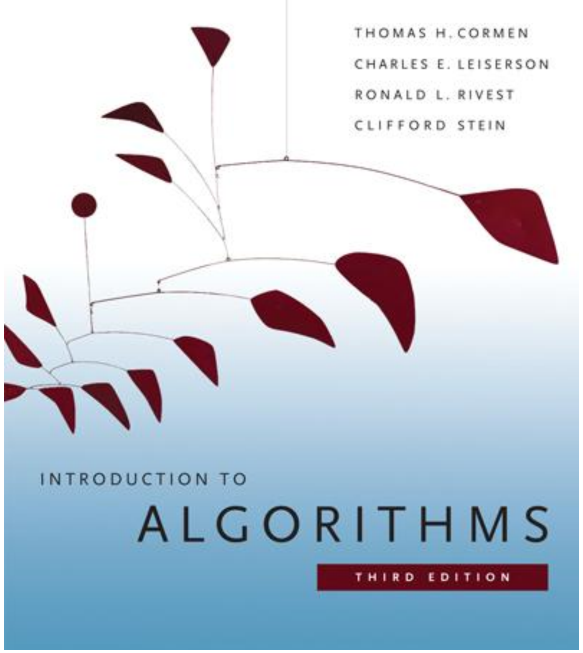
6.006- *Introduction to Algorithms*



Lecture 21

Prof. Constantinos Daskalakis

CLRS 15

Menu

- Text Justification
- Structured Dynamic Programming
  - Vertex Cover on trees
  - Parsimony: recovering the tree of life

Menu

- Text Justification
- Structured Dynamic Programming
  - Vertex Cover on trees
  - Parsimony: recovering the tree of life

Text Justification – Word Processing

- A user writes stream of text
- WP has to break it into lines that aren't too long
- obvious algorithm => greedy:
  - put as much on first line as possible
  - then continue to lay out rest
  - used by MSWord, OpenOffice
- Problem: suboptimal layouts !!

e.g.	blah blah blah		blah blah
	b l a h	vs	blah blah
	reallylongword		reallylongword

A Better Approach

- define a scoring rule
  - takes as input partition of words into lines
  - measures how good the layout is
- not an algorithm, just a metric
- find the layout with best score
  - here's where you think of algorithm

Layout Function

- Want to penalize big spaces. What objective would do that?
  - sum of leftover spaces?
  - then

blah blah blah b l a h reallylongword	as good as	blah blah blah blah reallylongword

  - i.e. it's the same for two layouts with the same number of lines (just total space minus number of characters)
- should penalize big spaces "extra"
  - (LaTeX uses sum of cubes of leftovers)

Formally

- input: array of words (lengths) w[0..n]
- split into lines L_1, L_2 …
- **badness of a line:**

  $$\text{badness}(L) = (\text{page width} - \text{total length}(L))^3$$

  – (or ∞ if total length of line > page width)
- **objective:** break into lines L_1, L_2… minimizing Σ_i badness(L_i)

Can We DP?

- Subproblems?
  - DP[i] = min badness for words w[i:n]

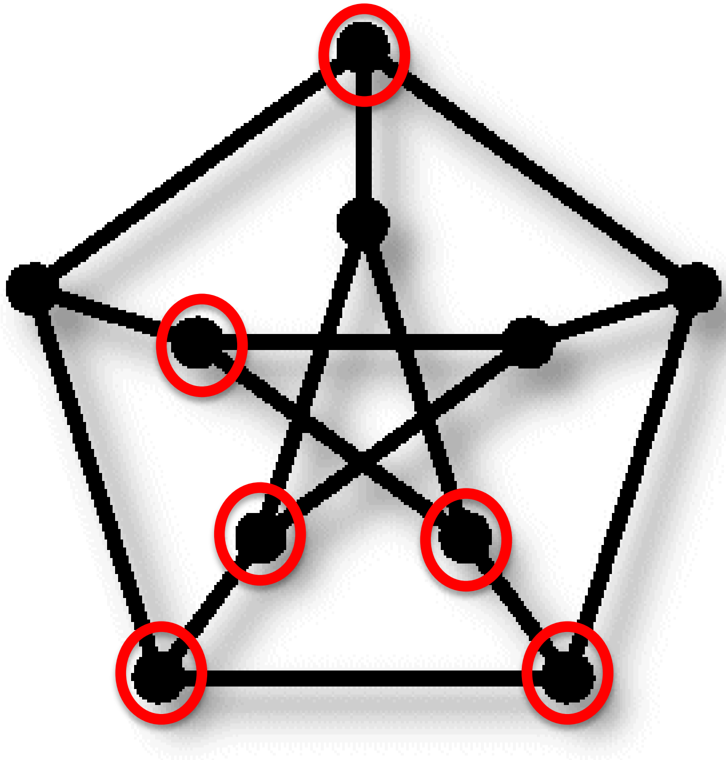
  (i.e. the score of the best layout of words w[i],…,w[n])
  - n subproblems where n is number of words
- Decision for problem i?
  - where to end first line in optimal layout for words w[i:n]
- Recurrence?
  - $DP[i] = \min_{j \text{ in range}(i+1,n)} (\text{badness}(w[i:j]) + DP[j])$
  - $DP[n+1]=0$
  - OPT = DP[0]
- Runtime? $O(n^2)$?

