Multiple Sequence Alignment

(from Chapter 6 of [DEKM])

Multiple Sequence Alignment

- Alignment of an arbitrary number of sequences, not just two sequences.
- Example with 3 DNA sequences:

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT--C
-ATTGC-G--ATTCGTAT------GGGACA-TGGATGCATGCAG-TGAC
```

- Allows simultaneous comparison of many sequences.
- Can find similarities in a family of sequences that are invisible in pairwise alignments.

Manual Alignment

Biologists create high quality multiple alignments of protein sequences by hand using expert knowledge:

- specific sorts of columns, such as highly conserved residues or buried hydrophobic residues
- influence of secondary and tertiary structure, such as alternation of hydrophobic and hydrophilic columns in exposed beta sheet
- expected patterns of insertions and deletions, that tend to alternate with blocks of conserved sequence
- phylogenetic relationships

Automatic Alignment

Two main aspects:

- How to score a multiple alignment?
- How to compute a multiple alignment that maximises the score?

Issues in designing the scoring model:

- Should incorporate as many of an expert's evaluation criteria as possible.
- Should turn biological criteria into a numerical score.
- Should help to recognise a 'good' multiple alignment.

What a Multiple Alignment Means

- Homologous residues among a set of sequences are aligned together in columns.
- 'Homologous' is meant in both the
  - structural sense: aligned residues occupy similar three-dimensional structural position
  - and evolutionary sense: aligned residues all diverge from a common ancestral residue
- Often it is impossible to identify homologous positions unambiguously, and there is no unique 'correct' multiple alignment.
- For clearly homologous (30% identical) protein sequences, usually only 50% of residues are structurally superposable. Globin family is an exception.

Idealised Scoring Approach

- Specify a complete probabilistic model of sequence evolution.
- Given the correct phylogenetic tree for the sequences, the model assigns a probability to each possible multiple alignment (taking into account the evolutionary events necessary to produce that alignment).
- High-probability alignments would be good structural and evolutionary alignments.

Problem: Model would be too complex and have too many parameters.

Practical Scoring Approach

- Let m be a multiple alignment of N sequences.
- Let m_i be the i-th column of the alignment m.
- Gap score $G(m)$: score for all gaps in alignment m.
- Define score $S(m_i)$ for the aligned residues in column i.
- Score $S(m)$ of alignment m can now be defined as

$$S(m) = G(m) + \sum_i S(m_i)$$

- For linear gap scores, the gap scores can be incorporated into the $S(m_i)$, giving:

$$S(m) = \sum_i S(m_i)$$

Minimum Entropy Scoring

- Let m_i be a column of aligned residues (no gaps).
- Let c_{ia} be the number of occurrences of residue a in column i.
- Let $p_{ia} = \frac{c_{ia}}{\sum_b c_{ib}}$, the relative frequency of a in column i.
- Entropy score of column i is:

$$S(m_i) = \sum_a c_{ia} \log_2 p_{ia}$$

  where we use the convention $0 \log 0 = 0$.
- A column of identical residues has score 0.
- A column with residues A,A,C,C,G,G,T,T has score $2 \log 1/4 + 2 \log 1/4 + 2 \log 1/4 + 2 \log 1/4 = -16$

Sum of Pairs (SP) Scoring

- SP score for a column is defined as the sum of all pairwise scores $s(a, b)$ for all pairs of residues in column i.
- Pairwise scores defined using a substitution matrix, e.g. PAM or BLOSUM matrices.
- Let m_i^k be the symbol in row k of column i.
- SP score can be written as: $S(m_i) = \sum_{k<l} s(m_i^k, m_i^l)$
- For linear gap costs, define $s(a, -) = s(-, a) = -d$ and $(-, -) = 0$.
- For general gap costs, score gaps separately (e.g. affine gap costs).

Comments on SP Scores

- SP scores are a popular standard method for scoring multiple alignments.
- Use of SP scores based on substitution matrix with log-odds scores cannot be easily justified using a probabilistic model.

$$\log \frac{p_{abc}}{q_a q_b q_c} \quad \text{vs.} \quad \log \frac{p_{ab}}{q_a q_b} + \log \frac{p_{bc}}{q_b q_c} + \log \frac{p_{ac}}{q_a q_c}$$

- Column with N identical residues L gets SP score $5\frac{N(N-1)}{2}$. Column with $N-1$ residues L and one G gets SP score $9(N-1)$ smaller. The relative decrease is $\frac{9(N-1)}{5N(N-1)/2} = \frac{18}{5N}$, decreasing with N (counter-intuitive!?).

