Informatica per le biotecnologie

9. Alberi di filogenesi

Estratto dal testo di J. Setubal e J. Meidanis: Introduction to Computational Molecular Biology

NOTE.

- Questa dispensa contiene solo gli aspetti più elementari dell'argomento. Il testo citato sopra è un'ottima fonte per approfondimenti. Il primo di questi dovrebbe riferirsi alla costruzione di alberi di filogenesi a partire da una **matrice di distanza** tra le specie.

- Ricordiamo che la classe dei problemi **NP-completi** contiene i problemi che si sanno risolvere solo in tempo esponenziale, e per i quali l'opinione corrente è che non esistano algoritmi polinomiali.

CHAPTER 6

PHYLOGENETIC TREES

In this chapter we describe the problem of reconstructing phylogenetic trees. This is a general problem in biology. It is used in molecular biology to help understand the evolutionary relationships among proteins, for example. We present models and algorithms for two basic kinds of input data: characters and distances.

Modern science has shown that all species of organisms that live on earth undergo a slow transformation process through the ages. We call this process *evolution*. One of the central problems in biology is to explain the evolutionary history of today's species and, in particular, how species relate to one another in terms of common ancestors. This is usually done by constructing trees, whose leaves represent present-day species and whose interior nodes represent hypothesized ancestors. These kinds of trees are called *phylogenetic trees*. One example is shown in Figure 6.1. According to this tree human beings and chimpanzees are genetically closer to each other than to the other primates in the tree. This means that they have a common ancestor that is not an ancestor of any of the other primates. The tree also shows a similar situation between gibbons and siamangs.

The problem with phylogenetic tree construction is that generally we do not have enough data about distant ancestors of present-day species; and even if we did, we could not be 100% sure that a particular fossil belongs to a species that *really* is the ancestor of two or more current species. Therefore we have to *infer* the evolutionary history of current organisms, and *re-create* their phylogenetic tree, generally using as primary data comparisons between today's species. For this reason the tree shown in Figure 6.1 is not necessarily the true one; it is simply a hypothesis.

Phylogenetic trees, or more simply, **phylogenies**, have been proposed since the last century for all sorts of groups of organisms. The methods used up to the 1950s were generally based on researchers' experience and intuition. Gradually mathematical formalisms were introduced and these were incorporated into numerical methods. Today there is an immense body of work on the mathematics and algorithmics of phylogeny reconstruction. There are also software packages offering a host of programs to aid scientists in re-creating phylogenies.

It is not our aim in this chapter to survey computational methods for phylogeny reconstruction; a whole book would be required for this. Our goal is rather to study some

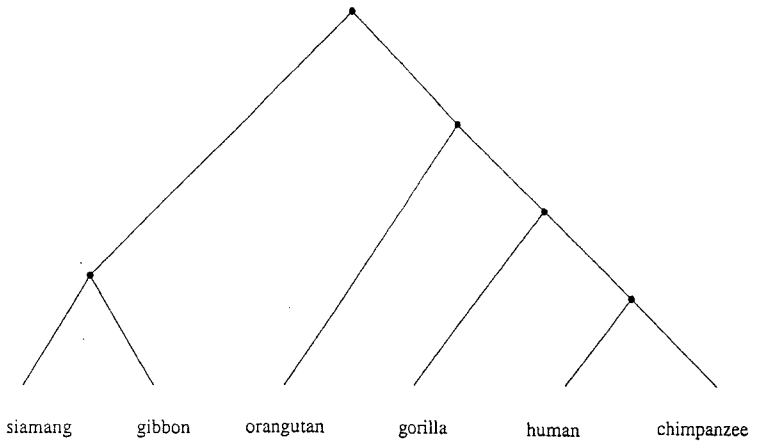


FIGURE 6.1

A phylogenetic tree for some primates.

problems of this area that are amenable to algorithmic solution. The reader should be aware that this is a very narrow and biased treatment, considering the extent and diversity of the literature on this topic. Pointers to more general texts on the subject are given in the bibliographic notes for this chapter.

