Scanning Phylogenetic Networks Is NP-hard

Vincent Berry[1], Celine Scornavacca[2], and Mathias Weller[3(✉)]

[1] LIRMM, Université de Montpellier, Montpellier, France
vberry@lirmm.fr
[2] CNRS, Université de Montpellier, Montpellier, France
celine.scornavacca@umontpellier.fr
[3] CNRS, LIGM, Université Paris Est, Marne-la-Vallée, France
mathias.weller@u-pem.fr

Abstract. Phylogenetic networks are rooted directed acyclic graphs used to depict the evolution of a set of species in the presence of reticulate events. Reconstructing these networks from molecular data is challenging and current algorithms fail to scale up to genome-wide data. In this paper, we introduce a new width measure intended to help design faster parameterized algorithms for this task. We study its relation with other width measures and problems in graph theory and finally prove that deciding it is NP-complete, even for very restricted classes of networks.

1 Introduction

Phylogenetic networks are rooted directed acyclic graphs used to depict the evolution of a set of species in the presence of reticulate events such as hybridizations, where two species combine their genetic material to create a new species (see nodes H_1 and H_2 in Fig. 1(left)) [9]. Herein, leaves represent the studied species and the root their most recent common ancestor, from which time flows away (as indicated by the direction of the arcs). Internal vertices represent either speciation events (a single parent) or reticulation events (several parents). Each arc represents the evolution of a species in time, during which each gene in the species genome can change due to mutations, allowing different forms of a gene (*alleles*) to appear among species, and even among individuals within the same species. Though the species history is modeled by a network, the evolution of a single non-recombinant gene can always be depicted by a tree, see Fig. 1(center), embedded in the species network, see Fig. 1(right).

Usually, a species network is inferred from a DNA dataset $S = \{S_1, \ldots, S_L\}$ composed of L genes sequenced from the genome of one or several individuals for each studied species [16]. To find the best phylogenetic network explaining S, a possibility is to sample many different networks N and compute the probability $P(S|N)$ of each N given S. Without giving all details here (they can be found for instance in [16]), $P(S|N)$ can be computed from the individual probabilities $P(G_i|N)$ of gene trees $G_1, \ldots, G_L$ for the L loci given N. In turn, each $P(G_i|N)$ can be computed from the probabilities of all possible embeddings of G_i in N, weighted by their respective probability depending on S_i, i.e. $P(G_i|N, S_i)$.

A. Chatzigeorgiou et al. (Eds.): SOFSEM 2020, LNCS 12011, pp. 519–530, 2020.
https://doi.org/10.1007/978-3-030-38919-2_42

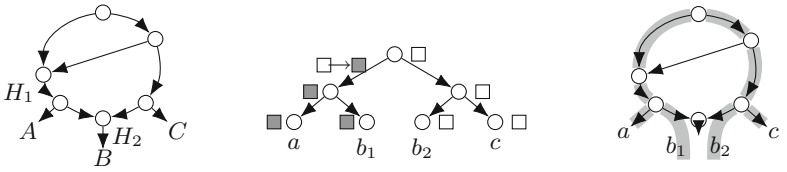


Fig. 1. Left: A phylogenetic network N depicting the evolutionary history of species A, B, and C. **Center:** An evolution scenario for a gene, given the sequences of one individual from species A, one from C and two from B, where different alleles (boxes) are observed: gray for A and for one individual from B, and white for the other individuals. The arc containing the mutation from the white to the gray allele is marked. **Right:** An embedding (gray arcs) in N of this gene evolution scenario.

See Fig. 1(center) for a gene tree and Fig. 1(right) for one of its possible embeddings within the network. Thus, heavy computations are needed to obtain $P(S|N)$ and current algorithms fail to scale to genome-wide data. To design faster algorithms, it is possible to integrate out the possible gene trees and embeddings, as done in [4]. To apply this technique to network inference we designed new partial likelihood formulae to compute $P(S|N)$ and stumbled on a new width parameter for DAGs that clearly puts into evidence why our approach is faster than existing ones, allowing us to handle several real-world datasets within minutes instead of weeks [13]. In this paper, we introduce this new parameter, which we call *scanwidth*, we study its relation with other parameters and problems in graph theory and finally prove that deciding it is NP-hard. A common and intuitive idea when working with phylogenetic networks is to exploit the observation that reticulation should be rare in practice to design algorithms that are fast for only mildly reticulate networks. This tree-likeness is often measured by the tree-width of the input. However, tree decompositions are in no way obligated to follow the leaf-to-root structure that phylogenies naturally impose and this makes dynamic programming on decomposition trees unnecessarily complicated. The scanwidth remedies this problem by forcing the leaves of the network to correspond to the leaves of the decomposition tree, yielding a form of tree-like cutwidth. Thus, our work broadens the arsenal of width measures that can be – and recently have been – used to attack hard problems in phylogenetics [5,8,12]. To get an intuition, imagine a (possibly red) scanner line traversing a network from the leaves to the root; at any moment, its *width* is the number of arcs it cuts. As the line moves up, it traverses nodes, changing the set of arcs it cuts and, hence its width. The cutwidth of the network is the largest width achieved by such a traversing line. Now, consider multiple independent scanner lines, each one scanning an arc incoming to a different leaf of the network. Whenever a node could be passed by two different lines, they are merged to form a single one. This naturally generalizes the cutwidth to a stronger (that is, smaller) width measure that we call *scanwidth*. As with the cutwidth, different orders in which the nodes are passed imply different values of the final width and the goal is to minimize it. In many optimization approaches for phylogenetic networks, a network is