Menu

- Text Justification
- Structured Dynamic Programming
  - Vertex Cover on trees
  - Parsimony: recovering the tree of life

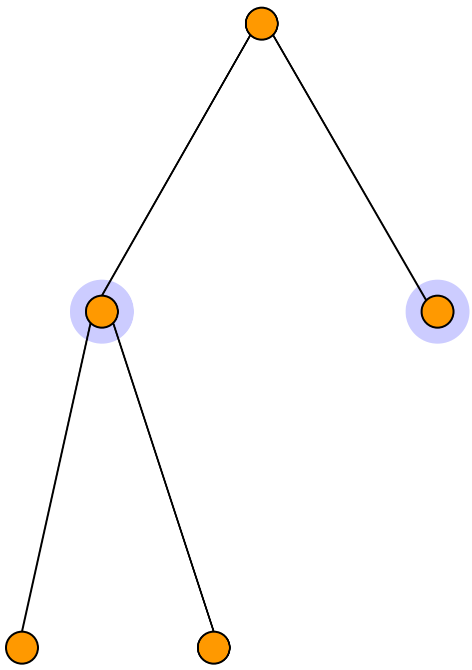
Vertex cover

- Find a minimum set of vertices that contains at least one endpoint of each edge
- (like placing guards in a house to guard all corridors)
- NP-hard in general

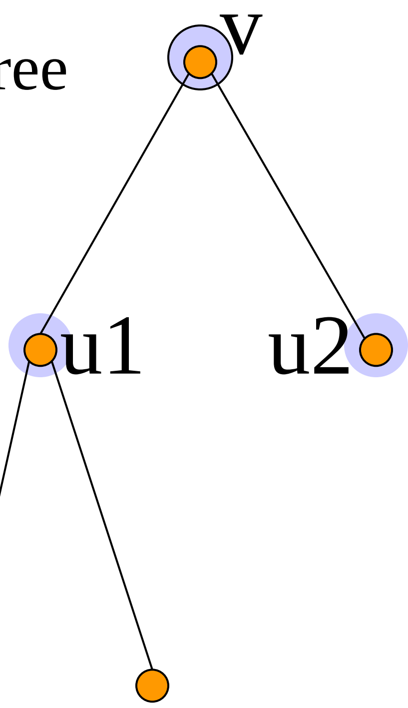


Vertex cover

- Find a minimum set of vertices that contains at least one endpoint of each edge
- (like placing guards in a house to guard all corridors)
- NP-hard in general
- We will see a polynomial (in n) time algorithm for trees of size n
- Ideas ?



Vertex cover: algorithm



- Let cost(v,b) be the min-cost solution of the sub-tree rooted at v, assuming v's status is $b \in \{YES,NO\}$
- Recurrence for cost(v,b)

cost(v,YES)= $1+\min_{b1}$ cost(u1,b1)+ $\min_{b2}$cost(u2,b2)+…

cost(v,NO)= cost(u1,YES)+cost(u2,YES)+….