In evolutionary biology, we can build phylogenetic trees for species, populations, genera, or other *taxonomical units*. (Taxonomy is the "science of classification." The term *numerical taxonomy* is used to describe the collection of numerical methods used for any kind of grouping. This chapter's topic can be seen as a subfield of numerical taxonomy.) Because nucleic acids and proteins also evolve, we can also build phylogenies for them, which explains why molecular biologists are also interested in methods for phylogeny reconstruction. To keep the exposition as general as possible we will use the term object for the taxonomical units for which we want to reconstruct a phylogeny.

Let us recall the definition of trees from Section 2.2. A tree is an undirected acyclic connected graph. We distinguish between its exterior nodes, or *leaves*, and the *interior nodes*. Leaves have degree one, whereas interior nodes have degree greater than one. To the leaves we associate the objects under study. Thus we can say that a leaf is *labeled* by an object or a set of objects.

Scientists are interested in two main aspects of phylogenetic trees. One is its *topology*, that is, how its interior nodes connect to one another and to the leaves. (Some authors use the term *branching pattern* for this.) Another important aspect is the *distance* between pairs of nodes, which can be determined when the tree edges are weighted. This distance is an estimate of the evolutionary distance between the nodes. Depending on many assumptions that we will not discuss, the distance between an interior node (ancestral object) to a leaf (present-day object) can be interpreted as an estimate of the time

it took for one to evolve into the other. But the reader should be aware evolutionary distance in general is not equivalent to elapsed time.

Another important feature of a phylogenetic tree is whether it has a root. A tree with a root implies ancestry relationships between interior nodes. However, in many of the problems that we will study there is not enough information in the data to allow us to determine the root; in those cases the reconstructed tree will be unrooted.

We mentioned that phylogenies are reconstructed based on comparisons between present-day objects. We can classify input data for phylogeny reconstruction into two main categories:

1. *Discrete characters,* such as beak shape, number of fingers, presence or absence of a molecular restriction site, etc. Each character can have a finite number of *states*. The data relative to these characters are placed in an objects × characters matrix, which we call **character state matrix**.
2. *Comparative numerical data,* which we call *distances* between objects. The resulting matrix is called the **distance matrix**.

Each of these categories gives rise to different methods for phylogeny reconstruction. One significant difference between the categories is that character state matrices are in general not square matrices, since the number of objects need not be equal to the number of characters. On the other hand, distance matrices are triangular matrices, because we have a distance for each pair of objects and these distances are symmetric. In the following sections we present representative algorithms for several of these methods. There is also a third category of *continuous character data,* but we will not cover this case.

6.1 CHARACTER STATES AND THE PERFECT PHYLOGENY PROBLEM

We start this section with some general remarks about assumptions on characters and the meaning of states in interior tree nodes.

The basic assumption regarding characters is a somewhat obvious one: that the characters being considered are "meaningful" in the context of phylogenetic tree reconstruction. The task of defining what is meaningful and checking that the chosen characters are meaningful according to this definition rests largely with the scientist, or "user" of the tree. Nevertheless, we briefly mention two assumptions that underlie all character-based phylogeny reconstruction methods.

The first is that characters can be inherited independently from one another; this is crucial for all algorithms we will discuss. Another assumption is that all observed states for a given character should have evolved from one "original state" of the nearest common ancestor of the objects being studied. Characters that obey this assumption are called *homologous*. This is also important when the objects are biological sequences. In such cases, a character will be a position in the sequence, and its states, in the case of

DNA, will be A, T, C, and G. If we are comparing position, say, 132 of sequence s to position 722 of sequence t, we should make sure that these two positions owe their present state to the same position in an ancestral sequence.

