The Parsimony Principle in Phylogenetics

by

Alex Musselwhite

MA4K8 Scholarly Report

Submitted to The University of Warwick

Mathematics Institute

April, 2019

Contents

1 Introduction

Ever since Charles Darwin first presented his theory of natural selection in 1859, it has intrigued society with its potential to answer the question of life's origins. The elegant theory suggested that all life on earth was the result of millions of years of trial and error. New species would appear through random mutation and the successful would survive to be the progenitors of yet more new species.

As such, the theory has raised the question of whether it might be possible to use the evidence of the species we have today to try and build a history of evolution; whether it might be possible to build a complete history of the sequence of random mutations through which all the diversity of modern life came to be. The field seeking to do this is phylogenetics.

The field of phylogenetics concerns itself with phylogenetic reconstruction, the use of evidence from modern species and the fossil record to build a picture of how the many species we have on Earth today relate to each other. Historically, this has been done through observation of a species' physical characteristics.

The results of this historical model are still taught today; various families of organisms are defined by whether they are warm- or cold- blooded, whether they have a bony endoskeleton or chitinous exoskeleton, whether they lay eggs or give birth to live young. This study has produced a rich tapestry relating life, but it is far from perfect. Questions of convergent evolution (such as the complicated, independent evolution of different types of eye) and evolutionary oddities which don't exactly fit the definitions of any biological group (such as the platypus) are difficult for it to manage.

However, the development of genome sequencing techniques and the millionfold increase in computing power which has unfolded over the past half-century have made possible new techniques for reconstructing phylogenetic histories by examining a species' DNA - the fundamental genetic code which determines all the characteristics of an organism.

In this report, we will provide information pertaining to one such method in particular: the parsimony principle, one of the oldest methods for constructing phylogenetic trees still in widespread use. We will establish the basic background in evolution necessary to understand the principles in use, then we will proceed to define parsimony as a concept before examining several available methods for using it to reconstruct phylogenetic trees. The ultimate goal is to provide the reader enough information to develop and implement such a method independently.

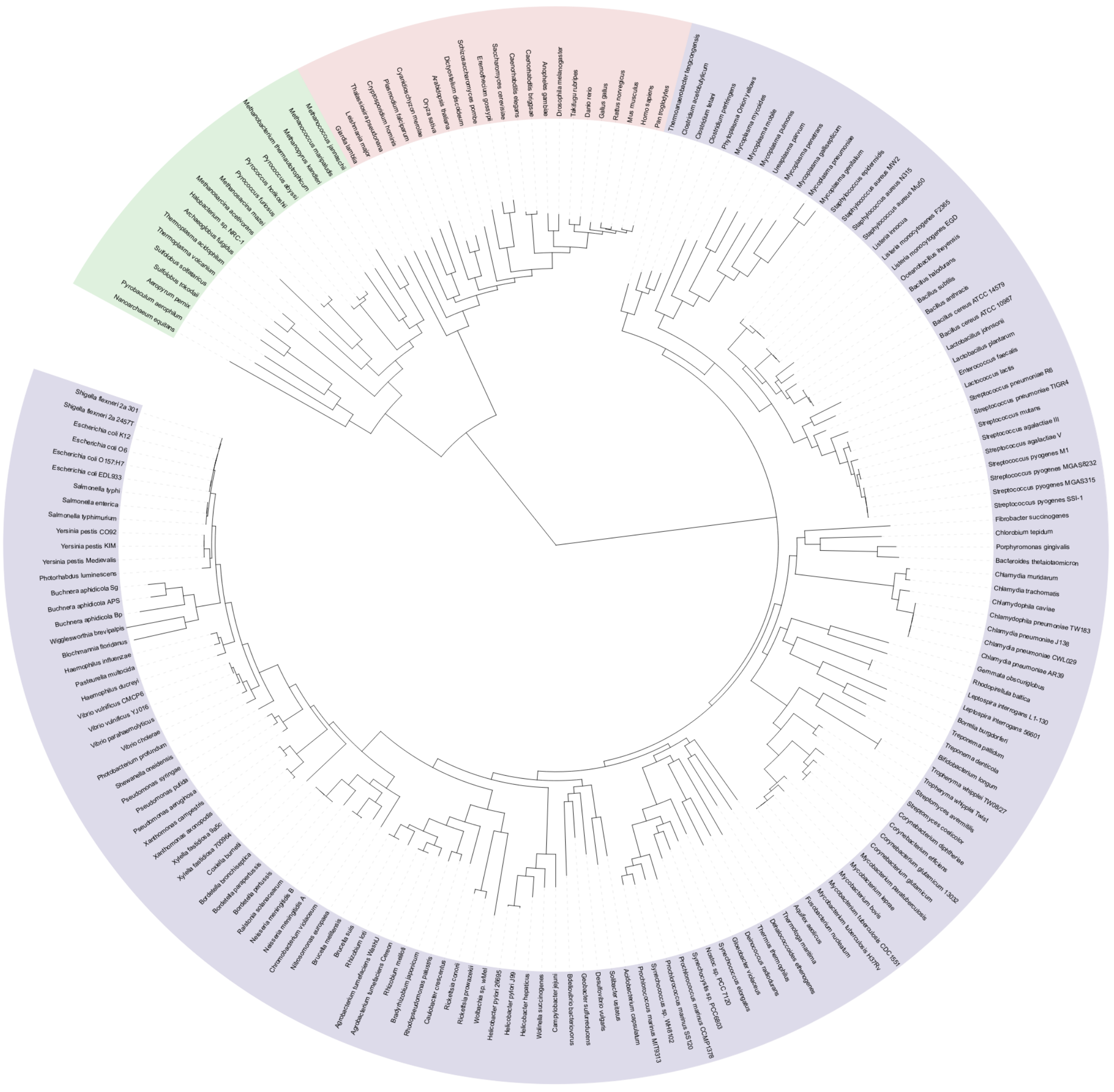


Figure 1: A version of the tree of life, taken from https://itol.embl.de/

2 Evolution

We will begin with an overview of the relevant biological background and graph theory for the rest of our discussion of phylogenetic trees and their construction.

2.1 DNA

Any modern method for the reconstruction of an evolutionary history relies on an understanding of DNA and the mechanisms of evolution. The DNA (or *genome*) of a species determines the structure of every protein which makes up a creature's body, controls its development, etc. Accordingly, DNA provides a rich source of information for understanding species and their relationships to each other.

DNA is composed of two *polynucleotide* strands coiled around each other in a double helix structure. Each strand is a chain composed of smaller units (monomers) called *nucleotides.* There are four nucleotides: two purines (*adenine* and *guanine*) and two pyrimidines (*cytosine* and *thymine*). As such, a DNA strand is usually written as a sequence with its entries in the set $\{A, C, G, T\}$ denoting the bases adenine, cytosine, guanine and thymine respectively.

In the DNA helix, each of these nucleotides always bonds with a specific *complementary* nucleotide on the other strand. Adenine bonds only to guanine, cytosine only to thymine, and vice versa. This means that for any DNA sequence, the base sequence of either of the two strands can be used to define the structure of the DNA. At any given point within the DNA strand, one of these strands (the *coding strand*) is copied into messenger RNA, which in turn encodes the structure of a given protein. Usually when DNA is written out as a nucleotide sequence, the coding strand is used.

Each protein is itself a chain of smaller units, this time one of 20 *amino acids.* Each possible sequence of three nucleotides (called a *codon*) corresponds to an amino acid in the protein whose structure the DNA encodes. Because there are $4^3 = 64$ possible 3-character nucleotide sequences and only 20 amino acids, each amino acid has multiple possible codons which code for its inclusion in a protein. In particular, often a change to the third nucleotide within a codon has no effect on the structure of the protein coded for by the DNA sequence.

Definition 2.1.1 (Notation of Genetic Sequences)**.** *A genetic sequence is denoted by a sequence whose entries are in the set $\{A, C, G, T\}$ corresponding to the four canonical bases, with spaces between each codon.*

For example, the sequence **AAC GCG TTA ATG** is an example of a (very short!) genetic sequence.

2.2 Mutation

The theory of evolution posits that all species on Earth ultimately descend from a common ancestor, the differences in characteristics between modern species emerging gradually over time as a result of incremental changes from one generation to the next accumulated over millions of years. In genetic terms, these mutations take the form of minor errors in the copying of DNA when an organism reproduces.

For the purposes of phylogenetic reconstruction, we are primarily interested in the effects of small-scale mutations, that is mutations which affect only a small number of nucleotides rather than whole chromosomes. Some examples of these are as follows:

Substitutions: the nucleotide at a particular entry in the genetic sequence is changed e.g. **AAC** $\rightarrow$ **AAT**.

