Theoretical Computer Science 401 (2008) 153–164

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Seeing the trees and their branches in the network is hard☆

Iyad A. Kanj [a,*], Luay Nakhleh [b], Cuong Than [a], Ge Xia [c]

[a] School of Computer Science, Telecommunications, and Information Systems, DePaul University, 243 S. Wabash Avenue, Chicago, IL 60604-2301, USA
[b] Department of Computer Science, Rice University, 6100 Main Street, MS 132 Houston, TX 77005-1892, USA
[c] Department of Computer Science, Acopian Engineering Center, Lafayette College, Easton, PA 18042, USA

ARTICLE INFO

Article history:
Received 22 December 2007
Accepted 12 April 2008
Communicated by D.-Z. Du

Keywords:
Phylogenies
Networks
Complexity
Parameterized algorithms

ABSTRACT

Phylogenetic networks are a restricted class of directed acyclic graphs that model evolutionary histories in the presence of *reticulate* evolutionary events, such as horizontal gene transfer, hybrid speciation, and recombination. Characterizing a phylogenetic network as a collection of trees and their branches has long been the basis for several methods of reconstructing and evaluating phylogenetic networks. Further, these characterizations have been used to understand molecular sequence evolution on phylogenetic networks.

In this paper, we address theoretical questions with regard to phylogenetic networks, their characterizations, and sequence evolution on them. In particular, we prove that the problem of deciding whether a given tree is contained inside a network is NP-complete. Further, we prove that the problem of deciding whether a branch of a given tree is also a branch of a given network is polynomially equivalent to that of deciding whether the evolution of a molecular character (site) on a network is governed by the *infinite site model*. Exploiting this equivalence, we establish the NP-completeness of both problems, and provide a parameterized algorithm that runs in time $O(2^{k/2}n^2)$, where n is the total number of nodes and k is the number of recombination nodes in the network, which significantly improves upon the trivial brute-force $O(2^k n)$ time algorithm for the problem. This reduction in time is significant, particularly when analyzing recombination hotspots.

1. Introduction

Phylogenies, i.e., evolutionary histories, play a major role in representing the relationships among biological entities. Their pervasiveness has led biologists, mathematicians, and computer scientists to design a variety of methods for their reconstruction. Until recently, most of these methods were designed to construct trees. Yet, biologists have long recognized that trees oversimplify our view of evolution in certain cases, since they cannot model events such as hybrid speciation, horizontal gene transfer (HGT), and recombination. These events, which are collectively referred to as *reticulation events* or *reticulate evolutionary events*, give rise to non-treelike evolutionary histories which are best modeled by *phylogenetic networks*.

Reconstructing and evaluating the quality of phylogenetic networks is very important, given the emerging evidence of the ubiquity of reticulation events and the evolutionary roles they play. Hybrid speciation is a main mode of evolution in large groups of plants, fish, and frogs [19]. Horizontal gene transfer is believed to be ubiquitous among prokaryotic organisms [6, 26], and recent evidence shows massive HGT in some plants [25,3,4]. Interspecific recombination's role in evolutionary

☆ A preliminary version of this paper appeared in proceedings of the Tenth Italian Conference on Theoretical Computer Science (ICTCS'07).

* Corresponding author.
E-mail addresses: ikanj@cs.depaul.edu (I.A. Kanj), nakhleh@cs.rice.edu (L. Nakhleh), cvthan@cs.rice.edu (C. Than), gexia@cs.lafayette.edu (G. Xia).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.04.019

genomics has long been acknowledged [35,34]. Finally, meiotic recombination plays a major role in genomic diversification in populations, and detecting it bears great implications on genotype–phenotype associations [9,1,24].

Relationships between phylogenetic networks on one hand, and the trees and their branches on the other, have great significance. From the computational perspective, these relationships form the basis for the wide array of methods that have been devised for reconstructing phylogenetic networks; e.g., see [18,23] for extensive surveys of these methods. From the biological perspective, these relationships shed light on how molecular sequences evolve down these networks. Events such as recombination, hybrid speciation, and lateral gene transfer break up the genomic history into many small pieces, each of which has a strictly treelike pattern of descent [21]. Identifying these trees and reconciling their discordance is the basis for several phylogenetic network reconstruction methods [22,11,8,33,20,2,30]. Understanding the relationship between a phylogenetic network and its branches, particularly in terms of the *clusters* (or *splits*) of taxa that they induce, has been the basis for another category of reconstruction methods, which includes the methods of [5,12]. Very recently, Nakhleh and colleagues introduced new approaches for augmenting a tree into a phylogenetic network to fit the evolution of a set of sequences based on the parsimony [28,13,15,16] and likelihood [14] criteria.

Almost all of the aforementioned methods are based on understanding relationships among networks, trees, and clusters of taxa. Further, some of them rely on analysis of the evolution of sequences on networks. In this paper, we provide a theoretical treatment of the computational complexity of establishing some of these relationships. In [32,31], the authors devised efficient algorithms for restricted cases of some of these problems, while leaving the computational complexity of the general cases as open questions. In this paper, we prove that the problem of deciding whether a given tree is contained inside a network is NP-complete. Further, we prove that the problem of deciding whether a branch of a given tree is also a branch of a given network is polynomially equivalent to that of deciding whether the evolution of a molecular character (site) on a network is governed by the *infinite site model*. Exploiting this equivalence, we establish the NP-completeness of both problems, and provide a parameterized algorithm that runs in time $O(2^{k/2}n^2)$, where n is the total number of nodes and k is the number of recombination nodes in the network, which significantly improves upon the trivial brute-force $O(2^k n)$ time algorithm for the problem. This improvement is very significant in practice [29]. In [17], the authors considered the problem of character compatibility on a different model of phylogenetic networks that is used in historical linguistics. Whereas the NP-hardness result from that work is modified and used here, that is not the case, however, for the new parameterized algorithm that we present. The algorithmic techniques used in [17] do not carry over to the biologically-motivated model of phylogenetic networks that we consider in this paper.

2. Phylogenetic networks, trees, and the infinite site model

Let $T = (V, E)$ be a tree, where V and E are the *tree nodes* and *tree edges*, respectively, and let $L(T)$ denote its leaf set. Further, let $\mathcal{X}$ be a set of taxa (species). Then, T is a phylogenetic tree over $\mathcal{X}$ if there is a bijection between $\mathcal{X}$ and $L(T)$. A tree T is said to be *rooted* if the set of edges E is directed and there is a single distinguished internal node r with in-degree 0.

A character c labeling the leaves of T is a function $c : L(T) \to \{0, 1\}$. Biologically, such a character corresponds to a single SNP, and the two states it takes are the two possible *alleles* that the SNP may exhibit.[1] The commonly assumed model of evolution of SNPs is the *infinite site model*, which states that when a character (site) mutates, it changes its state to a new one that is not observed anywhere else in the tree. We denote by $c(v)$ the state of character c for node v. A *haplotype* of length ℓ is a sequence of such characters $c_1 \cdots c_\ell$. A *full labeling*, or labeling for short, for character c on the tree is an extension, $\hat{c}$, of character c to label all the nodes of T; i.e., $\hat{c} : V(T) \to \{0, 1\}$ and $\hat{c}(v) = c(v)$ for every $v \in L(T)$. In this paper, we focus on characters that exhibit *exactly* two states.

Definition 2.1. A character c is *compatible* on tree T if there is a labeling $\hat{c}$ which extends c such that there exists exactly one edge $e = (u, v) \in E(T)$ where $\hat{c}(u) \neq \hat{c}(v)$, and for all other edges $e' = (u', v') \neq e$, $\hat{c}(u') = \hat{c}(v')$.