Regarding interior nodes, we recall that they represent hypothetical ancestor objects. The algorithms we describe assign specific states to these nodes, but the set of states of an interior node does not necessarily have biological significance. By this we mean that the characterization of interior nodes should not be taken as a back-prediction of the features of ancestral objects; the main goal is always the grouping of present-day objects, and the reconstruction of ancestor objects is just a means to this end. Nevertheless we have to be cautious about the assignment of states to interior nodes, in order to obtain a tree as close as possible to the true tree.

Having made these introductory remarks, we pass on to a formal definition of the *character state matrix*. We define the character state matrix as a matrix M with n rows (objects) and m columns (characters). Thus M_{ij} denotes the state the object i has for character j. There can be at most r states per character and states are denoted by nonnegative integers. A given row of this matrix is the *state vector* for an object. Since we will also assign states to interior nodes in the tree, we will also say that interior nodes have associated state vectors. An example of a character state matrix is seen in Table 6.1.

TABLE 6.1

A character state matrix.

	Character				
Object	c_1	c_2	c_3	c_4	c_5
A	1	1	0	0	0
B	0	0	1	0	1
C	1	1	0	0	1
D	0	1	1	1	0
E	1	1	0	0	1

When trying to create a phylogeny from a character state matrix, we encounter some difficulties. The first arises when two or more objects have the same state for the same character. Almost all methods for phylogeny reconstruction are based on the assumption that objects that share the same state are genetically closer than objects that do not. However there exists the possibility that two objects share a state but are not genetically close. A case in point is the presence of wings in bats and birds. Such phenomena are called *convergence* or *parallel evolution*. We observe in nature that such cases are rare; therefore, we will deal with this difficulty in the definition of our tree reconstruction problems in the following manner: Convergence events should not happen, or their number should be minimized.

The second difficulty has to do with the relationships among different states of the same character. Let us illustrate this problem using the matrix shown in Table 6.1. Suppose that there was an ancestral object X from which objects A and B evolved. Which state should we assign to X relative to character c_1? Note that $c_1 = 1$ for A and $c_1 = 0$ for B. If we make $c_1 = 0$ for X we are saying that 0 is the *ancestral state* and that 1 is the

derived state. Assume we have made such a choice, and now suppose that objects C and D have as ancestral object $Y \neq X$. Further assume we have decided that Y's state for c_1 is 1. In this case notice that object D presents a *reversal* to the ancestral state 0 for character c_1. In this particular case, since the characters are binary, we could interpret this situation in terms of gains and losses of the character. That is, A has acquired the character with respect to its ancestor X, whereas D has lost it with respect to its ancestor Y. As in the case of the difficulty discussed in the previous paragraph, we observe in nature that such reversals are uncommon. Therefore we will handle reversals the same way we did with convergence events: We will require reversals not to happen, or that their number be minimized. Notice that we may not know beforehand which state is ancestral and which is derived. If we do not, we will have to choose, and then remain consistent with this choice in order to avoid or minimize reversals.

In the previous example we dealt with a binary character. Because in general there can be r states for a given character, the relationships between them may be more complex. Depending on how much we know about these relationships, characters can be classified as **ordered** or **unordered**. For an unordered character we assume nothing about the way states can change; that is, any state can change into any other. In the case of ordered characters extra information is known. For example, a given character with four states may have the following *linear* order: $3 \leftrightarrow 1 \leftrightarrow 4 \leftrightarrow 2$. This means, among other things, that in the reconstructed tree there should be no change of state 3 directly into state 4; state 1 should always be an intermediate. In other words, if there is a node with state 3 for the given character, there should be no edge linking it to another node with state 4. Characters may also be *partially ordered*, in which case there is a *derivation tree* that shows how states are derived from one another by means of a tree. Note that even with ordered characters we are not saying anything about the *direction* in which state changes must take place; characters for which the direction of change is known are said to be **directed**. In the literature unordered characters are also known as *qualitative characters* and ordered characters as *cladistic characters*. Directed characters are also known as *polar characters*. Intuitively, ordered characters provide us with much more information on how we should build the phylogeny, and this intuition is useful in understanding some computational complexity results below.