Insertions: a sequence of nucleotides is inserted into an existing genetic sequence e.g. **AAC AGT** $\rightarrow$ **AAC TCT AGT**

Deletions: a subsequence of an existing genetic sequence is deleted e.g. **AAC TCT AGT** $\rightarrow$ **AAC AGT**

Most phylogenetic reconstruction techniques are particularly interested in the effect of substitutions.

Now these mutations occur slowly over time, and furthermore according to the theory of evolution every difference between species is the result of many of these mutations accumulating. So beginning with a common ancestor with a given genome, two or more different lineages emerge. Each is the result of a different sequence of mutations and hence possesses a very slightly different genome. Over time, the number of mutations experienced by each new species increases, and so the two species acquire more distinct genomes.

Following this logic in reverse, we can conclude that two species with very similar genomes are likely to have diverged from a common ancestor more recently than two species with very different genomes. The subject of DNA-based phylogenetic reconstruction is to exploit this in order to build up a complete evolutionary history encoded as a **phylogenetic tree**.

2.3 Aligned Orthologous Sequences

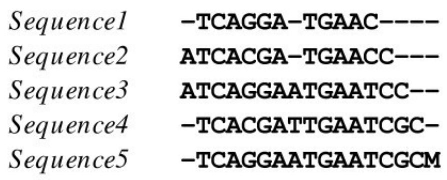
In order to perform phylogenetic reconstruction using the above logic, it needs to be the case that the DNA sequences we are comparing are "descended" from the same sequence. That is, each was produced from the same genetic sequence in a common ancestor by means of some finite sequence of mutations. If this assumption is false, then we are comparing unrelated segments of our species' genomes and therefore cannot draw meaningful conclusions. If this assumption is true, then we call the genetic sequences *orthologous.*

Most methods of phylogenetic analysis operate by taking orthologous sequences and aligning them so that one sequence may be transformed to the other by the smallest number of substitution mutations. The presence of insertions and deletions complicates this somewhat. In order to compare multiple sequences which have undergone insertions or deletions, we need to generate a *multiple sequence alignment*:

Definition 2.3.1 (Multiple Sequence Alignment)**.** *Given m genetic sequences S_i, create new sequences S_i' through repeated insertion of the 'gap' character '$-$' such that:*

- *There is no j, such that $S_{i,j} = -$ for all i.*

- *All the S_i' have the same length*



```
Sequence1   -TCAGGA-TGAAC----
Sequence2   ATCACGA-TGAACC---
Sequence3   ATCAGGAATGAATCC--
Sequence4   -TCACGATTGAATCGC-
Sequence5   -TCAGGAATGAATCGCM
```

Figure 2: Example of a Multiple Sequence Alignment. Note the gap characters.

Finding the multiple sequence alignment which maximises the genetic similarity between the sequences being compared is a difficult problem, and we will not address it here. For the rest of this report, we will be assuming that we have the luxury of a set of aligned, orthologous sequences from the taxa we wish to compare, and thus may use phylogenetic reconstruction techniques (in particular parsimony) to generate our phylogenetic tree.

2.4 Phylogenetic Trees

A phylogenetic tree is a binary tree encoding the relationships between a certain set of species (or *taxa*) in terms of their common ancestors. Recall the following definitions from graph theory:

Definition 2.4.1 (Graph Definitions)**.**

- A *graph* G is a vertex set V together with an edge set $E \subset (V \times V)$. An element (v_1, v_2) of E is said to join v_1 to v_2.
- The *neighbourhood* of a vertex v is the set of vertices joined to v by an edge.
- The *degree* of a vertex v is the size of its neighbourhood.
- A graph $G_1 = (V_1, E_1)$ is a *subgraph* of $G = (V, E)$ if and only if $V_1 \subset V$ and $E_1 \subset E$
- A *walk* from v_1 to v_2 is a sequence of vertices in V where adjacent vertices in the sequence are joined by an edge in the graph.
- A *path* from v_1 to v_2 is a walk from v_1 to v_2 containing no repeated vertices. The *distance* from v_1 to v_2 is the length of the shortest path from v_1 to v_2.
- A *tree* is a graph containing no cycles - that is, there is no path from any vertex in the graph to itself.
- A *directed graph* is a graph whose edges are ordered pairs (v_1, v_2) with v_1 called the *parent* and v_2 called the *child.*
- A *rooted* graph is a directed graph containing a vertex - called the *root* - which is the child of no vertex.

- A *rooted binary tree* is a rooted tree where each vertex has at most two children.
- An *unrooted binary tree* is a tree where all vertices have degree either one or three.
- A *weighted graph* is a graph G together with a function $w : E \to \mathbb{R}$. $w(e)$ is called the *weight* or *length* of the edge e.
- A *leaf* is a vertex in a tree with degree of exactly one. During this report, we will denote the set of leaves of a tree by L.
- Given a graph G, a *labelling* is a function $V \to I$ where I is some set of labels, often a subset of $\mathbb{N}$.

Furthermore, recall the following properties possessed by trees:

Definition 2.4.2 (Tree Properties)**.** *Any tree $T = (V, E)$ has the following properties:*

- Any two distinct vertices in T are joined by a unique path.
- $|E| = |V| - 1$.

We are now in a position to define what we mean by a phylogenetic tree.

Definition 2.4.3. *A **phylogenetic** or **leaf-labelled** tree is a binary tree (which may be rooted or unrooted) together with a labelling function $l : L \to Tax$ where Tax is some set of taxa.*

In a phylogenetic tree, each node corresponds to a species. The leaves of the tree correspond to the species (or taxa) we are comparing. Meanwhile, the internal nodes of the tree represent the common ancestor of their children. These nodes almost always represent species for which we do not have explicit genetic data available.

In figure 1 below, the species x represents the common ancestor of the taxa A and B. Meanwhile species y is the common ancestor of C and another species, itself the common ancestor of D and E.

3 Parsimony

We are now ready to move on to the method of phylogenetic reconstruction on which we will be focusing for this report - *parsimony.*

The parsimony principle is inspired by *Occam's Razor*, stating broadly that in cases where there are multiple competing explanations for a given phenomenon, we should assume that the simplest one is correct unless confronted with new data which disprove it. The word parsimonious means miserly, or stingy with money. We therefore say that a phylogenetic tree which assumes fewer changes than another is the *more parsimonious* of the two,

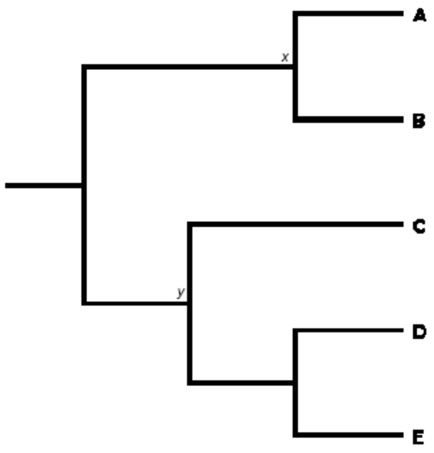


Figure 3: Example of a phylogenetic tree. x is the common ancestor of A and B.

despite (as we shall see) it having a lower parsimony score. The problem of finding the most parsimonious tree possible for a given set of trees is likewise referred to as the problem of maximum parsimony.

As applied to phylogenetic reconstruction, this is taken to mean: 'given multiple competing evolutionary histories, the one we should assume is correct is the one which requires the smallest number of significant changes to produce the modern species'. It is therefore an *optimality criterion* which allows us to evaluate a phylogenetic tree T and evaluate its merit as a hypothesised evolutionary history.

The correctness of this criterion applied to genetic evolution is debated. It is easy to see that the parsimony principle is likely to underestimate the total number of mutations which have actually occurred. Importantly, it does not generally account for 'hidden mutations' such as, for example, $A \to C \to T$. Instead, it will tend to assume that the single mutation AT. For this reason, parsimony-based analyses tend to return the best results when there is a high degree of similarity between the sequences being compared, as the small amount of evolution which has occurred in such cases can be taken to mean that relatively few hidden mutations will have occurred.

3.1 Miminal Extensions

Suppose we have a phylogenetic tree T comparing a set X of taxa that we wish to evaluate. Also suppose that Each element in X has associated to it some form of data which we can use to infer its evolutionary properties. We define this as a function $\chi : X \to S$ called a $|S|$-*state character* where S is some set dependent on the nature of the data we have to compare. For example, if we are comparing DNA sequences associated with our taxa, then the set S would be $\{A, C, G, T\}$ and the function χ might be "the nucleotide at position i

in the genetic sequence". Usually we will have a sequence of characters, such as an aligned DNA sequence, rather than being forced to rely on a single one.

