CONSTRUCTING COMPUTER VIRUS PHYLOGENIES

LESLIE ANN GOLDBERG*, PAUL W. GOLDBERG†, CYNTHIA A. PHILLIPS‡, AND GREGORY B. SORKIN§

Abstract. There has been much recent algorithmic work on the problem of reconstructing the evolutionary history of biological species. Computer virus specialists are interested in finding the evolutionary history of computer viruses — a virus is often written using code fragments from one or more other viruses, which are its immediate ancestors. A phylogeny for a collection of computer viruses is a directed acyclic graph whose nodes are the viruses and whose edges map ancestors to descendants and satisfy the property that each code fragment is "invented" only once. To provide a simple explanation for the data, we consider the problem of constructing such a phylogeny with a minimal number of edges. In general this optimization problem cannot be solved in quasi-polynomial time unless NQP=QP; we present positive and negative results for associated approximation problems. When tree solutions exist, they can be constructed and randomly sampled in polynomial time.

* leslie@dcs.warwick.ac.uk. Department of Computer Science, University of Warwick, Coventry CV4 7AL, United Kingdom. Part of this work was performed at Sandia National Laboratories and was supported by the U.S. Department of Energy under contract DE-AC04-76AL85000. Part of this work was supported by the ESPRIT Basic Research Action Programme of the EC under contract 7141 (project ALCOM-IT).

† goldbepw@helios.aston.ac.uk. Department of Applied Mathematics and Computer Science, Aston University, Aston Triangle, Birmingham B4 7ET, United Kingdom. Part of this work was performed at Sandia National Laboratories and was supported by the U.S. Department of Energy under contract DE-AC04-76AL85000.

‡ caphill@cs.sandia.gov. Sandia National Labs, P.O. Box 5800, Albuquerque NM 87185. This work was performed under U.S. Department of Energy contract DE-AC04-76AL85000.

§ sorkin@watson.ibm.com. IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights NY 10598.

1. Introduction. There are now several thousand different computer viruses in existence, with new ones being written at a rate of 3 to 4 per day. Most of these are based upon previous ones: someone copies and modifies a virus, or creates a new virus with subroutines borrowed from one or more **ancestors.**

For most purposes, a computer virus can be regarded as a fixed string of bytes, each byte consisting of eight bits. If one virus is based on another, long substrings of the ancestor, say 20 bytes or more, will appear in the descendant. Using probability models similar to those employed in speech recognition it is possible to estimate the probability that a given byte string occurs in several viruses *by chance* [13]; if the probability is low but the string does occur in several viruses then we conclude that it was written for one virus, and copied into the others. More details of the pattern-matching approach used to identify duplicated byte strings are given in Section 1.3.

We wish to infer an evolutionary or phylogenetic history for a set of computer viruses.[1] As most phylogenetic literature to date has been based upon biological evolution, we adopt that terminology. Thus, the viruses in the input set $\mathcal{S} = \{s_1, ..., s_n\}$ are called **species.** The species are defined by a set of **binary characters** $\mathcal{C} = \{c_1, \ldots, c_k\}$. A binary character is a function $c : \mathcal{S} \rightarrow \{0, 1\}$. (In general, the range of a character can be arbitrary, but the presence or absence of byte strings can be modeled with binary characters.) Each character c corresponds to a byte string, with $c(s) = 1$ if the string occurs in species s and $c(s) = 0$ otherwise. If $c(s) = 1$, we say that species s *has* or *contains* character c. In analogy with terminology from the logic synthesis area of computer circuit design, we define the **on-set** S_c of a character c to be the set of all species on which its value is 1: $S_c = \{s \in \mathcal{S} \mid c(s) = 1\}$. A character c is **trivial** if $|S_c| \leq 1$. A trivial character can be ignored since it imposes no constraints on possible solutions.

We assume that the input species are all related: that the bipartite graph joining species to characters that have them is connected. Otherwise, the connected components can be considered independently.

We also assume that each code fragment is invented only once. For sufficiently long fragments this is justified by differences in programming style, the many possible orderings of unconstrained events, etc. We model the evolution of a set of viral species with a directed graph in which an edge $s_i \rightarrow s_j$ indicates that species s_i is an ancestor of species s_j (i.e. s_j inherited some character(s) from s_i).

Definition: A **phyloDAG** for input species $\mathcal{S}$ and characters $\mathcal{C}$ is a directed acyclic graph (DAG) with node set $\mathcal{S}$. For each character $c \in \mathcal{C}$, the subgraph induced by on-set S_c is connected, in the sense that from a single **archetype** $a_c \in S_c$ there is a directed path, within S_c, to every other $s \in S_c$.

The phyloDAG model allows the possibility that a species may be derived from several ancestors rather than from a single ancestor. For computer viruses this is appropriate, since a virus author could appropriate code from a variety of sources. It is also a plausible model for evolution of bacteria populations which have inherited genes via infection by bacteriophages[7]: the genes evolve only once, and can be transferred from host to host by these viruses.

A phyloDAG exists for any inputs $(\mathcal{S}, \mathcal{C})$: for any chronology ascribed to the species (i.e. any total ordering of the species set), the directed graph with edges from each species to all later species is a phyloDAG. However, every pair of species is related by an edge in this graph. Since

[1] One of us (Sorkin) is a member of the computer anti-virus group at IBM Research. The group produces a commercial software product, IBM ANTIVIRUS, and conducts related theoretical research. Phylogenetic information may be useful for tracking global virus trends, while the byte strings serving as characters can be used for virus detection.

most virus species presumably have few ancestors, we seek a **Minimum PhyloDAG**, one with a minimal number of directed edges.

We assume that the input is given in the following compact format: for each species $s \in S$, we are given a list of the characters c for which $c(s) = 1$.

Definition: The **input length** $\ell = \ell(\mathcal{S}, \mathcal{C}) = \sum_{c \in C} |S_c|$. The **size** is $n = |\mathcal{S}|$. The number of characters is $k = |C|$.