If we want to altogether avoid convergence events and reversals, the considerations above require that the desired tree T have the following property: for each state s of each character c, the set of all nodes u (leaves and interior nodes) for which the state is s with respect to c must form a subtree of T (that is, a connected subgraph of T). This means that the edge e leading to this subtree is uniquely associated with a transition from some state w to state s. A phylogeny with this property is a *perfect phylogeny*. We are now ready to state the central problem of phylogeny reconstruction based on character state matrices. This problem is known as the *perfect phylogeny problem:*

PROBLEM: PERFECT PHYLOGENY
INSTANCE: A set O with n objects, a set C of m characters, each character having at most r states (n, m, r positive integers).
QUESTION: Is there a perfect phylogeny for O?

For the matrix shown in Table 6.1, assuming 0 is ancestral and 1 is derived, such a phylogeny does not exist, as will be shown in the next section. For the matrix shown in Table 6.2 a perfect phylogeny does exist and is shown in Figure 6.2. A character labeling

an edge in the figure indicates that a transition from state 0 to state 1 takes place along that edge, such that the subtree below the edge contains all objects that have state for 1 that character and no others.

TABLE 6.2

Another character state matrix.

	Character					
Object	c_1	c_2	c_3	c_4	c_5	c_6
A	0	0	0	1	1	0
B	1	1	0	0	0	0
C	0	0	0	1	1	1
D	1	0	1	0	0	0
E	0	0	0	1	0	0

n

m

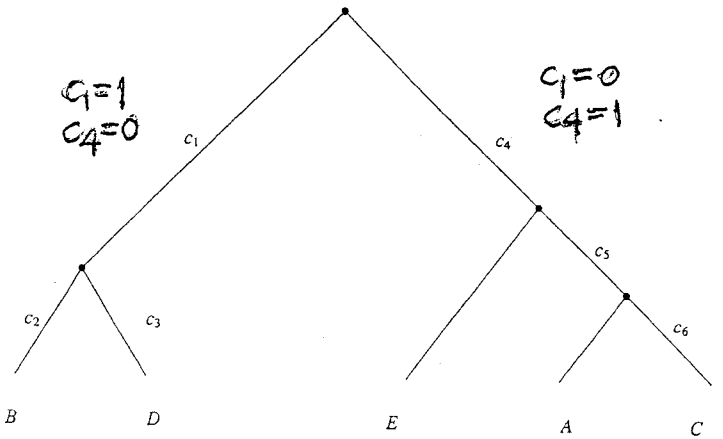


FIGURE 6.2

A phylogeny corresponding to the matrix in Table 6.2. Edges where state transitions take place are labeled by the corresponding character.

Whenever a set of objects defined by a character state matrix admits a perfect phylogeny, we say that the defining characters are **compatible**. This notion of compatibility will be later refined for the special cases of the problem that we will study.

Having defined the problem, the first question we would like to ask is this: Is there an efficient algorithm for solving it? We shall answer this question in a moment, but first let us have an idea of how many different trees can we build for n objects. Suppose we want only unrooted binary trees. Recall that we are interested in trees where objects are the leaves; hence in general it does matter which object is what leaf. ~~For three objects there~~

Si può dimostrare che per n oggetti esistono > n! alberi.

It turns out that whether the perfect phylogeny problem is computationally hard depends on the nature of the characters. If they are unordered, we have bad news: The problem is NP-complete. But if they are ordered, then it can be efficiently solved. Having these two facts as background, we now proceed to study in more detail some special cases of the problem that represent a sample of algorithmic results for character state matrices.

6.2 BINARY CHARACTER STATES

First we examine the special case of the perfect phylogeny problem in which characters are binary, such as those from Tables 6.1 and 6.2. Notice that for this case the distinction between ordered and unordered characters depends on knowing whether the characters are directed. This means that if we know that state 0 is ancestral and 1 is derived (or vice versa) then the character is ordered; otherwise it is unordered. As we will see, with binary characters (ordered or unordered) the perfect phylogeny problem can be solved efficiently; we present in this section an algorithm that runs in time $O(nm)$.

