Case study II: Parsimony

- Principle of parsimony = find the simplest explanation.
- In evolution - find the evolutionary history requiring the least amount of total change.
- It is unlikely (but not impossible) that the same feature evolved independently - common features indicate common ancestry.

Example: One "feature" could be which nucleotide (A,C,G,T) is present at a particular position in the DNA.

Question Given features present in existing population - which features were present in extinct population? Can we make guesses as to the content of the ancestral sequences?

Parsimony on a fixed tree

Formalising the definition.

INPUT

- A rooted binary tree $T = (V, E)$ with leaves labelled by species.
- An assignment of states to the leaves of T.
- A cost matrix M, where $M[i, j]$ is the cost of changing from state i to state j.

We want to assign states to *all* of the vertices of T. The cost of such an assignment is the sum of the costs of the changes across each edge in the tree. We want to minimize the cost.

Example

Looking for a recursion: first attempt

To optimize over the whole tree we optimize over the subtrees. For each vertex v let $f(v)$ be the minimum cost of assigning states to the vertices in the subtree rooted at v.

Suppose v has two children u_1 and u_2.

- Find a minimum assignment for the subtrees rooted at u_1 and u_2.
- Find a minimum assignment for subtree rooted at v by choosing the element to put at v.
- Compute $f(v)$.

Problem: doesn't always find minimum

Moral: Not every division into subproblems works.

Second attempt

For each vertex v and each state x, let $f(v, x)$ denote the minimum cost of an assignment to vertices in the subtree rooted at v, **subject** to the condition that x is assigned to v.

Recursion:

If v is a leaf then we have already assigned some state x to v. Put $f(v, x) = 0$ and $f(v, y) = \infty$ for all $y \neq x$.

Suppose that v has two children u_1 and u_2. Then for each x,

$$f(v, x) \leftarrow \min\{f(u_1, y) + M[x, y]\} + \min\{f(u_2, z) + M[x, z]\}$$

Dynamic Programming Solution

1. For all vertices v and states x, set $F[v,x] \leftarrow \infty$.
2. Let v_0 be the root of T.
3. **for** all states x **do**
4. $\quad MinCost(v,x)$
5. **return** the minimum of $F[v_0,x]$ over all x.

$MinCost(v,x)$

1. **if** $F[v,x] \neq \infty$ **then**
2. $\quad$ **return** $F[v,x]$.
3. **else**
4. $\quad$ **if** v is a leaf **then**
5. $\quad\quad$ **if** x is the state already assigned to v **then**
6. $\quad\quad\quad F[v,x] \leftarrow 0$
7. $\quad\quad$ **else**
8. $\quad\quad\quad F[v,x] \leftarrow \infty$
9. $\quad$ **else**
10. $\quad\quad$ let u_1 and u_2 be the children of v.
11. $\quad\quad F_1 \leftarrow \min\{MinCost(u_1,y) + M[x,y]\}$
12. $\quad\quad F_2 \leftarrow \min\{MinCost(u_2,z) + M[x,z]\}$
13. $\quad\quad F[v,x] \leftarrow F_1 + F_2$
14. $\quad$ **return** $F[v,x]$.

Recursive free version

The solution of the subproblem for (v, x) depends on the solution of (u_i, y) for each child u_i and each state y.

We want to traverse the vertices in *post-order* (the children processed before the parents).

The algorithm then becomes:

$MinCost2(T)$
1. **for** all vertices of v in a *post-order* traversal **do**
2. **if** v is a leaf **then**
3. **if** x is the state already assigned to v **then**
4. $F[v, x] \leftarrow 0$
5. **else**
6. $F[v, x] \leftarrow \infty$
7. **else**
8. let u_1 and u_2 be the children of v.
9. $F[v, x] \leftarrow \min\{F[u_1, y] + M[x, y]\} + \min\{F[u_2, z] + M[x, z]\}$
10. **return** minimum of $F[v_0, x]$. [v_0 = root of T]

Recovering a minimum cost assignment

As before, we use a recursive procedure. Let v_0 be the root and suppose that x_0 minimizes $F[v_0, x]$ we call $FindMinAssign(v_0, x_0)$.

$FindMinAssign(v, x)$
1. Assign x to vertex v.
2. **if** v is a not a leaf **then**
3. Let u_1 and u_2 be the children of v.
4. Find y that minimizes $F[u_1, y] + M[x, y]$
5. Find z that minimizes $F[u_2, z] + M[x, z]$
6. $FindMinAssign(u_1, y)$.
7. $FindMinAssign(u_2, z)$.