Notice that if SNP c evolves under the infinite site model, then there is a tree on which it is compatible. Hence, the compatibility criterion reflects this model of evolution. A sequence of characters $c_1 \cdots c_k$ is compatible on tree T if every character c_i, $1 \leq i \leq \ell$, is compatible on T. By this definition of compatibility, it suffices to establish the computational complexity of and develop algorithms for testing the compatibility of single characters. Therefore, from this point on, we focus on the case of a single character. Testing whether a character is compatible on a tree T with n leaves can be done in $O(n)$ time, as shown in [27].

As explained in Section 1, when reticulation events occur, the evolutionary history of a set of sequences is best modeled by a phylogenetic network. A phylogenetic network $N = (V, E)$ is a rooted directed acyclic graph, with set $L(N)$ of leaves, such that there is a bijection between a set of taxa $\mathcal{X}$ and $L(N)$. A network N has three types of nodes: (1) one node r with in-degree 0, which corresponds to the root; (2) nodes with in-degree 1, which correspond to *coalescence* events; and (3) nodes with in-degree 2, which correspond to recombination. Nodes with in-degree 2 can be called reticulation nodes. Fig. 1 shows an example of a phylogenetic network on four taxa A, B, C, and D.

A phylogenetic network N induces, or contains, a set of trees; these trees model the evolutionary histories of sets of non-recombining segments (or, genes) in the genomic sequences. We denote by $\mathcal{T}(N)$ the set of all trees contained inside network

[1] Even though SNPs may exhibit all four states (A, C, T, and G), bi-allelic SNPs, i.e., SNPs that exhibit two states, are the most common.

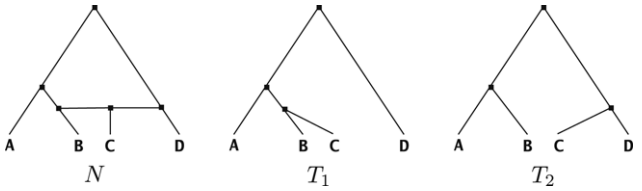

Fig. 1. A phylogenetic network N on four taxa A, B, C, and D, and the two trees T_1 and T_2 it contains. Character c_1, where $c_1(A) = c_1(D) = 0$ and $c_1(B) = c_1(C) = 1$ is compatible on tree T_1, but not compatible on tree T_2. Character c_2, where $c_2(A) = c_2(B) = 0$ and $c_2(C) = c_2(D) = 1$ is compatible on T_2 but not on T_1. It can be easily checked that there does not exist any tree T on which both characters are compatible. However, both characters are compatible on network N. The two horizontal edges in N are directed towards the parent of C.

N. Each such tree is obtained by the following two steps: (1) for each node of in-degree 2, remove one of the incoming edges, and then (2) for every node x of in-degree and out-degree 1, whose parent is u and child is v, remove node x and its two adjacent edges, and add a new edge from u to v. If node x is the root and its out-degree is 1, remove x and make its only child the new root for the tree. Fig. 1 shows the two trees contained inside network N. The membership problem of trees and networks, which is heavily used in network reconstruction methods, is formulated as follows.

Problem 1 (*Tree Containment (*TC)).

Input: *A phylogenetic network N and tree T over the same set $\mathcal{X}$ of taxa.*
Question: *Is $T \in \mathcal{T}(N)$?*

In the next section, we prove that the TC problem is NP-complete. The notion of character compatibility is extended to phylogenetic networks so as to reflect the biological fact that the evolutionary history of a character is modeled by one of the trees inside the network.

Definition 2.2. A character c is *compatible* on network N if c is compatible on at least one tree $T \in \mathcal{T}(N)$.

The problem of testing the infinite site model on a phylogenetic network can be defined as follows.

Problem 2 (*Infinite Site on Phylogenetic Networks (*ISPN)).

Input: *Phylogenetic network N and a binary character c labeling the leaves of N.*
Question: *Is c compatible on N?*

Given a network N with k nodes of in-degree 2, the size of $\mathcal{T}(N)$ is $O(2^k)$. Therefore, The ISPN problem is solvable in $O(2^k n)$ time, given the algorithm in [27] for solving the problem when N is a tree. In the next sections, we prove that ISPN is NP-complete and introduce a more efficient algorithm for solving it.

Further, we establish equivalence between the ISPN problem and another problem from phylogenetics, namely the Cluster Containment problem [31]. Let T be a phylogenetic tree on a set $\mathcal{X}$ of taxa. We say that edge e *induces*, or *defines*, cluster $X \subseteq \mathcal{X}$, where X is the set of all leaves reachable from the root of T through edge e. We denote by $\mathcal{C}(T)$ the set of all clusters defined by tree T. This notion is extended to networks by $\mathcal{C}(N) = \cup_{T \in \mathcal{T}(N)} \mathcal{C}(T)$. The Cluster Containment problem is defined as follows.

Problem 3 (*Cluster Containment (*CC)).

Input: *A phylogenetic Network N and set $\mathcal{X}$ of taxa.*
Question: *Is $X \in \mathcal{C}(N)$?*

In [31], a polynomial time algorithm was devised for a restricted version of the CC problem, yet its complexity in the general case was left open. We show that CC and ISPN are polynomially equivalent, thus establishing NP-completeness of the former problem as well.

3. Computational Complexity of the TC Problem

In this section we will prove that the TC problem is hard.

Theorem 3.1. *The problem* TC *is NP-complete.*

Proof. TC is in NP, since given a network N, a tree T, and a certificate in the form of a tree $T' \in \mathcal{T}(N)$ (equivalently, the certificate can be the set of network edges used to induce the tree T'), we can verify in polynomial time whether $T = T'$. It remains to be shown that TC is NP-hard. The proof is by a reduction from the Node-disjoint Paths problem, which is stated as follows: given a directed graph $G = (V, E)$ and a set of disjoint node pairs $\{(s_1, t_1), \ldots, (s_k, t_k)\}$, does G contain k mutually node-disjoint paths $\{p_1, p_2, \ldots, p_k\}$, where p_i is a path from s_i to t_i? The problem is NP-complete, even when G is a DAG [36].

We construct from DAG $G = (V, E)$ a network $N = (V', E')$, where $V' = V \cup \{S, L\} \cup \{s_{l_i} : 1 \leq i \leq k\} \cup \{t_{l_i} : 1 \leq i \leq k\}$ and $E' = E \cup \{(s_i, s_{l_i}), (t_i, t_{l_i}), (S, s_i) : 1 \leq i \leq k\} \cup \{(v, L) : v \in V,\ outdeg(v) = 0,\ \text{and}\ v \neq s_i, t_i\} \cup \{(S, v) : v \in V,\ indeg(v) = 0,\ \text{and}\ v \neq s_i, t_i\} \cup \{(S, L)\}$.

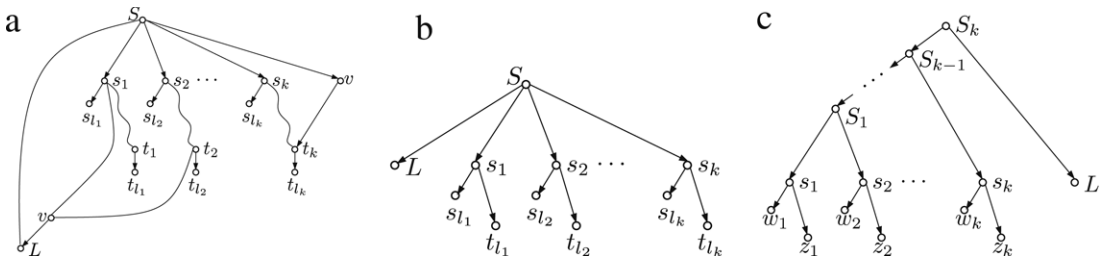

Fig. 2. Illustrations of the reduction used in the proof of Theorem 3.1. (a) The network N constructed from G, and (b) the tree T. (c) Converting non-binary trees generated by the reduction to binary ones.