We present the algorithm as working in two phases. (In an actual implementation these two phases can be combined, but for presentation purposes this separation is better.) In the first phase the algorithm *decides* whether the input matrix M admits a perfect phylogeny. If it does, then the second phase of the algorithm will construct one possible phylogeny.

In the algorithm to be presented we assume that state 0 is ancestral and state 1 is derived. We will see later on how to drop this restriction. Notice that because we are dealing with directed binary characters, a rooted tree T that is a perfect phylogeny for input matrix M will have the following property, already mentioned in the previous section: To every character in input matrix M there corresponds an edge in T, and this edge marks the transition from state 0 to state 1 for that character. Such edges will be labeled by their respective characters. The root always has character state vector $(0, 0, \ldots, 0)$. This means that when we traverse the path linking an object i in a leaf to the root the edge labels found along the way correspond to characters for which object i has state 1. (See Figure 6.2.)

In the special case under study, there is a very simple necessary and sufficient condition that enables us to tell whether a given matrix M admits a perfect phylogeny. To state this condition, we first need a few definitions.

Each column j of M is a character. Thus we use without distinction the terms *column* and *character*. Each row i of M is an object, and we also use without distinction the terms *row* and *object*.

DEFINITION 6.1 For each column j of M, let O_j be the set of objects whose state is 1 for j. Let $\overline{O}_j$ be the set of objects whose state is 0 for j.

The simple condition is the following:

LEMMA 6.1 A binary matrix M admits a perfect phylogeny if and only if for each pair of characters i and j the sets O_i and O_j are disjoint or one of them contains the other.

Proof. Let us assume that M admits a perfect phylogeny. Because M is binary, we know that to each character i we can uniquely associate an edge (u, v) in the tree. Moreover, the subtree rooted at v (assuming it is the deeper of the two nodes) contains all nodes having state 1 for character i, and any node having state 0 for character i does not belong to this subtree. Suppose now that there are three objects A, B, and C such that

"only if" part

$A, B \in O_i, C \notin O_i$ and $B, C \in O_j, A \notin O_j$. This means that, according to character i, A and B belong to the same subtree but C does not. However, according to character j, B and C do belong to the same subtree, which is a contradiction.

Let us now suppose that all pairs of columns of M satisfy the condition stated in the lemma. We will show inductively how to build a rooted perfect phylogeny. Assume we have only one character, say 1. This character partitions the objects into two sets, $A = O_1$ and $B = \overline{O}_1$. Create a root and then node a for set A and node b for set B. Link node a to the root with an edge labeled by 1. Link node b to the root by an unlabeled edge. As a final step, we split each child of the root into as many leaves as there are objects in them. The tree thus created is clearly a perfect phylogeny; this is our base case. Now assume we have built a tree T for k characters, but without having executed the final step (that is, there are no leaves; nodes still contain sets of objects). We now want to process character $k + 1$. This character also induces a partition in the object set, and using it we should be able to get tree T' from T. We will be able to do this as long as the partition induced by character $k + 1$ separates objects belonging to the *same* node. If this were not the case, we would be forced to label *two* edges with label $k + 1$ and the resulting phylogeny would not be perfect. But such a situation cannot happen. Suppose it did; that is, suppose character $k + 1$ separates objects belonging to the nodes a and b. Because these are different nodes, there must be a character i that led them to be in different nodes in the first place. It is the case that $O_i \cap O_{k+1} \neq \emptyset$, because either the objects in a or those in b belong to O_i. But it is also the case that O_i does not contain nor is contained in O_{k+1}, because either the objects in a or those in b are *not* in O_i. But this would contradict our hypothesis, and the lemma is proved. ■

"if" part

From this lemma it is easy to see why the character state matrix shown in Table 6.1 does not admit a perfect phylogeny: Columns c_1 and c_5 do not satisfy the lemma's condition. We say that those characters are not **compatible**. The notion of compatibility between characters can here be given a precise meaning and is just another way to state Lemma 6.1: A collection of binary directed characters is compatible if and only if they are pairwise compatible, where pairwise compatibility is given by the set relationships in the lemma's statement.