traversed from the leaves up to its root, while computing some quantities. For some applications, computations on tree-parts can be done independently for each arc but, when meeting a reticulation node, computations on both arcs entering the node have to be considered jointly. This inter-dependence makes computing the required quantities more time consuming. In such cases, one really wants to process the network while minimizing the numbers of arcs considered jointly. This is captured by the scanwidth parameter.

In this work, we show that deciding the scanwidth of a network relates to an old problem in program optimization called Register Sufficiency (PO1 of Garey and Johnson [7]). Our proof comprises a non-trivial adaptation of an NP-hardness proof [14] for the latter problem to a very restricted class of rooted DAGs, on which Register Sufficiency coincides with deciding the cutwidth and the scanwidth (offset by 1). This hardness proof, as well as the scanwidth parameter itself, may be of independent interest to the design of algorithms for other problems on DAGs.

Note that computing the scanwidth and using it as a parameter for other algorithms are two different pairs of shoes and, though a parameterized algorithm may require a tree extension (see Sect. 2) to be given, there is still hope that the scanwidth can be approximated efficiently. Thus, in analogy with other highly successful (width) parameters such as the treewidth, the hybridization number or the hybridization level [2,3,11,15], we point out that being NP-complete to compute does not hurt the practical usefulness of the scanwidth.

We defer some proofs to a long version of this paper.

2 Preliminaries

Phylogenetic Networks. Let G be a leaf-labelled, directed, acyclic graph with a single source (which is called "root"). The in-degree of a vertex v in G is $\deg_G^-(v)$ and its out-degree is $\deg_G^+(v)$, the sum of those being the degree of v. If all vertices of G have either in-degree one and out-degree zero (*leaves*), in-degree at most one and out-degree at least two (*tree-vertices*), and in-degree at least two and out-degree one (*reticulation*), then G is called *rooted phylogenetic network* (henceforth *network*). Note that the root is a special tree-vertex. We denote the set of leaves of G by $\mathcal{L}(G)$, the set of vertices by $V(G)$ and the tree-vertices by $V_T(G)$. If the root has degree two, the internal vertices have degree three, and the leaves have degree one, then G is called *binary*. If G contains a u-v-path for vertices u and v, we say that u is an ancestor of v (and v is a descendant of u) and we write $v <_G u$.

Vertex Orderings. A linear ordering σ of a subset V' of the vertices of a network G is called *G-respecting* if $u <_G v \Rightarrow u <_\sigma v$ for all $u, v \in V'$. A G-respecting ordering σ over $V(G)$ is called an *extension* (or "reverse topological order") of G, see Fig. 2. We call a tree Γ on $V(G)$ a *tree extension* for G if $x <_G y \Rightarrow x <_\Gamma y$ for all $x, y \in V(G)$. We denote the vertex at position i in σ by $\sigma(i)$ and $\sigma^{-1}(u)$ returns the position of the vertex u in sigma. Since positions and vertices are in bijection, we sometimes use vertices to represent their positions. A position i of

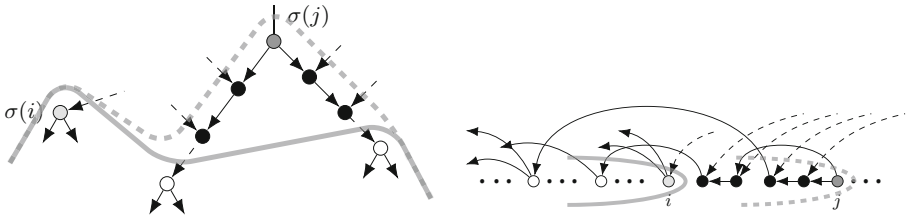


Fig. 2. Left: For an extension σ each position i induces a cut through G separating the vertices in $\sigma[1..i]$ (below gray line) from $\sigma[i+1..]$ (above gray line). **Right:** G with vertices linearly arranged according to σ.

σ is called a *milestone* if $\sigma(i)$ is a tree-vertex and σ is called *stable* if all maxima (wrt. $\leq_G$) in $\sigma[1..i]$ for any milestone i are tree-vertices (that is, each reticulation in $\sigma[1..i]$ has a parent in $\sigma[1..i]$). We denote the sub-order of σ restricted to the elements of a set U by $\sigma[U]$ and we abbreviate $\sigma[\{\sigma(i), \sigma(i+1), \ldots, \sigma(j)\}] =: \sigma[i..j]$. For disjoint orders σ and π let $\sigma \circ \pi$ denote the concatenation of σ with π (that is, σ followed by π). For a set X, we let (X) denote any order on the elements of X. Further, for distinct vertices or disjoint vertex sets $X_1, X_2, \ldots$, we abbreviate $(X_1) \circ (X_2) \circ \ldots =: (X_1, X_2, \ldots)$.