Multiple Alignment Algorithms

Dynamic Programming Approach

- Extension of the pairwise alignment algorithm.
- For aligning N sequences, use N-dimensional dynamic programming matrix F.
- $F(i_1, i_2, i_3, \ldots, i_N)$ is the score of the best alignment of prefixes of length i_1, i_2, $\ldots$, i_N of the N sequences.
- Running-time for N sequences of length ℓ becomes $O((2\ell)^N)$.
- Practical only for very small values of N.

Example with 3 Sequences

- Given $N = 3$ sequences: X, Y and Z
- Computation of $F(i, j, k)$:

$$F(i,j,k) = \max \begin{cases} F(i-1, j-1, k-1) + s(X_{i-1}, Y_{j-1}, Z_{k-1}) \\ F(i, j-1, k-1) + s(-, Y_{j-1}, Z_{k-1}) \\ F(i-1, j, k-1) + s(X_{i-1}, -, Z_{k-1}) \\ F(i-1, j-1, k) + s(X_{i-1}, Y_{j-1}, -) \\ F(i, j, k-1) + s(-, -, Z_{k-1}) \\ F(i-1, j, k) + s(X_{i-1}, -, -) \\ F(i, j-1, k) + s(-, Y_{j-1}, -) \end{cases}$$

- In general: maximum of $2^N - 1$ expressions.

Progressive Alignment Methods

- Progressive alignment methods work by construction of successive pairwise alignments.
- Begin with pairwise alignment of two sequences, then extend alignment by adding more sequences.
- These are **heuristic** methods: They do not guarantee finding optimal alignments.
- Fast and efficient, often give reasonable alignments.
- Three main aspects:
  - Order in which alignments are done.
  - Single or multiple growing alignments.
  - How to align sequences/alignments against each other.

General Progressive Alignment Approach

- Begin by aligning the most similar pairs of sequences first. These are (hopefully) the most reliable alignments.
- Form larger alignments by aligning sequences or alignments to existing alignments.
- Process of forming larger and larger alignments can be represented as a tree structure ('guide tree'). Leaves represent sequences, the root represents the final multiple alignment.
- The guide tree is not necessarily a good phylogenetic tree, it only helps finding a good multiple alignment.

Feng–Doolittle Algorithm

- Progressive multiple alignment algorithm by Feng & Doolittle 1987
  - (i) Calculate pairwise alignments between all $\binom{N}{2}$ pairs of sequences, and convert the alignment scores into 'distances'.
  - (ii) Construct a guide tree from the distance matrix using a clustering algorithm (Fitch & Margoliash, 1967).
  - (iii) For each node of the tree (in the order in which nodes were added to the tree), align the two child nodes (which can be sequences or alignments) to form a new alignment.

Converting Scores to Distances

- Define distance D between sequences X and Y by

$$D = -\log S_{\text{eff}} = -\log \frac{S_{\text{obs}} - S_{\text{rand}}}{S_{\text{max}} - S_{\text{rand}}}$$

  where
  - S_{obs}: score of alignment of X and Y
  - S_{rand}: score of aligning two random strings consisting of the same residues as X and Y
  - S_{max}: average of the score of aligning X to itself, and of aligning Y to itself
- D is 0 if X and Y are identical, and tends to ∞ as the effective score S_{eff} tends to zero.

Aligning Alignments

- To align a sequence X to an alignment m:
  - Align X to all sequences in m.
  - Use the best of these alignments to determine the alignment between X and m.
- To align an alignment m to another alignment m':
  - Consider pairwise alignments of all sequences in m to all sequences in m'.
  - Use the best of these alignments to determine the alignment between m and m'.
- When an alignment is completed, replace all gap symbols in it by a neutral X character. (Aligning X to any other character or gap symbol has cost 0.) This ensures 'once a gap, always a gap'.

Disadvantage of Feng–Doolittle

- All alignments are determined by pairwise alignments between sequences.
- This ignores important aspects:
  - When a sequence is aligned to a group of aligned sequences, the degree of sequence conservation at each point should be taken into account.
  - Gap penalties might be reduced in positions where lots of gaps occur in the group's alignment.
- Better approach: profile alignment. Can align a sequence/alignment to another alignment directly. (Groups of aligned sequences are also called 'profiles'.)