Our approach to the evolution problem corresponds to a *restricted* model of evolution: one in which we are not allowed to introduce hypothetical species outside of the input set. This model is well-suited to computer viruses, where because of good world-wide communications, sharing of data between anti-virus organizations, and the brief history involved, there are likely to be very few gaps in our viral database — a situation quite different from that in biology. Previous work on restricted models of evolution will be discussed in Section 1.2. For our model, if additional species could be introduced into a phyloDAG, there would always be a trivial sparse phyloDAG: a star graph with the center an added species s such that $c(s) = 1$ for all $c \in \mathcal{C}$.

If a phyloDAG's vertices are labeled with the values of one character, the postulate that no character is "invented" twice corresponds to an assertion that there is at most one directed edge labeled 0→1. Thus the sequence of labels along any source-to-leaf path is described by the regular expression 0*1*0*, that is, zero or more 0's, followed by zero or more 1's, and finally zero or more 0's again.

1.1. Paper organization and results. We will show in Section 3.3 that the Minimum PhyloDAG problem is "hard": if $\mathrm{NTIME}(n^{\mathrm{polylog}\, n}) \neq \mathrm{DTIME}(n^{\mathrm{polylog}\, n})$ (a plausible complexity-theoretic assumption related to P ≠ NP), it cannot be approximately solved within better than a logarithmic factor (let alone solved exactly) in polynomial time. In fact, we know of no way to approximate Minimum PhyloDAG within a logarithmic factor: Section 3.3 shows that various natural greedy strategies (including randomized ones) do not even approximate within a factor of cn.

Because of the difficulty of the phyloDAG problem, we consider two variants. In the first, we require that each species have just one ancestor, so that the phyloDAG is an arborescence (a tree with edges directed away from a root). In Section 2 we define a **0–1–0 phylogeny** to be an arborescent phyloDAG's underlying undirected tree. Species $\mathcal{S}$ and characters $\mathcal{C}$ may be consistent with zero, one, or multiple 0–1–0 phylogenies. We give two polynomial-time algorithms to randomly sample 0–1–0 phylogenies if any exist.

The first **atomic-set** algorithm (Section 2.1) computes a concise data structure that represents all 0–1–0 phylogenies for the input data and can be used to select a phylogeny uniformly at random in time $O(n\ell)$. When no solution exists the algorithm returns a **witness set**: a concise indication of *why* no there can be no phylogenetic tree.

The second **minimum spanning tree** algorithm (Section 2.2) characterizes a 0–1–0 phylogeny of the input species set as a minimum spanning tree (MST) of a particular undirected edge-weighted graph. With it, 0–1–0 phylogenies can be constructed in deterministic time $O(\ell n + n^2 \log n)$ or (with high probability) in randomized time $O(\ell n)$, and sampled uniformly at random in time $O(\ell n + M(n))$, where $M(n)$ is the time needed to multiply two $n \times n$ matrices. It does not produce a concise witness when there is no 0–1–0 phylogeny.

The second variant of phyloDAG is simply its undirected analogue. A **phylograph** for species $\mathcal{S}$ and characters $\mathcal{C}$ is an undirected graph with with vertex set $\mathcal{S}$, with the property that the subgraph induced by the on-set of each character $c \in \mathcal{C}$ is connected. The **Minimum Phylograph** problem is find a phylograph with the minimum number of edges. Theorem 3.1

shows that it is hard to approximate Minimum Phylograph within a factor less than $(1/16)\log\ell$, while Theorem 3.3 shows that approximating it within a factor of $\ln\ell$ is easy.

The model of computation used in this paper is the uniform-cost random-access machine.

1.2. Related work. Previous work in phylogeny has focused on constructing phylogenetic *trees*. However, the problem of modeling virus evolution is more suited to phylographs and phyloDAGs, in which undirected cycles may arise. As far as we know, ours is the first phylogenetic work that allows cycles.

There is substantial literature on character-based phylogenies where each subgraph induced by all species sharing a state for a character is required to be connected. This problem is called the **perfect phylogeny** problem, and is NP-complete for the "unrestricted" case (where putative species may be added) with general characters [3, 17]. For the unrestricted case with binary characters Gusfield gives an elegant $O(nk)$ algorithm [11], and for the restricted case with general characters Goldberg et al. [10] give an algorithm analogous to the MST algorithm of Section 2.2.

Our 0–1–0 phylogeny problem is similar to a restricted version of the **general character compatibility** problem of Benham et al. [2]. There a character c maps each species s to a *subset* $c(s) \subseteq \{0,1,2\}$ rather than to a single value; the *leaves* of the tree are the species $\mathcal{S}$; for each c and s a single value from $c(s)$ is chosen as a label; and the goal is to find a rooted perfect phylogeny in which the sequence of labels along any root-to-leaf path is of the form $0 \to 1 \to 2$. The problem is NP-hard [2].

1.3. Choosing characters. For the bulk of this paper the set of species and the characters are simply inputs, but at an earlier stage they too must be determined algorithmically. In one mathematical model of the problem, we are given a collection of byte strings representing computer viruses (there might be 5,000 such strings, each typically 2,000 bytes long) and a similar collection representing legitimate computer code (perhaps 40,000 strings, each typically 20,000 bytes long); we wish to find all strings of 20 bytes or more that occur in at least 2 viruses but in no legitimate programs. Of course the numerical parameters might be varied, or another criterion might be substituted, e.g. the string's appearance in at least 10 times as many viral programs as legitimate programs.

