Incomplete Directed Perfect Phylogeny*

Itsik Pe'er[†] Tal Pupko[‡] Ron Shamir[§] Roded Sharan[¶]

September 2003

Abstract

Perfect phylogeny is one of the fundamental models for studying evolution. We investigate the following variant of the model: The input is a species-characters matrix. The characters are binary and directed, i.e., a species can only gain characters. The difference from standard perfect phylogeny is that for some species the states of some characters are unknown. The question is whether one can complete the missing states in a way that admits a perfect phylogeny. The problem arises in classical phylogenetic studies, when some states are missing or undetermined. Quite recently, studies that infer phylogenies using inserted repeat elements in DNA gave rise to the same problem. Extant solutions for it take time $O(n^2m)$ for n species and m characters. We provide a graph theoretic formulation of the problem as a graph sandwich problem, and give near-optimal $\tilde{O}(nm)$-time algorithms for the problem. We also study the problem of finding a single, general solution tree, from which any other solution can be obtained by node splitting. We provide an algorithm to construct such a tree, or determine that none exists.

1 Introduction

When studying evolution, the divergence patterns leading from a single ancestor species to its contemporary descendants are usually modeled by a tree structure, called *phylogenetic tree*, or *phylogeny*. Extant species correspond to the tree leaves, while their common progenitor corresponds to the root. Internal

*Portions of this paper appeared in the Proceedings of the Eleventh Annual Symposium on Combinatorial Pattern Matching [20]. Other parts appeared in the Proceedings of the Eighth Scandinavian Workshop on Algorithm Theory [21].

[†]Dept. of Molecular Genetics, Weizmann Institute of Science, Rehovot 76100, Israel. peer@wicc.weizmann.ac.il.

[‡]Dept. of Cell Research and Immunology, George S. Wise Faculty of Life Sciences, Tel Aviv University, Tel Aviv 69978, Israel. talp@post.tau.ac.il.

[§]School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. rshamir@tau.ac.il.

[¶]International Computer Science Institute, 1947 Center St., Berkeley CA 94704. roded@icsi.berkeley.edu.

nodes correspond to hypothetical ancestral species, which putatively split up and evolved into distinct species. Tree branches model changes through time of the hypothetical ancestor species. The common case is that one has information regarding the leaves, from which the phylogenetic tree is to be inferred. This task, called *phylogenetic reconstruction* (cf. [8]), was one of the first algorithmic challenges posed by biology, and the computational community has been dealing with problems of this flavor for over three decades (see, e.g., [13]).

The character-based approach to tree reconstruction describes contemporary species by their attributes or *characters*. Each character takes on one of several possible *states*. The input is represented by a matrix $\mathcal{A}$ where a_{ij} is the state of character j in species i, and the i-th row is the *character vector* of species i. The output sought is a hypothesis regarding evolution, i.e., a phylogenetic tree along with the suggested character vectors of the internal nodes. This output must satisfy properties specified by the problem variant.

One important class of phylogenetic reconstruction problems concerns finding a *perfect phylogeny*. The property required from such a phylogeny is that for each possible character state, the set of all nodes that have that state induces a connected subtree. The general perfect phylogeny problem is NP-hard [5, 23]. When considering the number of possible states per character as a parameter, the problem is fixed parameter tractable [2, 16]. For *binary characters*, having only two possible states, perfect phylogeny is linear-time solvable [12].

When no perfect phylogeny is possible, one option is to seek a largest subset of characters which admits a perfect phylogeny. Characters in such a subset are said to be *compatible*. Compatibility problems have been studied extensively (see, e.g., [18]).

Another common optimization approach to phylogenetic reconstruction is the *parsimony* criterion. It calls for a solution with fewest state changes altogether, counting a change whenever the state of a character changes between a species and its ancestor species. This problem is known to be NP-hard [9]. A variant introduced by Camin and Sokal [6] assumes that characters are binary and *directed*, namely, only $0 \to 1$ changes may occur on any path from the root to a leaf. Denoting by 1 and 0 the presence and absence, respectively, of the character, this means that characters can only be gained throughout evolution. Another related binary variant is Dollo parsimony [7, 22], which assumes that a $0 \to 1$ change may happen only once, i.e., a character can be gained once, but it can be lost several times. Both of these variants are polynomially solvable (cf. [8]).

In this paper, we discuss a variant of binary perfect phylogeny which combines assumptions of both Camin-Sokal parsimony and Dollo parsimony. The setup is as follows: The characters are binary, directed, and can be gained only once. As in perfect phylogeny, the input is a matrix of character vectors, with the difference that some character states are missing. The question is whether one can complete the missing states in a way admitting a perfect phylogeny. We call this problem *Incomplete Directed Perfect phylogeny (IDP).*

The problem of handling incomplete phylogenetic data arises whenever some of the data are missing. It is also encountered in the context of morphological characters, where for some species it may be impossible to reliably assign a state to a character. The problem of determining whether a set of incomplete *undirected* characters is compatible was shown to be NP-complete, even in the case of binary characters [23]. Indeed, the popular PAUP software package [24] provides an exponential solution to the problem by exhaustively searching the space of missing states.

Quite recently, a novel kind of genomic data has given rise to the same problem: Nikaido et al. [19] use inserted repetitive genomic elements, particularly SINEs (Short Interspersed Nuclear Elements), as a source of evolutionary information. SINEs are short DNA sequences that were copied and randomly reinserted into various genomic loci during evolution. The distinct insertion loci are identifiable by the flanking sequences on both sides of the insertion site (see Figure 1). These insertions are assumed to be unique events in evolution, because the odds of having separate insertion events at the very same locus are negligible. Furthermore, a SINE insertion is assumed to be irreversible, i.e., once a SINE sequence has been inserted somewhere along the genome, it is practically impossible for the exact, complete SINE to leave that specific locus. However, the inserted segment along with its flanking sequences may be lost when a large genomic region, which includes them, is deleted. In that case we do not know whether a SINE insertion had occurred in the missing site prior to its deletion. One can model such data by assigning to each locus a character, whose state is '1' if the SINE occurred in that locus, '0' if the locus is present but does not contain the SINE, and '?' if the locus is missing. The resulting reconstruction problem is precisely Incomplete Directed Perfect phylogeny.

The IDP problem becomes polynomial when the characters are directed: Benham et al. [4] studied the compatibility problem on generalized characters. Their work implies an $O(n^2m)$-time algorithm for IDP, where n and m denote the number of species and characters, respectively. Another problem related to IDP is the consensus tree problem [3, 14]. This problem calls for constructing a consensus tree from binary subtrees, and is solvable in polynomial time. One can reduce IDP to the latter problem, but the reduction itself takes $\Omega(n^2m)$ time.

Our approach to the IDP problem is graph theoretic. We first provide several graph and matrix characterizations for solvable instances of binary directed perfect phylogeny. We then reformulate IDP as a *graph sandwich* problem: The input data is recast as two nested graphs, and solving IDP is shown to be equivalent to finding a graph of a particular type "sandwiched" between them. This formulation

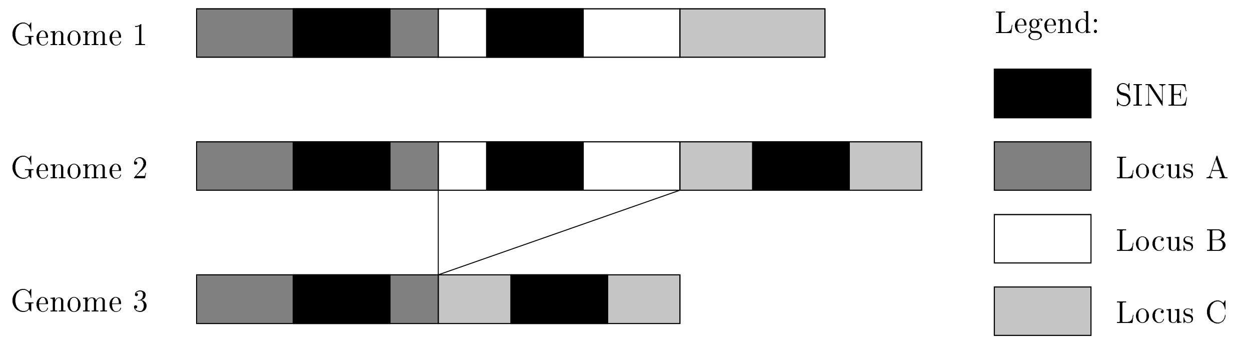

Figure 1: SINEs (black boxes) repeat in different loci (different shades of grey) across distinct genomes. A SINE insertion transformed Genome 1 into Genome 2. A deletion of a locus transformed Genome 2 into Genome 3. Given Genomes 1 and 3, we can identify that the SINE on locus C is not present in Genome 1, by its flanking sequence. However, locus B is missing in Genome 3.

allows us to devise efficient algorithms for IDP.

We provide two algorithms for IDP, which we call Algorithms A and B. Algorithm A has two possible implementations: deterministic and randomized. Its deterministic complexity is $O(nm + k\log^2(n+m))$, for an instance with k 1-entries in the species-characters matrix. The randomized version of Algorithm A takes $O(nm+k\log(l^2/k)+l(\log l)^3\log\log l)$ expected time, where $l = n+m$. Algorithm B is deterministic and takes $O(l^2\log l)$ time. For both algorithms, the improved complexity is obtained by using dynamic data structures for maintaining the connected components of a graph [25, 14, 15]. Since an $\Omega(nm)$ lower bound was shown by Gusfield for directed binary perfect phylogeny [12], our algorithms have near optimal time complexity.