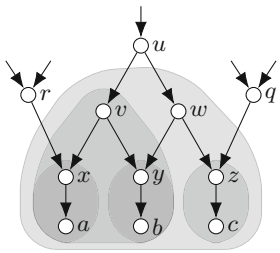
(Directed) Cutwidth. For an extension σ of a DAG G and a position i, we will use $\mathrm{C}_i(\sigma)$ to denote the set of arcs from a vertex in $\sigma[i+1..]$ to a vertex in $\sigma[1..i]$ and $\mathrm{cw}_i(\sigma) := |\mathrm{C}_i(\sigma)|$ is called the *cutwidth* of σ at position i. The cutwidth of σ is $\mathrm{cw}(\sigma) := \max_i \mathrm{cw}_i(\sigma)$ and the cutwidth of G, denoted $\mathrm{cw}(G)$, is the minimum of $\mathrm{cw}(\sigma)$ over all extensions σ of G. We allow i to be a vertex instead of a position, as σ is a bijection between the two.

(Directed) Register width. For an extension σ of G and a position i, we will use $\mathrm{RW}_i(\sigma)$ to denote the set of vertices in $\sigma[1..i]$ that have a parent in $\sigma[i+1..]$ and $\mathrm{rw}_i(\sigma) := |\mathrm{RW}_i(\sigma)|$ is called the *register width* (also known as "vertex cut" or "separation" [6]) of σ at position i (again, we allow i to be a vertex instead of a position, as σ is a bijection between the two). The register width of σ is $\mathrm{rw}(\sigma) := \max_i \mathrm{rw}_i(\sigma)$ and the register width of G, denoted $\mathrm{rw}(G)$, is the minimum over all extensions σ for G of $\mathrm{rw}(\sigma)$.

Theorem 1. *For all binary networks G, we have* $\mathrm{cw}(G) = \mathrm{rw}(G) + 1$.

Scanwidth. Let σ be an extension for G and let $i \in \mathbb{N}$. We define $\mathrm{SW}_i(\sigma)$ as the set of all arcs $uv \in \mathrm{C}_i(\sigma)$ for which v and $\sigma(i)$ are weakly connected in $G[\sigma[1..i]]$ (see Fig. 3(left)). $\mathrm{sw}_i(\sigma)$ is defined as $|\mathrm{SW}_i(\sigma)|$, while the scanwidth of σ is $\mathrm{sw}(\sigma) := \max_i \mathrm{sw}_i(\sigma)$ and the scanwidth of G, denoted by $\mathrm{sw}(G)$, is the minimum of $\mathrm{sw}(\sigma)$ over all extensions σ for G. Again, in our notations we allow i to be a vertex instead of a position, as σ is a bijection between the two.

Alternatively, $\mathrm{sw}(G)$ can be defined as follows. For a tree extension Γ for G, we define $\mathrm{GW}_v(\Gamma)$ as the set of arcs $(x, y) \in E(G)$ with $x >_\Gamma v \geq_\Gamma y$. Further,



$\sigma(i)$	$\mathrm{SW}_i(\sigma) = \mathrm{GW}_{\sigma(i)}(\Gamma)$
a	$\{xa\}$
b	$\{yb\}$
c	$\{zc\}$
x	$\{rx, vx\}$
y	$\{vy, wy\}$
z	$\{wz, qz\}$
v	$\{rx, uv, wy\}$
w	$\{rx, uv, uw, qz\}$
...	...

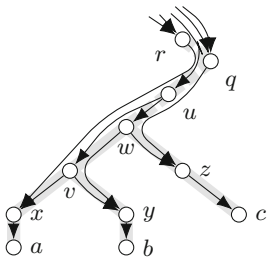


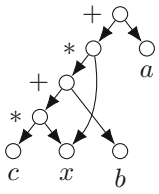
Fig. 3. Illustration of the two definitions of scanwidth. **Left:** Lower part of a graph G where gray zones represent the weakly connected components induced by $\sigma[1..i]$ for $\sigma = (a, b, c, x, y, z, v, w, \ldots)$ and $i \leq 8$. Here, $\mathrm{SW}_v(\sigma) = \{rx, uv, wy\}$ since x, y, and v are weakly connected in $G[a, b, c, x, y, z, v]$. **Middle:** table indicating $\mathrm{SW}_i(\sigma)$ for $i \leq 8$ corresponding to σ. **Right:** part of a tree Γ with $\mathrm{GW}_{\sigma(i)}(\Gamma) = \mathrm{SW}_i(\sigma)$ for all $i \leq 8$. For the extension $\pi = (c, a, z, b, y, x, v, w, \ldots)$, we also have $\mathrm{GW}_{\pi(i)}(\Gamma) = \mathrm{SW}_i(\pi)$ for all $i \leq 8$.

we let $\gamma\mathrm{w}(\Gamma) := \max_v |\mathrm{GW}_v(\Gamma)|$ and $\gamma\mathrm{w}(G) := \min_\Gamma \gamma\mathrm{w}(\Gamma)$. Although a tree extension is defined independently of a (full) extension for G, there is a link between the two notions. Indeed, the sets $\mathrm{GW}_v(\Gamma)$ in an optimal tree extension correspond to the sets $\mathrm{SW}_v(\sigma)$ in one or several optimal extensions σ (see Fig. 3).