The problems can be solved in linear space and time by a straightforward application of suffix trees [6]. All viral and legitimate strings are concatenated together, separated by a special character, and a suffix tree is constructed. The leaves of the suffix tree represent all suffixes of the input string, and the internal nodes — viewed as paths from root part-way to leaf — denote prefixes of suffixes, which is to say substrings of the input string. Depth-first search can be used to propagate, from leaves to root, the number of times each substring appears, and in fact the number of times it appears in viruses and (separately) in legitimate strings; this provides the solution to the stated problem.[2]

2. Computing a 0–1–0 phylogeny. The case in which each species has only one ancestor is of obvious special interest, and corresponds to cases in which the phyloDAG is an arborescence — a tree with all edges directed as a flow away from some root. There is a straightforward n:1 correspondence between arborescences and undirected trees: the undirected graph underlying an arborescence is a tree; and each of the n possible rootings of a tree is an arborescence. There exist phyloDAGs whose underlying graphs are trees but which are not arborescences.[3] These

[2] For the sake of accuracy, we note that this is not precisely the approach employed in practice at IBM; we have simplified here for the sake of exposition.

[3] An example, for species with characters (a), (ab), and (b), is $(a) \to (ab) \leftarrow (b)$.

instances, however, hypothesize multiple ancestors for a species, and therefore are not solutions to our problem. Therefore we concentrate on undirected 0–1–0 phylogenies:

Definition: An (undirected) **0–1–0 phylogeny**, or **phylogenetic tree**, is a tree T on species $\mathcal{S}$ with characters $\mathcal{C}$ such that each on-set S_c induces a sub-tree of T.

If $\vec{T}$ is a phyloDAG whose underlying graph is a tree T, then T is a 0–1–0 phylogeny as defined above: as each on-set S_c was connected in $\vec{T}$, it is connected in T. Also, if T is a 0–1–0 phylogeny, any arborescence based on T is a phyloDAG: the archetype of any character c is the species in S_c closest to the root. In this section, we will show how to generate 0–1–0 phylogenies, and how to generate them uniformly at random. Given a uniformly random phylogenetic tree, choosing a root uniformly at random generates a uniformly random arborescent phyloDAG.

Because an arborescence can be rooted anywhere, a 0–1–0 phylogeny alone does not determine an evolutionary chronology, but it can be useful in combination with external information. For example if the first species' identity is known, the rest of the evolutionary history follows.

2.1. The atomic-set algorithm for computing 0–1–0 phylogenies. As described in the Introduction, our atomic-set algorithm produces a data structure, an AS-tree, which concisely represents all 0–1–0 phylogenies for species $\mathcal{S}$ and characters $\mathcal{C}$, and can be used to generate an arbitrary solution or a solution chosen uniformly at random.

Generalizing the definition of the on-set of a character, define the on-set of a collection of characters to be the species having *all* those characters: $S_C = \bigcap_{c \in C} S_c$.

Definition: Let $\hat{C} \subseteq \mathcal{C}$ be a maximal (not necessarily *maximum*) set of characters for which $|S_{\hat{C}}| \geq 2$. Then $A = S_{\hat{C}}$ is an **atomic set** with **defining characters** $\hat{C}$.

LEMMA 2.1. *For any atomic set A and character C, either $S_c \supseteq A$ (c is a defining character), or $|S_c \cap A| = 1$ (C is a* **non-defining** *character), or $S_c \cap A = \emptyset$ (c is an* **avoiding** *character).*

Proof. The only logical possibility missing is that $|S_c \cap A| \geq 2$ but $S_c \cap A \neq A$, which would contradict the maximality of A's set of defining characters. □

Where $n = |\mathcal{S}|$ and $k = |\mathcal{C}|$, an atomic set can be constructed in time $O(kn)$: start with $\hat{C} = \emptyset$ (so $S_{\hat{C}} = \mathcal{S}$), sweep through all characters $c \in \mathcal{C}$ in turn, reject c if $|S_C \cap S_c| \leq 1$, but otherwise add c to the defining set, so $\hat{C} := \hat{C} \cup \{c\}$. An $O(\ell)$-time algorithm to find an atomic set is given in the Appendix.

LEMMA 2.2. *Suppose all species in $\mathcal{S}$ are connected, i.e. the bipartite graph joining characters to species that have them is connected. Then if $s_1, s_2 \in \mathcal{S}$ have no characters in common, no phylogeny contains the edge (s_1, s_2).*

Proof. Suppose a phylogenetic tree T contained (s_1, s_2), and delete (s_1, s_2) to create a forest T', consisting of two trees. For any character c and any $s, s' \in S_c$, T has a path $s, \ldots, s'$ within S_c. The path does not include the edge (s_1, s_2), since not both s_1 and s_2 can be in S_c, so T' contains the same path. Thus in T' there is a path from any species having character c to any other. Given the connectedness of the species–character graph, a series of such paths joins any species in $\mathcal{S}$ to any other, contradicting the fact that T' is not a connected graph. □

LEMMA 2.3. *If A is an atomic set, then in any 0–1–0 phylogeny A's induced subgraph is a subtree.*

Proof. In a 0–1–0 phylogenetic tree T, the on-set of any character $c \in \mathcal{C}$ induces a connected subgraph, therefore a subtree. A is the intersection of the subtrees corresponding to A's defining characters, and the intersection of subtrees is itself a subtree. □

LEMMA 2.4. *For any phylogeny T and atomic set A, if the subtree T_A is replaced by any other tree T'_A on the set A, the resultant overall tree T' is also a phylogeny.*

Proof. For any character c and species $s, s' \in S_c$, consider the (unique) path $s, \ldots, s'$ in T. If $S_c \cap A = \emptyset$, the path never enters A, so it is unaffected (i.e. the identical path exists in T'). If $|S_c \cap A| = 1$, the path touches at most one vertex in A, hence no edges within A, and is unaffected. Otherwise (by Lemma 2.1) $S_c \supseteq A$, and if the path through T included any sub-paths through T_A (in fact there can be at most one), those sections could be replaced by sub-paths through T'_A (and thus still within S_c). So connectedness of all characters in T implies the same for T', and T' is a phylogeny. □

LEMMA 2.5. *For any phylogeny T and atomic set A, if A is* **collapsed** *— replaced by a single species "a" having all defining and non-defining characters of A (but not its avoiding characters), and the subtree T_A is contracted to the single species a, then the resultant overall tree T' is a phylogeny for $\mathcal{S}' = (\mathcal{S} \setminus A) \cup \{a\}$.*