Now we also have the tree T encoding some hypothesised evolutionary relationships between the taxa in X. In order to evaluate T according to the parsimony principle, we need to evaluate how many changes in state are required if T represents the "true" evolutionary history. To judge this, we need to hypothesise the values taken by χ at the internal nodes of the tree.

Definition 3.1.1 (Extension). *A function χ' is an extension of χ to T if χ' labels all vertices of T and $\chi'(x) = \chi(x)$ for all x leaves of the tree T. A set of characters $\{\chi_1, \chi_2, ..., \chi_r\}$ is usually called a character sequence.*

Intuitively, a minimal extension assigns to each species in the evolutionary history implies by T a hypothesised value of the character for that species. For example, if a character sequence $\{\chi_i\}$ is a genetic sequence for the taxa X, then $\{\chi_i'\}$ takes this and builds a speculative genetic sequence for each common ancestor of the species in X.

Given such an extension, the number of changes in state it requires can be defined as follows:

Definition 3.1.2 (State-Change). *Given $T = (V, E)$ and χ' an extension of χ to T, define $\delta : (v_1, v_2) \mapsto \{1, 0\}$ as follows:*

$$\delta(e, \chi') = \begin{cases} 1 \text{ if } \chi'(v_1) = \chi'(v_2) \\ 0 \text{ otherwise} \end{cases}$$

Denote the set of such extensions of χ to T by $Ext_T(\chi)$

Now we can also define the *state-change count* or *cost* of our extension χ' as a function to the natural numbers given by

$$c(\chi', T) = \sum_{e \in E} \delta(e, \chi')$$

Phylogenetic trees are also weighted. In a parsimony-based analysis, the weight of an edge e is given by $\sum_{i=1}^{r} \delta(e, \chi')$. Intuitively, this is the minimum number of evolutionary changes which must have occured given

Finally, we are in a position to define the parsimony score of χ over the tree T. For a given character χ and tree T, the *parsimony score* of T given χ is defined as follows:

$$ps_\chi(T) = \min_{\chi' \in Ext_T(\chi)} c(\chi', T)$$

We call an extension χ' *minimal* if and only if $c(\chi', T) = ps_\chi(T)$.

Finally, we need to extend this definition to one which allows for sequences of characters, as we would like to be able to work with entire DNA sequences.

Definition 3.1.3 (Parsimony Score)**.** *Given a sequence of character data* $C = \{\chi_1, \chi_2, ..., \chi_r\}$ *and a tree* T*, we define the parsimony score of* T *as:*

$$ps_{\{\chi_i\}}(T) = \sum_{i=1}^{r} ps_{\chi_i}(T)$$

3.2 The Fitch Algorithm

Now the idea of parsimony-based phylogenetic reconstruction is to find a *most parsimonious tree* given the character data C; that is, a tree T' with the lowest possible parsimony score. In order to do this, we need to be able to do two things: efficiently calculate the parsimony score of a given tree (called the *small parsimony problem*), and efficiently search the space of possible phylogenetic trees to find a tree with minimal parsimony score (called the *large parsimony problem*). The quality of our method will depend on how quickly and accurately we may solve these problems.

For now, we will examine the small parsimony problem. There exists an algorithm to efficiently calculate the parsimony score of a given phylogenetic tree:

Algorithm 3.2.1 (The Fitch Algorithm)**.**

The Fitch algorithm finds the parsimony score $ps_\chi(T)$ of a tree T by means of the following process:

1. If T is unrooted, then arbitrarily root it by selecting an edge (v_1, v_2), deleting it, and creating a vertex ρ connected to both v_1 and v_2. ρ is now the root of the tree. Denote the newly-rooted tree by T^ρ. Otherwise if the tree is already rooted then denote its root by ρ.

2. Assign to each of the leaves of T^ρ corresponding to the taxa in $x \in X$ the pair $(0, \{\chi(x)\})$.

3. Now select an internal node of T^ρ both of whose children have been assigned a pair (n, A) where $n \in \mathbb{N}$ and $A \subset S$. Denote these pairs by (n_1, A_1) and (n_2, A_2). Assign to this node the pair:

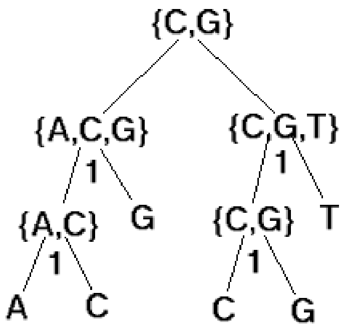


Figure 4: A visualisation of the Fitch Algorithm.

$$\begin{cases} (\mathrm{n}_1 + n_2 + 1, A_1 \cup A_2) \text{ if } A_1 \cap A_2 = \emptyset \\ (\mathrm{n}_1 + n_2, A_1 \cap A_2) \text{ otherwise} \end{cases}$$

4. If ρ has been assigned a pair (n_ρ, A_ρ) then return n_ρ. Otherwise, repeat step 3.

A similar algorithm by Hartigan does not require that we root the tree. However, due to the nature of the parsimony criterion, it turns out that the location of the root ρ has no impact on the parsimony score of the tree:

Lemma 3.2.2. *Regardless of the placement of the root ρ, $ps_\chi(T) = ps_\chi(T^\rho)$*

Proof: Note that the node ρ has two neighbours, say v_1 and v_2. Now by definition of parsimony, we have some minimal extension χ' such that $ps_\chi(T^\rho) = c(\chi', T^\rho)$. Consider the value of $\chi'(\rho)$.

We see that either $\chi'(\rho) = \chi'(v_1)$ or $\chi'(\rho) = \chi'(v_2)$: if this were not the case then we could construct an extension θ' with fewer state changes by setting $\theta'(\rho) = \chi'(v_1)$ and $\theta'(v) = \chi'(v)$ for all other vertices, and χ' would therefore not be minimal.

So assume without loss of generality that $\chi'(\rho) = \chi'(v_1)$. Now create an extension ζ' on T by $\zeta'(v) = \chi'(v)$ for all vertices of v.

We see that $\delta((v_1, \rho), \chi') + \delta((v_2, \chi), \chi') = \delta((v_1, v_2), \zeta')$. All other edges e are in both T and T^ρ, so $ps_\chi(T^\rho) = c(\zeta', T) \leq ps_\chi(T)$.

Similarly, consider a minimal extension γ' of χ on T. Define an extension η' of T^ρ by $\eta'(v) = \gamma'(v)$ for all vertices except ρ, and $\eta'(\rho) = \gamma'(v_1)$. The cost of this extension is the parsimony of T, so we have by definition that $ps_\chi(T) \leq ps_\chi(T^\rho)$. $\square$

Having demonstrated that it does not matter that the Fitch algorithm creates a root ρ, we will now establish that it is able to correctly find the parsimony score of the tree T.

Theorem 3.2.3 (Correctness of the Fitch Algorithm). *The output of the Fitch algorithm* *described above is the parsimony score $ps_\chi(T)$. Furthermore, the set A_ρ assigned to ρ by the algorithm is the set of labels assigned to ρ by at least one minimal extension of χ over T^ρ.*

We proceed by induction on the size of the unrooted tree T.

Denote the children of ρ by v_1 and v_2. By T_i, denote the binary subtree of T^ρ rooted at v_i. Denote by X_i the labels of the leaves of T_i so that X is the disjoint union of X_1 with X_2. Finally, let χ_i denote the restriction of χ to X_i.

For the case where $|T| = 2$, it is clear that the algorithm is correct, returning $ps_\chi(T) = 0$ if $\chi(v_1) = \chi(v_2)$ and 1 otherwise.

Now suppose $|T| = n > 2$. Let χ' be some minimal extension of χ to T^ρ. Denote by χ'_i the restriction of χ' to the subtree T_i. By the inductive hypothesis, the Fitch algorithm assigns to v_i the pair $(ps_\chi(T_i), A_i)$ with $\chi'_i(v_i) \in A_i$.

By the same logic we used in the proof of the lemma above, the minimality of χ' means we have two cases:

1. $\chi'(\rho) = \chi'(v_1) = \chi'(v_2)$
2. $\chi'(\rho) = \chi'(v_i) \neq \chi'(v_j)$ with $i \neq j$

Consider first case (1). Now it must be the case that at least one of the χ'_i is minimal: Suppose that neither are. Observe that the cost of χ' is the sum of the costs of χ'_1 and χ'_2. Then choose minimal extensions ζ'_1, ζ'_2 of T_1, T_2 respectively. We have that $c(\zeta'_i, T_i) \leq c(\chi'_i, T_i) - 1$ by definition of minimality. We then construct an extension ζ' by defining $\zeta'(v) = \zeta'_i(v)$ for all v in T_i, and set $\zeta'(\rho) = \zeta'(v_1)$. Now the cost of $zeta'$ is at most the sum of the costs of ζ_1 and ζ_2 plus one if $\zeta'(\rho) \neq \zeta'(v_2)$. This is strictly greater than the cost of χ', contradicting its minimality.