We also study the issue of multiple solutions for IDP. Often there is more than one phylogeny that is consistent with the data. When the input matrix is complete, and has a solution, there is always a tree $\mathcal{T}^*$ that is *general*, i.e., it is a solution, and every other tree consistent with the data can be obtained from $\mathcal{T}^*$ by node splitting. In other words, $\mathcal{T}^*$ describes all the definite information in the data, and ambiguities (nodes with three or more children) can be resolved by additional information. This is not always the case if the data matrix is incomplete: There may or may not be a general solution tree. In that case, using a particular solution and additional information, one can conclude that the data is inconsistent, even though the additional information may be consistent with another solution. It is thus desirable to know if a general solution exists and to generate such a solution if the answer is positive.

We provide answers to both questions. We prove that Algorithm A provides the general solution of a problem instance, if such exists. We also give an algorithm which determines if the solution $\mathcal{T}$ produced by Algorithm A is indeed general. The complexity of the latter algorithm is $O(nm+kd)$, where d denotes the maximum out-degree of $\mathcal{T}$.

The paper is organized as follows: In Section 2 we provide some preliminaries, and formalize the IDP problem. In Section 3 we characterize binary matrices admitting a directed perfect phylogeny, and provide the graph sandwich formulation for IDP. In Section 4 we present algorithms for IDP. Finally in Section 5 we analyze the generality of the solution produced by Algorithm A.

2 Preliminaries

We first specify some terminology and notation. We reserve the terms *nodes* and *branches* for trees, and use the terms *vertices* and *edges* for other graphs. Matrices are denoted by an upper-case letter, while their elements are denoted by the corresponding lower-case letter.

Let $G = (V, E)$ be a graph. We denote its set of vertices also by $V(G)$, and its set of edges also by $E(G)$. Let $\emptyset \neq V' \subseteq V$ be a subset of the vertices. The subgraph *induced* by V' is the graph $(V', E \cap (V' \times V'))$. We say that V' is *connected* in G, if V' is contained in some connected component of G. The *length* of a path in G is the number of edges along it.

Let $\mathcal{T}$ be a rooted tree with leaf set S, where branches are directed from the root towards the leaves. The *out-degree* of a node x in $\mathcal{T}$ is its number of children, and is denoted by $d(x)$. For a node x in $\mathcal{T}$ we denote the leaf set of the subtree rooted at x by $L(x)$. $L(x)$ is called a *clade* of $\mathcal{T}$. For consistency, we consider $\emptyset$ to be a clade of $\mathcal{T}$ as well, and call it the *empty clade*. $S, \emptyset$ and all singletons are called *trivial clades*. We denote by $triv(S)$ the collection of all trivial clades. Two sets are said to be *compatible* if they are either disjoint, or one of them contains the other.

Observation 1 *(cf. [18]) A collection $\mathcal{S}$ of subsets of a set S is the set of clades of some tree over S if and only if $\mathcal{S}$ contains $triv(S)$ and its subsets are pairwise compatible.*

A tree $\mathcal{T}$ is uniquely characterized by its set of clades. The transformation between a branch-node representation of a tree and a list of its clades is straightforward. Thus, we hereafter identify a tree with the set of its clades, and use the notation $S \in \mathcal{T}$ to indicate that S is a clade of $\mathcal{T}$. If $\hat{S}$ is a subset of the leaves of $\mathcal{T}$, then the subtree of $\mathcal{T}$ *induced* on $\hat{S}$ is the collection $\{\hat{S} \cap S' : S' \in \mathcal{T}\}$ (which defines a tree).

Throughout the paper we denote by $S = \{s_1, \ldots, s_n\}$ the set of all species and by $C = \{c_1, \ldots, c_m\}$ the set of all (binary) characters. For a graph K, we define $C(K) \equiv C \cap V(K)$ and $S(K) \equiv S \cap V(K)$. Let $\mathcal{B}_{n\times m}$ be a binary matrix whose rows correspond to species, each row being the character vector of its corresponding species. That is, $b_{ij} = 1$ if and only if the species s_i has the character c_j. A *phylogenetic tree for $\mathcal{B}$* is a rooted tree $\mathcal{T}$ with n leaves corresponding to the n species of S, such that each character is associated with a clade S' of $\mathcal{T}$, and the following properties are satisfied:

(1) If c_j is associated with S' then $s_i \in S'$ if and only if $b_{ij} = 1$.

(2) Every non-trivial clade of $\mathcal{T}$ is associated with at least one character.

For a character c, the node x of $\mathcal{T}$ whose clade $L(x)$ is associated with c, is called the *origin* of c with respect to $\mathcal{T}$. Characters associated with $\emptyset$ have no origin.

A $\{0, 1, ?\}$ matrix is called *incomplete*. For convenience, we consider binary matrices as incomplete. Let $\mathcal{A}_{n \times m}$ be a $\{0, 1, ?\}$ matrix in which $a_{ij} = 1$ if s_i has c_j, $a_{ij} = 0$ if s_i lacks c_j, and $a_{ij} =?$ if it is not known whether s_i has c_j. For a character c_j and a state $x \in \{0, 1, ?\}$, the *x-set* of c_j in $\mathcal{A}$ is the set of species $\{s_i \in S : a_{ij} = x\}$. c_j is called a *null character* if its 1-set is empty. For subsets $\hat{S} \subseteq S$ and $\hat{C} \subseteq C$, define $\mathcal{A}|_{\hat{S},\hat{C}}$ to be the submatrix of $\mathcal{A}$ induced on $\hat{S} \cup \hat{C}$.

A binary matrix $\mathcal{B}$ is called a *completion* of $\mathcal{A}$ if $a_{ij} \in \{b_{ij}, ?\}$ for all i, j. Thus, a completion replaces all the ?-s in $\mathcal{A}$ by zeroes and ones. If $\mathcal{B}$ has a phylogenetic tree $\mathcal{T}$, we say that $\mathcal{T}$ is a *phylogenetic tree for* $\mathcal{A}$ as well. We also say that $\mathcal{T}$ *explains* $\mathcal{A}$ *via* $\mathcal{B}$, and that $\mathcal{A}$ is *explainable*. An example of these definitions is given in Figure 2.

$$
\begin{array}{cc}
 & \text{Characters} \\
\text{Species} & \begin{array}{|ccccc|} \hline 1 & ? & 0 & 0 & 1 \\ ? & ? & 0 & 1 & 0 \\ ? & 0 & 1 & ? & ? \\ \hline \end{array}
\end{array}
\qquad
\begin{array}{cc}
 & \begin{array}{ccccc} c_1 & c_2 & c_3 & c_4 & c_5 \end{array} \\
\begin{array}{c} s_1 \\ s_2 \\ s_3 \end{array} & \begin{array}{|ccccc|} \hline 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ \hline \end{array}
\end{array}
$$

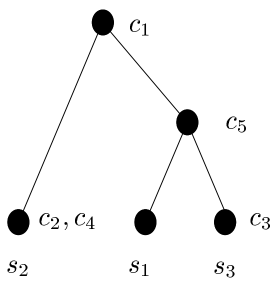

Figure 2: Left to right: An incomplete matrix $\mathcal{A}$, a completion $\mathcal{B}$ of $\mathcal{A}$, and a phylogenetic tree that explains $\mathcal{A}$ via $\mathcal{B}$. Each character is written to the right of its origin node.

The following lemma, closely related to Observation 1, has been proven independently by several authors:

Lemma 2 (cf. [18]) *A binary matrix $\mathcal{B}$ has a phylogenetic tree if and only if the 1-sets of every two characters are compatible.*

An analogous lemma holds for undirected characters (cf. [12]). In contrast, for incomplete matrices, even if every pair of columns has a phylogenetic tree, the full matrix might not have one. An example of such a matrix was provided in [8] for incomplete undirected characters. We provide a simpler example for incomplete *directed* characters in Figure 3. Indeed, if we consider columns 1 and 2 in the example then the missing entry on column 1 should be completed to 1 and the one on column 2 should be completed to 0. However, in such a completion the characters on columns 2 and 3 are not compatible.

$$
\begin{array}{cc}
 & \text{Characters} \\
\text{Species} & \begin{array}{|ccc|}
\hline
1 & 0 & 0 \\
1 & 1 & 0 \\
? & 1 & 1 \\
0 & ? & 1 \\
\hline
\end{array}
\end{array}
$$

Figure 3: An incomplete matrix which has no phylogenetic tree although every pair of its columns has one.

We are now ready to state the IDP problem:

Incomplete Directed Perfect Phylogeny (IDP):

Instance: An incomplete matrix $\mathcal{A}$.

Goal: Find a phylogenetic tree for $\mathcal{A}$, or determine that no such tree exists.

In the next section we characterize complete binary matrices that admit a perfect phylogeny. In the section that follows we present our algorithmic approaches for IDP.

3 Characterizations of Explainable Binary Matrices

3.1 Forbidden Subgraph Characterization

Let $\mathcal{B}$ be a species-characters binary matrix of order $n \times m$. Construct the bipartite graph $G(\mathcal{B}) = (S, C, E)$ with $E = \{(s_i, c_j) : b_{ij} = 1\}$. A Σ *subgraph* is an induced subgraph of $G(\mathcal{B})$ that includes three vertices from S, two vertices from C, and exactly four edges, forming a path of length four in $G(\mathcal{B})$ (see Figure 4). A bipartite graph with no induced Σ subgraph is called Σ-*free*.

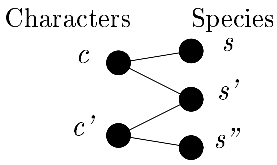

Figure 4: The Σ subgraph.

The following theorem restates Lemma 2 in terms of graph theory.

