Lecture 4 : Complexity of Maximum Parsimony

MATH285K - Spring 2010 *Lecturer: Sebastien Roch*

References: [SS03, Chapter 5], [DPV06, Chapter 8]

Previous class

Recall:

THM 4.1 (Splits-Equivalence Theorem) *Let Σ be a collection of X-splits. Then, $\Sigma = \Sigma(\mathcal{T})$ for some X-tree $\mathcal{T}$ if and only if the splits in Σ are pairwise compatible. Such tree is unique up to isomorphism.*

1 Beyond Perfect Phylogenies

Viewing (full, i.e., defined on all of X) binary characters as X-splits, the Splits-Equivalence Theorem and its proof via the Tree Popping procedure provide an algorithmic solution to the problem of checking whether a collection of binary characters are compatible and, if so, of constructing a minimal X-tree on which they are convex. (For a discussion of the more general non-binary, non-full problem, see [SS03, Chapters 4, 6].)

However, typical data may not be compatible and a more flexible approach is needed.

DEF 4.2 (Parsimony Score) *Let C be a character state space with $|C| \geq 2$, let χ be a (full) character on X and let $\mathcal{T} = (T, \phi)$ be an X-tree with $T = (V, E)$. Let $\bar{\chi}$ be an extension of χ to V. The* changing number $\mathrm{ch}(\bar{\chi})$ *is*

$$\mathrm{ch}(\bar{\chi}) = |\{\{u, v\} \in E\} \,:\, \bar{\chi}(u) \neq \bar{\chi}(v)|.$$

The parsimony score $\ell(\chi, \mathcal{T})$ *of χ on $\mathcal{T}$ is the minimum value of $\mathrm{ch}(\bar{\chi})$ over all extensions of χ on $\mathcal{T}$. For a collection $\mathcal{C} = \{\chi_1, \ldots, \chi_k\}$ of characters, the parsimony score of $\mathcal{C}$ on $\mathcal{T}$ is*

$$\ell(\mathcal{C}, \mathcal{T}) = \sum_{i=1}^{k} \ell(\chi_i, \mathcal{T}).$$

A maximum parsimony tree $\mathcal{T}^*$ *for* $\mathcal{C}$ *is an* X*-tree which minimizes* $\ell(\mathcal{C}, \mathcal{T})$ *over all* X*-trees. The corresponding parsimony score is denoted by* $\ell(\mathcal{C})$*. A natural generalization of the parsimony score is obtained by considering a metric* δ *on* C *and replacing* $\mathrm{ch}(\bar{\chi})$ *with*

$$\sum_{e=\{u,v\}\in E} \delta(\bar{\chi}(u), \bar{\chi}(v)).$$

We then use the notation ℓ_δ*.*

Given a character χ on X, an X-tree $\mathcal{T}$ and a metric δ on C, one can compute the parsimony score $\ell_\delta(\chi, \mathcal{T})$ using a technique known as *dynamic programming*. Choose an arbitrary root ρ on $\mathcal{T}$. If $v = \phi(x)$ for some $x \in X$, for each $\alpha \in C$ let

$$l(v, \alpha) = \begin{cases} 0, & \text{if } \chi(x) = \alpha, \\ +\infty, & \text{otherwise.} \end{cases}$$

(By convention, we assume that the parsimony score is $+\infty$ if two different states are assigned to the same node of $\mathcal{T}$.) For all $v \notin \phi(X)$, let $v_1, \ldots, v_m$ be the children of v (i.e., the immediate descendants of v in the partial order defined under the above rooting of $\mathcal{T}$) and for each $\alpha \in C$ define

$$l(v, \alpha) = \sum_{i=1}^{m} \min_{\beta \in C} \{\delta(\alpha, \beta) + l(v_i, \beta)\}.$$

Then, it is straighforward to check by induction that

$$\ell_\delta(\chi, \mathcal{T}) = \min_{\alpha \in C} l(\rho, \alpha),$$

which can be computed recursively from the leaves up to the root. For a collection of characters $\mathcal{C}$, one can compute $\ell_\delta(\mathcal{C}, \mathcal{T})$ by computing the parsimony scores of each character separately. Computing $\ell_\delta(\mathcal{C}, \mathcal{T})$ is known as the *Fixed Tree Problem*.

As it turns out, computing $\ell_\delta(\mathcal{C})$ is much harder and, as we now explain, no efficient procedure is likely to exist for it.

2 Computational Complexity: A Brief Overview

We will use the notation $g(n) = O(f(n))$ to indicate that there is $K > 0$ such that $g(n) \leq K f(n)$ for all $n \geq 1$. The following definitions are intentionally informal. For more details, see [Pap94].