Profile Alignment

- Consider profile alignment for SP scoring model and linear gap costs.
- Align two alignments m and m'.
- Score for column i in resulting alignment will be sum of:
  - scores $s(a, b)$ for symbol pairs from m in column i
  - scores $s(a, b)$ for symbol pairs from m' in column i
  - scores $s(a, b)$ for pairs of one symbol from m and one symbol from m' in column i
- The total contribution of the first two cases is independent of the alignment between m and m'.
- Thus, it suffices to consider only the score contributed by the third case.

Scoring Columns in Profile Alignment

- This means that the score for aligning column i of m and column j of m' can be calculated as:

$$s(m_i, m'_j) = \sum_k \sum_l s(m_i^k, m_j'^l)$$

where k ranges over the rows of m and l over the rows of m'.

- Example: If we align column $\begin{matrix}\mathrm{A}\\\mathrm{A}\\\mathrm{N}\end{matrix}$ of m with column $\begin{matrix}\mathrm{R}\\\mathrm{A}\end{matrix}$ of m', the score for the resulting column is:

$$s(\mathrm{A},\mathrm{R}) + s(\mathrm{A},\mathrm{R}) + s(\mathrm{N},\mathrm{R}) + s(\mathrm{A},\mathrm{A}) + s(\mathrm{A},\mathrm{A}) + s(\mathrm{N},\mathrm{A})$$

Profile Alignment Algorithm

- We know the score for aligning column i of m to column j of m'.
- Using $s(a, -) = s(-, a) = -d$ and $s(-, -) = 0$, we also have scores for aligning a column of one alignment to a gap in the other alignment.
- Therefore, we can use standard dynamic programming for pairwise alignment (Needleman-Wunsch) also to
  - compute optimal alignments between two profiles (alignments), or
  - compute optimal alignments between a profile (alignment) and a sequence.

CLUSTALW Progressive Alignment

- Progressive multiple alignment algorithm by Thompson, Higgins & Gibson 1994:
  - (i) Calculate pairwise alignments between all $\binom{N}{2}$ pairs of sequences, and convert the alignment scores into evolutionary 'distances' (using a model of Kimura, 1983).
  - (ii) Construct a guide tree by a neighbour-joining clustering algorithm (Saitou & Neil, 1987).
  - (iii) Progressively align at tree nodes in order of decreasing similarity, using sequence–sequence, sequence–profile and profile–profile alignment.

Additional CLUSTALW Heuristics

- Sequences are weighted to compensate for biased representation in large subfamilies, thus compensating defects of SP scores.
- Different substitution matrices are used depending on similarity of expected alignment (e.g. BLOSUM50 for distant sequences, BLOSUM80 for closely related ones).
- Position-specific gap-open penalties (based on observed gap frequencies in known structurally based alignments).
- Increased gap-open and gap-extension penalties in positions without gaps but where gaps occur nearby.
- Guide tree is adjusted on the fly if low-scoring alignments are encountered.

Iterative Refinement Methods

- Another approach to multiple alignment is based iterative refinement.
- Start with a multiple alignment computed with some other method.
- Take out one or several sequences, and realign these to the profile of the remaining sequences.
- If the score is improved, replace the old alignment by the new one.
- Repeat this process until the alignment can no longer be improved.

Barton-Sternberg Algorithm

- Multiple alignment algorithm by Barton & Sternberg 1987:
  - (i) Find the two sequences with highest pairwise similarity and align them using standard pairwise dynamic programming alignment.
  - (ii) Find the sequence that is most similar to the profile of the alignment of the first two sequences, and align it to the first two using profile–sequence alignment. Repeat until all sequences are included in the alignment.
  - (iii) For each sequence, remove it from the alignment and re-align it using profile–sequence alignment.
  - (iv) Repeat step (iii) for a fixed number of iterations or until the alignment score converges.

Alignment using HMMs (sketched)

- Multiple alignments can also be computed using methods based on Hidden Markov Models.
- 'Profile HMM': HMM that produces sequences belonging to a family with larger probability than other sequences.
- Sequence–profile alignment determined by considering the profile HMM and calculating the most probable state path for generating the sequence (using Viterbi algorithm).
- Use training methods to obtain profile HMM for the given group of sequences, then compute multiple alignment by aligning each sequence to that profile HMM (see [DEKM] for details).