Proof. Same as previous. □

LEMMA 2.6. *If $(\mathcal{S}, \mathcal{C})$ has an atomic set A, with species $s_1, s_2 \in A$ owning non-defining characters c_1, c_2 respectively, and if $S_{c_1} \cap S_{c_2} \neq \emptyset$, then there is no 0–1–0 phylogeny for $\mathcal{S}$.*

Proof. Suppose there is a phylogeny T for $\mathcal{S}$. Root T at any $s_3 \in S_{c_1} \cap S_{c_2}$, and let s_x be the lowest common ancestor of s_1 and s_2. Then the path (all paths in a tree are unique) from s_1 to s_2 passes through s_x; the path from s_3 to s_1 passes through s_x (since s_x is an ancestor of s_1); and the path from s_3 to s_2 passes through s_x (since s_x is an ancestor of s_2). By Lemma 2.3, A induces a subtree, so $s_1, s_2 \in A$ implies that the s_1–s_2 path is contained in A, and in particular $s_x \in A$. Similarly $s_1, s_3 \in S_{c_1}$ implies $s_x \in S_{c_1}$, and $s_2, s_3 \in S_{c_2}$ implies $s_x \in S_{c_2}$. Therefore $s_x \in A \cap S_{c_1} \cap S_{c_2}$. But c_1 and c_2 are nondefining characters with distinct owners, so $A \cap S_{c_1} \cap S_{c_2} = \emptyset$, a contradiction. □

If the hypotheses of Lemma 2.6 are satisfied, we say that the atomic set A, characters c_1, c_2, and species s_1, s_2 provide a **witness** attesting to the non-existence of any 0–1–0 phylogenetic tree.

LEMMA 2.7. *Let A be an atomic set, and suppose that no s_1, s_2, c_1, c_2 satisfy the conditions of Lemma 2.6. As before, "collapse" A to the single species a having all defining and non-defining characters of A. If $\mathcal{S}' = (\mathcal{S} \setminus A) \cup \{a\}$ has a phylogeny, so does $\mathcal{S}$.*

Proof. Let T' be a phylogeny for $\mathcal{S}'$. Delete a and its incident edges, and replace them with the set A and any tree on A. Additionally, replace each edge (s, a) with a single edge as follows.

By Lemma 2.2, s and a must share some character(s), which (since a has them) must be defining or nondefining characters of A. If s and a share any *non-defining* characters, those characters must have a single owner s' (or else A, these characters, their owners, and s are a negative witness), in which case add the edge (s, s'). Otherwise, s and a only share defining characters of A, in which case add any edge (s, s') with $s' \in A$.

Replacement of each edge (s, a) with an edge (s, s'), $s' \in A$, means that the tree components created by a's deletion are all connected to the tree on A, creating a single tree T. Using arguments similar to those in Lemma 2.4, all characters induce connected components in T as they did in T'. □

In fact, the constructive nature of the proof of Lemma 2.7 immediately suggests the **atomic-set algorithm.** Starting from $\mathcal{S}_0 := \mathcal{S}$, repeatedly, find an atomic set A_i and check for a witness as per Lemma 2.6. If one is found, terminate negatively. Otherwise, collapse A_i to a single new species a_i, and re-define the species set to be $\mathcal{S}_i := (\mathcal{S}_{i-1} \setminus A_i) \cup \{a_i\}$. Since each atomic set contains at least two species, this reduces the number of species, and needs to be performed at most $n - 1$ times.

We construct the AS-tree during this contraction phase. The leaves of the AS-tree are the species in $\mathcal{S}$, and all elements of any set A_i have a_i as their parent. Equivalently, the final a_i is the

root of the AS-tree, and each a_j has all species in A_j as children. This tree concisely represents all possible phylogenies.

Now, starting at the root of the AS-tree, we expand any node a_i whose parent is already expanded using the method suggested by the proof of Lemma 2.7: Replace a_i with A_i and form any tree T_i on A_i. For each old edge (s, a_i), if s has a nondefining character c of A_i, add edge $(s, \text{owner}_{A_i}(c))$; otherwise s must have only defining characters, in which case add any edge (s, s'), $s' \in A_i$.

THEOREM 2.8. *The algorithm above produces a phylogeny for $\mathcal{S}, \mathcal{C}$ if one exists, and otherwise produces a negative witness. If the algorithm is implemented to choose trees T_i uniformly at random, and to choose $s' \in A_i$ uniformly at random for defining-character edges (s, s'), then it produces a uniformly random undirected 0–1–0 phylogeny.*

Proof. The first assertion follows directly from the preceding sequence of lemmas. If we detect a negative witness, we correctly terminate negatively by Lemma 2.6 coupled with Lemma 2.5. Otherwise, by Lemmas 2.5 and 2.7, we can collapse the atomic set, solve the problem on the new set, and "expand" the collapsed set to a 0-1-0 phylogeny. The choices made in the expansion phase are independent and lead to different phylogenies. The uniform generation of phylogenies follows from this one-to-one correspondence between phylogenies, and choices in the algorithm. □

Since we can generate a random 0–1–0 phylogeny from the AS-tree, it concisely represents all possible 0–1–0 phylogenies.

The atomic-set algorithm produces an AS-tree in time $O(n\ell)$: in each of the $O(n)$ collapsing iterations, we find an atomic set, check for a witness, and collapse the set each in time $O(\ell)$.

The expansion can be completed in time $O(n\ell)$. There are $O(n)$ expansions. To expand node a_i, we can produce a random tree on the set A_i in time $O(|A_i|)$, since a labeled tree on r nodes can be selected uniformly at random in $O(r)$ time. (See, for example, [15].) If we store pointers to owners of non-defining characters when constructing the AS-tree, we can connect this tree to its neighbors in time $O(\ell)$.