In a *search problem*, we are given an *instance* $\mathcal{I}$ and we are asked to find a *solution* $\mathcal{S}$, that is, an object that meets certain requirements (or indicate that no such solution exists). For example, in the SAT problem, we are given a formula f over a Boolean vector $x = (x_1, \ldots, x_n)$ and we are asked to find an assignment for x such that $f(x)$ is TRUE—if such an assignment exists.

An *algorithm* $\mathcal{A}$ for a search problem is said to be *efficient* if the number of elementary operations it performs on any instance $\mathcal{I}$ is bounded by a polynomial in the size of the input, that is, there is a constant $K > 0$ such that the running time of $\mathcal{A}$ on an input of size n is $O(n^K)$.

EX 4.3 (Fixed Tree Problem: Dynamic Programming) *Consider again the dynamic programming algorithm for solving the Fixed Tree Problem. For each vertex and each character state, we must perform a calculation which takes $O(m|C|)$ where m is the number of children of that particular vertex. Summing over all vertices and character states, we get a running time of $O(|V| \times |C|^2)$. The input here is a character, a tree and a metric, the size of which is $O(|X| + |V| + |C|^2)$. Hence, the dynamic programming procedure is efficient.*

EX 4.4 (Maximum Parsimony: Exhaustive Search) *Suppose we are given a collection of characters $\mathcal{C} = \{\chi_1, \ldots, \chi_k\}$ on X and we seek to compute $\ell^{(2)}_\delta(\mathcal{C})$ for a metric δ, where $\ell^{(2)}_\delta$ indicates the maximum parsimony score restricted to binary phylogenetic trees on X. The input size is $O(k|X| + |C|^2)$. If we perform an exhaustive search over all binary phylogenetic trees and use dynamic programming on each of them to compute its parsimony score, the running time is $O(b(|X|) \times k|V| \times |C|^2)$, which is not polynomial in the size of the input.*

EXER 4.5 (Tree Popping) *Show that the Tree Popping algorithm is efficient. What is its running time?*

The class of all search problems for which there exists an efficient algorithm is called **P**. Another important class of search problems is **NP**, which is defined as those problems for which a solution can be verified efficiently. For example, SAT is in **NP** as, given a solution x, it is easy to check whether $f(x)$ is TRUE. (The standard definition involves decision problems which we will not discuss here.) An important conjecture is that **P**$\neq$**NP**, that is, there exist problems in **NP** for which there is no efficient algorithm. In particular, it is possible to define a sub-class of **NP** consisting of the "hardest" problems within **NP** in the sense that the existence of an efficient algorithm for any such problem would lead to an efficient algorithm for any problem in **NP**. Such problems are called **NP**-complete and require the notion of a reduction to be defined. A *reduction* from a search problem A to a search problem B is:

> An efficient algorithm f that transforms any instance $\mathcal{I}$ of A into an instance $f(\mathcal{I})$ of B, together with another efficient algorithm h that maps any solution $\mathcal{S}$ of $f(\mathcal{I})$ back into a solution $h(\mathcal{S})$ of $\mathcal{I}$.

See [DPV06] for several examples of reductions. Then, the class of **NP**-complete problems is defined as follows:

> A search problem is **NP**-complete if all other search problems in **NP** reduce to it.

It is well-known that SAT is **NP**-complete.

3 Maximum Parsimony is NP-complete

A natural way to transform an *minimization problem* such as Maximum Parsimony into a search problem is to add to the input a threshold g and ask for a solution with objective function below g. Then, the following was shown by Graham and Foulds:

THM 4.6 (Complexity of Maximum Parsimony) *The search problem corresponding to Maximum Parsimony is* **NP***-complete.*

In other words, it is unlikely that an efficient algorithm exists for Maximum Parsimony.

Further reading

The definitions and results discussed here were taken from Chapter 5 of [SS03] and Chapter 8 of [DPV06]. The rigorous theory of computational complexity is described at length in [Pap94]. The proof that Maximum Parsimony is **NP**-complete can be found in [GF82].

References

[DPV06] S. Dasgupta, C. Papadimitriou, and U. Vazirani. *Algorithms*. McGraw-Hill, 2006.

[GF82] R. L. Graham and L. R. Foulds. Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time. *Math. Biosci.*, 60:133–142, 1982.

[Pap94] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, Reading, MA, 1994.

[SS03] Charles Semple and Mike Steel. *Phylogenetics*, volume 24 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2003.