The construction produces a network N with a unique root and set $\mathcal{X} = L \cup \{s_{l_i}, t_{l_i} : 1 \leq i \leq k\}$ as its leaf set. Further, given that G is a DAG, by construction so is N.

The tree T has set $\mathcal{X}$ of leaves, internal nodes s_i and the root S. Fig. 2(a) and (b) illustrate how network N and tree T generated by the reduction. The reduction is clearly polynomial time. We now show that there are k mutually node-disjoint paths in G, as specified above, if and only if $T \in \mathcal{T}(N)$.

Suppose that there are such k node-disjoint paths in G. The tree $T \in \mathcal{T}(N)$ can be obtained as follows: (1) remove all network edges coming into the leaf L, if any, except the edge (S, L); (2) for each s_i, retain the edge (S, s_i), and delete all other network edges coming into it, if any; (3) if a node v on a path from s_i to t_i is a tree node, retain the edge coming into it. Otherwise, v is a network node and we remove all network edges coming into it, except the edge lying on the path from s_i to t_i. The removal of network edges coming into v does not affect any other path from s_j to t_j, where $j \neq i$, because these paths do not share any nodes; and, finally (4) every node v, including t_i, on a path from s_i to t_i is redundant in T, and hence we perform forced contraction on all nodes v on the path, leaving t_{l_i} as a child of s_i. The result of these four steps is the tree $T \in \mathcal{T}(N)$.

Conversely, suppose that G does not contain k mutually node-disjoint paths as specified above. There are two possibilities. (1) There do not exist any paths between some pair (s_i, t_i). Therefore, no combinations of different choices of network edges in N would connect s_i with t_i. Consequently, $T \notin \mathcal{T}(N)$. (2) There are two paths $s_i \rightsquigarrow t_i$ and $s_j \rightsquigarrow t_j$, where $j \neq i$, that share at least one common node v. Clearly v must be a network node, because there are at least two edges coming into it. Suppose that we choose to retain the network edge coming into v that lies on the path $s_i \rightsquigarrow t_i$. Then in order for N to induce T, there must exist another path connecting the two nodes s_j and t_j. That path, then, must share at least one node with some other path $s_m \rightsquigarrow t_m$; otherwise, all paths $s_i \rightsquigarrow t_i$ are node-disjoint, contradicting the assumption. If $m = i$, then it is not possible to make both t_{l_i} and t_{l_j} children of s_i and s_j. In the case $m \neq i$, we must choose to retain the network lying on the path $s_j \rightsquigarrow t_j$. Then, there exists another path connecting s_m and t_m, and this path shares a node with some other path, and so on. This process must stop after a finite number of steps (because k is finite). Therefore, there must be two at least two leaves t_{l_i} and t_{l_j} connected to a node s_i, which means that $T \notin \mathcal{T}(N)$. □

Even though the above reduction may produce non-binary trees, the problem is NP-hard for binary trees as well, since the non-binary tree can be easily converted into a binary one, as illustrated in Fig. 2(c). Further, the reduction may generate networks that contain nodes of in-degree higher than 2. However, such a network can be converted to one where all nodes have in-degree at most 2, without affecting the correctness of the reduction, in the following way. For each node v of in-degree d ($d > 2$), and $u_1, \ldots, u_d$ as its parents, (1) delete all edges (u_i, v), for every $1 \leq i < d$; (2) add $d - 2$ new nodes $x_1, \ldots, x_{d-2}$; (3) add edges (x_i, x_{i+1}) for every $1 \leq i < d - 2$, and edge (x_{d-2}, v); and (4) add edges (u_{i+1}, x_i) for every $1 \leq i < d - 1$ and edge (u_1, x_1). Clearly, the resulting network has nodes of in-degree at most 2.

4. The ISPN Problem: Complexity and a Parameterized Algorithm

4.1. NP-completeness of ISPN

Kanj et al. [17] proved the NP-completeness for the problem of character compatibility on phylogenetic networks when the network edges are bi-directional. We modify their proof to make it work for the ISPN problem and present the theorem.

Theorem 4.1. ISPN *is NP-complete.*

Proof. To show ISPN is in NP, a nondeterministic Turing machine guesses a tree $T \in \mathcal{T}(N)$ for the input network N, and verifies that the character is compatible on it, which can be done in linear time using the algorithm from [27].

We will reduce 3-SAT to ISPN. Given a formula $\varphi = \theta_1 \wedge \theta_2 \wedge \cdots \wedge \theta_m$, where each θ_i, $1 \leq i \leq m$, is a disjunction of exactly three literals, the 3-SAT problem asks if φ is satisfiable. Suppose that φ consists of n variables $x_1, x_2, \ldots, x_n$. For each variable x_i, we construct a variable gadget as follows.

- Create two nodes for x_i and $\overline{x_i}$.

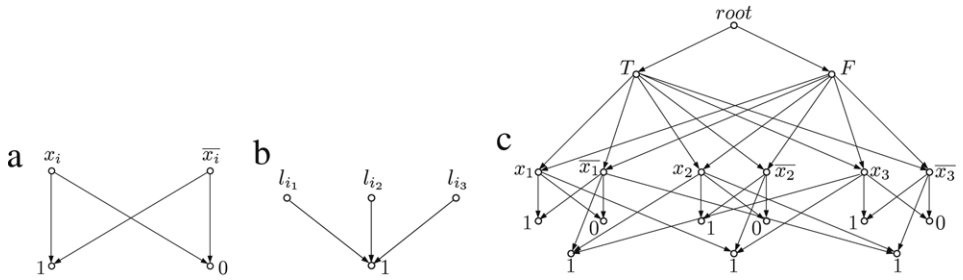

Fig. 3. Illustrations of the reduction used in the proof of Theorem 4.1. (a) The variable gadget. (b) The clause gadget. (c) The network for $\varphi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3})$.

- Create two leaves corresponding with truth values *false* and *true*, and assign them the character α values 0 and 1, respectively.
- From each of the nodes x_i and $\overline{x_i}$, add two edges to the *false* and *true* leaves.

The gadget built in this way ensures that, if α is compatible on the network we are constructing, nodes x_i and $\overline{x_i}$ cannot be connected to the same leaf. Therefore, x_i and $\overline{x_i}$ will not receive the same truth value. Fig. 3(a) illustrates a variable gadget.

For each clause $\theta_i = l_{i_1} \vee l_{i_2} \vee l_{i_3}$, we construct a clause gadget as follows.

- Create a new leaf node corresponding to θ_i and assign the α value 1 to it.
- Add three edges from nodes corresponding to l_{i_1}, l_{i_2} and l_{i_3} in variable gadgets to this leaf.

The formula φ is satisfiable if and only if every disjunction θ_i is satisfiable, and θ_i is satisfiable if and only if at least one of its literals is assigned *true*. The clause gadget ensures that, if α is compatible on the network, each clause will have at least one literal assigned *true*. Fig. 3(b) illustrates a clause gadget.

From these variable and clause gadgets, we build a network N by creating two nodes F and T and add edges from them to all literal nodes x_i and $\overline{x_i}$ in the variable gadgets. We then create a root and connect two edges from it to nodes F and T. It is easy to verify that the resulting network satisfies the properties of a phylogenetic network mentioned above. Fig. 2(c) illustrates the reduction.

Suppose that the formula φ is satisfiable. Then, there is a truth assignment that makes all disjunctions θ_i *true*. We label nodes corresponding to literal l_i as 1 if it is assigned *true*, and 0 if it is assigned *false*. We also label node T and the root of N as 1, and node F as 0. We next prove that we can induce a tree from N such that all nodes labeled 0 are connected and so is the set of nodes labeled 1. That tree is induced by retaining:

- edges from T to literal nodes labeled 1, and edges from F to literal nodes labeled 0;
- edges from literal nodes labeled 1 to leaves labeled 1 in the variable gadget, and edges from literal nodes 0 to leaves 0 in the variable gadget; and
- edges from literal nodes labeled 1 to the leaves corresponding to clauses.