Case (1a): suppose first that both χ_1 and χ_2 are minimal. Then the parsimony of T^ρ is $ps_{\chi_1}(T_1) + ps_{\chi_2}(T_2)$ the cost of χ'. This is also the value returned by the algorithm. Now for any $a \in A_\rho$ we have that $a \in A_1, A_2$. Therefore there are minimal extensions η'_i with $\eta'_i(v_i) = a$. We can then construct a minimal extension η' by letting $\eta'(v) = \eta'_i(v)$ for all $v \in T_i$ and setting $\eta'(\rho) = a$.

Case (1b): Now suppose without loss of generality that only χ_1 is minimal. Then pick θ'_2 some minimal extension of χ_2 on T_2. Now define a new extension θ' which agrees with χ' on the vertices of T_1 and on ρ, but which agrees with θ'_2 on the vertices of T_2. The number of state-changes required by θ' on the vertices of T_2 is strictly less than by χ' by assumption. Furthermore θ' has created at exactly one new state change, namely on the edge $\{v_2, \rho\}$. It follows that θ' is minimal and therefore $ps_\chi(T^\rho) = ps_\chi(T_1) + ps_\chi(T_2) + 1$. Furthermore clearly A_1 and A_2 are disjoint, as otherwise we could use the process in case (1a) to create an extension with cost $ps_\chi(T_1) + ps_\chi(T_2)$ and so the Fitch algorithm returns the correct answer.

Now, take $a \in A_\rho = A_1 \cup A_2$. Suppose without loss of generality that $a \in A_1$. Then there is a minimal extension κ' with $\kappa'(\rho) = a$ constructed as follows: Take κ'_1 a minimal extension of χ_1 with $\kappa'_1(v_1) = a$ - this must exist by the inductive hypothesis. Take κ'_2 to be any minimal extension of χ_2. Then let κ' take the value a at ρ and agree with the κ'_i on the other vertices of T^ρ.

Finally, we consider case (2). In this case, suppose without loss of generality that $\chi'(\rho) = \chi'(v_1)$.

First, suppose that χ'_2 is not a minimal extension of χ_2. If this is the case, then we can construct an extension λ' by selecting a minimal extension λ'_2 of χ_2 and defining λ' to agree with λ'_2 on the vertices of T_2 but agree with χ' on the other vertices of T^ρ. Then λ has a cost strictly less than the cost of χ', contradicting the minimality of χ'.

On the other hand, suppose that χ'_1 is not a minimal extension of χ_1. We can define a new minimal extension ψ' agreeing with χ' except with $\psi'(\rho) = \chi'(v_2)$. Observe that ψ' and χ' have the same cost. We can then construct a new extension with strictly lower score than ψ' using an argument similar to the one above, contradicting the minimality of χ' in the same way.

It follows that the parsimony score of T^ρ is $ps_\chi(T_1) + ps_\chi(T_2) + 1$. Furthermore the sets A_1 and A_2 must be disjoint, else we could apply the same process we used in case (1a) to contradict the minimality of χ'. As a result, the Fitch algorithm gives the correct result in this case. $\square$

3.3 Computational Complexity

Having demonstrated that the Fitch algorithm is able to always produce the correct parsimony score for a tree T, we now turn to a discussion of its efficiency. In particular, in order to assess the use of the algorithm in a practical setting it is important to know its computational complexity.

Recall:
Definition 3.3.1 (Big Oh Notation). *Given two functions f, g from $\mathbb{N}$ to $\mathbb{N}$, we say that $f = \mathcal{O}(g)$ if and only if there is some constant $0 < c < \infty$ such that $f(n) < c.g(n)$ for all n.*

In particular, we are interested in determining the time T taken by the Fitch algorithm to complete as we vary the size of the input. This can be considered a measure of how "efficiently" the algorithm works in terms of how easily it is able to handle large amounts of data. Knowing the complexity of the Fitch algorithm will allow us to judge the sizes of the inputs we can use while still having the algorithm complete in a reasonable amount of time.

It turns out that the Fitch algorithm is remarkably efficient. In fact, it completes with *linear time* with respect to the size of the input data. That is to say, with respect to some parameter l measuring the size of the input data, the algorithm completes in a time $T(l) = \mathcal{O}(l)$.

Specifically, the Fitch algorithm is able to evaluate a phylogenetic tree T with a set X of n taxa and $\{\chi_i\}$ a length k sequence of s-state characters in time $\mathcal{O}(skn)$.

In order to see this, we walk through the algorithm: each pass of the Fitch algorithm evaluates one character χ_i, so it needs to be run exactly k times in order to calculate the parsimony score for the tree given the whole sequence $\{\chi_i\}$. Accordingly it completes in linear time with respect to k.

Furthermore, at each node, the algorithm must consider up to s possible states in turn when determining whether the sets A_i are disjoint. Accordingly, the algorithm completes in linear time with respect to s.

Finally, the algorithm considers each internal vertex of the tree T^ρ in turn. It is easy to prove by induction that a rooted binary tree with n leaves has $n-1$ internal vertices. Therefore the algorithm completes in linear time with respect to n.

An algorithm with linear time complexity is considered to be extremely efficient. The Fitch algorithm therefore provides an excellent solution to the small parsimony problem.

3.4 Weighted Parsimony

Now the basic version of parsimony we developed above treats all substitution mutations as carrying equal weight in determining the true evolutionary history. However, this need not necessarily be true - it may be (and usually is) that some mutations are noticeably more probable than others. As such, it can be helpful to weight the different mutations differently. For example, we might wish to impose a different penalty on the parsimony score of T for a mutation from a purine to a pyrimidine than for a mutation from a purine to a purine. If we do so, then instead of the state-change count delta we define a *weight function* w.

Suppose we have some s-state character χ. For each pair of states s_i, s_j we wish to weight the transition from state i to state j differently. We define a $s \times s$ matrix $W = (w_{ij})$. The entry w_{ij} represents the weight we wish to assign to the transition $s_i \to s_j$.

We replace δ with a weight function. Suppose we have a rooted phylogenetic tree $T = (V, E)$ with a minimal extension χ' and let $e = (v_1, v_2)$ be an edge of T. Say that v_1 is closer to the root ρ of T. Then:

$$w(e, \chi') = w_{ij} \text{ with } i = \chi'(v_1) \text{ and } j = \chi'(v_2)$$

$$ps_\chi(T) = \sum_{e \in E} w(e, \chi')$$

Notice that it is possible that $w_{ij} \neq w_{ji}$ so the placement of the root now matters in determining the parsimony of T. Also notice that we have not constrained ourselves to the case $w_{ii} = 0$.

For the calculation of weighted parsimony we can use the following modification of the Fitch algorithm:

Algorithm 3.4.1 (Sankoff's Algorithm)**.**

1. Assign to each leaf x an s-element vector $\mathbf{c}$. Index the entries of $\mathbf{c}$ by the elements of S. $\mathbf{c}$ is determined as follows:
   $c_i = \begin{cases} 0 \text{ if i} = \chi(x) \\ \infty \text{ otherwise} \end{cases}$
2. Choose a vertex v both of whose children v_1, v_2 have been assigned vectors $\mathbf{a}, \mathbf{b}$ respectively. Assign v the vector $\mathbf{c}$ with

   $$c_i = \min_{j \in S}(a_i + w_{ij}) + \min_{j \in S}(b_i + w_{ij})$$

   Repeat this step until the root ρ has been assigned such a vector.
3. If the vector of ρ is c, then output the lowest of the entries of c as the parsimony score of T given χ.

The entries of the vectors assigned to each vertex v can be thought of as encoding the lowest possible total weight of the subtree with v as its root given each possible value in s an extension might assign to v.

The proof of this algorithm's correctness is very similar to the one for Fitch's algorithm, so we omit it here.

There is a second type of weighting we can perform, where we weight the individual characters χ_i in a sequence. This might be done when some characters provide less information than others about evolution. For example, as mentioned briefly in the previous section, redundancy in the way DNA codes for various proteins means that often the third nucleotide in a given codon could take one of two or three possible values and the codon would still code for the exact same amino acid in the corresponding protein chain. Accordingly, we might assign character weights (y_i) and modify our calculation of a tree's final parsimony score by using

$$ps_{\{\chi_i\}}(T) = \sum_{i=1}^{r} y_i . ps_{\chi_i}(T)$$

3.5 Rooting Trees

If we are not using weighted parsimony or are using certain types of weighted parsimony then the placement of the root will be irrelevant to the parsimony score; our analysis will conclude that all possible positions for the root are somehow equally viable.