We mentioned that we would present the algorithm for the binary case in two phases: In the first we decide whether the matrix admits a perfect phylogeny, and in the second we construct one, if possible. Lemma 6.1 already gives us an algorithm for the decision phase: Simply consider each column in turn and check whether it is compatible with all others. Each such check takes $O(n)$ and we have to make $O(m^2)$ checks, resulting in an algorithm with running time $O(nm^2)$. From the lemma's proof it is also possible to develop an algorithm for the construction phase, with the same time bounds.

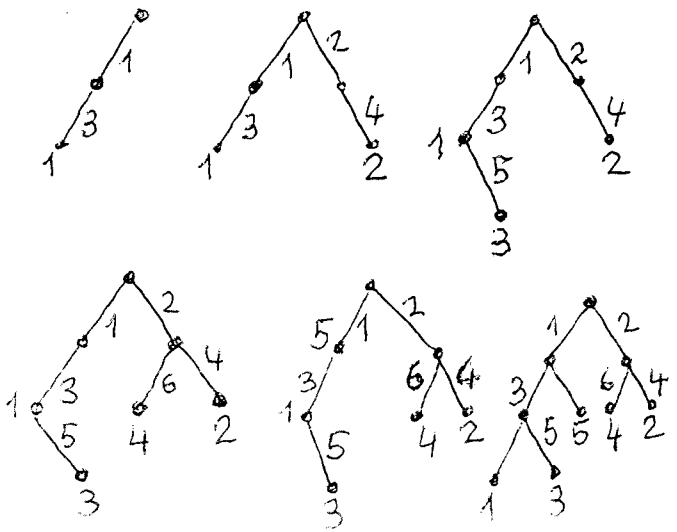
Esiste anche un algoritmo che funziona in tempo $\Theta(n \cdot m)$.

	C_1	C_2	C_3	C_4	C_5	C_6
A	1	0	1	0	0	0
B	0	1	0	1	0	0
C	1	0	1	0	1	0
D	0	1	0	0	0	1
E	1	0	0	0	0	0

La matrice di Fig. 6.2 con i caratteri rinominati per numero non-crescente di 1 (per es. C_4 è il precedente C_4 ecc.)

L'algoritmo indica gli oggetti con l'indice i di riga (es. A=1 ecc.) e i caratteri con gli indici j di colonna

Effetto dell'algoritmo a ogni ciclo di i, e passo finale



The Construction Phase

The idea in the construction is to process objects in turn, building the tree as we go. We start with a tree with a single node: the root. For each object, we look at the characters for which its state is 1. Whenever we see that there is no edge so far labeled with such a character, we create a new node and link it to the current node by an edge labeled by this character. The new node becomes the current node. If there already exists an edge labeled with this character, we simply move on to the next character, letting the current node be the endpoint of this edge. The algorithm is described formally in Figure 6.6. Notice that the tree obtained is rooted, but we can unroot it simply by removing the root and connecting its children. Let us now show that the tree constructed in this algorithm is a perfect phylogeny for M, and that the running time is $O(nm)$.

Algorithm *Perfect Binary Phylogeny Construction*
input: binary matrix M with columns sorted in nonincreasing order by number of 1s
output: perfect phylogeny for M
Create root
for each object i **do**
  curNode ← *root*
  for $j \leftarrow 1$ **to** m **do**
    if $M_{ij} = 1$ **then**
      if there already exists edge (*curNode*, u) labeled j **then**
        curNode ← u
      else
        Create node u
        Create edge (*curNode*, u) labeled j
        curNode ← u
  Place i in *curNode*
for each node u except *root* **do**
  Create as many leaves linked to u as there are objects in u

FIGURE 6.6

An algorithm that constructs a perfect phylogeny from directed binary characters.