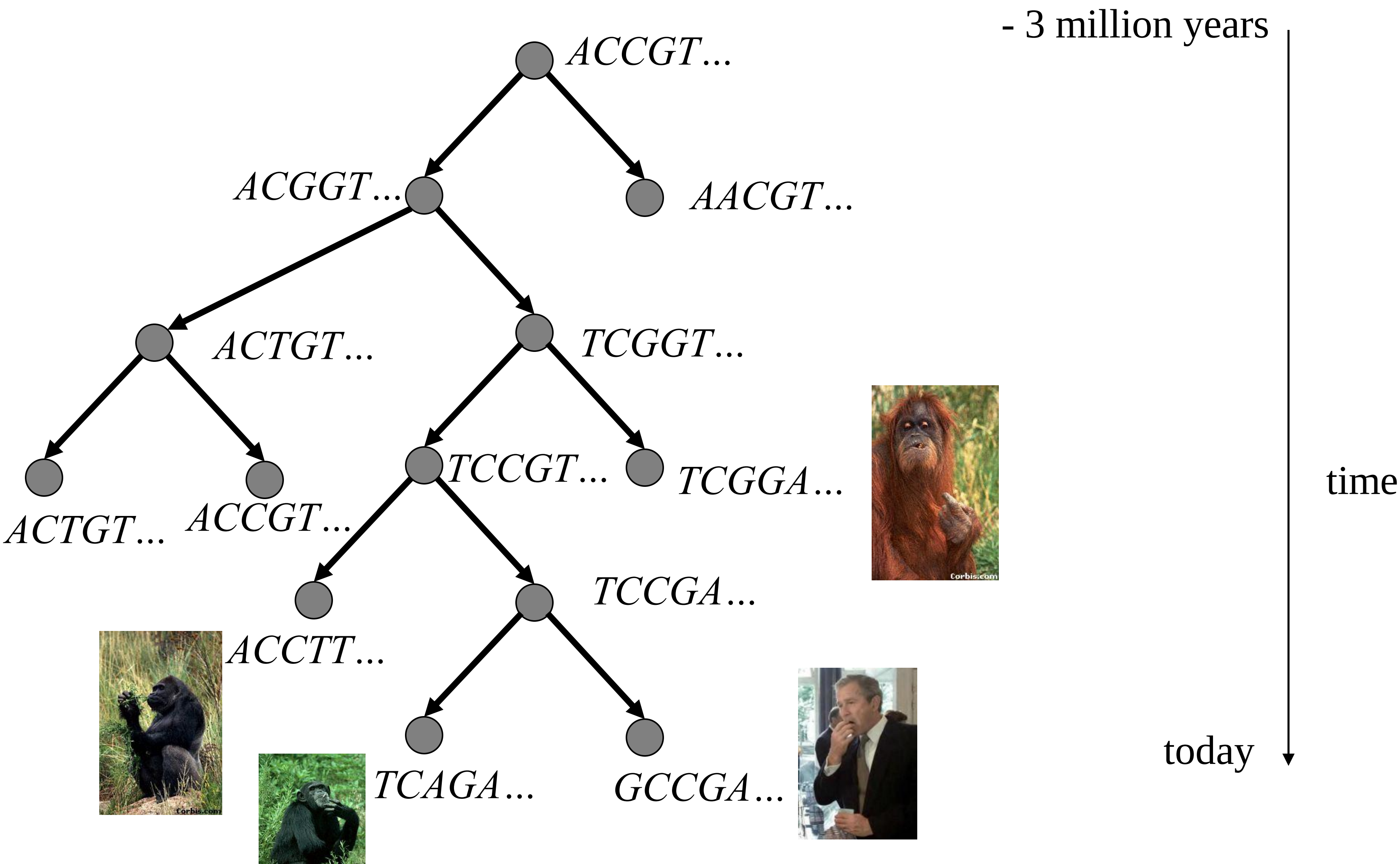
- Base case v=leaf:

cost(v,YES)=1

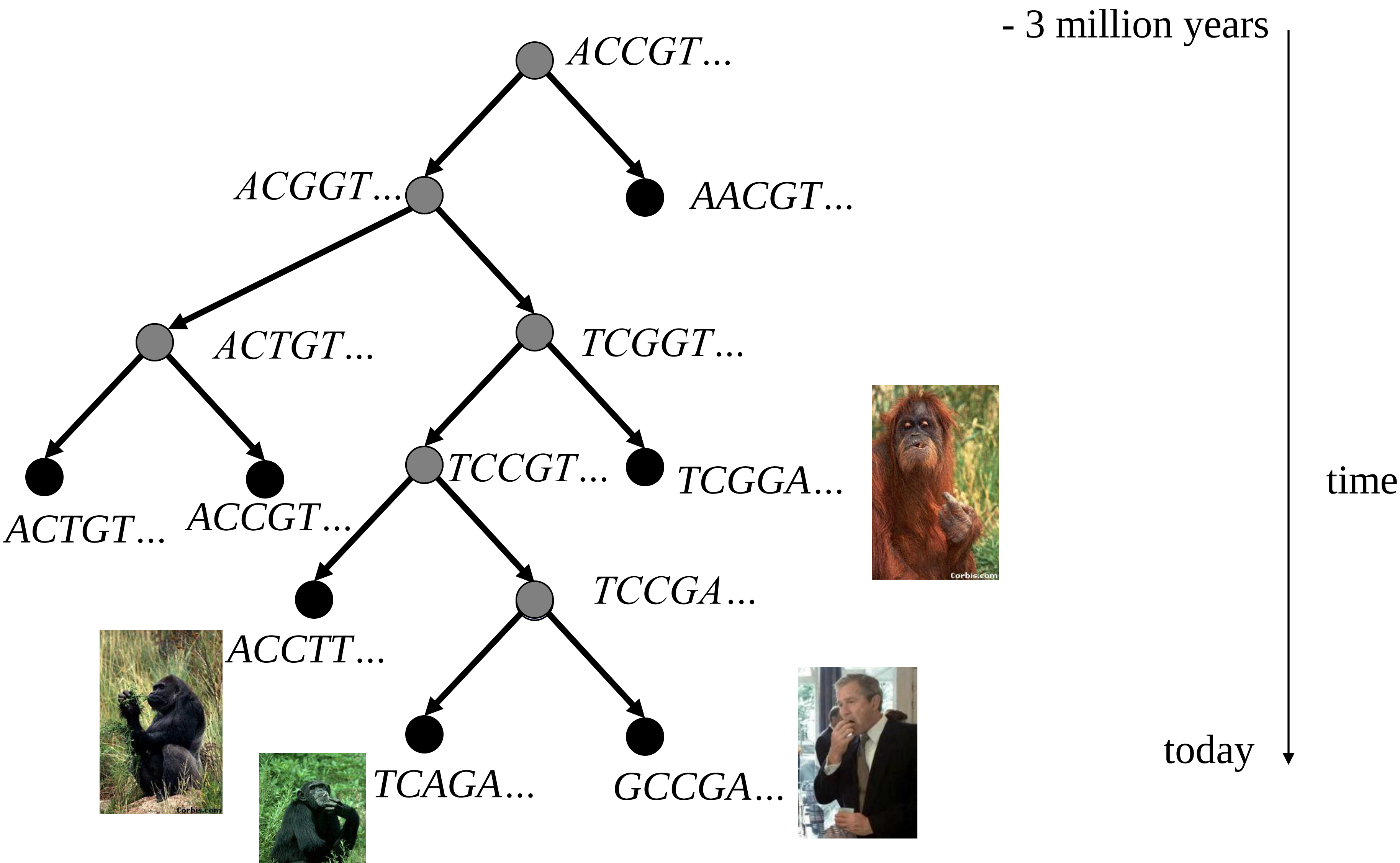
cost(v,NO)=0

- Running time ? O(n)