Theorem 3 *$\mathcal{B}$ has a phylogenetic tree if and only if $G(\mathcal{B})$ is Σ-free.*

Corollary 4 *Let $\hat{S} \subseteq S$ and $\hat{C} \subseteq C$ be subsets of the species and characters, respectively. If $\mathcal{A}$ is explainable then so is $\mathcal{A}|_{\hat{S},\hat{C}}$.*

Observation 5 *Let $\mathcal{A}$ be a matrix explained by a tree $\mathcal{T}$ and Let $\hat{S} = L(x)$ be a clade in $\mathcal{T}$, where x is a node of $\mathcal{T}$. Then the submatrix $\mathcal{A}|_{\hat{S},C}$ is explained by the subtree of $\mathcal{T}$ rooted at x.*

For a subset $S' \subseteq S$ of species, we say that a character c is *S'-universal* in $\mathcal{B}$, if its 1-set (in $\mathcal{B}$) contains S'.

Proposition 6 *If $G(\mathcal{B})$ is connected and Σ-free, then there exists a character which is S-universal in $\mathcal{B}$.*

Proof: Suppose to the contrary that $\mathcal{B}$ has no S-universal character. Consider the collection of all 1-sets of characters in $\mathcal{B}$. Let c be a character whose 1-set is maximal with respect to inclusion in this collection. Let s'' be a species which lacks c. Since $G(\mathcal{B})$ is connected, there exists a path from s'' to c in $G(\mathcal{B})$. Consider a shortest such path P. Since $G(\mathcal{B})$ is bipartite, the length of P is odd. However, P cannot be of length 1, by the choice of s''. Furthermore, if P is of length greater than 3, then its first 5 vertices induce a Σ subgraph, a contradiction. Thus $P = (s'', c', s', c)$ must be of length 3. By maximality of the 1-set of c, it is not contained in the 1-set of c'. Hence, there exists a species s which has the character c but lacks c'. Together with s, the vertices of P induce a Σ subgraph, as depicted in Figure 4, a contradiction. ■

Let Ψ be a graph property. In the Ψ *sandwich problem* one is given a vertex set V and a partition of $V \times V$ into three disjoint subsets: E_0 - *forbidden edges*, E_1 - *mandatory edges*, and $E_?$ - *optional edges*. The objective is to find a supergraph of (V, E_1) which satisfies Ψ and contains no forbidden edges. Hence, the required graph (V, F) must be "sandwiched" between (V, E_1) and $(V, E_1 \cup E_?)$. The reader is referred to articles [10, 11] for a discussion of various sandwich problems.

For the property "containing no induced Σ subgraph" (a property of bipartite graphs) the sandwich problem is defined as follows:

Σ-free sandwich:

Instance: A vertex set $V = S \cup C$ with $S \cap C = \emptyset$, and a partition $E_0 \cup E_? \cup E_1$ of $S \times C$.

Goal: Find a set of edges F such that $F \supseteq E_1$, $F \cap E_0 = \emptyset$, and the graph (V, F) is Σ-free, or determine that no such set exists.

Theorem 3 motivates looking at the IDP problem with input $\mathcal{A}$ as an instance $((S, C), E_0^{\mathcal{A}}, E_?^{\mathcal{A}}, E_1^{\mathcal{A}})$ of the Σ-free sandwich problem. Here, $E_x^{\mathcal{A}} = \{(s_i, c_j) : a_{ij} = x\}$, for $x = 0, ?, 1$. In the sequel, we omit the superscript $\mathcal{A}$ when it is clear from the context.

Proposition 7 *The Σ-free sandwich problem is equivalent to IDP.*

Note, that there is an obvious 1-1 correspondence between completions of $\mathcal{A}$ and possible solutions of the corresponding sandwich instance $((S, C), E_0, E_?, E_1)$. Hence, in the sequel we refer to matrices and their corresponding sandwich instances interchangeably.

3.2 Forbidden Submatrix Characterizations

A binary matrix $\mathcal{B}$ is called *good* if it can be decomposed as follows:
(1) Its left $k_1 \geq 0$ columns are all ones.
(2) There exist good matrices $\mathcal{B}_1, \ldots, \mathcal{B}_l$, such that the rest (0 or more) of the columns of $\mathcal{B}$ form the block-structure illustrated in Figure 5.

A matrix $\mathcal{A}$ is *canonical* if $\mathcal{A} = [\mathcal{B}, \mathcal{C}]$ where $\mathcal{B}$ is a zero submatrix and $\mathcal{C}$ is good. We say that a matrix $\mathcal{B}$ *avoids* a matrix $\mathcal{X}$, if no submatrix of $\mathcal{B}$ is identical to $\mathcal{X}$.

Theorem 8 *Let $\mathcal{B}$ be a binary matrix. The following are equivalent:*

1. *$\mathcal{B}$ has a phylogenetic tree.*
2. *$G(\mathcal{B})$ is Σ-free.*
3. *Every matrix obtained by permuting the rows and columns of $\mathcal{B}$ avoids the following matrix:*

$$\mathcal{Z} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

4. *There exists an ordering of the rows and columns of $\mathcal{B}$ which yields a canonical matrix.*
5. *There exists an ordering of the rows and columns of $\mathcal{B}$ so that the resulting matrix avoids the following matrices:*

$$\mathcal{X}_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \mathcal{X}_2 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, \mathcal{X}_3 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \mathcal{X}_4 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Proof:

1$\Leftrightarrow$2 Theorem 3.

2$\Leftrightarrow$3 Trivial.

1⇒4 Suppose $\mathcal{T}$ is a tree that explains $\mathcal{B}$. Assign to each node of $\mathcal{T}$ an index which equals its position in a preorder visit of $\mathcal{T}$. Sort the characters according to the indices of their origin nodes, letting null characters come first. Sort the species according to the indices of their corresponding leaves in $\mathcal{T}$. The result is a canonical matrix.

4⇒5 It is easy to verify that canonical matrices avoid $\mathcal{X}_1, \ldots, \mathcal{X}_4$.

5⇒3 Suppose to the contrary that $\mathcal{B}$ has an ordering of its rows and columns, so that rows i_1, i_2, i_3 and columns j_1, j_2 of the resulting matrix form the submatrix $\mathcal{Z}$. Consider the permutations $\theta_{row}, \theta_{col}$ of the rows and columns of $\mathcal{B}$, respectively, which yield a matrix avoiding $\mathcal{X}_1, \ldots, \mathcal{X}_4$. In this ordering, row $\theta_{row}(i_1)$ necessarily lies between rows $\theta_{row}(i_2)$ and $\theta_{row}(i_3)$ or, else, the submatrix $\mathcal{X}_4$ occurs. Suppose that $\theta_{row}(i_2) < \theta_{row}(i_3)$ and $\theta_{col}(j_1) < \theta_{col}(j_2)$, then $\mathcal{X}_3$ occurs, a contradiction. The remaining cases are similar.

■

Note that a matrix which avoids $\mathcal{X}_4$ has the consecutive ones property in columns. Gusfield [12, Theorem 3] has proven that a matrix which has an *undirected* perfect phylogeny can be reordered so as to satisfy this property. In fact, for explainable binary matrices, the reordering used by Gusfield's proof essentially generates a canonical matrix. Note also that Σ-free graphs are bipartite convex as they avoid $\mathcal{X}_1$, $\mathcal{X}_2$ and $\mathcal{X}_3$ (see, e.g., [1]).

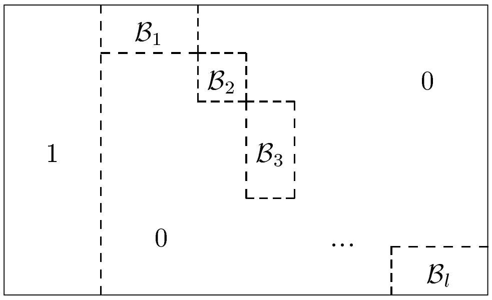

Figure 5: Construction of a good matrix. Each $\mathcal{B}_i$ is a good matrix. A canonical matrix is formed from it by appending columns of zeros on the left.

The reader is referred to article [17] for other problems of permuting matrices to avoid forbidden submatrices.

4 Algorithms for Solving IDP

We now return to the problem of completing an incomplete binary matrix. Let $\mathcal{A}$ be the input matrix, and define $G(\mathcal{A}) = (S, C, E_1^{\mathcal{A}})$. For a nonempty subset $S' \subseteq S$, we say that a character is *S'-semi-universal* in $\mathcal{A}$ if its 0-set does not intersect S'. The following lemmas motivate a divide and conquer approach to IDP, which is the basis of our algorithms for solving it.

Lemma 9 *Let $\mathcal{A}$ be an incomplete matrix with a Σ-free completion $\mathcal{B}$. Let c be S-semi-universal in $\mathcal{A}$. Let $\mathcal{B}'$ be the matrix obtained from $\mathcal{B}$ by setting all entries in the column of c to 1. Then $\mathcal{B}'$ is also a Σ-free completion of $\mathcal{A}$.*

Proof: Suppose to the contrary that $\{s_1, c_1, s_2, c_2, s_3\}$ induce a Σ subgraph in $G(\mathcal{B}')$. Since $G(\mathcal{B})$ is Σ-free, if follows that at least one of the Σ edges was added to $\mathcal{B}'$. But then one of c_1 and c_2 is c, a contradiction. ■

Lemma 10 *Let $\mathcal{A}$ be an incomplete matrix with a Σ-free completion $\mathcal{B}$. Let $(K_1, \ldots, K_r)$ be a partition of $S \cup C$ such that each K_i is a union of one or more connected components of $G(\mathcal{A})$. Let $\mathcal{B}'$ be the matrix obtained from $\mathcal{B}$ by setting all entries between vertices of K_i and K_j to 0, for all $i \neq j$. Then $\mathcal{B}'$ is also a Σ-free completion of $\mathcal{A}$.*

Proof: Suppose to the contrary that $\{s_1, c_1, s_2, c_2, s_3\}$ induce a Σ subgraph in $G(\mathcal{B}')$. Then one of the non-edges (s_1, c_2) or (c_1, s_3) contains one vertex from K_i and the other from K_j, for $i \neq j$. It follows that there is a path in $G(\mathcal{B}')$ between a vertex of K_i and a vertex of K_j, a contradiction. ■

We now describe two efficient $\tilde{O}(nm)$-time algorithms for solving IDP.