It is easy to see that in the tree induced, all nodes labeled 0 form a connected component while nodes 1 also form another connected component. Therefore, the character α is compatible on N.

Conversely, suppose that the character α is compatible on N. Then, there exists a tree that is induced from N and that α is compatible on. If the edge coming from l_i to the leaf corresponding to a disjunction φ_j in φ is chosen to induce that tree, we assign *true* to l_i; all other literals are assigned *false*. First, this is a legal truth assignment. We have to assign both x_i and $\overline{x_i}$ truth value *true* if in the tree induced, they both connect to leaves 1 corresponding clauses in φ. But this cannot happen because the leaf 0 in the variable clause must belong either to node x_i or node $\overline{x_i}$, and because the character α is compatible on this tree. Second, this assignment satisfies every disjunction φ_j because at least one of its literal, l_i, is assigned *true*. Therefore, φ is satisfiable.

The reduction above clearly can be performed in polynomial time. Therefore, 3-SAT is reducible to ISPN. Since 3-SAT is NP-complete, it follows that ISPN is NP-complete.

Remark. As defined in Section 2, the in-degree of a node v in a phylogenetic network is bounded by 2. However, the reduction we devised in this section may generate a node whose in-degree is 3 (see Fig. 3(c) in the appendix). To satisfy the constraint on the in-degree of nodes, the phylogenetic network produced by the reduction can be modified as follows. For every node v, with edges (u_1, v), (u_2, v), (u_3, v) incident into it, we do the following: (1) delete edges(u_1, v) and (u_2, v); (2) add a new node x; and (3) add edges (u_1, x), (u_2, x), and (x, v). This transformation does not affect the correctness of the reduction. Further, it takes $O(n)$ time, where n is the number of leaves in the network, and the resulting network satisfies all conditions of the definition in Section 2. □

4.2. A Parameterized Algorithm for ISPN

A Prelude to the Algorithm

An instance of a parameterized problem is a pair consisting of an input instance x of size n and a parameter k. A parameterized problem is *fixed-parameter tractable* if it can be solved in time $f(k)n^{O(1)}$, where f is a computable function of the parameter k. We refer the reader to [7] for a detailed discussion on parameterized complexity. A parameterized algorithm for an intractable problem implies that the exponential growth in the running time of the algorithm can be confined to the parameter rather than the input size. Therefore, if the parameter is small, as is the case for many practical instances of NP-hard problems, the algorithm can derive a solution to an instance of the problem in a feasible amount of time.

Naturally, the ISPN problem can be parameterized by the number of *recombination nodes* (nodes of in-degree 2) k in the phylogenetic network, which is usually much smaller than the total number of nodes in the network [29]. Every recombination node in N has two incoming edges, and hence two possible parents. Deciding the parent of each recombination node in N *induces* a tree from N, and N is compatible if and only if there exists an induced tree from N that is compatible. Since there are $O(2^k)$ such induced trees, the ISPN problem can be solved in $O(2^k n)$ time, where n is the number of nodes in N, by enumerating all possible induced trees then checking whether any of them is compatible using the linear time algorithm described in [27]. We shall improve on this trivial upper bound next by presenting a simple branch-and-search algorithm that runs in $O(2^{k/2}n^2)$ time.

For two nodes u and v in N, we denote by the ordered pair (u, v) the directed edge from u to v (in case the edge exists in N). A node u in N is an *internal* node if u is not a leaf in N, that is, if the out-degree of u is greater than 0.

Definition 4.2. For a node $u \in N$ we define the *weight* of u, denoted $wt(u)$, to be the in-degree of u minus 1 if u is not the root of N, and to be 0 if u is the root of N.

Initially, every recombination node in N has weight 1 and every other node has weight 0. Therefore, the number of recombination nodes in N, which is the parameter k, is equal to $\sum_{u \in N} wt(u)$. The branch-and-search algorithm will branch by reducing the weight of some nodes in N, that is, by removing incoming edges to certain nodes. Therefore, during the execution of the algorithm, the network N may no longer satisfy the initial definition of a phylogenetic network (the nodes in N have in-degree 0, 1, or 2 and only the leaves in N are labeled), and we will refer to it by the *network* N. When the weight of every node in N becomes 0, the algorithm will check whether the resulting tree is compatible. As the algorithm progresses, the weight of a recombination node can either increase or decrease due to the operations performed by the algorithm. If the weight of a recombination node becomes 0, then the node ceases to be a recombination node. It is also possible that the weight of a recombination node exceeds 1. We formalize this notion in the following definition.

Definition 4.3. A node u in N is a said to be a *recombination node* if it is weight is greater or equal to 1, otherwise, u is said to be a *non-recombination node*.

An important notion to the algorithm is the notion of a *partition* node. If N is compatible and T is a compatible induced tree from N, then there will be a node p in T at which the nodes will be partitioned into two sets: all the nodes in the subtree of T rooted at p will be labeled with the one label (either 0 or 1), and all the remaining nodes in T will be labeled with the other label. We formally define this notion next.

Definition 4.4. Let N be a network. A node p in N is said to be a *partition node* if there exists an induced compatible tree T from N such that there is a valid labeling for the nodes in T with all the nodes in the subtree rooted at p in T labeled with the same label, and all the other nodes in T labeled with the other label.

While applying the branch-and-search process, the algorithm will label some of the internal nodes in the network. Therefore, the network will get partially labeled as the algorithm progresses. In many cases the (resulting) network can be simplified, or even, its compatibility can be inferred easily. We describe next some of the scenarios in which the compatibility of the network can be directly decided. We also describe some operations that simplify the network. The algorithm will make use of these operations and simplifications.

Proposition 4.5. *If there is at most one leaf of label* 0 *(similarly* 1*) in* N *then* N *is compatible.*

Proof. If there is at most one leaf in N of label 0, then every induced tree from N is compatible. This can be seen by picking an arbitrary induced tree from N and labeling all its internal nodes 1. □

Proposition 4.6. *Let* $u \in N$ *be a node. Suppose that* u *has two children that are non-recombination nodes. Suppose further that these two children have different labels and none of them is a partition node. Then* N *is not compatible.*

Proof. Since the two children of u are non-recombination nodes, they will remain children of u in any tree induced from N. Since the two children have different labels and none of them is a partition node, no labeling of a tree induced from N will make that tree compatible. □

Proposition 4.7. *If a labeled node* $u \in N$ *has a non-recombination child* v *such that* $label(v) \neq label(u)$ *and* v *is not a partition node, then* N *is not compatible.*

Proof. The proof of this proposition is very similar to that of Proposition 4.6. □

Propositions 4.5–4.7, illustrate some of the cases when the compatibility of N can be decided directly and easily.

The proof of the following proposition can be easily verified by the interested reader.

Proposition 4.8. *Let u be a recombination node in N and let (u', u) be an incoming edge to u. Suppose that $label(u) \neq label(u')$. Let N' be the network resulting from N by removing the edge (u', u). Then N is compatible if and only if N' is. Moreover, if p is a partition node in N then p is also a partition node in N'.*

Proposition 4.8 implies that we can remove every edge incoming to a recombination node from a node with different label, provided that the recombination node is not a partition node. When removing such an edge, the weight of the recombination node needs to be decreased by 1.