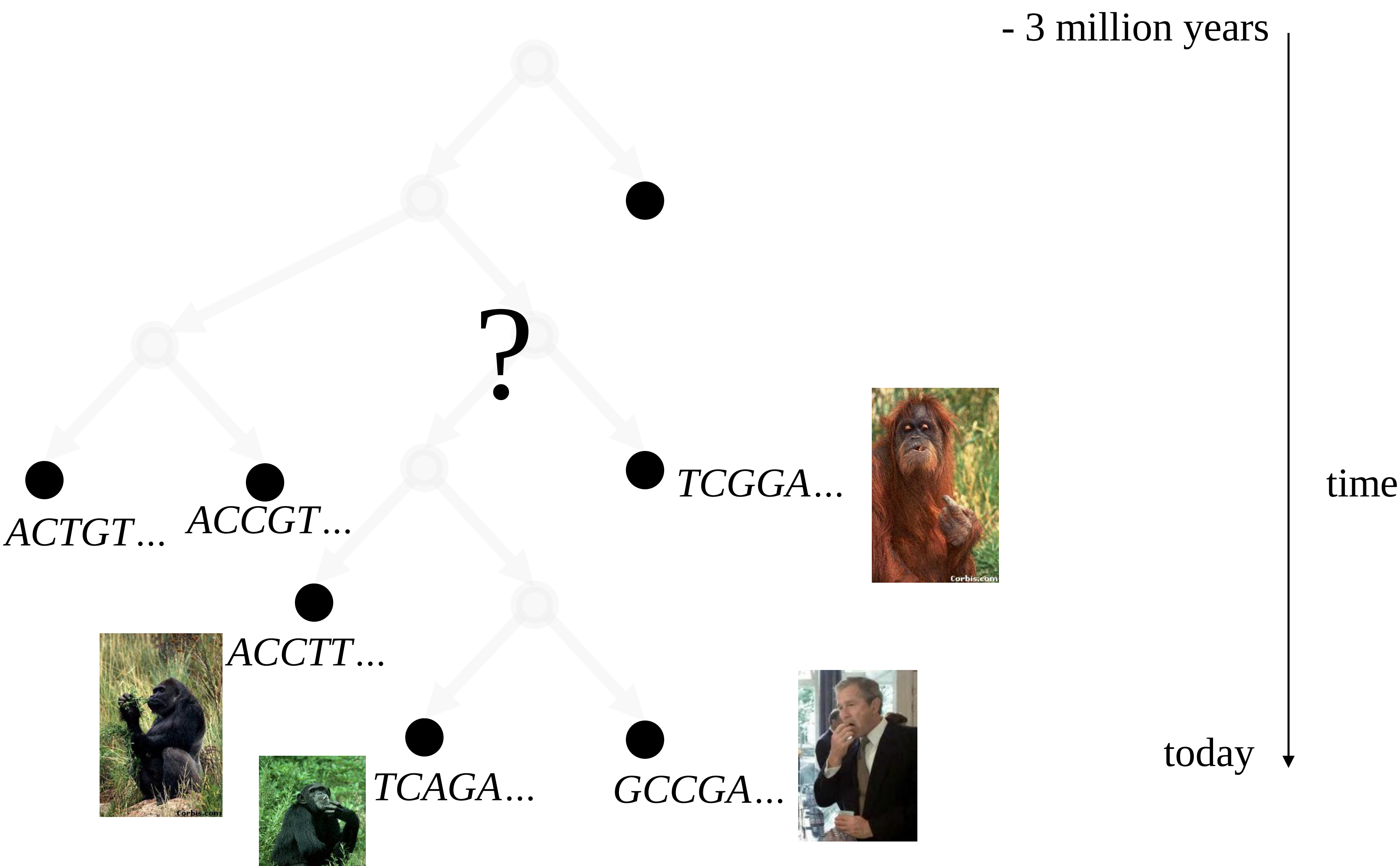
The Tree of Life



The Computational Problem



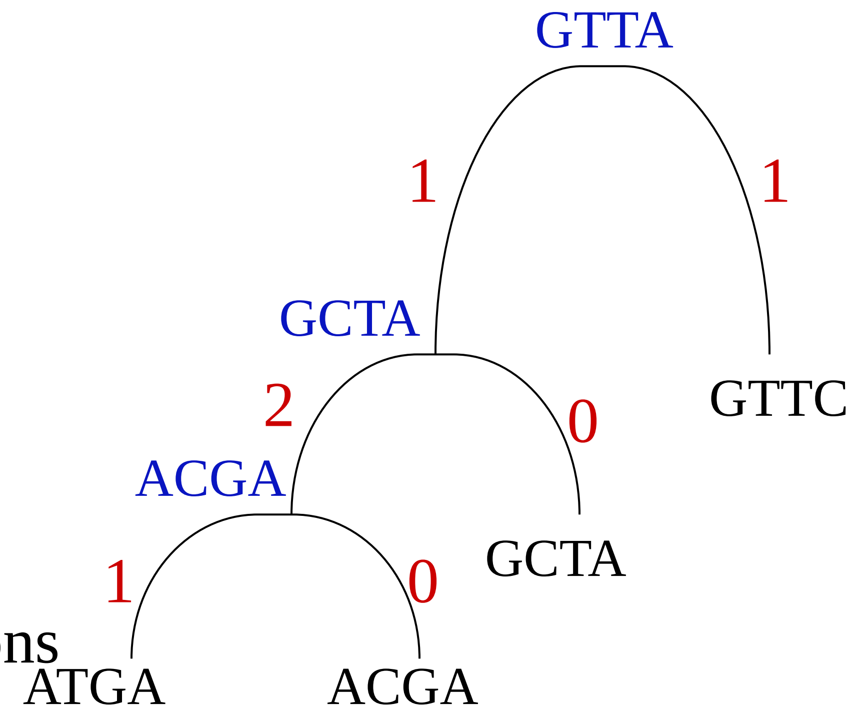
The Computational Problem



How to score a proposed tree?

- A desired property of a plausible tree:

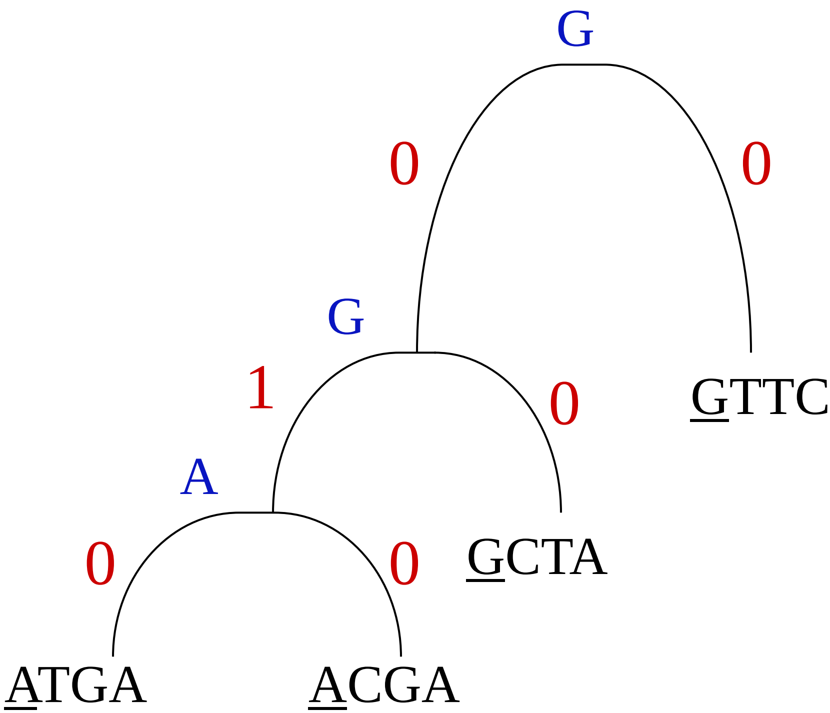
  Explains how the observed DNA sequences came about using few mutations.
- Such tree has "**high parsimony**".
- Algorithmic problem. Given:
  - n "leaf strings" of length m each, with letters from {A, C, G, T}
  - a tree
- Goal: find **"inner node" sequences** that minimize the sum of all mutations along all edges
- This is the **parsimony** of the tree.
- Algorithmic Ideas ?



parsimony = 5

Parsimony: algorithm

- Observation I: we can consider one letter at a time
- Observation II: can use dynamic programming to find the best inner-node letters