4.1 Algorithm A

Algorithm A is described in Figure 6. The algorithm outputs the set of non-empty clades of a tree explaining $\mathcal{A}$, or outputs *False* if no such tree exists. The algorithm is recursive and is initially called with Alg_A($\mathcal{A}$) .

Theorem 11 *Algorithm A correctly solves IDP.*

Proof: Suppose that the algorithm outputs *False*. Then there exists a recursive call Alg_A($\mathcal{A}'$) in which the graph $G' = (S', C', E')$, obtained in Step 1b, was found to be connected. Suppose to the contrary that $\mathcal{A}$ has a phylogenetic tree. Then by Corollary 4, there exists some edge set F^*, which

Alg_A($\mathcal{A} = ((S, C), E_0, E_?, E_1)$)**:**

1. **If** $|S| > 1$ then **do**:

   (a) Remove all S-semi-universal characters and all null characters from $G(\mathcal{A})$.

   (b) If the resulting graph G' is connected then output *False* and **halt**.

   (c) Otherwise, let $K_1, \dots, K_r$ be the connected components of G', and let $\mathcal{A}_1, \dots, \mathcal{A}_r$ be the corresponding submatrices of $\mathcal{A}$.

   (d) **For** $i = 1, \dots, r$ **do**: Alg_A($\mathcal{A}_i$) .

2. Output S.

Figure 6: Algorithm A for solving IDP.

solves $\mathcal{A}'$. The graph $G^* = (S', C', F^*)$ is connected and by Theorem 3, it is also Σ-free. Therefore, by Proposition 6 there exists an S'-universal character in G^*. That character must be S'-semi-universal in $\mathcal{A}'$. By Algorithm A this vertex should have been removed at Step 1a, a contradiction.

To prove the other direction, we will show that if the algorithm outputs a collection $\mathcal{T}' = \{S_1, \dots, S_l\}$ of sets, then $\mathcal{T} = \mathcal{T}' \cup \{\emptyset\}$ is a tree which explains $\mathcal{A}$. We first prove that the collection $\mathcal{T}$ of sets is pairwise compatible, implying by Observation 1 that $\mathcal{T}$ is a tree. Associate with each S_i the recursive call Alg_A($\mathcal{A}_i$) at which it was output. Observe that each such call makes recursive calls associated with disjoint subsets of S_i. By induction, it follows that $S_i \subseteq S_j$ if and only if the recursive call associated with S_i is nested within the one associated with S_j. Otherwise, $S_i \cap S_j = \emptyset$. Hence, $S_1, \dots, S_l$ are pairwise compatible and, thus, $\mathcal{T}$ is a tree.

It remains to show that $\mathcal{T}$ is a phylogenetic tree for $\mathcal{A}$. Associate each null character with the empty clade. Each other character $\hat{c}$ is removed at Step 1a only once in the course of the algorithm, during some recursive call Alg_A($\hat{\mathcal{A}}$). Associate $\hat{c}$ with the clade $\hat{S}$ which was output at that recursive call. Observe that each non-trivial clade $\hat{S} \in \mathcal{T}$ is associated with at least one character. Finally, define a binary matrix $\mathcal{B}_{n\times m}$ with $b_{sc} = 1$ if and only if s belongs to the clade S_c associated with c. Since $a_{sc} \neq 1$ for all $s \notin S_c$ and $a_{sc} \neq 0$ for all $s \in S_c$, $\mathcal{B}$ is a completion of $\mathcal{A}$. The claim follows. ■

Let $h \leq \min\{m, n\}$ be the height of the reconstructed tree. Each recursive call increases the height of the output tree by at most one. The work at each level of the tree requires: (1) Finding semi-universal vertices; and (2) finding connected components in disjoint graphs whose total number of edges is at most mn. Hence, the total work is $O(mn)$ per level, and a naive implementation requires $O(hmn)$ time. We give a faster implementation below.

Theorem 12 *Algorithm A has a deterministic implementation which takes* $O(nm + |E_1| \log^2(n+m))$ *time, and a randomized implementation which takes* $O(nm + |E_1| \log(l^2/|E_1|) + l(\log l)^3 \log\log l)$ *expected time, where* $l = n + m$.

Proof: For the complexity proof we give an alternative, non-recursive implementation of Algorithm A, shown in Figure 7. This iterative version mimics the recursive one, but traverses the tree of recursive calls in a breadth first manner, rather than a depth first manner. Consequently, the implementation deals with a single graph, rather than a different graph per each recursive call. The reduction in complexity is primarily due to the use of an efficient dynamic data structure for graph connectivity. The data structure maintains the connected components of the graph while edge deletions occur.

We now analyze the running time of this implementation. Step 1 takes $O(nm)$ time. Each iteration of the 'while' loop (Step 2) splits the (potential) clades added in the previous one. Thus, Algorithm A performs one iteration of this type per each level of the tree returned, and at most h iterations.

Step 2b requires explicitly computing the connected components of G. Both data structures that we use for storing the connected components of G (see below) maintain a spanning tree for each connected component of G, and allow computing the connected components in $O(n+m)$ time per iteration, or $O(h(m+n)) = O(nm)$ time in total.

The loop of Step 2c is performed at most $\min\{2n-1, m\}$ times altogether, as in each (successful) iteration at least one character is removed from G (Step 2(c)vii), and at least one clade is added to $\mathcal{T}$. Thus, Step 2(c)i takes $O(\min\{n,m\})$ time altogether, and Step 2(c)ii takes $O(nm)$ time in total. Step 2(c)iii takes $O(nm)$ time in total, as it considers each species-character pair only once throughout the execution of the algorithm.

In order to analyze the complexity of Step 2(c)iv, observe that the following invariants hold in this step for each character $c \in C(K_i)$:

- $d_c^? = |\{(s,c) \in E_? | s \in S(K_i)\}|$, as guaranteed by Step 2(c)iii.
- $d_c^1 = |\{(s,c) \in E_1 | s \in S(K_i)\}| = |\{(s,c) \in E_1 | s \in S\}|$, as initialized in Step 1b, since species are never removed, and each species adjacent to c must be in its connected component until c is removed.

Given $d_c^1, d_c^?$ and $|S(K_i)|$, one can check in $O(1)$ time whether c is $S(K_i)$-semi-universal, and thus Step 2(c)iv takes $O(|C(K_i)|)$ time, or $O(hm)$ time in total.

Alg_A_fast$(\mathcal{A} = ((S,C), E_0, E_?, E_1))$ **:**

1. Initialize:

   (a) Set $t \leftarrow 0$, $\mathcal{K}^0 \leftarrow \{S \cup C\}$, $G \leftarrow G(\mathcal{A})$, $\mathcal{T} \leftarrow triv(S)$.

   (b) **For** each character c, and $i \in \{1, ?\}$ **do:**

   Set $d_c^i \leftarrow |\{s \in S | (s,c) \in E_i\}|$.

   (c) Remove all S-semi-universal and all null characters from G.

   (d) Initialize a data structure for maintaining the connected components of G.

2. **While** $E(G) \neq \emptyset$ **do**:

   (a) Increment t.

   (b) Explicitly compute the set $\mathcal{K}^t$ of connected components $K_1, \ldots, K_r$ of G.

   (c) **For** each connected component $K_i \in \mathcal{K}^t$ such that $|E(K_i)| \geq 1$ **do**:

   i. Pick any character $c' \in C(K_i)$.

   ii. Compute $S' = S(K') \setminus S(K_i)$, where K' is the component in $\mathcal{K}^{t-1}$ which contains c'.

   iii. **For** each species-character pair $(s,c) \in S' \times C(K_i)$ **do**:

   If $(s,c) \in E_?$ **then** decrement $d_c^?$.

   iv. Compute the set U of all characters in K_i which are $S(K_i)$-semi-universal in $\mathcal{A}$.

   v. **If** $U = \emptyset$ **then** output *False* and **halt**.

   vi. Set $\mathcal{T} \leftarrow \mathcal{T} \cup \{S(K_i)\}$.

   vii. Remove U from G and update the data structure of connected components accordingly.

3. Output $\mathcal{T}$.

Figure 7: An iterative presentation of Algorithm A.

Since each set added to $\mathcal{T}$ in Step 2(c)vi corresponds to at least one character, and each character is associated with exactly one such set, updating $\mathcal{T}$ requires $O(nm)$ time in total. This also implies an $O(nm)$ bound on the size of the output produced in Step 3.

It remains to discuss the cost of the dynamic data structure, which is charged for Step 2(c)vii. Using the dynamic algorithm of [15], the connected components of G can be maintained during $|E_1|$ edge deletions at a total cost of $O(|E_1| \log^2(n+m))$ time spent in Step 2(c)vii. Alternatively, using the Las-Vegas type randomized algorithm of [25] for decremental dynamic connectivity, the edge deletions can be supported in $O(|E_1| \log(l^2/|E_1|) + l(\log l)^3 \log\log l)$ expected time. The complexity follows. ■

4.2 Algorithm B

We now describe another deterministic algorithm for IDP, which is faster than Algorithm A whenever $|E_1| = \omega((n+m)^2/\log(n+m))$. Algorithm B uses the dynamic-connectivity data structure of [14], which supports deletion of batches of edges from a graph, while detecting after each batch one of the new connected component in the resulting graph (if new components were formed).

Algorithm B is described in Figure 8. For an instance $\mathcal{A}$ it outputs the non-empty clades of a tree explaining $\mathcal{A}$ (except possibly the root clade, if it has no matching character), or *False* if no such tree exists. It is initially called with Alg_B($\mathcal{A}$).