Proposition 4.9. *Let u be a labeled non-recombination node in N with the incoming edge (u', u). Suppose that u' is unlabeled. Let N' be the network obtained from N by setting $label(u') = label(u)$ if u is not the partition node, and $label(u') = 1 - label(u)$ if u is the partition node. Then N is compatible and p is a partition node in N if and only if N' is compatible and p is a partition node in N'.*

Proof. Since u is a not a recombination node, u remains a child of u' in any induced tree from N. Therefore, if N is compatible, in any valid labeling of a compatible tree induced from N we have: $label(u') = label(u)$ in case u is the partition node, and $label(u') \neq label(u)$ (and hence $label(u') = 1 - label(u)$) in case u is not the partition node. □

According to Proposition 4.9, we can always label a node u' having an edge (u', u) to a labeled non-recombination node u (provided that a partition node is given).

The proof of the following proposition can be easily verified by the interested reader.

Proposition 4.10. *Let u be a node in N and let (u', u) be an incoming edge to u. Suppose that $label(u) = label(u')$. Suppose further that u is a recombination node and let (u'', u) be another incoming edge to u. Let N' be the network resulting from N by removing the edge (u'', u). Then N is compatible with a partition node p if and only if N' is compatible with a partition node p.*

From Proposition 4.10, if a node u in N has the same label as another node u' where (u', u) is an edge in N, then we can remove all the other incoming edges to u.

Proposition 4.11. *Let w be a node in N such that all its children are leaves labeled with the same label. Let N' be the network obtained from N by:* (1) *replacing w and its children with a leaf w' labeled with the same label as the children of w,* (2) *making every incoming edge to w an incoming edge to w', and* (3) *making every incoming edge to a child of w an incoming edge to w'. Then N is compatible if and only if N' is. Moreover, if p is a partition node of N then p is also a partition node of N', unless p is either w or one of its children, and in which case w' is a partition node in N'.*

Proof. If N' is compatible, then since w' is labeled 0, by replacing w' with w and its children and labeling w 0, we obtain a compatible induced tree from N. Conversely, suppose that N is compatible and consider an induced compatible tree from N. If we remove w and all its leaf-children from T except one leaf-child that we rename w', the resulting tree is an induced compatible tree from N'. The statement about the partition node can be easily verified by the reader. □

The above proposition describes an operation that applies to a node w whose children are leaves labeled with the same label. The operation collapses the node together with its children to a single leaf w'. In particular, if a node w has a single child which is a leaf, then this operation replaces w and its leaf-child with a single leaf w'. Note that the weight of the new node w' should be set to the the weight of w plus the sum of the weights of all the children of w.

We describe below a procedure that simplifies the network according to the operations and simplifications described above. The procedure **Simplify** is given in Fig. 4.

Proposition 4.12. *Let N be a network with a given partition node p. If the procedure* **Simplify** *decides the instance N, then its decision is correct, and if it applies an operation to N to obtain a network N' with a partition node p', then N is compatible with p as a partition node if and only if N' is compatible with p' as a partition node.*

Proof. The correctness of the decision made in step 2 of **Simplify** follows from Proposition 4.5. The correctness of the decision made in step 3 follows from the fact that N has at least two leaves of each label at this point (by step 2), and that leaves have no outgoing edges. The correctness of the decisions made in steps 4 and 5 of **Simplify** follow from Propositions 4.6 and 4.7, respectively.

The correctness of the operation performed in step 1 is easy to see. The correctness of the operations performed in steps 6–9 follows from Propositions 4.8–4.11, respectively. □

Lemma 4.13. *Let N be a network with a partition node p and suppose that the procedure* **Simplify** *if applied to N does not decide N nor does it perform any operation to N. Then there exists a node $w \in N$ satisfying the following properties:* (1) *w has at least two children and all the children of w are leaves;* (2) *there are at least two children of w with different labels;* (3) *w is unlabeled; and* (4) *every child of w is a recombination node.*

Simplify(N)

1. **while** there is a node $u \in N$ of out-degree 0 **do** remove u from N;
2. **if** there is only one leaf in N with label 0 (or 1) **then return** (TRUE);
3. **if** the partition node p is a leaf **then return** (FALSE);
4. **if** there exists a node $u \in N$ that has two children with different labels that are non-recombination nodes and such that none of them is a partition node **then return** (FALSE);
5. **if** there exists a labeled node $u \in N$ that has a non-recombination child v such that $label(v) \neq label(u)$ and v is not a partition node **then return** (FALSE);
6. **if** there exists a recombination node $u \in N$ and an edge $(u', u) \in N$ such that $label(u') \neq label(u)$ **then** remove (u', u) and decrease $wt(u)$ by 1;
7. **if** u' is unlabeled and has a labeled child u such that u is a non-recombination node **then**
   if u is designated as the partition node **then** set $label(u') = 1 - label(u)$;
   else set $label(u') = label(u)$;
8. **if** there exists a recombination node $u \in N$ and an edge $(u', u) \in N$ such that $label(u') = label(u)$ **then for** every edge $(u'', u) \in N$ where $u'' \neq u$ **do** remove (u'', u) and decrease $wt(u)$ by 1;
9. **if** there exists a node $w \in N$ such that all the children of w are leaves labeled with the same label **then** {$*$ note that w at this point must be unlabeled or labeled as its children $*$}
   9.1. remove w and its children and replace them with a leaf w';
   9.2. label w' with the same label as the children of w;
   9.3. make all incoming edges to w and its children incoming edges to w' and set $wt(w')$ to be $wt(w)$ plus the sum of the weights of the children of w;
   9.4. **if** w is designated as the partition node **then** designate w' as the partition node in the resulting network;

Fig. 4. The procedure **Simplify**.

ISPN-Solver (N, k)
{$*$ k is the total weight of all the nodes in N $*$}
1. **if** $k = 0$ and N is not compatible **then** reject;
2. **while** the procedure **Simplify** is applicable to N **do** apply it;
3. let w be a node satisfying the statement of Lemma 4.13; branch as follows:
   first side of the branch: set $label(w) = 1$;
   second side of the branch: set $label(w) = 0$;

Fig. 5. The algorithm **ISPN-Solver**.

Proof. Let ℓ be a leaf in N such that the root-leaf path P to ℓ has maximum length. Note that ℓ must exist by step 1 of **Simplify** (every path starting at the root of N must lead to a leaf). Let w be the parent of ℓ on the path P. By the maximality of P, all the children of w must be leaves. If all the children of w are labeled with the same label, then step 9 of **Simplify** would apply to w. This shows that w has at least two children labeled with different labels, and properties (1) and (2) about w have been established.

Suppose, to get a contradiction, that w is labeled. Let u be a child of w such that $label(u) \neq label(w)$.

By step 6 of **Simplify**, u must be a non-recombination node otherwise the edge (w, u) would be removed. By step 5 of **Simplify**, v must be a partition node. But then by step 3 of **Simplify** the procedure would have rejected the instance, contradicting the statement of the lemma. It follows that w is unlabeled establishing property (3) about w.

Finally, if w had a child that is a non-recombination node, then by step 7 of **Simplify**, w would have been labeled contradicting property (3) shown above. This establishes property (4) about w and completes the proof. □

The Algorithm

The algorithm **ISPN-Solver** is given in Fig. 5. The algorithm implicitly assumes that the partition node p is given. This assumption can be removed by trying every node in N as the partition node, then calling the algorithm with that node as the partition node. This will increase the running time of the algorithm by an $O(n)$ factor. If the algorithm **ISPN-Solver** returns TRUE on any of these calls then N must be compatible. To keep the presentation of the algorithm concise, we will not enumerate the partition nodes, but we will compensate for that by multiplying the running time of the algorithm by a linear factor at the end.

The algorithm **ISPN-Solver** is a branch-and-search process. Each stage of the algorithm starts with an instance (N, k) of the problem, where k is the total weight of all the nodes in N, and then tries to reduce k either by branching or by simplifying the network. Then the algorithm recursively works on the reduced instances. We implicitly assume that after each step, the network N and the parameter k are updated accordingly.