This may be unsatisfactory, in which case it would be good to have a method for determining the optimal placement for the root ρ.

There are several methods which may be employed. However, the most popular method is called the outgroup method. The nature of the method means that it is best suited

This method relies on possession of sequence data from a species y (the outgroup) which is distantly related to each of the other species in our set X of taxa (the ingroup) for comparison. In such a case, it is reasonable to assume that this outgroup diverged from its common ancestor with the in group before the members of the in group became distinct species. In outgroup rooting, we add y to our set X of taxa. When we are evaluating the parsimony of an unrooted tree T relating the elements of X, we always add the root ρ at the vertex incident to y at step (1) of the Fitch algorithm.

After the analysis is complete, we can then remove the vertices ρ and y and root T at the other child v of ρ if we do not wish to include y in our final analysis.

4 Searching Tree Space

Now that we have successfully recovered an efficient solution to the small problem, we turn our attention to the large parsimony problem. Namely, we attempt to find the following set of *most parsimonious trees*:

$$\{T | ps_{\{\chi_i\}}(T) = \min_{T'} ps_{\{\chi_i\}}(T')\}$$

In order to do this, we need some way of searching the space of all possible phylogenetic trees T' comparing $\{\chi_i\}$ in such a manner as to efficiently recover the most parsimonious tree. Sadly, it appears to be impossible to perform this task in a manner which is both computationally efficient and guarantees that we have found a most parsimonious tree.

4.1 Exhaustive Search

The simplest way to search tree space is to enumerate all the phylogenetic trees comparing out taxa X and computing the parsimony score for each of them. The computational complexity of this method will depend on the size of tree space with respect to the number of taxa $n = |X|$.

Theorem 4.1.1 (Size of Tree Space)**.** *If $|X| = n$ then there are $(2n-5)!!$ possible unrooted phylogenetic trees taking the taxa of X as leaves, where $!!$ is the double factorial defined recursively by $n!! = n(n-2)!!$ for $n > 2$, $0!! = 1!! = 1$.*

Similarly, there are $(2n-3)!!$ rooted phylogenetic trees.

In order to prove this theorem we will use the following lemma, the correctness of which can be easily proven by induction on the number of vertices of T:
Lemma 4.1.2. *An unrooted binary tree with n leaves has $(2n-3)$ edges.*

We will also make use of the following process, to which we will refer as *inserting* the leaf v:

1. Choose an edge $e = (v_1, v_2)$ of our tree T
2. Replace this edge with a vertex ρ connected to both v_1 and v_2
3. Create a new vertex v and connect it to only ρ

This takes an unrooted binary tree with leaves L and creates a new unrooted binary tree with leaves $L \cup \{v\}$. We can also perform this process in reverse to remove the leaf v. These processes are used in many inductive proofs on phylogenetic trees and are also used in many algorithms which construct them explicitly.

Proof of Theorem: We proceed by induction on the number n of leaves of T.

Clearly for the case $n = 3$ the supposition holds as there is only one unrooted tree with three leaves, namely the claw graph:

Suppose then we have a tree with $k-1$ leaves. Now we can create a new binary tree with k vertices by inserting a leaf v at one of the edges of T as described above. By the lemma, we have (2k - 5) edges at which we can insert v, each of which must produce a different tree.

Now suppose we have a set X of k taxa. Order these elements $\{x_1, x_2, ..., x_k\}$.

If we have a tree such T taking the elements of X as its leaves, then we can remove the leaf x_k to produce a tree with the $k-1$ leaves $\{1, 2, ..., k-1\}$. It follows that for each tree T with k leaves there is a unique tree with $k-1$ leaves to which we can insert x_k at some edge e to obtain T.

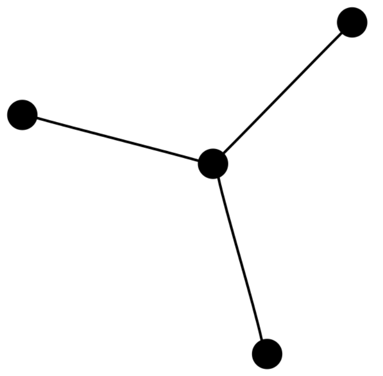


Figure 5: The claw graph with three leaves

So for each tree with $k-1$ leaves we can generate $(2k-5)$ distinct trees with k leaves. By the inductive hypothesis we have that the number of trees with $k-1$ leaves is $(2k-7)!!$. So the number of trees with k leaves is $(2k-5)(2k-7)!!$ as required.

Now consider that each rooted binary tree T^ρ may be generated from an unrooted tree T by selecting an edge on which to root the tree. By the lemma there are $(2k-3)$ edges for an unrooted binary tree with k leaves, so the number of rooted binary trees with k leaves is $(2k-3)(2k-5)!! = (2k-3)!!$ as required. $\square$

So in order to exhaustively search tree space for the most parsimonious trees we would need to consider $(2n-5)!!$ trees. As is shown in figure 5 below, this search space rapidly becomes far too large to search.

n	Number of unrooted binary trees with n leaves
3	1
5	15
9	135135
20	221643095476699771875

Figure 6: The size of the search space increases rapidly from 1 to over 221 quintillion

As a result, we clearly need a more efficient method than exhaustive search to find a parsimonious tree for a large number of taxa.

4.2 Branch and Bound

One approach which allows us to search tree space more efficiently is the branch and bound algorithm. This uses an upper bound on the minimum possible parsimony score to exclude large areas of the search space which we know do not contain a maximally parsimonious tree. It relies on the following property of parsimony:

Lemma 4.2.1. *Given a phylogenetic tree T generated by inserting a vertex v into a phylogenetic tree T', we have that $ps_{\chi}(T) \geq ps_{\chi}(T')$.*

It is easy to see that this lemma must hold, as the removal of a leaf v from a phylogenetic tree cannot increase the number of state changes in a minimal extension χ' of χ.

Algorithm 4.2.2 (Branch and Bound)**.**

Assume that we have a set $X = \{x_1, x_2, ...x_r\}$ of taxa together with character data $\{\chi_i\}$.

Begin by setting $B = \infty$. B will be our bound on the minimum parsimony score.

Now we build a search tree G as follows:

Let the root of the tree v correspond to the unrooted binary tree T_2 containing the two vertices x_1 and x_2 as leaves.

Now execute the following process:

1. If the tree associated to v does not contain all the x_i as leaves, connect vertices to v corresponding to all possible trees which can be generated from T by insertion of the vertex v_{j+1}, where j is maximal such that T has x_j as a leaf. Calculate each of their parsimony scores using the Fitch algorithm.

   If it does, then v is a leaf of G. Set g equal to $ps_{\{\chi_i\}}$ and record the tree corresponding to v.

2. Select a child vertex w of v such that the tree associated with w has a parsimony score of at most g and the least parsimony score among all trees associated with the children of v. Repeat step (1) with w in place of v.

3. Once there are no more valid vertices in G to consider, return g and all phylogenetic trees found which have a parsimony score of g. This is the set of maximally parsimonious trees for the character sequence $\{\chi_i\}$.

The branch and bound algorithm works by excluding trees from its search which are known to contain subtrees with a parsimony score exceeding the upper bound g. Because of this, it is often able to avoid checking a large number of G.

However, the efficiency of the branch and bound algorithm is dependent on being able to quickly obtain a good bound g. If we are unlucky, it may be that the bound g is higher than the parsimony of most trees and therefore that we still need to check a large number of trees.

Even with a good bound for the minimum parsimony score, the branch and bound algorithm still cannot guarantee us an efficient search. For larger numbers of taxa (say 21 or above) it is necessary to abandon the guarantee of finding the most parsimonious tree, and use a heuristic method which will return an estimate for the most parsimonious tree.

4.3 Local Search

Local search methods are an optimisation technique to find a minimum or maximum value of a function f (in our case the maximum parsimony of a character sequence $\{\chi_i\}$) used when we have a large search space. These methods are characterised as either maximising some score or *fitness function*, or as mimimising some weight or *cost function.* It is not hard to see that these are equivalent problems.

As opposed to a global search, which explicitly considers the entire search space g, a local search method begins with an initial estimate for the best solution and checks all "local" solutions to try and find a better one. As a result, local search methods can be used to find good (if not guaranteed to be optimal) values

The simplest form of local search method is a hill-climbing approach, which would be:

Algorithm 4.3.1 (Hill Climbing)**.**

1. Generate an initial tree T
2. Use Fitch's algorithm to calculate the parsimony of all trees T' in the neighbourhood $N(T)$ of T.
3. If no tree T' is found with strictly lower parsimony score than T then stop and return T.
4. Otherwise, replace T with some T' such that T' has least parsimony score of all trees in $N(T)$. Then repeat this process from step (2).