Alg_B($\mathcal{A} = ((S,C), E_0, E_?, E_1)$)**:**

1. **If** $|S| = 1$ or $G(\mathcal{A})$ has an S-semi-universal vertex **then** output S.

2. **If** $|S| > 1$ **then** do:

   (a) Remove all S-semi-universal characters and all null characters from $G(\mathcal{A})$.

   (b) **If** the resulting graph G' contains a new connected component K **then** do:

      i. Let $\mathcal{A}_1, \mathcal{A}_2$ be the submatrices of $\mathcal{A}$ induced on $V(K)$ and $V(G') \setminus V(K)$, respectively.

      ii. **For** $i = 1, 2$ **do**: Alg_B($\mathcal{A}_i$).

   (c) **Else** output *False* and **halt**.

Figure 8: Algorithm B for solving IDP.

Theorem 13 *Algorithm B correctly solves IDP in $O((n+m)^2 \log(n+m))$ deterministic time.*

Proof: **Correctness:** We prove correctness by induction on the problem size. If G' is connected (at Step 2c), then by Proposition 6 $\mathcal{A}$ has no phylogenetic tree, and indeed the algorithm outputs *False*. Otherwise, let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the sub-instances induced on K and $K' = V(G') \setminus K$, respectively, as detected in Step 2b. If $\mathcal{A}$ has a phylogenetic tree then by Corollary 4 so do $\mathcal{A}_1$ and $\mathcal{A}_2$. On the other hand, let $\mathcal{T}_1, \mathcal{T}_2$ be phylogenetic trees for $\mathcal{A}_1, \mathcal{A}_2$, respectively. Note that by definition, $\mathcal{T}_2$ must contain the trivial clade $S(K')$, which is not necessarily a clade in a phylogenetic tree for $\mathcal{A}$ (if K' has no semi-universal character). To remedy that define $\mathcal{T}_2' = \mathcal{T}_2$ if the algorithm outputs $S(K')$, and $\mathcal{T}_2' = \mathcal{T}_2 \setminus \{S(K')\}$ otherwise. Then $\mathcal{T}_1 \cup \mathcal{T}_2' \cup \{S\}$ is a phylogenetic tree for $\mathcal{A}$.

Complexity: The data structure of [14] dynamically maintains a graph $H = (V, E)$ through batches of edge deletions, with each batch followed by a query for a newly created connected component in the resulting graph. If we denote by b_0 the number of batches which do not result in a new component, then as shown in [14], the total cost of answering the queries and performing the batch deletions, if eventually all edges are deleted, is $O(|V|^2 \log |V| + b_0 \min\{|V|^2, |E| \log |V|\})$.

We use this data structure to maintain $G(\mathcal{A})$ during all the recursive calls. As $b_0 = 1$ (since in case no new component is formed the algorithm outputs *False* and halts) and $|V| = n + m$, the total cost is $O((m+n)^2 \log(n+m))$ time. This expression dominates the complexity, as finding the semi-universal vertices at each recursive call costs in total only $O(nm)$ time (see proof of Theorem 12). ■

We remark, that an $\Omega(nm)$-time lower bound for (undirected) binary perfect phylogeny was proven by Gusfield [12]. A closer look at Gusfield's proof reveals that it applies, as is, also to the directed case. As IDP generalizes directed binary perfect phylogeny, any algorithm for this problem would require $\Omega(nm)$ time.

4.3 Greedy Approach Fails

We end the section by showing that a simple greedy approach to IDP fails. Let $\mathcal{A}$ be an incomplete matrix. We say that $a_{sc} =?$ is *forced* if there exists an assignment $x \in \{0, 1\}$ such that completing a_{sc} to x results in an induced Σ in the graph $(S, C, E_1^{\mathcal{A}'} \cup E_?^{\mathcal{A}'})$ corresponding to the completed matrix $\mathcal{A}'$. $\mathcal{A}$ is called *forced* if it has some forced ?-entry.

A naive greedy algorithm for IDP is as follows: At each step complete one ?-entry in the matrix. If there are no forced entries, choose any ?-entry and complete it arbitrarily. Otherwise, try to complete a forced entry. If such completion is not possible (an induced Σ is formed) report *False*.

Figure 9(A) shows an explainable instance with no forced entries. Setting the bottom-left ?-entry to 0 results in an instance which cannot be explained. A solution matrix is shown in Figure 9(B).

A (Characters across, Species down):

1	?	0	0
1	1	?	?
?	1	1	?
?	?	1	1
?	0	?	1

B (Characters across, Species down):

1	0	0	0
1	1	1	1
1	1	1	1
1	1	1	1
1	0	1	1

Figure 9: A counter-example to the greedy approach. A: The input matrix. B: A solution.

5 Determining the Generality of the Solution

A 'yes' instance of IDP may have several distinct phylogenetic trees as solutions. These trees may be related in the following way: We say that a tree $\mathcal{T}$ *generalizes* a tree $\mathcal{T}'$, and write $\mathcal{T} \subseteq \mathcal{T}'$, if every clade of $\mathcal{T}$ is a clade of $\mathcal{T}'$, i.e., the evolutionary scenario expressed by $\mathcal{T}'$ includes all the details of the scenario expressed by $\mathcal{T}$, and possibly more. Therefore, $\mathcal{T}'$ represents a more specific scenario, and $\mathcal{T}$ represents a more general one. We say that a tree $\mathcal{T}$ is the *general solution* of an instance $\mathcal{A}$, if $\mathcal{T}$ explains $\mathcal{A}$ and generalizes every other tree which explains $\mathcal{A}$. Figure 10 demonstrates the definitions and also gives an example of an instance that has no general solution.

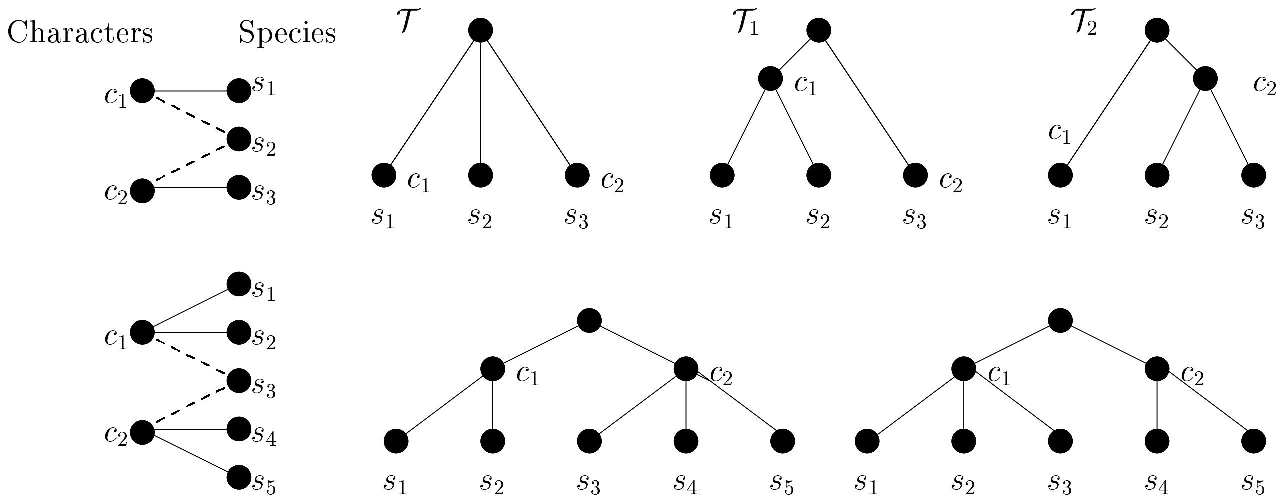

Figure 10: Top left: An IDP instance which has a general solution. Dashed lines denote $E_?$-edges, while solid lines denote E_1-edges. Top-right: $\mathcal{T}, \mathcal{T}_1$ and $\mathcal{T}_2$ are the possible solutions. $\mathcal{T}$ generalizes $\mathcal{T}_1$ and $\mathcal{T}_2$ (which are obtained by splitting the root node of $\mathcal{T}$), and is the general solution. Bottom left: An IDP instance which has no general solution. Bottom middle and bottom right: Two possible solutions. The only tree which generalizes both solutions is the tree composed of the trivial clades only, which is not a solution.

We give in this section a characterization of IDP instances that admit a general solution. We prove that whenever a general solution exists, Algorithm A finds it. We also provide an algorithm to determine whether the solution tree $\mathcal{T}$ returned by Algorithm A is general. The complexity of the latter algorithm is shown to be $O(mn + |E_1|d)$, where d is the maximum out-degree in $\mathcal{T}$.

The following notation is used in the sequel: Let $\mathcal{A}$ be an incomplete matrix and let $\hat{S} \subseteq S$. We denote by $W_{\mathcal{A}}(\hat{S})$ the set of $\hat{S}$-semi-universal characters in $\mathcal{A}$. Note that if $\mathcal{A}$ is binary, then $W_{\mathcal{A}}(\hat{S})$ is its set of $\hat{S}$-universal characters. We now define the operator $\tilde{}$ on incomplete matrices: We denote by $\widetilde{\mathcal{A}}$

the submatrix $\mathcal{A}|_{S,C\setminus W_{\mathcal{A}}(S)}$ of $\mathcal{A}$. In particular, $G(\widetilde{\mathcal{A}})$ is the graph produced from $G(\mathcal{A})$ by removing its set of S-semi-universal characters. A species set $\emptyset \neq S' \subseteq S$ is said to be *connected* in a graph G, if S' is contained in some connected component of G.