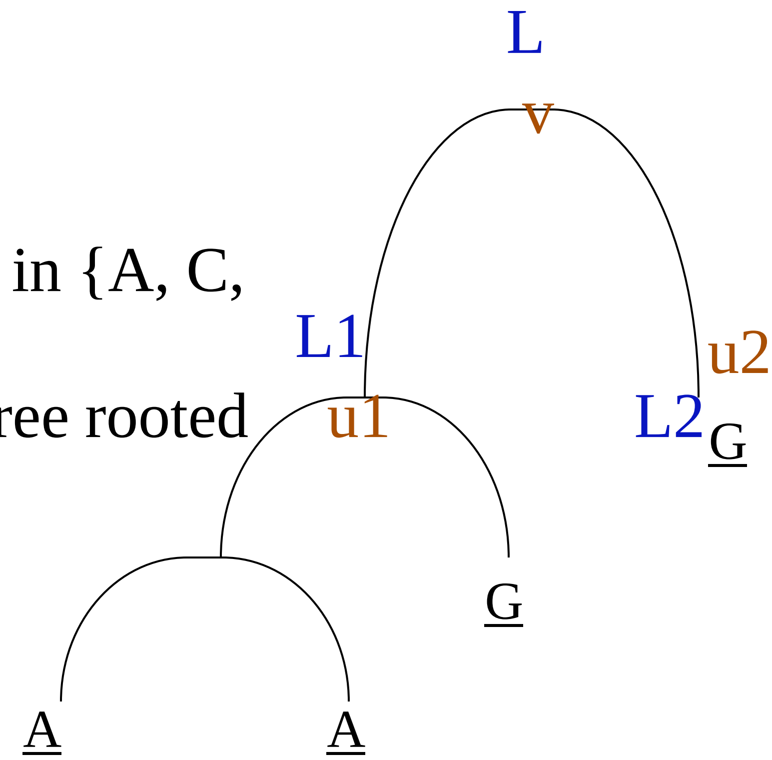


Parsimony: dynamic program

- Define letter distance as follows

  D(a,b)=0 if a=b and =1 otherwise

- For any node v of the tree and label L in {A, C, G, T}, define cost(v,L)
- This is the minimum cost for the subtree rooted at v, if v is labeled L
- solution=$\min_L$ cost(root,L)



- Recurrence for cost(v,L) ?

cost(v,L)=$\min_{L1,\ L2}$ D(L,L1)+D(L,L2)+cost(u1,L1)+ cost(u2,L2)

- Base case: if v is a leaf

  cost(v,L)= ∞*D(L,leaf_label(v))

Parsimony: analysis

- We have

$$\mathrm{cost}(v,L)=\min_{L1,\,L2} D(L,L1)+D(L,L2)+\mathrm{cost}(u1,L1)+\mathrm{cost}(u2,L2)$$

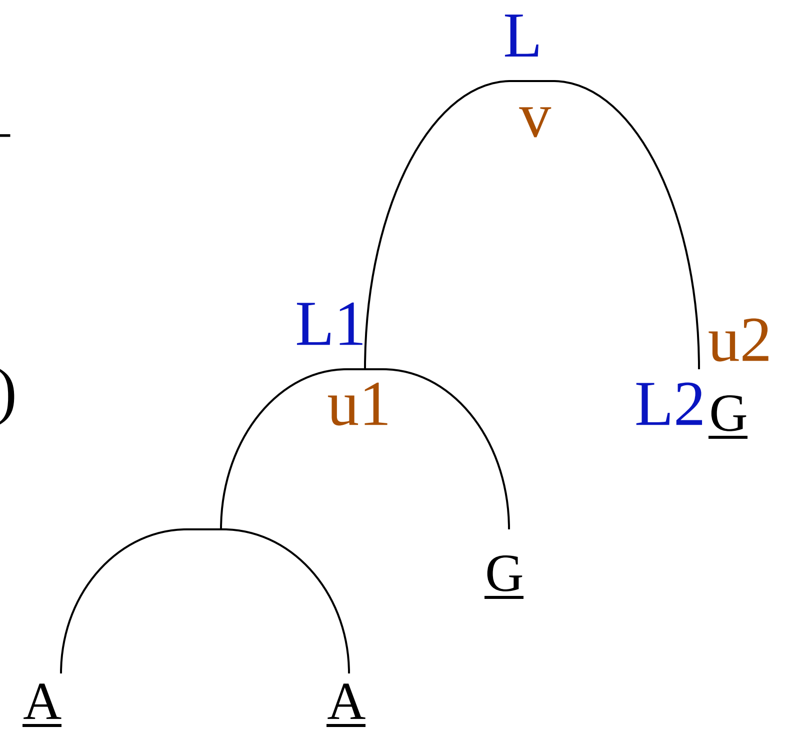
- Equivalently

$$\mathrm{cost}(v,L) = \min_{L1} D(L,L1)+\mathrm{cost}(u1,L1) \quad + \quad \min_{L2} D(L,L2)+\mathrm{cost}(u2,L2)$$

- Running time?

$$O(n\,k) * O(k) = O(nk^2)$$

where k is the alphabet size



Time for digression (extracurricular)

- Back to vertex cover
- What if the graph is not a tree ?
- Small separator good enough