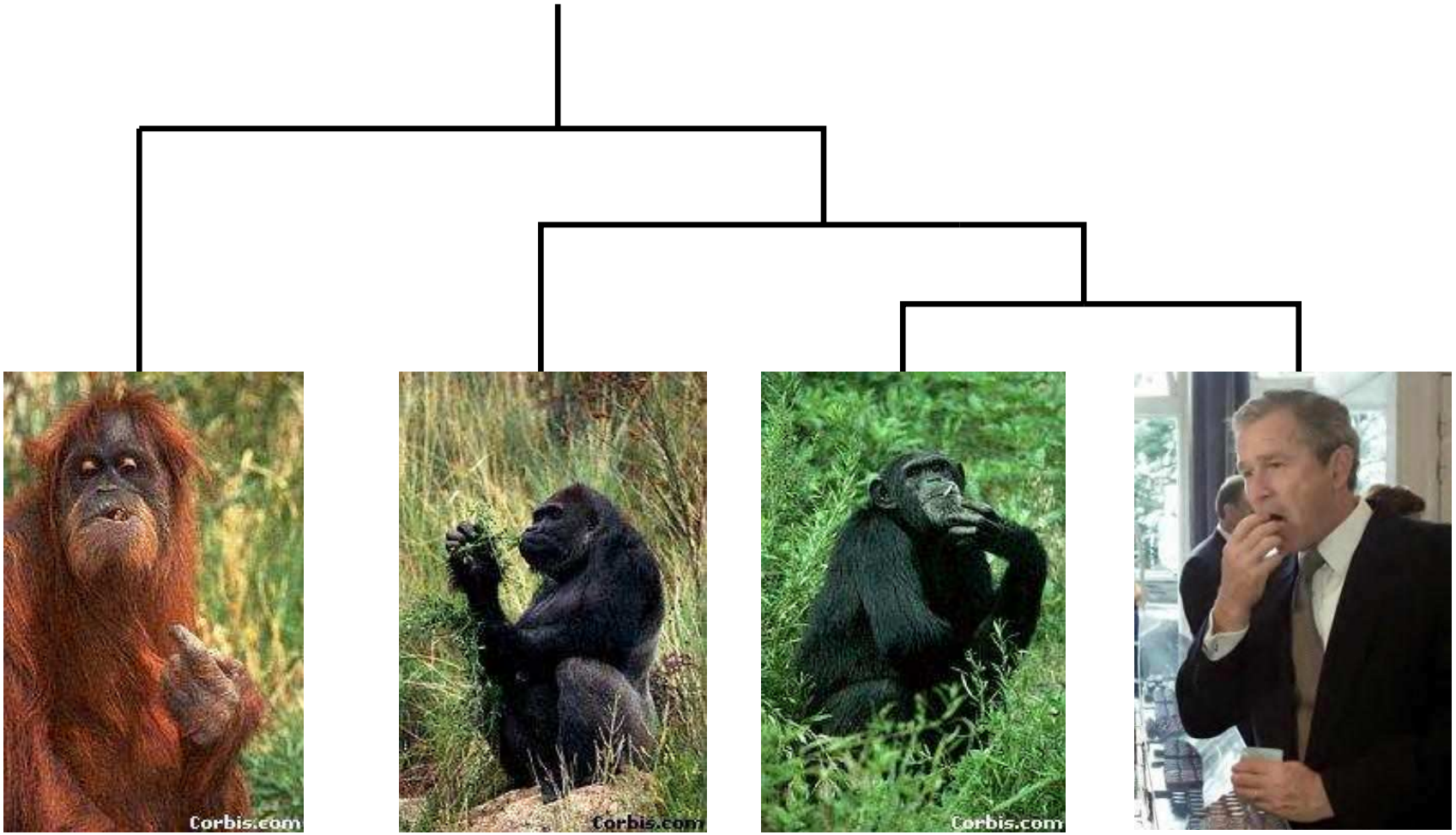
Phylogenetic Trees

(from Chapters 7–8 of [DEKM])

Reconstructing the Tree of Life

(from: Tree of Life web project)

Primates



Orangutan **Gorilla** **Chimpanzee** **Human**

Phylogenetic Inference Challenges

- Huge data sets
- Millions of species
- Difficult to choose the right modelling assumptions (phylogenetic trees should be good according to the model if they are as close to reality as possible)
- Many models lead to NP-hard optimisation problems

Phylogenies

- Phylogenetic trees are (binary) branching diagrams that represent the history of life, i.e., the ancestral relationships among a set of species.
- Internal nodes refer to (hypothetical) ancestors.