Proposition 1. *For any network G, (a) $\gamma\mathrm{w}(G) = \mathrm{sw}(G)$. Further, (b) if G has only one leaf, then $\mathrm{sw}(G) = \mathrm{cw}(G)$. (c) If G is also binary, then $\mathrm{sw}(G) = \mathrm{rw}(G) + 1$.*

Observe that the scanwidth differs largely from the directed path-width [1], which is always zero for DAGs. To relate the scanwidth to established parameters, let us mention that the scanwidth of any level-k network cannot exceed $k + 1$ but it might even be constant. Regarding width-measures, the scanwidth is bounded by the cutwidth from below and the treewidth (of the underlying undirected graph) from above.

3 NP-completeness



To compute the value of a given algebraic expression such as $(cx+b)x+a$ using a computer, we need to store the values of a, b, c, and x in registers which can then be processed by the CPU. As registers can be overwritten, expressions involving more variables than the number of available registers can be evaluated. The problem of deciding whether a given expression can be evaluated on a CPU with k registers (without recomputing sub-expressions or relying on the costly *spilling* technique) is called Register Sufficiency. We suppose that the input expression is given as a **rooted** DAG of necessary

computations. For example, to compute $(cx+b)x+a$, we need to compute $cx+b$, for which we need to compute cx (see figure on the right).

Register Sufficiency [PO1 in [7]] (RS)
Input: a rooted DAG G of an expression to be computed, $k \in \mathbb{N}$
Question: Can G be computed using at most k registers?

Register Sufficiency can be interpreted as a game played on G, where the player has k stones that have to be placed progressively on *all* vertices, using the following operations [14]:

1. remove a stone from any vertex
2. for a vertex p whose every child contains a stone,
   2a. place an available stone on p or
   2b. move a stone from a child of p to p,

so that each vertex receives a stone exactly *once* during these operations.

Stones represent registers and putting a stone on a vertex of the graph corresponds to computing the vertex and storing the result in that register (this is why we need stones on all children of a vertex when computing it). Removing a stone from a vertex corresponds to forgetting the value of the vertex, which should then be done only if we do not need it in other computations (as vertices cannot be recomputed), i.e. when all its parent vertices have already received a stone.

Winning the game means successfully computing the algebraic expression encoded in the graph while using at most k registers. In this context, an extension σ for a graph G indicates in which order the vertices receive stones. Note that the first stone enters G via applying Rule 2a to a leaf of G. Then, solving the optimization problem associated to Register Sufficiency can be seen as finding an extension of G that minimizes the number k of stones (registers) needed to win the game (compute the expression). As suggested by our formulation, this number equals the previously introduced "register width", $\mathrm{rw}(G)$.

Proposition 2. *A DAG G can be computed using $\leq k$ registers if and only if $\mathrm{rw}(G) \leq k$.*

With Proposition 2, the Register Sufficiency problem can be formulated as: given a rooted DAG G and some integer k, decide if $\mathrm{rw}(G) \leq k$. Following Sethi [14], we will use a special, "initial" vertex in our reduction.

Definition 1. *Let (G, k) be an instance of* Register Sufficiency *such that G has k leaves and all leaves have a common parent ψ. Then, we say that ψ is an* initial vertex *and that (G, k)* has the initial vertex property.

Lemma 1 (See [14]). *Let (G, k) be a yes-instance of* Register Sufficiency *with an initial vertex ψ. Let σ be an extension of G with $\mathrm{rw}(\sigma) \leq k$. Then, $\sigma(k+1) = \psi$ and $\sigma[1..k]$ contains the k leaves of G in any order. Moreover, there is a leaf whose only parent is ψ.*

A corollary of Lemma 1 is that, in a yes-instance (G, k) with the initial vertex property, all children of the initial vertex are leaves. Thus, $\mathrm{rw}_k(\sigma) = k$ for all extensions σ with $\mathrm{rw}(\sigma) \leq k$ and, thus, $\mathrm{rw}(G) = k$ for such yes-instances. Note that 3-SAT reduces to instances (G, k) of Register Sufficiency that have the initial vertex property [14].

Theorem 2 ([14]). *It is* NP*-hard to decide* Register Sufficiency *for instances* (G, k) *that have the initial vertex property.*

Below, we reduce Register Sufficiency on instances with the initial vertex property to Register Sufficiency on rooted, binary, single-leaf DAGs. To this end, we reduce from Weighted 2-Satisfiability instead of 3-Satisfiability and modify parts of the reduction in order to obtain a network that is already bifurcating in some crucial spots. Then, we present a number of polynomial-time executable transformation rules that take one such instance (G, k) of Register Sufficiency having the initial vertex property and replace all remaining high-degree vertices with binary ones without changing the answer for the instance. Finally, a reduction rule is given to ensure that the resulting DAG has a single leaf.

