Multi-State Perfect Phylogeny

Simon Rubin

Abstract

I present a polynomial time algorithm, implemented in the Julia Programming language, for determining whether a set of cladistic characters admits a perfect phylogeny. The application also answers the inverse question: Given a phylogenetic tree determine what subset of the characters are compatible.

1 Introduction

A phylogeny is a tree representation of the evolutionary history of a set of taxa. The taxa are described by the states they exhibit on a set of characters. It is desired that the character evolution exhibit homoplasy; characters mutate to a given state no more than once. The motivation being that it is unlikely for the same mutation to occur over short evolutionary distances. Sets of characters that admit phylogenies without homoplasy are said to be compatible. The perfect phylogeny problem asks if a set of characters is compatible.

A cladistic character has multiple state values that taxa can exhibit. The character is associated with a transition tree which specifies how the states are allowed to evolve. T is consistent with a character if the evolution of the character in T respects the transition tree. T is a perfect phylogeny for a set of cladistic characters C if T exhibits homoplasy and T is consistent with all characters.

2 Theory

Let M be an $n \times m$ binary matrix specifying the states of m characteristics for each of the n taxa. We have the following theorem:

Theorem (3-Gametes Condition): M has a perfect-phylogeny if and only if no pair of columns c, d contains the three binary pairs $(0,1), (1,0)$, and $(1,1)$.

Consider sorting the columns of M in decreasing order of the number of 1s contained in each column. Let f and g be two taxa, and let d be the rightmost character in M that taxa f and g both possess. The following lemma follows from the above theorem:

Shared Prefix Property: rows f and g in M must be identical from column one to column d.

The last piece of theory needed to answer the multi-state phylogeny problem will allow us to adapt the above properties of a binary state matrix to a multi-state matrix. A multi-state character can be factored as follows: Let t_c be the transition tree of character c and let v be a node of t_c. The binary factor of c associated with v is the binary character whose 1-state consists of all taxa that exhibit states in the subtree of t_c rooted at v. We

have the following lemma:

Lemma: A set C of cladistic characters is compatible if and only if the set of all binary factors of the characters in C is compatible.

3 Existence of a Perfect Phylogeny

The shared prefix property provides the following algorithm to determine the compatibility of a set of binary characters. If the characters are found to be compatible then the output will be a perfect phylogenetic tree:

Input:

- $M :=$ Binary character-state matrix

Output:

- Perfect Phylogeny T, or alert that one cannot be constructed.

```
1: function CONSTUCT_PHYLOGENY(M)
2:     sort the columns of M in nondecreasing order of #1s
3:     T ← with all 0 root
4:
5:     for taxon f ∈ M
6:         curr ← root
7:         for character c ∈ 1(f)
8:             if curr has edge leading to c
9:                 curr ← c
10:            else
11:                if c pre-exists in T
12:                    return Perfect Phylogeny does not exist
13:                else
14:                    add edge from curr to c
15:                    curr ← c
16:
17:        add edge from curr to taxon f
18:    return T
```

Initialize a tree with all 0 root. At $t = 1$, go through first row of matrix. Add a path from the root to a leaf labeled by taxon 1. Each edge in the path corresponds to a character possessed by taxon 1. The edges occur in the order that the characters in matrix row 1. At $t = i$, start at the root, and examine the characters possessed by taxon i. Traverse the tree as long as the successive characters on the path matches the successive characters possessed by taxon i. At the end of the traversal either all characters of i have been matched or a node v is reached that has no edge matching the next character possessed by taxon i. Then create a new path out of v containing all the characters of taxon i that have yet to be matched. If at any point a character that is already in T as a node is to be readded then the algorithm halts.

If a vertex preexists in the tree then the shared prefix property is not true. Suppose this event were to occur. Each path in the tree corresponds to a prefix of some taxa's row in M. The path to our current point describes a prefix of a taxon f. f possesses the

character c that is to be inserted. c already exists in the tree, so it must be on some separate path. Therefore, there is another taxon g that possesses c but not one of the other characters of f that comes before c. The failure of the shared prefix property implies that the 3-Gametes condition fails on the input matrix and that there is no perfect phylogeny for the input.

Constructing the phylogenetic tree requires traversing each postition of M row-wise. Therefore, if M consists of m characters for n taxa the algorithm runs in $\mathcal{O}(nm)$.

The above procedure provides a means to determine the compatibility of a set of binary characters. A multi-state matrix can be converted into a binary matrix by taking all binary factors, then using the above algorithm to determine compatibility. Note that if a character has a maximum of r states, then there are r binary factors for the character. In the case of cladistic characters the algorithm runs in $\mathcal{O}(nmr)$ time.

4 Evaluating a Phylogeny

This version of the problem specifies a phylogenetic tree T and a set of transition trees, and asks what characters are compatible T. T must evolve without homoplasy. It is assumed that the edges of the input tree are labeled with the locations of character-state transitions. The homoplasy condition is checked by determining whether a given mutation occurs more than once in T. To evaluate consistency, a character's transition tree must be checked against the evolution that occurs in T. Let E be the tree capturing how the character's states evolved in T. E is consistent with the transition tree, if for each node $v \in E$ the states that v evolves to is a subset of the possible states that v can evolve to in the transition tree.

The homoplasy condition can be checked in time proportional to the number of edges in the input tree. The maximal number of edges in the tree occurs if for each character of each taxa, the character mutates into all its possible states. Given n taxa, and m characters with a maximum number of states r, we have that this operation runs in $\mathcal{O}(nmr)$ time. For the consistency condition, deducing a character's evolution in the tree can be determined via a depth first search. This takes time proportional to the size of the tree $\mathcal{O}(nmr)$. The comparison of the transition tree to the evolutionary tree is proportional to the sizes of these trees which is bounded by the number of states that this character can exhibit, $\mathcal{O}(r)$ time. The most expensive operation is the depth first search, and it is performed for each of the m characters, requiring $\mathcal{O}(nm^2r)$ time.

5 Running

To have the program determine whether a perfect phylogeny exists enter the following command:

```
./compat.sh matrix [matrix_file] [character_transition_trees_file]
```

The matrix file must have the following format: on each line specify the taxon name followed by a space and then a numeric string, where the i^{th} position represents the taxon's state for the i^{th} character. Here is an example:

a 10201
b 11200
c 11000
d 01201

The character transition tree file has the following format:

0 -> 1
1 -> 2
end
0 -> 1
0 -> 2
end

The 'end' separates the description of one character's transition tree from another's. The line 0 -> 1 denotes, that in a given transition tree state 0 is allowed to evolve to state 1. A line in the file should only contain a single transition.

To have the program determine which characters are compatible with a given phylogenetic tree enter the following command:

```
./compat.sh tree [tree_file (dot format)] [character_transition_trees_file]
```

It is assumed that the input tree has edges labeled with where and to what state characters evolved.