This will always find a local minimum within G - a tree T which has the least parsimony score of all trees in its neighbourhood. However, there is in general no way to guarantee that this is the global minimum.

Now in order to be able to perform this process, we still need the following:

- A method for generating the initial tree T.
- A definition of the neighbourhood $N(T)$.

We start by examining how to construct an initial T. One method of generating a reasonably-good initial estimate for the most parsimonious tree is similar to Branch and Bound and is known as *stepwise addition*:

Algorithm 4.3.2 (Stepwise Addition)**.**

As with Branch and Bound, we begin with T being the unrooted phylogenetic tree T_2 containing only x_1 and x_2 as leaves.

We then consider all possible possible rooted phylogenetic trees which can be created by inserting the leaf x_{i+1} where T already contains the vertices $\{x_1, x_2, ..., x_i\}$ as leaves. Then we select the tree T' with the lowest parsimony score of all the ones considered to replace T. Then we repeat until we have a tree containing all the x_i as leaves.

Our bound for g is then the parsimony score of our final tree.

Notice that this tree would be the first bound we obtained when running branch and bound fully.

A second method for constructing our initial tree algorithmically is star-decomposition. Given a set X of n taxa with character data χ, we create our initial graph to be the star graph $Star_n$, with a single internal vertex ρ connected to all the leaves x_i. A new tree T_{ij} can be generated by creating a new vertex v_{ij} connected to the vertices ρ, x_i, x_j, and deleting the edges (ρ, x_i) and (ρ, x_j). All such possible trees T_{ij} are created and evaluated by parsimony score. The T_{ij} with least score is picked. The step is repeated until the vertex ρ has degree exactly three.

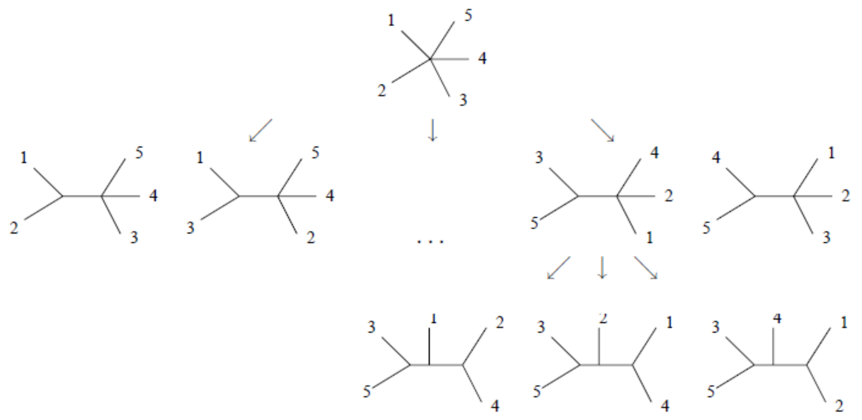


Figure 7: Illustration of Star Decomposition

Alternatively, we can generate our initial tree completely at random by once again beginning with T_2 and iteratively inserting the remaining x_i along random edges. This method will tend to produce initial trees T with higher parsimony score than one generated through stepwise addition, but has the advantage that we can easily run our hill climbing algorithm several times with different initial T to check our results.

4.4 Metrics on Tree Space

we now turn our attention to the construction of a neighbourhood $N(T)$ around T. The natural way to do this is to define some metric d on the space G_n of phylogenetic trees with n leaves. and let a neighbourhood be defined by $N(T) = \{T' : d(T, T' \leq k\}$ for some $k \in \mathbb{N}$.

The normal way to do this is to define some process $p : T \mapsto T'$ to rearrange a tree to create a new one. Then the distance $d_p(T, T')$ is the minimum number of times this process must be used to rearrange T to T' and vice versa. It is easy to see that this will be a metric provided that the process is reversible, that is that if we can rearrange T to T' using p once, then we can also use p once to rearrange T' to T. So we can define $N(T) = \{T' | d_p(T, T') \leq 1\}$

For example:

Definition 4.4.1 (Nearest Neighbour Interchange)**.** *Suppose we have an edge $\{u, v\}$ with four associated subtrees A, B, C, D arranged as shown below. A nearest neighbour interchange (or NNI) move swaps the position of two of these subtrees.*

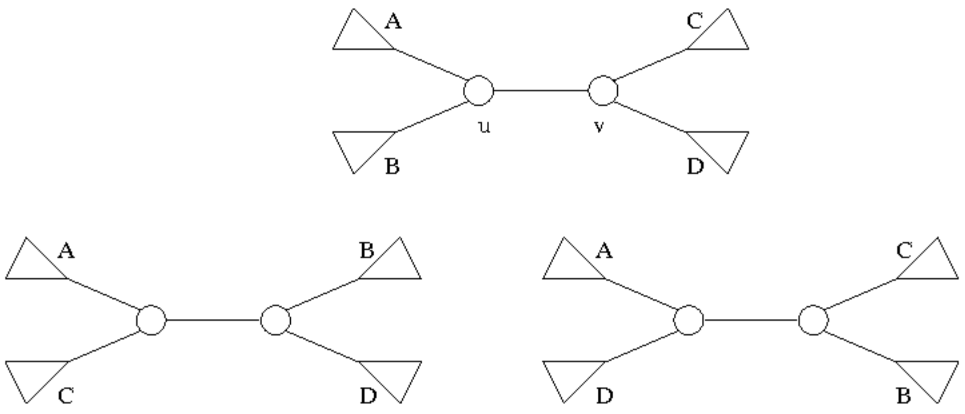


Figure 8: Illustration of NNI. The tree at the top can be rearranged to produce either of the trees below it.

Definition 4.4.2 (Subtree Pruning and Regrafting)**.** *Given a tree T containing an edge (u, v), a Subtree Pruning and Regrafting (or SPR) move does the following:*

- *Delete the edge (u, v), then delete the vertex v to leave two trees T_1 and T_2. Let T_1 be the tree containing u.*
- *Choose an edge of T_2 and create a vertex w at its midpoint.*
- *Create the edge (u, w), rejoining T_1 and T_2 to create a new tree T'*

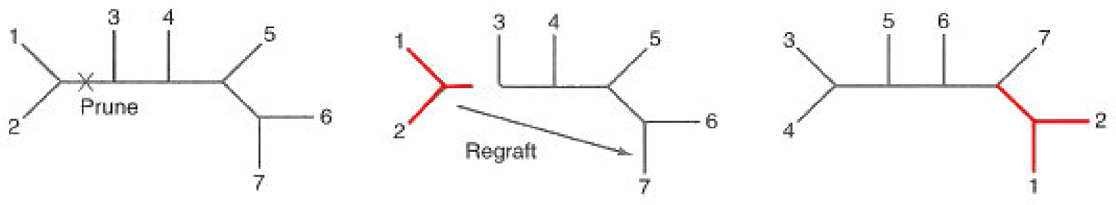


Figure 9: Example of an SPR move

Definition 4.4.3 (Tree Bisection and Reconnection)**.** *Once again given a tree T containing an edge (u, v), A Tree Bisection and Rearrangement (or TBR) move does the following:*

- *Delete the edge (u, v) and both vertices U, v to create two disjoint trees T_1, T_2.*
- *Select edges e_1 of T_1 and e_2 of T_2 respectively.*
- *Create vertices W_1, W_2 along e_1, e_2 respectively.*
- *Create the edge (w_1, w_2) to obtain a new tree T'.*

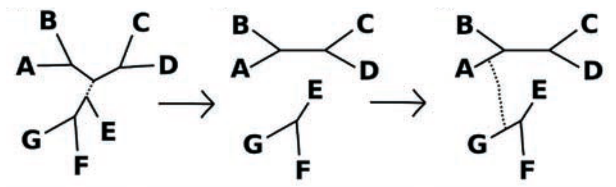


Figure 10: Example of a TBR move

The above moves each have their own associated metric, $d_{NNI}, d_{SPR}, d_{TBR}$ respectively. Furthermore, it is easy to see that they are related in the following way:
$N_{NNI}(T) \subset N_{SPR}(T) \subset N_{TBR}(T)$.

When selecting which of the above metrics to use, there is a trade-off of which we have to be aware. On the one hand, the larger a neighbourhood N, the more computationally intensive our hill-climbing algorithm will be as the number of trees we need to evaluate at each step increases. On the other hand, all other things being equal we would expect that the smaller the neighbourhood the more local minima will exist as the expected number of more parsimonious trees in the neighbourhood decreases.