3.1 An Adaptation of a Known NP-hardness Proof

We strengthen the construction presented by Sethi [14] to construct a *binary* DAG with a *single leaf* and *without degree-two vertices*. Our modifications to Sethi's construction come in two stages. First, instead of 3-SAT, we will reduce a 2-SAT variant called Monotone Weighted 2-Satisfiability (also known as Vertex Cover), which is also NP-hard [10]. In this variant, all variables occur non-negated in the instance formula φ, each variable is used at least once, and we ask for an assignment that satisfies φ while setting at most k variables to true. Second, we show how to "binarize" all remaining polytomies and establish a single leaf.

Construction 1 (See Fig. 4). *Given a formula* φ *in monotone 2-CNF on variables* $x_1, \ldots, x_n$ *and clauses* $C_1, \ldots, C_m$*, let* $y_{i,j}$ *denote the* j^{th} *literal in* C_i*. Construct the instance* (G, k')*, where* $k' = 8n + 3m + k + 2$ *and* G *is a rooted DAG on the vertex set* $A' \uplus B' \uplus C \uplus F' \uplus H \uplus P \uplus P' \uplus R' \uplus S' \uplus T' \uplus U \uplus W \uplus X \uplus X' \uplus X^* \uplus Z' \uplus \{\alpha, \psi, d, \rho\}\}$ *where* $R' := \bigcup_{i\in[n]} R'^i$*,* $S' := \bigcup_{i\in[n]} S'^i$*,* $T' := \bigcup_{i\in[n]} T'^i$ *and*

$$A' = \{a_i \mid i \in [2n + 1 + k]\} \qquad C = \{c_i \mid i \in [m]\} \qquad F' = \{f_{i,1}, f_{i,2} \mid i \in [m]\}$$
$$B' = \{b_i \mid i \in [3n - m]\} \qquad W = \{w_i \mid i \in [n]\} \qquad R'^i = \{r_{i,j} \mid \ j \in [2n - 2i + 2 + k]\}$$
$$U = \{u_{i,1}, u_{i,2}, u_{i,3} \mid i \in [n]\} \qquad X' = \{x'_i \mid i \in [n]\} \qquad S'^i = \{s_{i,j} \mid \ j \in [2n - 2i + 1 + k]\}$$
$$X = \{x_i, \overline{x}_i \mid i \in [n]\} \qquad X^* = \{x^*_i \mid i \in [n]\} \qquad T'^i = \{t_{i,j} \mid \ j \in [2n - 2i + 1 + k]\}$$
$$H = \{h_{i,1}, h_{i,2} \mid i \in [m]\} \qquad P = \{p_i \mid i \in [m]\} \qquad P' = \{p'_i, p''_i, p'''_i \mid i \in [m]\}$$
$$Z' = \{z_i \mid i \in [n + 1]\}$$

and the arc set is the union of the following sets:

$$E'_1 = \{\psi v \mid v \in A' \uplus B' \uplus F' \uplus U \uplus H \uplus \{\alpha\}\} \qquad E_5 = \{w_i u_{i,1}, w_i u_{i,2} \mid i \in [n]\}$$
$$E'_2 = \{v\psi \mid v \in R'^i \uplus S'^i \uplus T'^i\} \qquad E'_{7,1} = \{z_{i+1} w_i, z_{i+1} z_i \mid i \in [n]\}$$
$$E_4 = \{x_i z_i, \overline{x}_i z_i, x_i u_{i,1}, \overline{x}_i u_{i,2}, x'_i u_{i,3} \mid i \in [n]\} \qquad E'_{7,2} = \{c_i z_{n+1} \mid i \in [m]\}$$
$$E_{12} = \{r_{i,j} r_{i,j+1} \mid j \in [2n - 2i + k + 1], i \in [n]\} \qquad E'_9 = \{dv \mid v \in B' \uplus C\}$$
$$E_{13} = \{s_{i,j} s_{i,j+1}, t_{i,j} t_{i,j+1} \mid j \in [2n - 2i + k], i \in [n]\} \qquad E_{11} = \{x^*_i x'_i, x'_i x_i \mid i \in [n]\}$$

$$E'_3 = \{\rho v \mid v \in W \uplus X \uplus Z' \uplus \{\psi, d\} \uplus X^* \uplus \{p'_i, p'''_i \mid i \in [m]\} \uplus \{u_{i,3} \mid i \in [n]\} \uplus \{z_{n+1}\}\}$$
$$E'_6 = \{z_i r_{i,j}, x_i s_{i,j}, \overline{x}_i t_{i,j} \mid \ r_{i,j} \in R'^i,\ s_{i,j} \in S'^i,\ t_{i,j} \in T'^i, i \in [n]\}$$
$$E'_8 = \{c_i p_i, p_i f_{i,1}, p_i f_{i,2}, p'''_i f_{i,2}, p'''_i p''_i, p''_i p'_i, p''_i h_{i,2}, p'_i h_{i,1} \mid i \in [m]\}$$
$$E'_{10} = \{x^*_{i,1} f_{i,1}, x^*_{i,2} h_{i,2}, \overline{x}_{i,1} h_{i,1} \mid i \in [m]\}$$

where $x_{i,j}$ denotes the j^{th} variable in C_i and $\overline{x}_{i,j}$ the negation of $x_{i,j}$ (and their corresponding vertices with the same names) and $x^*_{i,j}$ the vertex in X^* such that $x^*_{i,j} x_{i,j} \in E_{11}$.