2.2. The Minimum Spanning Tree algorithm. In this section we give a second algorithm for computing 0–1–0 phylogenies. It is very simple, and is based on the observation that 0–1–0 phylogenies for species $\mathcal{S}$ and characters $\mathcal{C}$ correspond to minimum-weight spanning trees (MSTs) of a particular undirected edge-weighted graph $G(\mathcal{S}, \mathcal{C})$. (This observation was also used in [10] to obtain an algorithm finding restricted perfect phylogenies.)

The graph $G(\mathcal{S}, \mathcal{C})$ is a complete graph on $\mathcal{S}$, with edge weights $w(s_1, s_2) = k - |\{c \in \mathcal{C} \mid c(s_1) = c(s_2) = 1\}|$. It can be constructed in $O(\ell n)$ time.

THEOREM 2.9. *0–1–0 phylogenies for $(\mathcal{S}, \mathcal{C})$ are spanning trees of $G(\mathcal{S}, \mathcal{C})$ with weight $nk - \ell$. Furthermore, $G(\mathcal{S}, \mathcal{C})$ has no spanning trees of smaller weight.*

Proof. Every spanning tree of $G(S)$ has weight at least $nk - \ell$, since the contribution of each character c to the total weight is at least $(n-1) - (|S_c| - 1)$. Spanning trees of $G(S)$ with weight $nk - \ell$ correspond to trees in which each on-set S_c is connected (see [10]). □

Because of this correspondence, phylogenies can be constructed (or randomly sampled) by established algorithms for constructing (or randomly sampling) MSTs. Prim's algorithm [16, 9] constructs an MST of G in $O(m \log m)$ time, where m is the number of edges in G, and $m = \binom{n}{2}$ for $G = G(\mathcal{S}, \mathcal{C})$. If a faster algorithm is required, Karger, Klein and Tarjan's randomized algorithm constructs an MST, with high probability, in $O(m)$ time [12]. (Their model of computation is a unit-cost random-access machine with the restriction that the only operations allowed on edge weights are binary comparisons. See also the other algorithms discussed in [12].)

Given an unweighted n-vertex graph, an algorithm of Colbourn, Myrvold and Neufeld [4] selects a spanning tree uniformly at random in $O(M(n))$ time.[4] (Here $M(n) = O(n^{2.376})$ is the time needed to multiply two $n \times n$ matrices [5].) Colbourn and Jerrum note that the algorithm can be used to select an MST of a weighted graph G uniformly at random in $O(M(n))$ time: construct a random spanning tree on each connected component of the subgraph of G induced by the edges of minimum weight, put the spanning trees' edges into the final solution, contract the spanning trees, and repeat.

Compared with the atomic-set algorithm, the MST approach has the advantage of using an unusually widely understood and simple paradigm, a benefit echoed in the availability and efficiency of computer programs. However, it does not supply a structural representation of all possible phylogenies, nor a concise witness when no phylogeny exists.

3. Phylographs and phyloDAGs. Having considered the problem of constructing phylogenetic trees, we now turn to phylogenies that are not trees. In particular, we consider the phylograph and phyloDAG problems that were defined in the Introduction. In Section 3.1 we prove that it is hard to approximate the optimal phylograph within better than a logarithmic factor, and in Section 3.2 that the natural greedy algorithm gives an approximation within such a factor. In Section 3.3 we show both that it is hard to approximate the optimal phyloDAG within better than a logarithmic factor, and that in this case the natural greedy algorithm can perform very badly, even on average.

3.1. Hardness of approximation of phylograph. The following theorem states that unless nondeterministic quasi-polynomial time is equivalent to deterministic quasi-polynomial time, Minimum Phylograph is at best log-approximable:

THEOREM 3.1. *Let $c < 1/16$ be a constant. Unless* $\mathrm{NTIME}(n^{\mathrm{polylog}\, n}) = \mathrm{DTIME}(n^{\mathrm{polylog}\, n})$, *there is no polynomial-time algorithm that takes as input species $\mathcal{S}$ and characters $\mathcal{C}$ and outputs a phyloDAG $G = (\mathcal{S}, E)$ such that $|E|$ is within a factor of $c \log \ell$ of the minimum possible value.*

We use the following notation in the proof of Theorem 3.1.

Definition: The **neighborhood** of a vertex v of a graph $G = (V, E)$ is the set $N(v) = \{v\} \cup \{w : (v, w) \in E\}$. A **dominating set** of G is a set of vertices $D \subseteq V$ whose neighborhoods cover the graph: $\bigcup_{d \in D} N(d) = V$.

The following theorem, from [14], shows that it is hard to approximate **Minimum Dominating Set** to within better than a logarithmic factor; a similar result appears in [1].

THEOREM 3.2 (LUND AND YANNAKAKIS). *Let c be a constant in the range $0 < c < 1/4$. Unless* $\mathrm{NTIME}(n^{\mathrm{polylog}\, n}) = \mathrm{DTIME}(n^{\mathrm{polylog}\, n})$, *there is no polynomial-time algorithm that takes as input a graph G and outputs a dominating set D of G such that $|D|$ is within a factor of $c \log |V|$ of the minimum possible value.*

We are now ready to prove Theorem 3.1.

Proof. We use an approximation-preserving reduction from Minimum Dominating Set to Minimum Phylograph. Given an input $G = (V, E_G)$ with $|V| = \nu$, construct an instance P to Minimum Phylograph as follows: The species set is $\mathcal{S} = V \cup X$ where X is a set of ν^3 "auxiliary vertices". For each pair of vertices $\{v_1, v_2\} \in V^{(2)}$, define a character with on-set $\{v_1, v_2\}$. Thus any phylograph for P contains each edge in the complete graph on V. In addition, for each pair of vertices $(v, x) \in V \times X$ we define a character with on-set $\{x\} \cup N(v)$.

[4] Another randomized algorithm, due to Wilson [18], has an expected running time equal to the *mean hitting time* of the graph; this is often smaller than $M(n)$, but can be larger.