Lemma 14 *Let $\mathcal{T}$ be the general solution for an instance $\mathcal{A}$ of IDP. Let $S' = L(x)$ be a clade of $\mathcal{T}$, corresponding to some node x. Let $\mathcal{T}'$ be the subtree of $\mathcal{T}$ rooted at x, and let $\mathcal{A}'$ be the instance induced on $S' \cup C$. Then $\mathcal{T}'$ is the general solution for $\mathcal{A}'$.*

Proof: By Observation 5, $\mathcal{T}'$ explains $\mathcal{A}'$. Suppose that $\mathcal{T}''$ also explains $\mathcal{A}'$ and $\mathcal{T}' \not\subseteq \mathcal{T}''$. Then $\hat{\mathcal{T}} = (\mathcal{T} \setminus \mathcal{T}') \cup \mathcal{T}''$ explains $\mathcal{A}$, and $\mathcal{T} \not\subseteq \hat{\mathcal{T}}$, a contradiction. ■

A nonempty clade of a tree is called *maximal* if the only clade that properly contains it is S.

Lemma 15 *Let $\mathcal{T}$ be a phylogenetic tree for a binary matrix $\mathcal{B}$. A non-empty clade S' of $\mathcal{T}$ is maximal if and only if S' is the species set of some connected component of $G(\widetilde{B})$.*

Proof: Suppose that S' is a maximal clade of $\mathcal{T}$. We first claim that S' is contained in some connected component K of $G(\widetilde{B})$. If $|S'| = 1$ this trivially holds. If $|S'| > 1$, let c be a character associated with S'. c is adjacent to all vertices in S' and to no other vertex. Hence, c is not S-universal implying that all the edges $\{(c, s) : s \in S'\}$ are present in $G(\widetilde{B})$. This proves the claim. It remains to show that $S(K) = S'$. Suppose $S(K) \supset S'$. In particular, $|S(K)| > 1$. By Proposition 6, there exists a character c' in $G(\widetilde{B})$ whose 1-set is $S(K)$. Hence, $S(K)$ must be a clade of $\mathcal{T}$ which is associated with c', contradicting the maximality of S'.

To prove the converse, let S' be the species set of some connected component K of $G(\widetilde{B})$. We first claim that S' is a clade. If $|S'| = 1$, S' is a trivial clade. Otherwise, by Proposition 6 there exists an S'-universal character c' in $G(\widetilde{B})$. Since K is a connected component, c' has no neighbors in $S \setminus S'$. Hence, S' must be a clade in $\mathcal{T}$. Suppose to the contrary that S' is not maximal, then it is properly contained in a maximal clade S'', which by the previous direction is the species set of K, a contradiction.
■

Theorem 16 *Algorithm A produces the general solution for every IDP instance that has one.*

Proof: Let $\mathcal{A}$ be an instance of IDP for which there exists a general solution $\mathcal{T}^*$. Let $\mathcal{T}_{alg}$ be the solution tree produced by Algorithm A. By definition $\mathcal{T}^* \subseteq \mathcal{T}_{alg}$. Suppose to the contrary that $\mathcal{T}^* \neq \mathcal{T}_{alg}$. Let S' be the largest clade reported by Algorithm A, which is not a clade of $\mathcal{T}^*$ (S' must be non-trivial), and let S'' be the smallest clade in $\mathcal{T}_{alg}$ which properly contains S'. Let $\mathcal{A}'$ be the instance induced on

$S'' \cup C$. By Observation 5, $\mathcal{A}'$ is explained by the corresponding subtrees $\mathcal{T}'_{alg}$ of $\mathcal{T}_{alg}$ and $\mathcal{T}'^*$ of $\mathcal{T}^*$. By Lemma 14, $\mathcal{T}'^*$ is the general solution of $\mathcal{A}'$. Due to the recursive nature of Algorithm A, it produces $\mathcal{T}'_{alg}$ when invoked with input $\mathcal{A}'$. Thus, without loss of generality, one can assume that $S'' = S$ and S' is a maximal clade of $\mathcal{T}_{alg}$.

Suppose that $\mathcal{T}^*$ explains $\mathcal{A}$ via a completion $\mathcal{B}^*$, and let $G^* = G(\mathcal{B}^*)$. Since S' is a maximal clade, it is reported during a second level call of Alg_A($\cdot$) (the call at the first level reports the trivial clade S). Hence, it must be the species set of some connected component K in $G(\widetilde{\mathcal{A}})$. Since every S-universal character in G^* is S-semi-universal in $\mathcal{A}$, S' is contained in some connected component K^* of $G(\widetilde{B^*})$. Denote $S^* \equiv S(K^*)$. By Lemma 15, S^* is a maximal clade of $\mathcal{T}^*$. Since $S' \notin \mathcal{T}^*$, we have $S' \neq S^*$, and therefore, $S^* \supset S'$. But $\mathcal{T}^* \subseteq \mathcal{T}_{alg}$, implying that S^* is also a non-trivial clade of $\mathcal{T}_{alg}$, in contradiction to the maximality of S'. ■

We now characterize IDP instances for which a general solution exists. Let $\mathcal{A}$ be a 'yes' instance of IDP. Consider a recursive call Alg_A($\mathcal{A}'$) nested within Alg_A($\mathcal{A}$), where $\mathcal{A}' = \mathcal{A}|_{C',S'}$. Let $K_1, \ldots, K_r$ be the connected components of $G(\widetilde{\mathcal{A}'})$, computed in Step 1c. Observe that $S(K_1), \ldots, S(K_r)$ are clades to be reported by recursive calls launched during Alg_A($\mathcal{A}'$). A set U of characters is said to be (K_i, K_j)-*critical* if:

- Characters in U are both $S(K_i)$-semi-universal and $S(K_j)$-semi-universal in $\mathcal{A}'$.
- Removing U from $G(\widetilde{\mathcal{A}'})$ disconnects $S(K_i)$.

Note that by definition of U, $U \subseteq W_{\mathcal{A}'}(S(K_i))$, and $a'_{sc} =?$ for all $c \in U, s \in S(K_j)$. A clade $S(K_i)$ is called *optional* (with respect to $\mathcal{A}'$), if $r \geq 3$ and there exists a (K_i, K_j)-critical set for some index $j \neq i$. If $S(K_i)$ is not optional we say it is *mandatory.* In the example of Figure 10 (bottom), let $K_1 = \{s_1, s_2, c_1\}$, $K_2 = \{s_3\}$, and $K_3 = \{s_4, s_5, c_2\}$. The set $U = \{c_1\}$ is (K_1, K_2)-critical, so $S(K_1) = \{s_1, s_2\}$ is optional. In contrast, in Figure 10 (top) no clade is optional.

Theorem 17 *The tree produced by Algorithm A is the general solution if and only if all its clades are mandatory.*

Proof: $\Rightarrow$ Suppose that $\mathcal{T}_{alg}$ is the general solution of an instance $\mathcal{A}$. Suppose to the contrary that it contains an optional clade. Without loss of generality, assume it is maximal, i.e., during the recursive call Alg_A($\mathcal{A}$), $G' = G(\widetilde{\mathcal{A}})$ has $r \geq 3$ connected components, $K_1, \ldots, K_r$, and there exists a (K_i, K_j)-critical set U (for some $1 \leq i \neq j \leq r$). Let $\mathcal{A}_i, \mathcal{A}_j$ and $\mathcal{A}_{ij}$ be the sub-instances induced on K_i, K_j and $K_i \cup K_j$, respectively. Consider the tree $\mathcal{T}'$ which is produced by a small modification to the execution of Alg_A($\mathcal{A}$): Instead of recursively invoking Alg_A($\mathcal{A}_i$) and Alg_A($\mathcal{A}_j$), call Alg_A($\mathcal{A}_{ij}$). Then $\mathcal{T}'$ is a phylogenetic tree which explains $\mathcal{A}$ and includes the clade $S(K_i \cup K_j)$. Since removing U from $G(\widetilde{\mathcal{A}})$ disconnects $S(K_i)$, $|S(K_i)| \geq 2$ so $S(K_i)$ is non-trivial. Moreover, $S(K_i)$ is not a clade of $\mathcal{T}'$ for the same reason. Hence, $\mathcal{T}'$ does not contain all clades of $\mathcal{T}_{alg}$, in contradiction to the generality of $\mathcal{T}_{alg}$.

$\Leftarrow$ Suppose that $\mathcal{T}_{alg}$ is not the general solution of an instance $\mathcal{A}$, i.e., there exists a solution $\mathcal{T}^*$ of $\mathcal{A}$ such that $\mathcal{T}_{alg} \not\subseteq \mathcal{T}^*$. We shall prove the existence of an optional clade in $\mathcal{T}_{alg}$.. (The reader is referred to the example in Figure 13 for notation and intuition. The example follows the steps of the proof, leading to the identification of an optional clade.) Let $\mathcal{B}^*$ be a completion of $\mathcal{A}$ which is explained by $\mathcal{T}^*$, and denote $G^* = G(\mathcal{B}^*)$. Let $S' \in \mathcal{T}_{alg} \setminus \mathcal{T}^*$ be the largest clade reported by Algorithm A which is not a clade of $\mathcal{T}^*$. Without loss of generality (as argued in the proof of Theorem 16), S' is a maximal clade of $\mathcal{T}_{alg}$, and let $S' = S(K_1)$, where $K_1, \ldots, K_r$ are the connected components of $G(\widetilde{\mathcal{A}})$.