Theorem 4.14. *The algorithm* **ISPN-Solver** *correctly decides in time* $O(2^{k/2}n)$ *whether a phylogenetic network with a given partition node is compatible or not.*

Proof. Since we are going to try every node as the partition node, it suffices to show that the algorithm **ISPN-Solver**, which works under the assumption that the partition node is given, makes the correct decision.

Step 1 of the subroutine is correct because if $k = 0$ then N must be a phylogenetic tree, and the compatibility of N can be checked in linear time [27]. The correctness of step 2 follows from Proposition 4.12. Since Step 3 tries every possible label for w (there are only two possible labels for w), the correctness of the algorithm follows.

To analyze the running time of the algorithm **ISPN-Solver**, notice that the algorithm is a branch-and-bound process and its execution can be depicted by a search tree. The running time of the algorithm is proportional to the number of root-to-leaf paths, or equivalently the number of leaves in the search tree, multiplied by the time spent along each such path. Therefore, the main step in the analysis of the algorithm is deriving an upper bound on the number of leaves in the search tree. Let $\mathcal{T}$ be the search tree for the algorithm **ISPN-Solver** on an input instance (N, k), and let $T(k)$ be the number of leaves in $\mathcal{T}$. Let w be a node that the algorithm **ISPN-Solver** branches on in step 3.

Since all the children of w are leaves, the children of w are all labeled. Since all the children of w are recombination nodes by property (3) of Lemma 4.13, when the algorithm labels w in each of the two branches, at least one incoming edge to each child of w having the same label as w will be removed by step 8 of **Simplify** when applied next to the network. On the other hand, an incoming edge to every child of w whose label is different from w will be removed by step 6 of **Simplify**. Therefore, for every child of w, the weight of the child will be decreased by at least 1 in the next call to **Simplify**. Since w has at least two children by property (2) of Lemma 4.13, the total weight k of all the nodes in N is reduced by at least 2 in every side of the branch. It follows that the number of leaves $T(k)$ of the search tree $\mathcal{T}$ satisfies the recurrence relation $T(k) \leq 2T(k-2)$, and $T(k) = O(2^{k/2})$.

Now consider a root-leaf path in the search tree $\mathcal{T}$. On every node of this path the algorithm might need to call the procedure **Simplify**, which could take $O(n)$ time since the size of N is $O(n)$. However, this need not be the case with a careful implementation of this procedure. Instead of calling this procedure at each node of N, we only call it on the nodes on which the operation is applicable. The time spent by the procedure in each such call is proportional to the number of nodes/edges removed plus the number of nodes labeled in the call. Since we can only have $O(n)$ nodes/edges, the total time spent by the procedure on a root-leaf path of $\mathcal{T}$ is proportional to the size of the network, which is $O(n)$. It follows that the running time of the algorithm is $O(2^{k/2}n)$. □

Corollary 4.15. *The* ISPN *problem can be solved in time* $O(2^{k/2}n^2)$*, where* n *is the number of nodes and* k *is the number of recombination nodes, respectively, in the phylogenetic network.*

5. The cluster containment problem

Let T be phylogenetic tree on set $\mathcal{X}$ of taxa and rooted at node r. Each edge $e = (u, v)$ induces a cluster c_e of taxa, which is the set of leaves reachable from root r only through v. It is easy to see that the leaves in c_e are exactly the leaves of the subtree rooted at v. A cluster c_e is contained in a network N if it is a cluster in a tree induced from N.

We can easily determine if a cluster c is in a tree by finding the least common ancestor $lca(c)$ of leaves in c, and then comparing the leaf set under $lca(c)$ and c. The CC problem is hard because there are many different trees that can be induced from the network N. We will prove that CC is NP-hard by reducing the problem ISPN to it.

Theorem 5.1. *The problem* CC *is NP-hard.*

Proof. The proof is by a Turing reduction from the ISPN problem. Let $\langle N, \alpha\rangle$ be an instance of the ISPN problem. Assume the Cluster Containment problem is in P, and algorithm M solves it in polynomial time. We generate a program M' that solves the problem ISPN in polynomial time as follows:

- From $\langle N, \alpha\rangle$, generate two instances $\langle N, x\rangle$ and $\langle N, \bar{x}\rangle$, where $x = \{y \in L : \alpha(y) = 1\}$ and $\bar{x} = L - x$.
- M' outputs $M(\langle N, x\rangle) \vee M(\langle N, \bar{x}\rangle)$.

It is easy to see that the construction is computable in polynomial time. We now show the correctness of the reduction.

Assume $\langle N, \alpha\rangle$ is a yes-instance of the ISPN problem (i.e., character α is compatible on network N). If $\alpha(r) = 0$, where r is the root, then x is a cluster of N, and therefore $M(\langle N, x\rangle) \vee M(\langle N, \bar{x}\rangle)$ is true. The case where $\alpha(r) = 1$ is similar.

Now, assume that $\langle N, \alpha\rangle$ is a no-instance of the ISPN problem (i.e., character α is not compatible on network N). Then, α is not compatible on any tree T, for all $T \in \mathcal{T}(N)$. Then, for any tree T, and any labeling of T, any subtree t of T that contains all the leaves labeled with 0 must contain at least one leaf labeled with 1 (similarly, any subtree t of T that contains all the leaves labeled with 1, must contain at least one leaf labeled with 0). Hence, neither x nor $\bar{x}$ is a cluster of N. Therefore, $M(\langle N, x\rangle) \vee M(\langle N, \bar{x}\rangle)$ is false. □

Theorem 5.2. *The problems* CC *and* ISPN *are polynomially equivalent.*

Proof. The proof for Theorem 5.1 shows that we can polynomially reduce ISPN to CC. We need to show the reverse direction, that is, CC can be polynomially reduced to ISPN. Let $\langle N, c\rangle$ be an instance of the problem of cluster containment. We build an instance $\langle N', \alpha\rangle$ of ISPN as follows.

- Create a new node r' that will be the new root for N'.
- Connect r' to $r(N)$, and create a new leaf X and also connect r' to it.
- Label leaves in c as 1 while label all other leaves, including leaf X, as 0.

The construction is clearly done in linear time in the size of N.

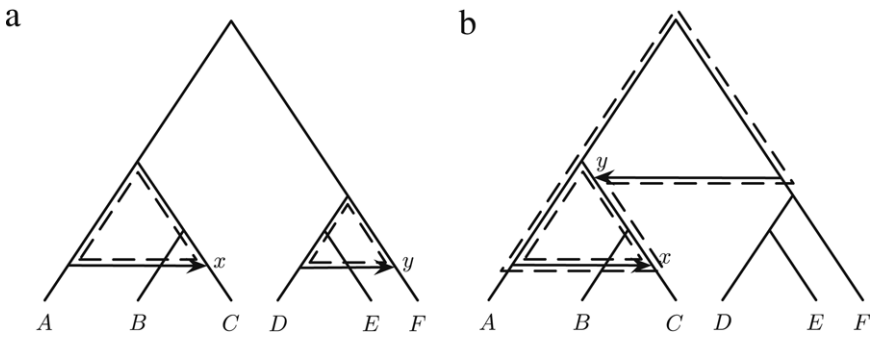

Fig. 6. The network in (a) is an example of a galled network. It has two reticulation nodes x and y and their associated galls are illustrated with dashed triangles. Note that these dashed triangles do not intersect with each other. The network in (b) is a general network because the reticulation node y is on recombination cycles associated with x, which are shown as dashed polygons.

Assume that c is a cluster in N. There must be a tree T induced from network N and an internal edge $e = (u, v)$ such that $c = c_e$. We label all internal nodes in the subtree rooted at v, including v, as 1. All other internal nodes in N' are labeled 0. By labeling in this way, all nodes assigned value 1 form a connected component and all nodes assigned value 0 form another connected component. Therefore, α is compatible on N'.