Also, one of the important considerations to take into account when selecting a neighbourhood to use in hill-climbing is the diameter of the search space G, denoted $\Delta(G)$. That is, the greatest distance between two elements of the search space. Intuitively, this is because the smaller $\Delta(G)$, the fewer steps we need to make in the hill-climbing algorithm in order to reach the global minimum and therefore the higher the chance we will find it. Clearly $\Delta_{NNI}(G) \geq \Delta_{SPR}(G) \geq \Delta_{TBR}(G)$.

We shall now attempt to quantify the relationship between these three metrics in terms of neighbourhood size and search space diameter more explicitly in order to allow a more informed choice on which to use.

Theorem 4.4.4 (Neighbourhood sizes). *Given the space G_n, the following hold:*

1. $|N_{NNI}(T) - \{T\}| = 2(n-3)$
2. $|N_{SPR}(T) - \{T\}| = 2(n-3)(2n-7)$
3. $|N_{TBR}(T)| \leq \mathcal{O}(n^3)$

Proof: We begin by proving (1). Consider a phylogenetic tree T. We wish to count the number of trees which can be made from T' by a NNI move. We start by counting the number of possible NNI moves per tree. Recalling the definition of NNI moves, we see that there are two such moves per edge of the tree not connected to a leaf. Now it is easy to show by induction on the number of leaves of T that there are $n-3$ such vertices. Therefore there are $2(n-3)$ such moves.

Furthermore we can see that each of these moves produces a different tree: The two distinct NNI moves which fix the same edge $\{u, v\}$ are clearly distinct, so suppose we have two edges $e_1 = (u_1, v_1)$, $e_2 = (u_2, v_2)$. Suppose e_1 connects the subtrees A, B, C, D and assume without loss of generality that e_2 is a vertex in A. Observe that the NNI moves fixing e_1 do not change the layout of any of the subtrees A, B, C, D, only the way in which they connect to each other. However, the NNI moves fixing e_2 change the layout of A.

Proof of part (2): We begin in the same way, by counting the number of possible SPR moves possible given a tree T. Recall that in an SPR move, we delete some edge (u, v) leaving two trees, each containing exactly one of u and v. We then choose an edge in one of these two trees, and attach to it whichever of u and v is not in the same tree.

As such, the number of SPR moves is given by the number of ordered pairs of edges in T, which is $(2n-3)(2n-4)$ since the initial T has $2n-3$ edges. However, unlike in the case of NNI, we cannot guarantee that each of these moves will produce a distinct tree. The following cases result in overcounting:

i Suppose u is connected to the three vertices v, w, x. If the first edge in our ordered pair is (u, v) and the second is either (u, x) or (u, w) then we will delete the edge (u, v) only to immediately reattach v to u, producing no change in the tree. If v is not a leaf, this logic also applies to v. So the number of such SPR moves is 2 per edge which meets one of the leaves of T, and 4 for each remaining edge, giving a total of $2n + 4(n-3) = 6(n-2)$.

ii Suppose we have two edges (u_1, v_1), (u_2, v_2) such that $(u_1, u_2) =: e$ is also an edge of the tree. In this case, the SPR move is equivalent to a NNI move on the edge e. There are only 2 NNI moves fixing e. However, e meets four edges. As such there are 8 possible ordered pairs of edges of the sort identified in this case. As such, we have overcounted the number of distinct trees by 6. This holds for every internal vertex e of T so we have overcounted by $6(n-3)$

Beyond these cases, every SPR move does in fact give a unique result: Suppose we perform an SPR move which deletes (u, v) and reattaches U at the edge (w, x). Call the vertex at which u is attached y. Suppose there were another SPR move which gave an identical tree. Then y would still have to be connected to all of u, w, x. In order to be distinct, this move would have to attach either w or x to y. Without loss of generality, assume w. Then either the single edge (u, x) already existed (which is case i) or both the edges (u, y) and (x, y) already existed (which is case ii).

Cases (i) and (ii) are distinct by definition, so the total number of SPR-neighbours of T

is $(2n-3)(2n-4)-6(n-2)-6(n-3)$ which rearranges to $2(n-3)(2n-7)$ as required.

Finally, finding such an explicit formula for the number of TBR-neighbours of a tree is difficult but we can attain a bound by the same sort of combinatorial argument as in the cases above. To perform a TBR move, we select three edges: one at which to bisect the tree and two more at which to reconnect it. The precise number of options for the pair of edges at which to reconnect the tree depends on the choice of inital edge at which to bisect it. However it is no more than $(2n-3)(2n-4)(2n-5$, which is cubic as required. $\square$

We observe that SPR results in neighbourhoods significantly larger than NNI, with roughly $4n^2$ additional elements of $N_{SPR}(T)$ compared to $N_{NNI}(T)$ and that TBR can give us larger neighbourhoods still. This needs to be taken into consideration when selecting which of the three we use for a local search algorithm.

As a result of this, nearest neighbour interchange frequently suffers from premature convergence to a local optimum; the nature of nearest neighbour interchange means that each tree T neighbouring T' is very similar to T. A method of hill climbing using nearest neighbour interchange cannot reasonably expect to assess enough of the search space G to have a good chance of finding the global maximum parsimony.

TBR, on the other hand, suffers from the opposite problem. While a TBR-based hill climbing algorithm searches a very large neighbourhood, the fact that the size of $N_{TBR}(T)$ is potentially cubic with respect to n means that it is very computationally expensive to use.

Perhaps unsurprisingly, many current implementations of local search-based parsimony optimisation use SPR as a middle ground between the two. The size of its neighbourhood scales quadratically, which is seen as providing an acceptable compromise between a thorough search and speed.

5 Alternative Search Methods

Even with SPR and TBR, the hill climbing algorithm is still prone to finding a mere local minimum rather than the global minimum. There are various amendments which can be made to the algorithm in order to mitigate this flaw.

Variable Neighbourhood

The fact that we have three different notions of neighbourhood, each of increasing size, means that we are not necessarily constricted to using the same one for the entire runtime of our algorithm. In fact we can combine them in such a way as to benefit from the desirable properties of each.

Suppose we use a normal hill climbing algorithm with nearest neighbour interchange. After some time, it will likely halt with a local minimum tree T. However, as discussed, it is quite probable that T is not the tree with maximum parsimony.

Therefore, once the algorithm is unable to find a better tree $T'_{NNI}(T)$, we can switch to a SPR or TBR in order to search a larger neighbourhood for such a T'. This allows us to run a version of hill climbing almost as efficiently as a purely NNI based method while providing a result which is guaranteed to be a local minimum in a much larger neighbourhood. The largest neighbourhood usually used in methods of this kind is N_{2-SPR}, which is the set of trees which can be obtained by two iterations of SPR. It is easy to see that this neighbourhood has a size which is $\mathcal{O}(n^4)$.

Accepting Worse Solutions

Another approach is to modify the hill-climbing method such that under certain conditions a worse solution, or a tree with higher parsimony score than T, will be accepted as the new candidate solution. The idea is that the algorithm can escape from "shallow" local minima and find a region of the search space with generally lower score. As such, these methods perform a more comprehensive assessment of the search space.

Such methods include:

5.1 Tabu Search

Tabu search is an algorithm which works similarly to hill climbing, except that as it proceeds it places each tree T it visits in a "Tabu list" which is thereafter excluded from the search space. This means that the algorithm is able to accept higher-cost solutions without the danger of revisiting areas of the graph previiously considered and becoming trapped in a loop. An example of Tabu Search applied to parsimony might look something like the following:

Algorithm 5.1.1 (Tabu Search)**.**

Begin by constructing an initial tree T with character data $\{\chi_1, \chi_2, ..., \chi_n\}$. Also let the set Ta of tabu solutions be initalised to $\emptyset$, and decide on a stopping criterion (usually a certain amount of time or a certain number of iterations of the process below). Furthermore, let T_{best} be the tree with the lowest parsimony score of all those considered so far. Let T_{best} be T initially.

Now:

1. Construct a neighbourhood $N(T)$ using e.g. SPR.
2. Evaluate $ps_{\{\chi_i\}}(T')$ for all T' in this neighbourhood.
3. Add T to the tabu list.

4. Select the new candidate solution to be the T' with lowest parsimony score which is not in the Tabu list. If the parsimony score of T' is better than that of T_{best}, then set T_{best} to T'.

5. If the stopping criterion is not satisfied, then repeat this process from step (1). Otherwise, return T_{best} as the candidate solution to the maximum parsimony problem.