If $P_0 = (S, E_0)$ is an optimal phylograph for P and D_0 is a minimum dominating set for G, then $|E_0| = \binom{\nu}{2} + |X|\,|D_0|$. To see this, observe that the complete graph on V added to $V \times D_0$ is a phylograph for P, so $|E_0| \leq \binom{\nu}{2} + |X|\,|D_0|$. On the other hand, every phylograph for P has at least $\binom{\nu}{2}$ edges connecting species in V and has at least $|D_0|$ edges adjacent to each $x \in X$.

Suppose we had an algorithm A that could produce a phylograph (S, E_A) for P with $|E_A| \leq c\log(\ell)\,|E_0|$ edges. By the construction of P, some vertex $x \in X$ is connected to a dominating set D for G with $|D| \leq |E_A|/|X| \leq c\log(\ell)|E_0|/|X|$ edges. Since $|E_0| = \binom{\nu}{2} + |X|\,|D_0|$, we have

$$|D| \leq c\log(\ell)\,(\tbinom{\nu}{2} + |X|\,|D_0|)/|X|.$$

Thus (since $|X| = \nu^3$ and $|D_0| \geq 1$), $|D| \leq c(1 + o(1))\log(\ell)|D_0|$. Now note that $\ell = \nu(\nu-1) + 2\nu|X| + 2|E_G|\,|X| = O(\nu^5)$. Thus, $|D| \leq 5c(1+o(1))\log(\nu)|D_0|$, which is contrary to Theorem 3.2 if $c < 1/20$, unless $\mathrm{NTIME}(n^{\mathrm{poly}\log n}) = \mathrm{DTIME}(n^{\mathrm{poly}\log n})$. Using $|X| = \nu^{2+\epsilon}$ instead of ν^3 gives the constant $1/16$. □

3.2. Greedy algorithm for phylograph. There is a natural greedy algorithm for the Minimum Phylograph problem. In a phylograph, every character's induced subgraph consists of a single connected component, so the greedy algorithm "grows" a solution by iteratively adding an edge that maximally reduces the number of connected components.

The same notation needed to define the algorithm more precisely can be used in the proof of its quality. Given species $\mathcal{S}$ and characters C, and a set of edges $E \subseteq \mathcal{S}^{(2)}$ define the "cost" of E to be

$$f(E) = \sum_{c \in C} \mathrm{components}(S_c) - |C|,$$

where $\mathrm{components}(S_c)$ denotes the number of connected components in the subgraph of (S, E) induced by the on-set of c. Thus $f(\emptyset) = \sum_{c \in C} |S_c| - |C| = \ell - |C|$, and if E is a phylograph, $f(E) = \sum_{c \in C} 1 - |C| = 0$.

For any edge set E and any edge e, let $\Delta_E(e) = f(E) - f(E \cup \{e\})$ be the amount by which e decreases the cost f. The greedy algorithm begins with each species an isolated vertex, and iteratively adds the edge which maximally decreases the cost, until the cost is 0. (See the Appendix for pseudocode).

THEOREM 3.3. *Suppose that for species S and characters C, of total input length ℓ, the minimum phylograph $\{e(1), \ldots, e(r)\}$ has cardinality r. Then the greedy algorithm produces a phylograph E_G of size $|E_G| \leq r\ln(\ell - |C|)$.*

Proof. If we have any partial solution, adding in all r edges of a minimum phylograph will certainly yield a phylograph. Since r more edges are enough to complete the job, some edge (one of these, even) must take care of at least $1/r$th of the cost. If the initial cost was $f(\emptyset)$, and the greedy algorithm reduces it by $1 - 1/r$ at each step, after $r\ln f(\emptyset)$ steps the cost must be reduced below 1, and the algorithm must have terminated.[5]

More formally, for any edge set $E(0)$ define a series of sets $E(0) \subseteq \cdots \subseteq E(r)$, where $E(i) = E(0) \cup \{e(1), \ldots, e(i)\}$ and the edges $e(i)$ are those of the minimum phylograph. Note that $E(r)$ is a phylograph, since it contains the minimum phylograph. Because components (with

[5] We observe even now that the same approach will not work for phyloDAG. Since directed cycles are forbidden, chosen edges constrain the addition of future ones, and even if there was a solution of size r initially, there may not be once some edges have been chosen sub-optimally.

respect to any character) only become more connected as i increases, for any e, if $i \leq j$ then $\Delta_{E(i)}(e) \geq \Delta_{E(j)}(e)$. Thus for any starting set E_0,

$$\begin{aligned} r \cdot \max_{e \in \mathcal{S}^{(2)}} \Delta_{E(0)}(e) &\geq \sum_{i=1}^{r} \Delta_{E(0)}(e(i)) \\ &\geq \sum_{i=1}^{r} \Delta_{E(i-1)}(e(i)) \\ &= \sum_{i=1}^{r} [f(E(i-1)) - f(E(i))] \\ &= f(E(0)) - f(E(r)) \\ &= f(E(0)). \end{aligned}$$

Comparing the first and last quantities, we conclude that there always exists an edge e for which $\Delta_{E(0)}(e) \geq f(E_0)/r$.

Therefore the greedy algorithm reduces the cost by a factor $1 - 1/r$ at each step. Since the initial cost is $\ell - |C|$, the cost after $r \ln(\ell - |C|)$ steps of the greedy algorithm is at most $(1 - 1/r)^{r \ln(\ell - |C|)}(\ell - |C|) \leq 1$. The greedy algorithm therefore terminates within $r \ln(\ell - |C|)$ steps, producing a phylograph of the same size. □

This complements the result of Theorem 3.1: Minimum Phylograph is apparently hard to approximate to better than a factor of $(1/16)\log \ell$, but easy to approximate to a factor $\ln(\ell - |C|) \leq \ln \ell$. It would be of some interest to derive better bounds on the constant c, $\log_2(e)/16 < c \leq 1$, for which $(c \ln \ell)$-approximability is possible.