Now assume that c is not a cluster in N. Let $\bar{c}$ be the set of leaves in network N that are not in c. There are two cases: $\bar{c}$ is also not a cluster in N, and $\bar{c}$ *is* a cluster in N. If both c and $\bar{c}$ are not clusters in N, then, by an argument similar to that in proof of Theorem 5.1, α is not compatible on N. In the other case, the root of N must be labeled 1 as leaves in c are in order to make α compatible. However, leaves in $\bar{c}$ and X are labeled 0, and they are separated by this node. Therefore, α cannot be compatible on N'.

This shows that CC is reducible to ISPN in linear time. Note also that the number of recombination nodes in both instances is the same. □

Corollary 5.3. *The* CC *problem when parameterized by the number of recombination nodes k in the network is solvable in time $O(2^{k/2}n^2)$, where n is the number of nodes in the network.*

6. Galled networks

In this section, we address a biologically-motivated restricted class of phylogenetic networks, called *gt-networks*, proposed by Wang et al. [37] and Gusfield et al. [10]. We adopt the definitions of [33].

Definition 6.1. Let N be a phylogenetic network in which the in-degree of every node is at most 2, and let w be a node that has two directed paths out of it that meet at a node x of in-degree 2. These two directed paths together define a **recombination cycle** Q. Node w is called the **coalescent node** of Q, and x is the **recombination node** of Q.

Definition 6.2. A recombination cycle in a phylogenetic network that shares no nodes with any other recombination cycle is called a **gall.**

Definition 6.3. We denote by Q_x^w a gall whose coalescent node is w and whose recombination node is x. We denote by $E(Q_x^w)$ the set of all edges on gall Q; formally, $E(Q_x^w) = \{e : e \text{ is an edge on a directed path from } w \text{ to } x\}$.

When the context is clear, we simply write Q for a gall, without explicitly naming the coalescent and recombination nodes.

Definition 6.4. A phylogenetic network N is called a **galled network** if every reticulation cycle is a gall.

Fig. 6 shows examples of a galled and a general network. In the general network in Fig. 6(b) recombination node x is associated with two reticulation cycles. Observe that based on the definition of galled networks, each recombination node is associated with exactly one gall, and that all nodes on the cycle defined by a gall have in-degree 1, except for the recombination node, which has in-degree 2.

We recall that a set of taxa is a cluster of N if it is a cluster of at least one induced tree of N, and N induces a tree by retaining one reticulation edge (and deleting the other one) for each reticulation node.

We now prove that the number of clusters in a galled network is linear in the number of leaves.

It is straightforward to establish that any edge that does not lie on a gall in a galled network contributes only a single cluster to the overall set of clusters of the network, which we formalize as follows.

Lemma 6.5. *Let $e = (u, v)$ be an edge such that $e \notin E(Q_x^w)$ for any gall Q in a galled network N. Further, let N_1 and N_2 be the two subnetworks obtained from deleting edge e in N, where N_1 is the subnetwork that contains node u, and we attached a leaf α to it, and N_2 is the subnetwork rooted at node v. Then,*

$$|\mathcal{C}(N)| = |\mathcal{C}(N_1)| + |\mathcal{C}(N_2)|.$$

Further, we now show that the number of clusters contributed by the set of edges in a gall is linear in the number of edges in that gall.

Lemma 6.6. *Let Q_x^w be a gall, with p_l and p_r the two parents of x, in galled network N. Further, denote by E_l the set of edges whose both endpoints lie on the path from w to p_l, and by E_r the set of edges whose both endpoints lie on the path from w to p_r. Let T_l and T_r be the two trees induced from N that differ only in the edges incident into x; that is, T_l has edge (p_l, x), while T_r has edge (p_r, x). Then,*

$$|\mathcal{C}(T_l)\triangle\mathcal{C}(T_r)| = 2(|E_l| + |E_r|),$$

where $A\triangle B$ is the symmetric difference of two sets A and B.

Proof. Let S be the subnetwork of N rooted at node x. Notice that S is also a galled network. Then, T_l' and T_r', the two trees obtained from T_l and T_r by restricting them to leaves in $L(N) - L(S)$, are isomorphic, and hence $\mathcal{C}(T_l') = \mathcal{C}(T_r')$. Notice that T_l' contains all edges in E_l, and T_r' contains all edges in E_r. Let C_l and C_r be the sets of clusters of T_l' and T_r' induced by the edges in E_l and E_r, respectively. Then, we have

- $\mathcal{C}(T_l) = \mathcal{C}(T_l') - C_l \cup \{c \cup L(X) : c \in C_l\}$, and
- $\mathcal{C}(T_r) = \mathcal{C}(T_r') - C_r \cup \{c \cup L(X) : c \in C_r\}$.

Therefore,

$$|\mathcal{C}(T_l)\triangle\mathcal{C}(T_r)| = 2|C_l| + 2|C_r| = 2(|E_l| + |E_r|). \quad \square$$

The combination of Lemmas 6.5 and 6.6 has two important consequences. First, a cluster induced by an edge e that is not in *any* gall is the same in all induced trees. Second, breaking a gall in two different ways only affects clusters induced by edges in that gall, not clusters induced by the other edges.

Lemma 6.7. *In any galled network with n leaves, the number of reticulation nodes is at most $n - 1$, and hence the number of galls in N is at most $n - 1$.*

Proof. Consider any phylogenetic tree T with n leaves. The number of internal nodes in T is at most $n - 1$ (which only happens when T is binary). If we add a new reticulation edge $(p(x), x)$ to T in order to buid a network N, then Q_x^w has at least one internal node of T. Galls in N cannot share any internal node in T. We also note that new nodes $p(x)$ and x cannot be shared with other galls in N. Therefore, we can have at most $n - 1$ galls in N. $\square$

Lemma 6.8. *Let $N = (V, E)$ be a galled network with n leaves. Then, $|\mathcal{C}(N)| = O(n)$.*

Proof. Let R be the set of reticulation nodes of N, and let $\bar{E} = E \setminus \bigcup_{x\in R} E(Q_x^w)$. By Lemma 6.5, clusters induced by edges in $\bar{E}$ are the same in all induced trees. Further, by Lemma 6.6, breaking a gall Q_x^w in two different ways creates at most $2|E(Q_x^w)|$ different clusters, regardless of any combination of breaking all other galls. Therefore, the number of different clusters induced by N is at most $|\bar{E}| + 2\sum_{x\in R}|E(Q_x^w)| = |E| + \sum_{x\in R}|E(Q_x^w)|$.

Any two galls in N do not share any nodes, so $\sum_{x\in R}|E(Q_x^w)|$ is bounded by $|E|$. From Lemma 6.7, the number of edges in a galled network is at most $(2n - 2) + (n - 1) = 3n - 3$. Therefore, the number of clusters induced by a galled network is at most $6n - 6$, which completes the proof. $\square$

Given this series of results, we now show that the Cluster Containment Problem is in P for galled networks.

Theorem 6.9. *The Cluster Containment Problem is solvable in polynomial time on galled networks.*

Proof. Given a network N with n leaves, every cluster induced by N has at most n leaves, and thus checking whether two clusters are equal takes polynomial time. The number of clusters induced by N, as shown in Lemma 6.8, is $O(n)$. Therefore, checking whether a given cluster X is equal to one of clusters induced by N takes polynomial time. $\square$

Given the polynomial equivalence of the CC and ISPN problems, and the fact the reduction in the proof of Theorem 5.2 does not modify the network, we obtain the following corollary.

Corollary 6.10. *The ISPN problem is solvable in polynomial time on galled networks.*

Acknowledgements

The first author was supported in part by a DePaul University Competitive Research Grant and the fourth author was supported in part by a Lafayette College Research Grant.