Example

Case study III - DNA sequence alignment

Input: two DNA sequences

```
AAACAGTTAACTTA
AACAGTCAGACTGA
```

An alignment involves inserting gaps to match up site for site.

```
AAACAGTTA-ACTTA
AA-CAGTCAGACTGA
```

To score an alignment we examine each site in turn.:

- If both are equal, score +1
- If sites are different score -1
- If one site is a gap, score -2.

Finding a recursion

Input sequences are $a_1, a_2, \ldots, a_m$ and $b_1, b_2, \ldots, b_n$.

For each pair (i, j) such that $1 \leq i \leq m$ and $1 \leq j \leq n$ we consider the subproblem: "what is the maximum score for an alignment of $a_1, a_2, \ldots, a_i$ and $b_1, b_2, \ldots, b_j$?"

For any alignment of $a_1, a_2, \ldots, a_i$ and $b_1, b_2, \ldots, b_j$ we can have one of three situations:

- The last column in the alignment is not a gap for either of the sequences.
- There is a gap for the first but not the second sequence.
- There is a gap for the second but not the first sequence.

still looking for a recursion

Let $f(i,j)$ denote the maximum score of an alignment of $a_1,\ldots,a_i$ and $b_1,\ldots,b_j$. For now, assume that $i > 0$ and $j > 0$.

Suppose that we have such a maximum cost alignment (i.e. it has cost $f(i,j)$).

- If the last column has no gaps then
  - if $a_i = b_j$ then $f(i,j) = f(i-1,j-1) + 1$
  - if $a_i \neq b_j$ then $f(i,j) = f(i-1,j-1) - 1$
- If the last column is a gap in the first sequence then $f(i,j) = f(i,j-1) - 2$
- If the last column is a gap in the second sequence then $f(i,j) = f(i-1,j) - 2$.

If $i = j = 0$ then $f(i,j) = 0$.

If $i > 0$ and $j = 0$ then $f(i,j) = f(i-1,0) - 2$.

If $i = 0$ and $j > 0$ then $f(i,j) = f(0.j-1) - 2$.

Dynamic programming algorithm

1. Initialise $F[i,j] \leftarrow -\infty$ for all i,j.
2. Call $Align(m,n)$.

$Align(i,j)$
1. **If** $F[i,j] \neq -\infty$ **then**
2. **return** $F[i,j]$.
3. **else**
4. **if** $i = j = 0$ **then**
5. $F[i,j] \leftarrow 0$
6. **else if** $i = 0$ and $j > 0$ **then**
7. $F[i,j] \leftarrow Align(i,j-1) - 2$
8. **else if** $i > 0$ and $j = 0$ **then**
9. $F[i,j] \leftarrow Align(i-1,j) - 2$
10. **else**
11. **if** $a_i = b_j$ **then**
12. $F[i,j] \leftarrow \max\{Align(i-1,j-1) + 1,$
$Align(i-1,j) - 2, Align(i,j-1) - 2\}$
13. **else**
14. $F[i,j] \leftarrow \max\{Align(i-1,j-1) - 1,$
$Align(i-1,j) - 2, Align(i,j-1) - 2\}$
15. **return** $F[i,j]$.

Recursion free version

To compute $f(i,j)$ we needed to have computed

- $f(i,j-1)$
- $f(i-1,j-1)$
- $f(i-1,j)$

We can process the subproblems in order of a topological sort using a simple loop:

$Align2$
1. **for** i from 0 to m **do**
2. **for** j from 0 to n **do**
3. **if** $i = j = 0$ **then**
5. $F[i,j] \leftarrow 0$
6. **else if** $i = 0$ and $j > 0$ **then**
7. $F[i,j] \leftarrow F[i,j-1] - 2$
8. **else if** $i > 0$ and $j = 0$ **then**
9. $F[i,j] \leftarrow F[i-1,j] - 2$
10. **else**
11. **if** $a_i = b_j$ **then**
12. $F[i,j] \leftarrow \max\{F[i-1,j-1] + 1,$
$F[i-1,j] - 2, F[i,j-1] - 2\}$
13. **else**
14. $F[i,j] \leftarrow \max\{F[i-1,j-1] - 1,$
$F[i-1,j] - 2, F[i,j-1] - 2\}$
15. **return** $F[m,n]$.