3.3. PhyloDAGs. We begin by observing that a phyloDAG cannot always be obtained by directing the edges of a phylograph. Consider four species with s_1 defined by characters (b, c, d), s_2 by (a, c, d), s_3 by (a, b, d), and s_4 by (a, b, c). The cycle s_1, s_2, s_3, s_4, s_1 is a 4-edge phylograph, but there is no way to direct the edges of the cycle to obtain a phyloDAG: any acyclic orientation will create two archetypes for some character's on-set.

We now prove the following theorem, which is analogous to Theorem 3.1.

THEOREM 3.4. . *Let $c < 1/16$ be a constant. Unless* $\mathrm{NTIME}(n^{\mathrm{poly}\log n}) = \mathrm{DTIME}(n^{\mathrm{poly}\log n})$, *there is no polynomial-time algorithm that takes as input species $\mathcal{S}$ and characters C and outputs a phyloDAG $G = (\mathcal{S}, E)$ such that $|E|$ is within a factor of $c \log \ell$ of the minimum possible value.*

Proof. The proof uses the same reduction as the proof of Theorem 3.1. Let E_0 be the edge set in an optimal phyloDAG for P. We must show that $E_0 = \binom{\nu}{2} + |X|\,|D_0|$. The direction that differs from the proof of Theorem 3.1 is showing that given a dominating set D_0, we can construct a phyloDAG of size $\binom{\nu}{2} + |X|\,|D_0|$. To do so, first construct a phylograph (as in the proof of Theorem 3.1). Then direct edges having both end-points in V according to a total order on the vertices in V, and direct all remaining edges from vertices in V toward vertices in X. The resulting digraph has no directed cycles and each character has a unique archetype. Therefore, it is a phyloDAG. The rest of the proof is the same as the proof of Theorem 3.1. □

As already noted, the natural greedy algorithm does not work well for phyloDAGs: the phyloDAG problem seems to be more difficult because the prohibition of cycles means that it is possible for the greedy algorithm to add a "bad" edge which prevents other "good" edges from being added later. In the remainder of this section, we give an example of a species set for which various natural greedy approaches for constructing a phyloDAG lead to an $\Omega(n)$ ratio between the size (number of edges) of the constructed phyloDAG and the size of the optimal

phyloDAG. A randomized strategy has an $\Omega(n)$ expected ratio and has a ratio of $\Omega(n/\log n)$ with high probability.

We construct a species set as follows. There are n species $s_1, \ldots, s_n$, and two distinguished species s' and s''. Now we add

- $2n$ characters shared by s' and s'';
- 2 characters shared by s' and s_i, for $i = 1, ..., n$
- 1 character shared by s'', s_i and s_j, for $1 \leq i, j \leq n, i \neq j$.

Duplicating characters forces the order in which a greedy algorithm connects species. We hide this duplication from an algorithm that checks for it by adding a set S_d of dummy species, where $|S_d| = \lceil \log(4n) \rceil$. There are $2^{\lceil \log(4n) \rceil} \geq 4n$ distinct subsets of S_d. We add one such subset to each of the $4n$ nonunique characters.[6] An optimal solution has $O(n)$ edges, consisting of an edge from s' to s'', edges from s' and s'' to each of the s_i, and edges from s' to each species in S_d.

A phyloDAG has exactly one archetype for each character. A greedy algorithm begins with each species an isolated node, thus an archetype for each character it contains. A natural edge to add in a greedy fashion is one that maximally reduces the number of archetypes (over all characters). Of course, we may not introduce directed cycles.

There may be times where we can choose the direction of the edge to be introduced (for example at the first iteration) and we show that the algorithm performs badly for any of the following strategies:

- The direction is chosen arbitrarily.
- The direction is chosen uniformly at random. (The expected performance of the algorithm is bad for this example, and the example can be modified so that the bad performance occurs with high probability.)
- The edge is directed out from the node with the larger number of characters. (This a natural way of breaking ties, since we expect ancestral nodes to have many characters.)

A greedy algorithm starts by putting an edge between s' and s'', and an edge between s' (or possibly s'') and each species in S_d. Then it adds edges between s' and the s_i. If directions are chosen arbitrarily we may assume that these edges are from s' to s'', and from each of the s_i to s'. Hence it is now impossible to add edges from s'' to any of the s_i, since they would create directed cycles. This means that in order to prevent there being two archetypes for a character shared by s'', s_i and s_j, species s_i must be connected to s_j by an edge. This results in $\binom{n}{2}$ edges.

Now consider the variant where the direction of an edge is chosen uniformly at random whenever it is equally good to direct it either way. With high probability (i.e. with complement probability that is exponentially small in n), there will be at least $n/4$ edges directed from the s_i's to s'. If the edge between s' and s'' is directed the wrong way (i.e. from s' to s'') then these s_i nodes will have to be connected in a clique, resulting in a quadratic number of edges. If we now consider a species set consisting of $\alpha \log(n)$ copies of the species set as described (for a positive constant α), we see that the optimal solution has $\Theta(n \log n)$ nodes and edges, and with probability at least $1 - n^{-\alpha}$, at least one of those copies will have the edge between s' and s'' directed the wrong way, resulting in $\Theta(n^2)$ edges.

If edges are directed away from nodes with higher numbers of characters, then the algorithm can be forced to take the "wrong" direction for the edges by adding dummy characters at the nodes from which we want the edges to be directed.

[6] An algorithm may also check for *domination*, where s_d contains a subset of the characters contained by s. We can remove the dominated species s_d from the instance and later direct an edge from s to s_d in the phylogeny for the reduced set. To avoid this situation here, we add a character $\{s_d(i), s_d(i+1)\}$ for $i = 1, \ldots, \lceil 4n \rceil$, which chains the dummies together. This does not change the asymptotic size of the optimal solution.