The idea behind Construction 1 is that the "variable-assignment phase" of Sethi [14] still works as before (with k more stones in each step to account for the k additional vertices we have in R'^i, S'^i, and T'^i). In more detail, this process is as follows: in the beginning, all k' stones have to go to all the leaves, at which point ψ is computed using one stone of a vertex in A', while the other $k + 2n$ stones of A' are now free (unlike stones on other leaves still having other parents). These $k + 2n$ stones need to go to R'^1 (otherwise we will not have enough stones later for these vertices), allowing to compute z_1, who will keep one stone. The $k + 2n - 1$ other stones from R'^1 are free to go to either S'^1 allowing to compute x_1 or to T'^1 allowing to compute $\overline{x}_1$. The chosen literal allows exactly one stone from U to move to w_1 (e.g. $u_{1,1}$ if x_1 is chosen and $u_{1,2}$, otherwise), who will keep this stone. Thus, $k + 2n - 2$ stones (from either S'^1 or T'^1) are now free to compute R'^2, followed by z_2. This process continues until w_n receives a stone, at which point we ended Sethi's variable assignment phase. Now, the stone on α moves to z_{n+1} and we are left with the k free stones, coming from either S'^n or T'^n, that we can spend on vertices $x'_j \in X'$ (which then move to $x^*_j \in X^*$) whose corresponding $x_j \in X$ has received a stone before.

Consider what happens if the described "variable assignment" phase chooses k vertices in X satisfying the formula and the k corresponding vertices of X^* receive a stone right after this phase. Consider the gadget corresponding to clause $C_i = (x_j \vee x_\ell)$ and recall that $f_{i,1}$, $f_{i,2}$, $h_{i,1}$, $h_{i,2}$ already hold stones. In analogy with Sethi [14], each c_i receives a stone as follows:

- If C_i is satisfied by x_j, then x^*_j holds a stone, so the stone on $f_{i,1}$ can move to p_i (this is allowed since p_i's children all hold stones).
- Otherwise, both $\overline{x}_j$ and x^*_ℓ hold stones. The first one allows the stone on $h_{i,1}$ to move to p'_i. The second one allows the stone on $h_{i,2}$ to move to p''_i and then to p'''_i, allowing the stone on $f_{i,2}$ to move to p_i.

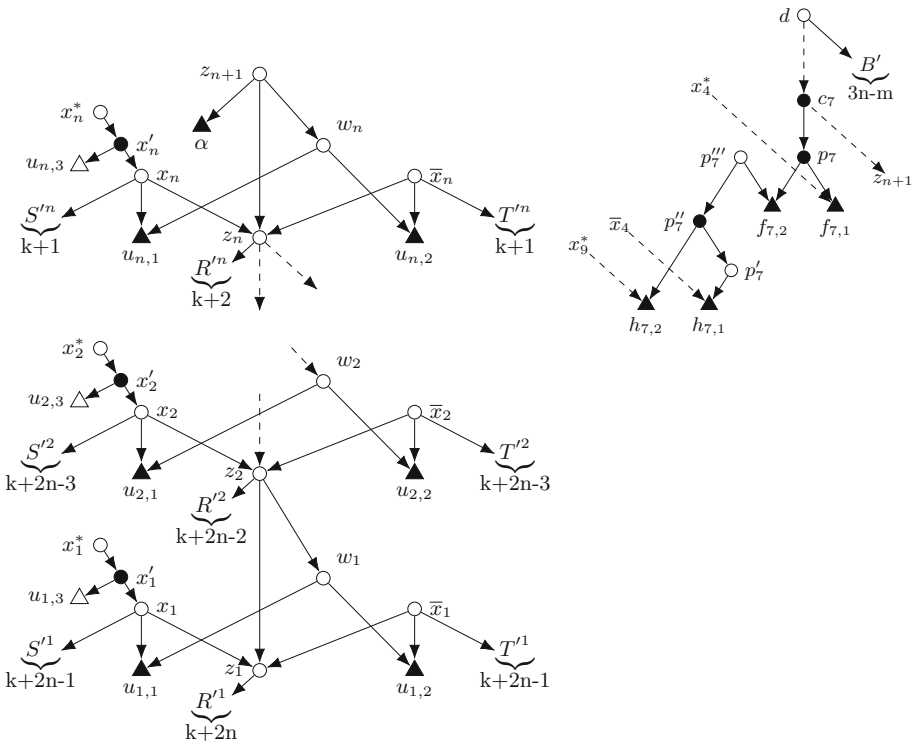


Fig. 4. Illustration of Construction 1; white vertices are children of the root ρ, triangles are leaves and children of ψ. This allows us to omit drawing ψ and ρ. We also omit the leaves in A'. **Left:** "variable-assignment" gadget (arcs of E_2' omitted). **Right:** clause gadget for the clause $C_7 = (x_4 \vee x_9)$. Note that all w_i, p_i, p_i'', p_i''' and c_i are bifurcating.

