.D. Gunawan, A. Labarre, S. Vialette, L. Zhang, Solving the tree containment problem in linear time for nearly stable phylogenetic networks, Discrete Appl. Math. 246 (2018) 62–79.
[10] A. Gunawan, Solving tree containment problem for reticulation-visible networks with optimal running time, preprint, arXiv:1702.04088, 2017.
[11] K.T. Huber, L. van Iersel, R. Janssen, M. Jones, V. Moulton, Y. Murakami, C. Semple, Rooting for phylogenetic networks, preprint, arXiv:1906.07430, 2019.
[12] P.J. Humphries, S. Linz, C. Semple, Cherry picking: a characterization of the temporal hybridization number for a set of phylogenies, Bull. Math. Biol. 75 (10) (2013) 1879–1890.
[13] L. van Iersel, C. Semple, M. Steel, Locating a tree in a phylogenetic network, Inf. Process. Lett. 110 (23) (2010) 1037–1043.
[14] L. van Iersel, R. Janssen, M. Jones, Y. Murakami, N. Zeh, A unifying characterization of tree-based networks and orchard networks using cherry covers, preprint, arXiv:2004.07677, 2020.
[15] R. Janssen, Y. Murakami, Solving phylogenetic network containment problems using cherry-picking sequences, preprint, arXiv:1812.08065, 2018.
[16] R. Janssen, Y. Murakami, Linear time algorithm for tree-child network containment, in: International Conference on Algorithms for Computational Biology, Springer, 2020, pp. 93–107.
[17] R. Janssen, M. Jones, Y. Murakami, Combining networks using cherry picking sequences, in: International Conference on Algorithms for Computational Biology, Springer, 2020, pp. 77–92.

[18] I.A. Kanj, L. Nakhleh, C. Than, G. Xia, Seeing the trees and their branches in the network is hard, Theor. Comput. Sci. 401 (1–3) (2008) 153–164.
[19] S. Linz, C. Semple, Attaching leaves and picking cherries to characterise the hybridisation number for a set of phylogenies, Adv. Appl. Math. 105 (2019) 102–129.
[20] D.A. Morrison, Networks in phylogenetic analysis: new tools for population biology, Int. J. Parasitol. 35 (5) (2005) 567–582.
[21] Y. Murakami, L. van Iersel, R. Janssen, M. Jones, V. Moulton, Reconstructing tree-child networks from reticulate-edge-deleted subnetworks, Bull. Math. Biol. 81 (10) (2019) 3823–3863.
[22] R. Huijsman, Tree-child network containment, Bachelor's thesis, Delft University of Technology, 2019.