Observe that a binary matrix has at most one phylogenetic tree. Thus, an application of Algorithm A to $\mathcal{B}^*$ necessarily outputs $\mathcal{T}^*$. Consider such an application, and let $\{S_i^*\}_{i=1}^t$ be the nested set of reported clades in $\mathcal{T}^*$ which contain S': $S = S_1^* \supset \cdots \supset S_t^* \supset S'$ (see Figure 11). For each $i = 1, \ldots, t$, let $\mathcal{B}_i^*$ be the instance invoked in the recursive call which reports S_i^*, and let H_i^* be the graph $G(\widetilde{\mathcal{B}_i^*})$, computed in Step 1a of that recursive call. Let C_i^* be the set of characters in H_i^*. Equivalently, C_i^* is the set of characters in $\mathcal{B}_i^*$ whose 1-set is non-empty and is properly contained in S_i^*. Furthermore, define H_i to be the subgraph of $G(\mathcal{A})$ induced on $S_i^* \cup C_i^*$. Observe that H_i^* is the subgraph of G^* induced on the same vertex set. Since G^* is a supergraph of $G(\mathcal{A})$, each H_i^* is a supergraph of H_i.

Claim 18 *S' is disconnected in H_t^*, and therefore also in H_t.*

Proof: Suppose to the contrary that S' is contained in some connected component K^* of H_t^*. K^* is thus computed during the t-th recursive call (with argument $\mathcal{B}_t^*$), and $S(K^*)$ is reported as a clade in $\mathcal{T}^*$ by a nested recursive call. Therefore, $S_t^* \supset S(K^*) \supset S'$, where the first proper containment follows from the fact that H_t^* is disconnected, and the second from the assumption that S' is not a clade of $\mathcal{T}^*$. Hence, we arrive at a contradiction to the minimality of S_t^*. ■

We now return to the proof of Theorem 17. Recall that S' is connected in $H_1 = G(\widetilde{A})$. Thus, the previous claim implies that $t > 1$. Let K_p be a connected component of $G(\widetilde{\mathcal{A}})$ such that $S(K_p) \subseteq S \setminus S_2^*$ (see Figure 11). Let l be the minimal index such that there exists some connected component K_i of $G(\widetilde{\mathcal{A}})$ for which $S(K_i)$ is disconnected in H_l. l is properly defined as $S(K_1) = S'$ is disconnected in H_t. $l > 1$, since otherwise some K_i is disconnected in H_1 and, therefore, also in its subgraph $G(\widetilde{\mathcal{A}})$, in contradiction to the definition of $K_1, \ldots, K_r$.

By minimality of l, $S_l^* \supseteq S(K_i)$. Also, $S_l^* \supseteq S_t^* \supset S' = S(K_1)$, so $S_l^* \neq S(K_i)$. We now claim that there exists some connected component K_j of $G(\widetilde{\mathcal{A}})$, $j \neq i$, such that $S(K_j) \subseteq S_l^*$. Indeed, if $i \neq 1$ then

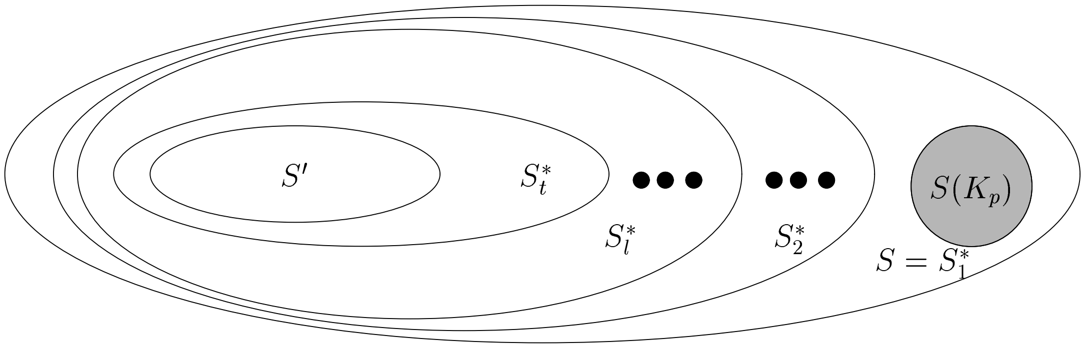

Figure 11: The clades $S = S_1^* \supset S_2^* \supset \cdots \supset S_t^* \supset S'$.

$j = 1$. If $i = 1$ then $l = t$ (by an argument similar to that in the proof of Claim 18), and since $S_l^* \setminus S'$ is non-empty, it intersects $S(K_j)$ for some $j \neq i$. By minimality of l, $S(K_j)$ is properly contained in $S_l^* \setminus S'$.

Define $U \equiv W_{G^*}(S_l^*)$. We now prove that U is a (K_i, K_j)-critical set. By definition all characters in U are S_l^*-universal in G^*, and are thus both K_i-semi-universal and K_j-semi-universal in $\mathcal{A}$. $S(K_i)$ is disconnected in $H_l = G(\mathcal{A}|_{C_l^*, S_l^*})$. Since K_i is a connected component of $G(\widetilde{\mathcal{A}})$, $S(K_i)$ is disconnected in $G(\mathcal{A}|_{C_l^*, S})$, implying that U is a (K_i, K_j)-critical set. Also, K_i, K_j and K_p are distinct, implying that $r \geq 3$ (see Figure 12). In conclusion, U demonstrates that $S(K_i)$ is optional.

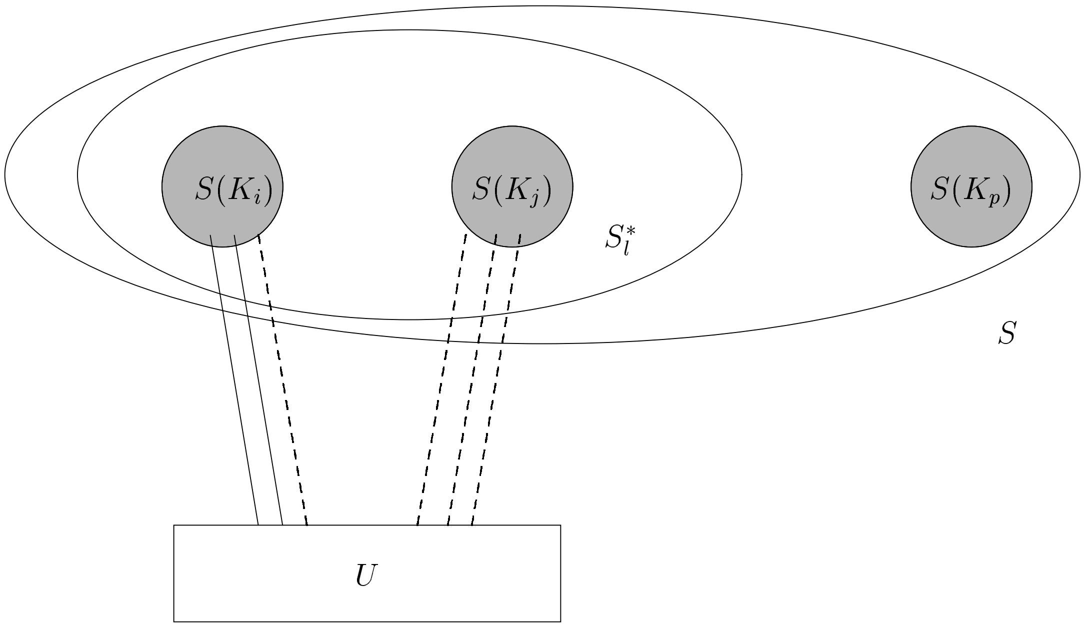

Figure 12: $S(K_i)$,$S(K_j)$ and $S(K_p)$. Note, that removing U disconnects $S(K_i)$.

■

The characterization of Theorem 17 leads to an efficient algorithm for determining whether a solution