Thus, in both cases p_i gets a stone which it then passes to c_i. Finally, when all c_i have received a stone, d receives a stone from one of them, freeing up $|B'| = 3n - m$ stones on the vertices in B' and $|C| - 1 = m - 1$ stones on the vertices in C. Since $k \leq n$, these $3n - 1$ stones can then be placed on T'^1 (if x_1 already holds a stone) or S'^1 (if $\overline{x}_1$ already holds a stone) and one of them can then move to $\overline{x}_1$ or x_1, respectively. In this way, all $2n$ vertices x_i and $\overline{x}_i$ progressively receive a stone. Since n of them already got stones in the variable assignment phase, this leaves us with $(3n-1)-n$ stones, $n-k$ of which are then put on the $n-k$ remaining stoneless vertices of X' which immediately move to the remaining vertices of X^*. At this point, all stones on all $h_{i,1}$ and $h_{i,2}$ move to p_i' and p_i'' followed by p_i''' if they did not already do so before. Finally, ρ receives a stone from any of its children.

Theorem 3. *Let $k \in \mathbb{N}$, let φ be a formula in monotone 2-CNF, and let (G, k') be an instance of* Register Sufficiency *constructed by Construction 1 on input (φ, k). Then, φ has a satisfying assignment with $\leq k$ true variables if and only if $\mathrm{rw}(G) \leq k'$.*

Note that networks G created by Construction 1 contain non-binary vertices, as well as many leaves. However, all non-binary vertices of G have nice properties that allow us to "binarize" them using reduction rules that we present in Sect. 3.2.

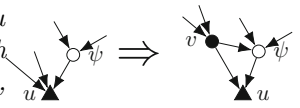
Observation 1. *Let (G', k') result from Construction 1 and let $u \in V(G')$.*

(a) If u is a non-leaf with $\deg^+(u) \geq 3$, then u has a child with in-degree 1.
(b) If u is a non-leaf with at least 3 parents, then the root ρ is a parent of u.
(c) If u is a leaf with at least 3 parents, then u has exactly 3 parents and ψ is one of them.

3.2 Reducing Nice Polytomies and Leaves

The following reduction rule is used to turn all leaves binary since many leaves constructed in Construction 1 have in-degree three.

Rule 1. *Let (G, k) have an initial vertex ψ and let u be a leaf in G with at least three parents, one of which is ψ. Then, add a new parent v to u, add the arc $v\psi$, and replace all xu by xv except ψu.*



The next rule splits vertices of in- and out-degree at least two into a reticulation and a tree-vertex.

Rule 2. *Let u be a vertex of G, let P and C be its parents and children, respectively, and let $|P| > 1$ and $|C| > 1$. Then, "split" u, i.e. add a new vertex v, add the arc uv and, for all $c \in C$, replace the arc uc by the arc vc.*

Rule 3 (See Fig. 5(left)). *Let u be a vertex with at least three children, let x and y be children of u such that y is a tree-vertex and x is either a tree-vertex or x has a parent $q \neq u$ that is comparable to u in G. Then, "split" u into ru (that is, create a new parent r for u and make all parents of u parents of r instead), subdivide ru with a new vertex r', for all parents q of x with $q >_G u$ replace qx with qr', add the arc rx, subdivide uy with a new vertex w, remove the arc ux and, unless x has a parent $q <_G u$ in G, add the arc wx.*

Correctness proofs of Rules 1–3 are deferred to the full version of this paper. Note that Rule 3 only increases $\deg^-_G(x)$ if x has no parents $q <_G u$ in G. But then, either x is a tree-vertex in G, in which case no new polytomies are created, or x has a parent $q >_G u$, in which case this parent becomes a parent of r' instead. Further, although Rule 3 may introduce degree-two vertices, all of them are parents of tree-vertices and can thus be removed using the following:

The following rule turns polytomous reticulations into binary ones. We make use of the fact that G has a root and an initial vertex (see Definition 1).

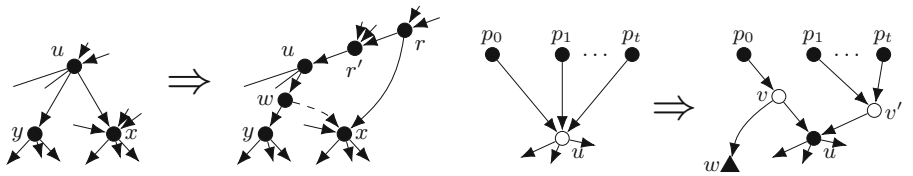
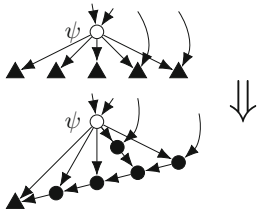


Fig. 5. Illustration of Rule 3 (left) and Rule 4 (right). Triangles are leaves and children of the initial vertex ψ, white vertices are children of the root ρ. Note that, on the left, u has a tree-vertex child before and after the modification.