REFERENCES

[1] M. Bellare, S. Goldwasser, C. Lund and A. Russell, Efficient Probabilistically Checkable Proofs and Applications to Approximation, *Proceedings of the 25th ACM Symposium on the Theory of Computing*, pp. 294–304, 1993.

[2] C. Benham, S. Kannan, M. Paterson and T. Warnow, *Hen's Teeth and Whale's Feet: Generalized Characters and their Compatibility*, to appear in *Journal of Mathematical Biology*.

[3] H. Bodlaender, M. Fellows, and T. Warnow, *Two strikes against perfect phylogeny*, in Proceedings of the 19th International Colloquium on Automata, Languages, and Programming, Springer Verlag, Lecture Notes in Computer Science (1992), pp. 273–283.

[4] C.J. Colbourn, W.J. Myrvold, and E. Neufeld, "Two algorithms for unranking arborescences", *Journal of Algorithms*, to appear.

[5] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions", *Journal of Symbolic Computation*, Vol. 9, 1990, pp. 251-280.

[6] M. Crochemore and W. Rytter, Text Algorithms, Oxford University Press, 1994.

[7] R. Dulbecco, H. S. Ginsberg, *Virology* (2nd Ed), J.B. Lippincott Company, Philadelphia, 1988.

[8] P. Ferragina and R. Grossi, "Optimal on-line search and sublinear time update in string matching", Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, pp. 604–612, 1995.

[9] A. Gibbons, Algorithmic Graph Theory, Cambridge University Press, 1985.

[10] L.A. Goldberg, P.W. Goldberg, C.A. Phillips, E. Sweedyk, and T. Warnow, "Computing the phylogenetic number to find good evolutionary trees", *Proceedings of the 6th Symposium on Combinatorial Pattern Matching*, July 1995.

[11] D. Gusfield, "Efficient algorithms for inferring evolutionary trees", *Networks*, Vol. 21, 1991, pp. 19–28.

[12] D.R. Karger, P.N. Klein and R.E. Tarjan, A randomized linear-time algorithm to find minimum spanning trees, *JACM*, Vol. 42(2), 1995.

[13] J.O. Kephart and W.C. Arnold, "Automatic extraction of computer virus signatures", *Proceedings of the 4th Virus Bulletin International Conference*, R. Ford, ed., Virus Bulletin Ltd., Abingdon, England, 1994, pp. 179–194.

[14] C. Lund and M. Yannakakis, "On the hardness of approximating minimization problems", *Proceedings of the 25th ACM Symposium on the Theory of Computing*, pp. 286–293, 1993.

[15] A. Nijenhuis and H.S. Wilf, *Combinatorial Algorithms for Computers and Calculators, Second Ed.*, Academic Press, 1978.

[16] R.C. Prim, Shortest connection networks and some generalizations, *Bell System Tech. J.*, Vol. 36, 1957, pp. 1389–1401.

[17] M.A. Steel, *The complexity of reconstructing trees from qualitative characters and subtrees*, Journal of Classification, Vol. 9, 1992, pp. 91–116.

[18] D.B. Wilson, "Generating random spanning trees more quickly than the cover time", 1995, submitted.

Acknowledgements: We thank Phil MacKenzie, Tom Martin, Madhu Sudan, Luca Trevisan, and David Wilson for useful discussions.

4. Appendix. This appendix contains additional proofs and details to be read at the discretion of the program committee.

4.1. The greedy phylodag algorithm.

Let $i := 0$ and $E_G(0) := \emptyset$
While $f(E_G(i)) > 0$ do
begin
  Let $i := i + 1$
  Let e be an edge maximizing $\Delta_{E_G(i-1)}(e)$
  Let $E_G(i) := E_G(i-1) \cup \{e\}$
end
Return the set $E_G = E_G(i)$

4.2. An $O(\ell)$-time algorithm to compute an atomic set. We iteratively reduce a set of active species until we find an atomic set. Initially all species are active. Initialize a set of $k+1$

buckets to be empty. The buckets are numbered $0, 1, \ldots, k$, corresponding to characters $c_1, \ldots, c_k$ and a dummy c_0. Keep a variable *min* which points to the smallest bucket which contains at least 2 species (equal to ∞ at the start of each iteration). Also keep a list *touched* of nonempty buckets and a list *defining* of defining characters (initially empty). We place active species into buckets according to their "next" (in sorted order) character (initially the first one). When there are no further characters for a species, its "next" character is c_0. We keep a list *done* of species that have no further characters to process, and contain all the defining characters. Process the active species in arbitrary order. To process species s, suppose it's next character has index c. If $c \leq min$, add the species to bucket c. If this is the first entry in this bucket, add c to the *touched* list. If $min > c$, then, if this is the second species in the bucket, update *min* to c. After all active species have been processed, if $min = \infty$ (no two species match in their next character), then keep this active set, advancing each species to its next character after adding any species in the 0th bucket to the *done* list. Otherwise, the species in the *min* bucket are the new active set. Add *min* to the defining set (as long as it is nonzero). If there are exactly 2 active species, then this active set is an atomic set. If $min = 0$, then this active set along with any species on the *done* list) is an atomic set. Otherwise, empty the *done* list, empty each bucket on the *touched* list (discarding the species), reset *touched* to empty and *min* to ∞, and repeat the process with the active list using the next character for each species.

At each iteration, the species in the active set contain all the defining characters. In iterations where no new defining character is added (ie. when $min = \infty$), each species' next character is unique to that species. If the species ultimately is part of the atomic set, it will be the owner of this nondefining character. If $min = 0$, there are no more characters to process and each species in the current active set of size at least 2 (along with the *done* list contains all defining characters and all other characters are possessed by at most one of the species.

The algorithm requires $O(\ell)$ time. A species is a member of an active list at most once for each character it contains. Processing a member of the active list requires constant time. Emptying the buckets after each iteration of the algorithm requires time $O(a_i)$ where a_i is the size of the active set on iteration i. But $\sum_i a_i \leq \ell$. Therefore we can complete all iterations and find an atomic set in time $O(\ell)$.