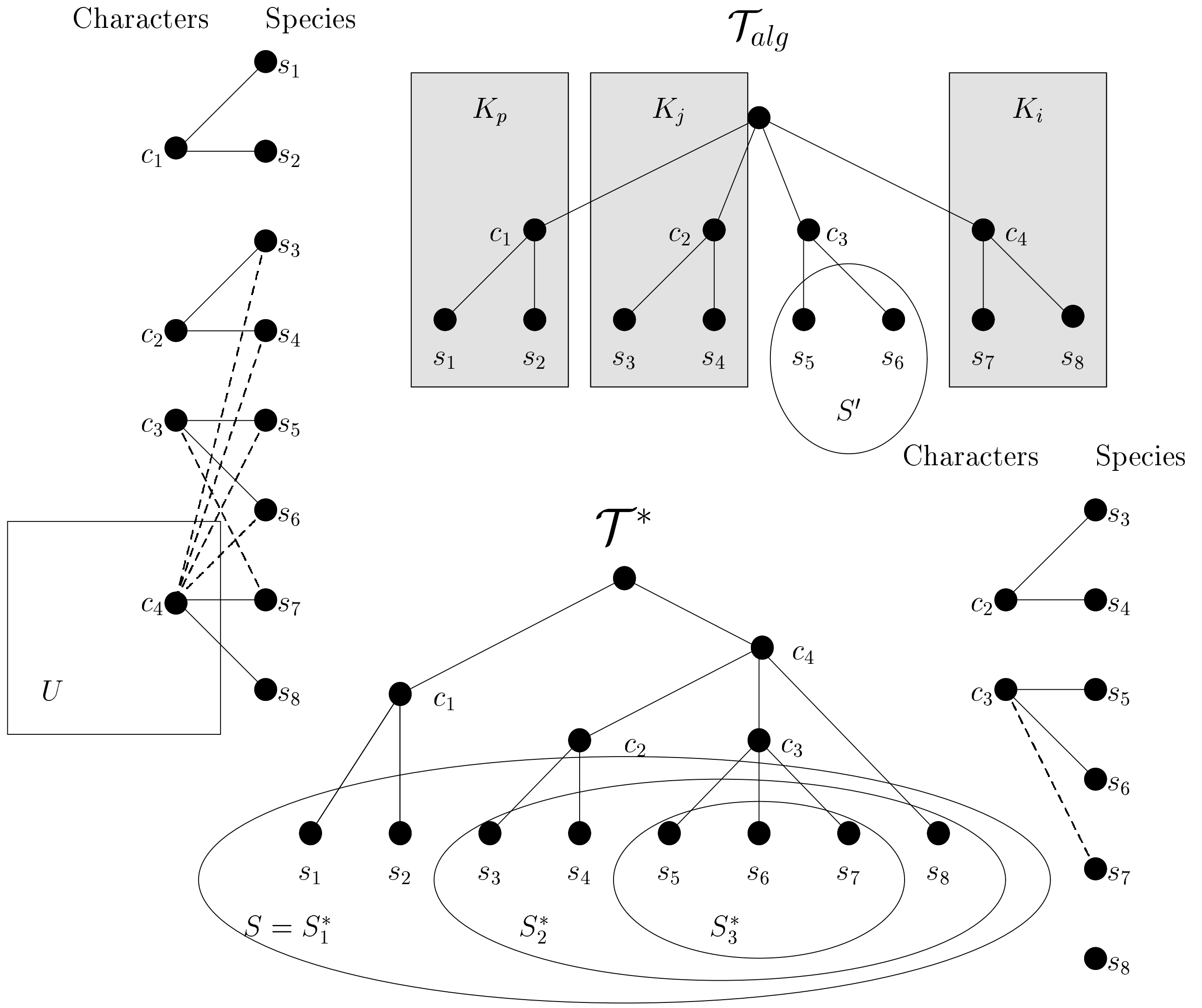

Figure 13: An example demonstrating the proof of the 'if' part of Theorem 17, using the notation in the proof. Left: A graphical representation of an input instance $\mathcal{A}$. Dashed lines denote $E_?$-edges, while solid lines denote E_1-edges. Top right: The tree $\mathcal{T}_{alg}$ produced by Algorithm A. Bottom middle: A tree $\mathcal{T}^*$ corresponding to a completion $\mathcal{B}^*$ that uses all the edges in $E_?$. Bottom right: The graphs H_2 (solid edges) and H_2^* (solid and dashed edges). $\mathcal{T}_{alg} \not\subseteq \mathcal{T}^*$, and $S' = \{s_5, s_6\}$. There are $t = 3$ clades of $\mathcal{T}^*$ which contain S': $S_1^* = \{s_1, \ldots, s_8\}$, $S_2^* = \{s_3, \ldots, s_8\}$, and $S_3^* = \{s_5, s_6, s_7\}$. The component $K_p = \{c_1, s_1, s_2\}$ has its species in $S \setminus S_2^*$. Since $W_{\mathcal{A}}(S) = W_{\mathcal{B}^*}(S) = \emptyset$, $H_1 = G(\mathcal{A})$. Since $W_{\mathcal{B}^*}(S_2^*) = \{c_4\}$, the species set of the connected component $K_i = \{s_7, s_8, c_4\}$ is disconnected in H_2, implying that $l = 2$. For a choice of $K_j = \{s_3, s_4, c_2\}$, the set $U = \{c_4\}$ is (K_i, K_j)-critical, demonstrating that S' is optional.

$\mathcal{T}_{alg}$ produced by Algorithm A is general.

Theorem 19 *There is an $O(nm+|E_1|d)$-time algorithm to determine if a given solution $\mathcal{T}_{alg}$ is general, where d is the maximum out-degree in $\mathcal{T}_{alg}$.*

Proof: The algorithm simply traverses $\mathcal{T}_{alg}$ bottom-up, searching for optional clades. For each internal node x visited, whose children are $y_1, \ldots, y_{d(x)}$, the algorithm checks whether any of the clades $L(y_1), \ldots, L(y_{d(x)})$ is optional. If an optional clade is found the algorithm outputs *False*. Correctness follows from Theorem 17.

We show how to efficiently check whether a clade $L(y_i)$ is optional. If $d(x) = 2$, or y_i is a leaf, then certainly $L(y_i)$ is mandatory. Otherwise, let U_i be the set of characters whose origin (in $\mathcal{T}_{alg}$) is y_i. Let U^i_j denote the set of characters in U_i which are $L(y_j)$-semi-universal, for $j \neq i$. The computation of U^i_j for all i and j takes $O(nm)$ time in total, since for each character c and species s we check at most once whether $(s,c) \in E^{\mathcal{A}}_?$, for an input instance $\mathcal{A}$.

It remains to show how to efficiently check whether for some j, U^i_j disconnects $L(y_i)$ in the appropriate subgraph encountered during the execution of Algorithm A. To this end, we define an auxiliary bipartite graph H^i whose set of vertices is $W_i \cup U_i$, where $W_i = \{w_1, \ldots, w_{d(y_i)}\}$ is the set of children of y_i in $\mathcal{T}_{alg}$. We include the edge (w_r, c_p) in H^i, for $w_r \in W_i, c_p \in U_i$, if $(c_p, s) \in E^{\mathcal{A}}_1$ for some species $s \in L(w_r)$. We construct for each $j \neq i$ a subgraph H^i_j of H^i induced on $W_i \cup (U_i \setminus U^i_j)$. All we need to report is whether H^i_j is connected.

For each i we construct H^i by considering all $E^{\mathcal{A}}_1$ edges connecting characters in U_i to species in $L(y_i)$. This takes $O(|E^{\mathcal{A}}_1|)$ time in total. There are $d(y_i)$ subgraphs H^i_j for every y_i. Hence, computing H^i_j for all j, and determining whether each H^i_j is connected, takes $O\left(|E(H^i)|d(y_i)\right)$ time. Since $\sum_i |E(H^i)| \leq |E^{\mathcal{A}}_1|$, the total time complexity is $O\left(mn + \sum_i |E(H^i)|d(y_i)\right) = O(mn + |E^{\mathcal{A}}_1| \cdot \max_{v \in \mathcal{T}_{alg}} d(v))$. ■

Acknowledgments

We thank Dan Graur for drawing our attention to this phylogenetic problem, and for helpful discussions. We thank Nati Linial for insightful comments. We thank Joe Felsenstein, Dan Gusfield, Haim Kaplan, Mike Steel, and an anonymous CPM '00 referee for referring us to helpful literature.

I. Pe'er was supported by the Clore foundation scholarship. R. Shamir was supported in part by a grant from the Israel Science Foundation (grant 309/02). R. Sharan was supported by a Fulbright grant and by an Eshkol fellowship from the Ministry of Science, Israel.

References

[1] N. Abbas and L.K. Stewart. Biconvex graphs: ordering and algorithms. *Discrete Applied Mathematics*, 103:1–19, 2000.

[2] R. Agarwala and D. Fernández-Baca. A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM Journal on Computing*, 23(6):1216–1224, 1994.

[3] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.

[4] C. Benham, S. Kannan, M. Paterson, and T.J. Warnow. Hen's teeth and whale's feet: generalized characters and their compatibility. *Journal of Computational Biology*, 2(4):515–525, 1995.

[5] H. L. Bodlaender, M. R. Fellows, M. T. Hallett, T. Wareham, and T. Warnow. The hardness of perfect phylogeny, feasible register assignment and other problems on thin colored graphs. *Theoretical Computer Science*, 244(1–2):167–188, 2000.

[6] J. H. Camin and R. R. Sokal. A method for deducing branching sequences in phylogeny. *Evolution*, 19:409–414, 1965.

[7] L. Dollo. Le lois de l'évolution. *Bulletin de la Societé Belge de Géologie de Paléontologie et d'Hydrologie*, 7:164–167, 1893.

[8] J. Felsenstein. *Inferring Phylogenies*. Sinaur Associates, Sunderland, Massachusetts, 2002. In press.

[9] L.R. Foulds and R.L. Graham. The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3:43–49, 1982.

[10] M. C. Golumbic. Matrix sandwich problems. *Linear algebra and its applications*, 277:239–251, 1998.

[11] M. C. Golumbic, H. Kaplan, and R. Shamir. Graph sandwich problems. *Journal of Algorithms*, 19:449–473, 1995.

[12] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991.

[13] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[14] M. Henzinger, V. King, and T.J. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, 24:1–13, 1999.

[15] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001.

[16] S. Kannan and T. Warnow. A fast algorithm for the computation and enumeration of perfect phylogenies. *SIAM Journal on Computing*, 26(6):1749–1763, 1997.

[17] B. Klinz, R. Rudolf, and G. J. Woeginger. Permuting matrices to avoid forbidden submatrices. *Discrete applied mathematics*, 60:223–248, 1995.

[18] C. A. Meecham and G. F. Estabrook. Compatibility methods in systematics. *Annual Review of Ecology and Systematics*, 16:431–446, 1985.

[19] M. Nikaido, A. P. Rooney, and N. Okada. Phylogenetic relationships among cetartiodactyls based on insertions of short and long interspersed elements: Hippopotamuses are the closest extant relatives of whales. *Proceedings of the National Academy of Science USA*, 96:10261–10266, 1999.

[20] I. Pe'er, R. Shamir, and R. Sharan. Incomplete directed perfect phylogeny. In *Eleventh Annual Symposium on Combinatorial Pattern Matching (CPM'00)*, pages 143–153, 2000.

[21] I. Pe'er, R. Shamir, and R. Sharan. On the generality of phylogenies from incomplete directed characters. In *Proceedings of the Eighth Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 2368 of *LNCS*, pages 358–367, 2002.

[22] W. J. Le Quesne. The uniquely evolved character concept and its cladistic application. *Systematic Zoology*, 23:513–517, 1974.

[23] M. A. Steel. The complexity of reconstructing trees form qualitative characters and subtrees. *Journal of Classification*, 9:91–116, 1992.

[24] D. L. Swofford. *PAUP, Phylogenetic Analysis Using Parsimony (and Other Methods).* Sinaur Associates, Sunderland, Massachusetts, 1998. Version 4.

[25] M. Thorup. Decremental dynamic connectivity. *Journal of Algorithms*, 33:229–243, 1999.