References

[1] S-A Bacanu, B. Devlin, K. Roeder, Association studies for quantitative traits in structured populations, Genetic Epidemiology 22 (2002) 78–93.
[2] R.G. Beiko, N. Hamilton, Phylogenetic identification of lateral genetic transfer events, BMC Evolutionary Biology 6 (2006).
[3] U. Bergthorsson, K.L. Adams, B. Thomason, J.D. Palmer, Widespread horizontal transfer of mitochondrial genes in flowering plants, Nature 424 (2003) 197–201.
[4] U. Bergthorsson, A. Richardson, G.J. Young, L. Goertzen, J.D. Palmer, Massive horizontal transfer of mitochondrial genes from diverse land plant donors to basal angiosperm Amborella, Proceedings of the National Academy of Sciences of United States of America 101 (2004) 17747–17752.
[5] D. Bryant, V. Moulton, NeighborNet: An agglomerative method for the construction of planar phylogenetic networks, in: R. Guigo, D. Gusfield (Eds.), Proc. 2nd Workshop Algorithms in Bioinformatics (WABI'02), in: Lecture Notes in Computer Science, vol. 2452, Springer Verlag, 2002, pp. 375–391.
[6] W.F. Doolittle, Y. Boucher, C.L. Nesbo, C.J. Douady, J.O. Andersson, A.J. Roger, How big is the iceberg of which organellar genes in nuclear genomes are but the tip?, Philosophical Transactions of the Royal Society London, Series B, Biological Science 358 (2003) 39–57.
[7] R. Downey, M. Fellows, Parameterized Complexity, Springer, New York, 1999.
[8] P. Gorecki, Reconciliation problems for duplication, loss and horizontal gene transfer. in: Proc. 8th Ann. Int'l Conf. Comput. Mol. Biol. (RECOMB04), 2004, pp. 316–325.
[9] R.C. Griffiths, P. Marjoram, Ancestral inference from samples of DNA sequences with recombination, Journal of Computational Biology 3 (1996) 479–502.
[10] D. Gusfield, S. Eddhu, C. Langley, Efficient reconstruction of phylogenetic networks with constrained recombination. in: Proceedings of Computational Systems Bioinformatics, CSB 03, 2003.
[11] M.T. Hallett, J. Lagergren, Efficient algorithms for lateral gene transfer problems, in: Proc. 5th Ann. Int'l Conf. Comput. Mol. Biol. (RECOMB01), ACM Press, New York, 2001, pp. 149–156.
[12] D.H. Huson, SplitsTree: A program for analyzing and visualizing evolutionary data, Bioinformatics 14 (1) (1998) 68–73.
[13] G. Jin, L. Nakhleh, S. Snir, T. Tuller, Efficient parsimony-based methods for phylogenetic network reconstruction, in: Proceedings of the European Conference on Computational Biology, ECCB 06, Bioinformatics 23 (2006) e123–e128.
[14] G. Jin, L. Nakhleh, S. Snir, T. Tuller, Maximum likelihood of phylogenetic networks, Bioinformatics 22 (21) (2006) 2604–2611.
[15] G. Jin, L. Nakhleh, S. Snir, T. Tuller, Inferring phylogenetic networks by the maximum parsimony criterion: A case study, Molecular Biology and Evolution 24 (1) (2007) 324–337.
[16] G. Jin, L. Nakhleh, S. Snir, T. Tuller, A new linear-time heuristic algorithm for computing the parsimony score of phylogenetic networks: Theoretical bounds and empirical performance, in: I. Mandoiu, A. Zelikovsky (Eds.), Proceedings of the International Symposium on Bioinformatics Research and Applications, 2007, pp. 61–72.
[17] I. Kanj, L. Nakhleh, G. Xia, Reconstructing evolution of natural languages: Complexity and parameterized algorithms, in: 12th Annual International Computing and Combinatorics Conference, in: Lecture Notes in Computer Science, vol. 4112, Springer, 2006, pp. 299–308.
[18] C.R. Linder, B.M.E. Moret, L. Nakhleh, T. Warnow, Network (reticulate) evolution: biology, models, and algorithms. in: The Ninth Pacific Symposium on Biocomputing, PSB, 2004. A tutorial.
[19] C.R. Linder, L.H. Rieseberg, Reconstructing patterns of reticulate evolution in plants, American Journal of Botany 91 (2004) 1700–1708.
[20] D. MacLeod, R.L. Charlebois, F. Doolittle, E. Bapteste, Deduction of probable events of lateral gene transfer through comparison of phylogenetic trees by recursive consolidation and rearrangement, BMC Evolutionary Biology 5 (2005).
[21] W.P. Maddison, Gene trees in species trees, Systematic Biology 46 (3) (1997) 523–536.
[22] V. Makarenkov, T-REX: Reconstructing and visualizing phylogenetic trees and reticulation networks, Bioinformatics 17 (7) (2001) 664–668.
[23] V. Makarenkov, D. Kevorkov, P. Legendre, Phylogenetic network reconstruction approaches, Genes, Genomics, and Bioinformatics 6 (2005).
[24] J. Marchini, P. Donnelly, L.R. Cardon, Genome-wide strategies for detecting multi loci that influence complex diseases, Nature Genetics 37 (4) (2005) 413–417.
[25] J.P. Mower, S. Stefanovic, G.J. Young, J.D. Palmer, Gene transfer from parasitic to host plants, Nature 432 (2004) 165–166.
[26] Y. Nakamura, T. Itoh, H. Matsuda, T. Gojobori, Biased biological functions of horizontally transferred genes in prokaryotic genomes, Nature Genetics 36 (7) (2004) 760–766.
[27] L. Nakhleh, Phylogenetic Networks. PhD Thesis, The University of Texas at Austin, 2004.
[28] L. Nakhleh, G. Jin, F. Zhao, J. Mellor-Crummey, Reconstructing phylogenetic networks using maximum parsimony. in: Proceedings of the 2005 IEEE Computational Systems Bioinformatics Conference, CSB2005, 2005, pp. 93–102.
[29] L. Nakhleh, D. Ringe, T. Warnow, Perfect phylogenetic networks: A new methodology for reconstructing the evolutionary history of natural languages. LANGUAGE, Journal of the Linguistic Society of America 81 (2) (2005) 382–420.
[30] L. Nakhleh, D. Ruths, L.S. Wang, RIATA-HGT: A fast and accurate heuristic for reconstrucing horizontal gene transfer. In: L. Wang (ed) Proceedings of the Eleventh International Computing and Combinatorics Conference (COCOON 05), 2005. pp. 84–93, LNCS #3595.
[31] L. Nakhleh, L.S. Wang, Phylogenetic networks: Properties and relationship to trees and clusters, LNCS Transactions on Computational Systems Biology II (2005) 82–99. LNBI #3680.
[32] L. Nakhleh, L.S. Wang, Phylogenetic networks, trees, and clusters. in: Proceedings of the 2005 International Workshop on Bioinformatics Research and Applications, IWBRA 05, 2005. pp. 919–926, LNCS #3515.
[33] L. Nakhleh, T. Warnow, C.R. Linder, Reconstructing reticulate evolution in species–theory and practice. In: Proc. 8th Ann. Int'l Conf. Comput. Mol. Biol. (RECOMB04), 2004, pp. 337–346.
[34] D. Posada, K.A. Crandall, The effect of recombination on the accuracy of phylogeny estimation, Journal of Molecular Evolution 54 (3) (2002) 396–402.
[35] D. Posada, K.A. Crandall, E.C. Holmes, Recombination in evolutionary genomics, Annual Review of Genetics 36 (2002) 75–97.
[36] A Schrijver, Combinatorial optimization: polyhedra and efficiency, vol. 24, Springer-Verlag, Berlin, 2004.
[37] L. Wang, K. Zhang, L. Zhang, Perfect phylogenetic networks with recombination, Journal of Computational Biology 8 (1) (2001) 69–78.