Rule 4 (see Fig. 5, right). *Let (G, k) have an initial vertex ψ, let ρ be the root of G, let u be a non-leaf with parents $p_0, p_1, \ldots, p_t, p_{t+1} = \rho$ $(t \geq 1)$. Then, add a new leaf w, increase k by one, subdivide p_0u with a vertex v, replace arc ρu by ρv, add a new parent v' of u, replace arc p_iu by p_iv' for all $i \in [t+1]$, and add the arcs $\rho v'$, vw, ψw.*

Note that Rule 4 effectively turns a vertex of in-degree $t + 2$ (for $t \geq 1$) into a vertex of in-degree $t + 1$. Further, note that the instance (G', k') constructed by Construction 1 has an initial vertex ψ.

Rule 5. *Let (G, k) have an initial vertex ψ, let X be the set of leaves of G and let $Y \subseteq X$ contain the leaves that have more than one incoming arc. Let $Y \neq \emptyset$ and let $x_1, x_2, \ldots, x_k$ be an arbitrary total order of X with $x_k \in Y$. Then, turn X into a path by adding the arc $x_{i+1}x_i$ for all i. Further, for all $y \in Y - x_k$, subdivide ψy with a new vertex z, and replace all arcs uy occurring in G by uz.*



Note that the graphs produced by Construction 1 satisfy $\emptyset \subsetneq Y \subsetneq X$.
Note that Rule 5 destroys the initial vertex property, preventing any further use of Rule 4 and Rule 5. However, Rule 5 does not create new polytomies and, for turning ψ binary by applying Rule 3, it is sufficient that ψ is "primal" (a weaker condition than being initial).

Theorem 4. Register Sufficiency *is* NP*-complete on rooted, single-leaf binary DAGs.*

Acknowledgments. We thank Fabio Pardi to have brought the problem to our attention and the Genome Harvest project, ref. ID 1504-006 ("Investissements d'avenir", ANR-10-LABX-0001-01).

References

1. Barát, J.: Directed path-width and monotonicity in digraph searching. Graphs Comb. **22**(2), 161–172 (2006)
2. Bordewich, M., Scornavacca, C., Tokac, N., Weller, M.: On the fixed parameter tractability of agreement-based phylogenetic distances. J. Math. Biol. **74**(1), 239–257 (2017)
3. Bordewich, M., Semple, C.: Computing the hybridization number of two phylogenetic trees is fixed-parameter tractable. IEEE/ACM Trans. Comput. Biol. Bioinform. **4**(3), 458–466 (2007)
4. Bryant, D., Bouckaert, R., Felsenstein, J., Rosenberg, N.A., RoyChoudhury, A.: Inferring species trees directly from biallelic genetic markers: bypassing gene trees in a full coalescent analysis. Mol. Biol. Evol. **29**(8), 1917–1932 (2012)
5. Bryant, D., Lagergren, J.: Compatibility of unrooted phylogenetic trees is FPT. Theor. Comput. Sci. **351**(3), 296–302 (2006)
6. Díaz, J., Petit, J., Serna, M.: A survey of graph layout problems. ACM Comput. Surv. **34**(3), 313–356 (2002)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman & Co., Ltd., New York City (1979)
8. Grigoriev, A., Kelk, S., Lekić, N.: On low treewidth graphs and supertrees. In: Dediu, A.-H., Martín-Vide, C., Truthe, B. (eds.) AlCoB 2014. LNCS, vol. 8542, pp. 71–82. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07953-0_6
9. Huson, D.H., Rupp, R., Scornavacca, C.: Phylogenetic Networks: Concepts: Algorithms and Applications. Cambridge University Press, Cambridge (2010)
10. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds.) Complexity of Computer Computations. The IBM Research Symposia Series, pp. 85–103. Springer, Boston (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
11. Kelk, S., Scornavacca, C.: Constructing minimal phylogenetic networks from soft-wired clusters is fixed parameter tractable. Algorithmica **68**(4), 886–915 (2014)
12. Kelk, S., Stamoulis, G., Wu, T.: Treewidth distance on phylogenetic trees. Theor. Comput. Sci. **731**, 99–117 (2018)
13. Rabier, C.E., Berry, V., Pardi, F., Scornavacca, C.: On the inference of complicated phylogenetic networks by Markov chain Monte-Carlo (submitted)
14. Sethi, R.: Complete register allocation problems. SIAM J. Comput. **4**(3), 226–248 (1975)
15. Whidden, C., Beiko, R.G., Zeh, N.: Fixed-parameter algorithms for maximum agreement forests. SIAM J. Comput. **42**(4), 1431–1466 (2013)
16. Zhang, C., Ogilvie, H.A., Drummond, A.J., Stadler, T.: Bayesian inference of species networks from multilocus sequence data. Mol. Biol. Evol. **35**(2), 504–517 (2018)