5.2 Simulated Annealing

Another popular method for mitigating the local minimum problem is Simulated Annealing. Rather than exhaustively search the neighbourhood around T, simulated annealing selects an element of it at random and moves to it with a probability p, usually 1 if the selected element is better than the current candidate solution. A variable $Temp$, called the *temperature*, is used to determine the probability p that the simulated annealing algorithm will accept a solution with a higher parsimony score than the current candidate solution. $Temp$ decreases over time, and the probability p decreases with it until simulated annealing reduces to normal hill climbing when $Temp$ reaches 0. The idea is by analogy to a physical system - the temperature represents the ability of the algorithm, which allows it to jump to states with a higher "energy", just as physical temperature allows particles to reliably achieve higher energy states.

The most important consideration to take into account when designing a simulated annealing scheme is determining how to decrease the temperature $Temp$. As a general rule, decreasing $Temp$ slowly will result in better solutions but the algorithm will obviously take longer to complete.

Algorithm 5.2.1 (Simulated Annealing)**.**

Begin with a candidate solution T. Select a decreasing function $Temp(\sigma) : \mathbb{N}-rightarrow[0, \infty)$ to determine the temperature, where σ is the number of possible candidate trees that have been assessed so far. It should be 0 at some predetermined number of steps σ_{max}. Also choose an increasing function $p() : [0, \infty) \to [0, 1]$ which will determine the probability that a worse candidate solution is accepted.

Now perform the following process:

1. Select a random tree T' from the neighbourhood of the candidate tree T.

2. If T' has lower parsimony score than T, then let T' be the new candidate solution.

3. If T' has higher parsimony score than T, then with probability p set the candidate solution to T', and with probability $1 - p$ maintain T as the candidate solution.

4. Increase σ by one.

5. Update and p.

6. Repeat from step (1) until the temperature is zero and there are no better solutions in the neighbourhood of T.
7. Return the tree T_{min} with the lowest parsimony score of all those visited.

While we have given example algorithms for both tabu search and simulated annealing, in reality they are both more families of approaches (or metaheuristics) rather than always being the exact same algorithm. Modifications to both can be made to better suit the nature of a given problem. For example, in tabu search there are a whole family of approaches for populating and updating the tabu list. There are also many adjustments one can make to simulated annealing; for example, one can allow the algorithm to revisit a previous solution if enough iterations have occured without an improvement in the best score found so far.

Unfortunately, there is no general method for determining how slowly the temperature has to be decreased in order to guarantee finding the global minimum. There are general proofs demonstrating that simulated annealing can find the correct global minimum with probability arbitrarily close to one, but the run times which would allow this tend to be far too long to be computationally feasible. Nonetheless, experience has demonstrated that in many problems simulated annealing does a good job finding solutions of comparable or higher quality to those found by hill climbing methods.

We shall examine one last metaheuristic before closing the report:

5.3 Genetic Algorithmns

Genetic Algorithms work by analogy to the physical process of evolution. As the wide variety of highly specialised organisms we have available to apply the knowledge in this report to attests, evolution appears to be very good at optimisation problems.

At initialisation, a pool of candidate solutions is created. Each of these solutions is assessed for "fitness" - in our case, the lower a tree's parsimony score the lower its fitness.

After this, a new pool of candidate solutions is created by a process of "breeding". For each new tree in the pool, two trees in the previous pools are selected and some process is applied to combine their properties. After this, there is a chance that a random mutation (in our case probably one of the neighbourhood-generating operations discussed in the previous section) may occur, which alters the tree further and helps maintain a diverse pool of candidate solutions.

After this process has been repeated a predetermined number of times, the most fit candidate solution in the current pool is selected as the output of the method.

We already have most of the concepts required to implement such an approach, but we are missing a way to combine two phylogenetic trees to produce a new one. There are a wide variety of such methods, and usually they benefit from being tailored to a specific problem. However, most of them do follow a general strategy of *subtree cutting and regrafting.* As can be guessed from the name, this is quite similar to the *subtree pruning and regrafting* method we defined earlier.

The process works as follows:

- Suppose we have two rooted phylogenetic trees T, T' comparing the same set of taxa X. Select some edge $e = (u, v)$ of T and cut it. This leaves two trees T_1 and T_2. Without loss of generality suppose that v and the root ρ of T are both in T_2.
- Let X_1 be the taxa which, together with u, form the leaves of T.
- Delete the leaves in X_1 from the tree T'.
- Select an edge e' of T'.
- Attach u to T' at e'.

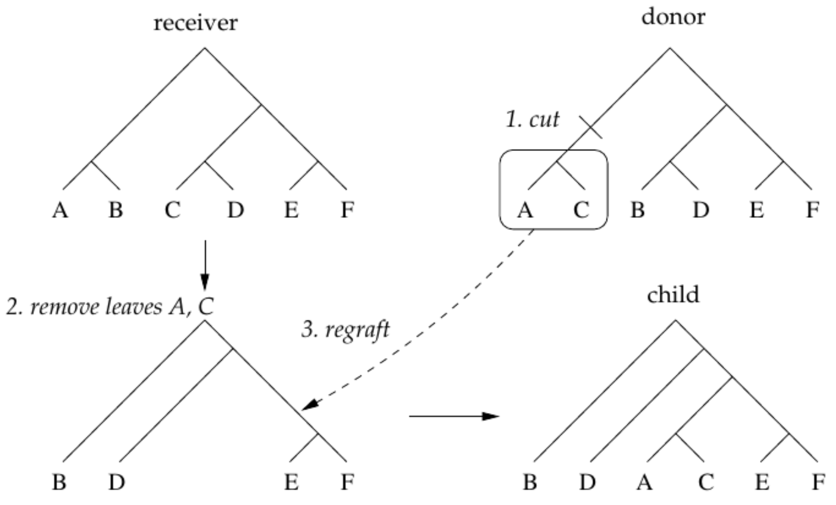


Figure 11: Illustrative Example of Subtree Cut and Regraft in a Genetic Algorithm

6 Conclusion

We now conclude our report with a summary of the material covered. In this report, we:

- Covered basic notions of evolutionary theory, most notably genetic divergence from a common ancestor due to mutation.
- Defined a notion of Parsimony, formalising the idea of selecting the simplest solution available from a pool of candidates.

- Gave an algorithm to calculate the parsimony of an arbitrary phylogenetic tree, and proved its correctness.
- Outlined several approaches to determine the most parsimonious tree possible comparing a given set of taxa.
- Developed the necessary theory to implement a few methods of non-exhaustively searching the space of phylogenetic trees for the tree with maximum parsimony, going into particular detail on hill climbing.

We hope that the information contained herein has been of interest, but acknowledge that phylogenetics is far too broad a field to do justice in one report. Interested readers may look further into any of the concepts explored here, but should also be aware that parsimony is far from the only phylogenetic reconstruction method available. There exist landscapes of similar depth surrounding other appraoches entirely such as distance-based, maximum likelihood and bayesian methods.

Nonetheless, thank you for taking the time to read our report.

Bibliography

[ONLINE] Noel Sturm, *DNA Mutation and Repair* , California State University, 2018, Last Visited April 2019, *http://www2.csudh.edu/nsturm/CHEMXL153/DNAMutationRepair.htm*

[NOTES] Vadim Lozin, *Lecture Notes on Graph Theory* , University of Warwick Institute of Mathematics, 2018, Last Visited April 2019

[ONLINE] David Baum, *Reading a Phylogenetic Tree: The Meaning of Monophyletic Groups* , Scitable, 2008, *https://www.nature.com/scitable/topicpage/reading-a-phylogenetic-tree-the-meaning-of-41956*

[BOOK] Elizabeth S. Allman, John A. Rhodes, *The Mathematics of Phylogenetics* , IAS/Park City Mathematics Institute, 2005

[IMAGE] *http://www.cs.ubc.ca/labs/beta/Courses/CPSC536A-01/Class10/tree4.png* , University of British Columbia

[IMAGE] https://www.mun.ca/biology/scarr/Tree_Pruning.jpg, Memorial University Newfoundland

[PAPER] Tobias Hill, *Development of New Methods for Inferring and Evaluating Phylogenetic Trees* , Uppsala University, 2007

[BOOK] T Kinene, J Wainaina, S Maina, LM Boykin, *Encyclopedia of Evolutionary Biology, Volume 3* University of Western Australia, 2016

[NOTES] Daniel Huson, *Algorithms in Bioinformatics I* , Center for Bioinformatics Tübingen, 2003

[IMAGE] http://mathworld.wolfram.com/images/eps-gif/Claw_700.gif, Wolfram Mathworld

[PAPER] Bhaskar DasGupta, Xin He, Tao Jiang, Ming Li, John Tromp, Louxin Zhang, *On Computing the Nearest Neighbour Interchange Distance* , 1999

[BOOK] Adrien Goëffon, Jean-Michel Richer, Jin-Kao Hao, *Heuristic Methods for Phylogenetic Reconstruction with Maximum Parsimony* , University of Angers, 2010