We first recall that each character in M must correspond to one and only one edge of the tree, and that is precisely what the algorithm does. In addition, in a path from the root to a leaf (object) i we should traverse the edges corresponding to the characters for which i has state 1, and no others, and again that is what happens. The running time is $O(nm)$ given that we look at each element from M exactly once, and we spend constant time per element. Applying the algorithm to the matrix in Table 6.2 we obtain the phylogeny shown in Figure 6.2.

6.4 PARSIMONY AND COMPATIBILITY IN PHYLOGENIES

We have seen in the previous sections that even though the perfect phylogeny problem is NP-complete for unordered characters, the fixed-parameter special cases are solvable in polynomial time. Fixed parameter algorithms might seem good enough in practice, but the problem is that real character state matrices are unlikely to admit perfect phylogenies. The reason is twofold: Experimental data in biology always carries errors, and the assumptions made by the algorithms mentioned (no reversals and no convergence) sometimes are violated. For these reasons other approaches for reconstructing phylogenies are necessary.

Ignoring errors but working around the assumptions, two approaches suggest themselves. The first is to allow reversal and convergence events, but to try to *minimize* their occurrence. This is known as the *parsimony criterion*. The second is to insist in avoiding reversals and convergence, but to exclude characters that cause such "problems." It is obviously a good idea to try to exclude as few characters as possible. This means that in this approach we try to find a maximum set of characters such that we can find a perfect phylogeny for them (or, in other words, a maximum set of characters that are compatible). This is known as the *compatibility criterion*.

Both criteria result in optimization problems. Such problems are in general harder than decision problems, such as the perfect phylogeny problem considered in Section 6.1. Because we know that deciding perfect phylogeny for unordered characters is an NP-complete problem, the problems just sketched should be at least just as hard; and so they are. But we might have better luck in the case of *ordered* characters, since, as noted in Section 6.1, for them the perfect phylogeny problem *is* polynomially solvable. Unfortunately we have no such luck, and both criteria outlined above result in NP-hard problems for ordered characters as well. Below we discuss these problems in more detail.

When using the parsimony criterion, we try to minimize reversal and/or convergence events, as already mentioned. In general this means that we want to minimize the number of *state transitions* (that is, edges) in the reconstructed phylogeny. If we allow

reversals but forbid convergence, we get what is known as the *Dollo parsimony criterion*. If, on the other hand, we forbid reversals but minimize convergence events we get the *Camin-Sokal parsimony criterion*. Both criteria result in NP-hard problems.

The concept of parsimony also appears in the literature in connection with a different problem. Given a set of objects *and* the tree through which they are related, assign states to the interior nodes such that a minimum number of state transitions occur from node to node. This is a problem for which efficient algorithms exist depending on the exact characterization of the problem. We will not cover such algorithms.

Ciò significa che i pacchetti software esistenti per risolvere esattamente questo problema sono utilizzabili solo per alberi estremamente piccoli. Esistono però algoritmi euristici per soluzioni approssimate (si veda il testo citato di Setubal-Meidanis).

The version of the compatibility problem to be shown NP-complete is the following:

PROBLEM: COMPATIBILITY
INSTANCE: a character state matrix M with n objects and m directed binary characters, and a positive integer $B \leq m$.
QUESTION: Is there a subset L of the characters that satisfies Lemma 6.1 with $|L| \geq B$?

THEOREM 6.4 COMPATIBILITY is NP-complete.

Dunque il problema rimane intrattabile anche se viene ristretto al caso molto semplificato relativo al Lemma 6.1, e anche questo caso deve essere affrontato con algoritmi euristici. Si noti che il parametro B serve a stabilire un limite inferiore al numero di caratteri di L che permettono di ottenere una filogenia perfetta (scartando gli altri caratteri non in L). Gli algoritmi euristici non considerano un valore per B, costruendo filogenie perfette significative ma senza garanzia che ne esistano altre con un maggior numero di caratteri.