- When a hypothetical common ancestor diverges, the two different resulting species are assumed to evolve independently.
- Edge lengths may be used to represent evolutionary distance.
- Trees can be rooted or unrooted.
- **Goal:** infer phylogenies from available data about a set of species

Data

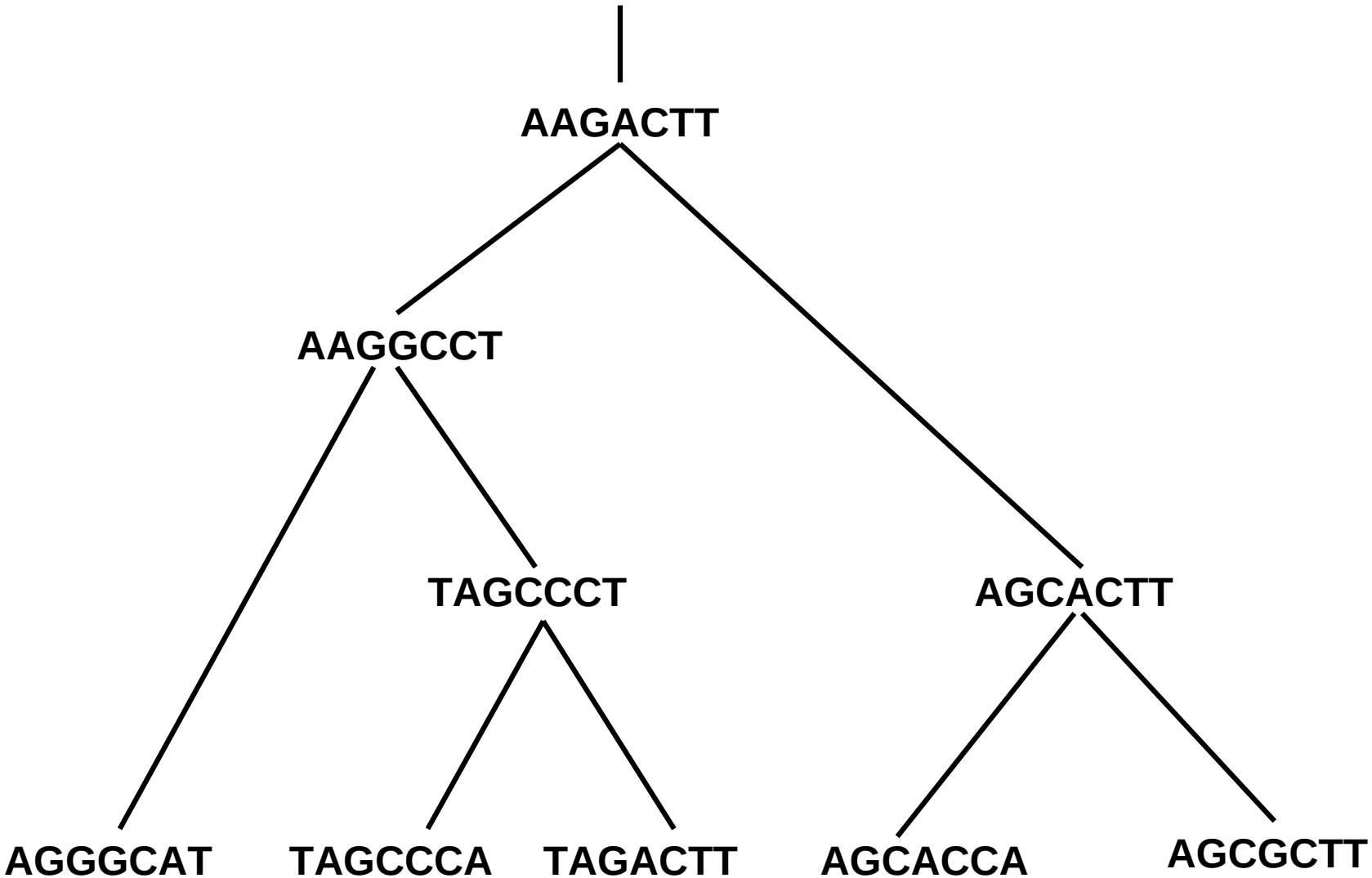
- Morphology data (form or shape of an organism or part thereof)
- Gene order data
- Sequence data (DNA, RNA, amino acids) in a multiple alignment

Abstraction of sequence or morphology data: Each species is identified by a map from characters (sites) to state sets.

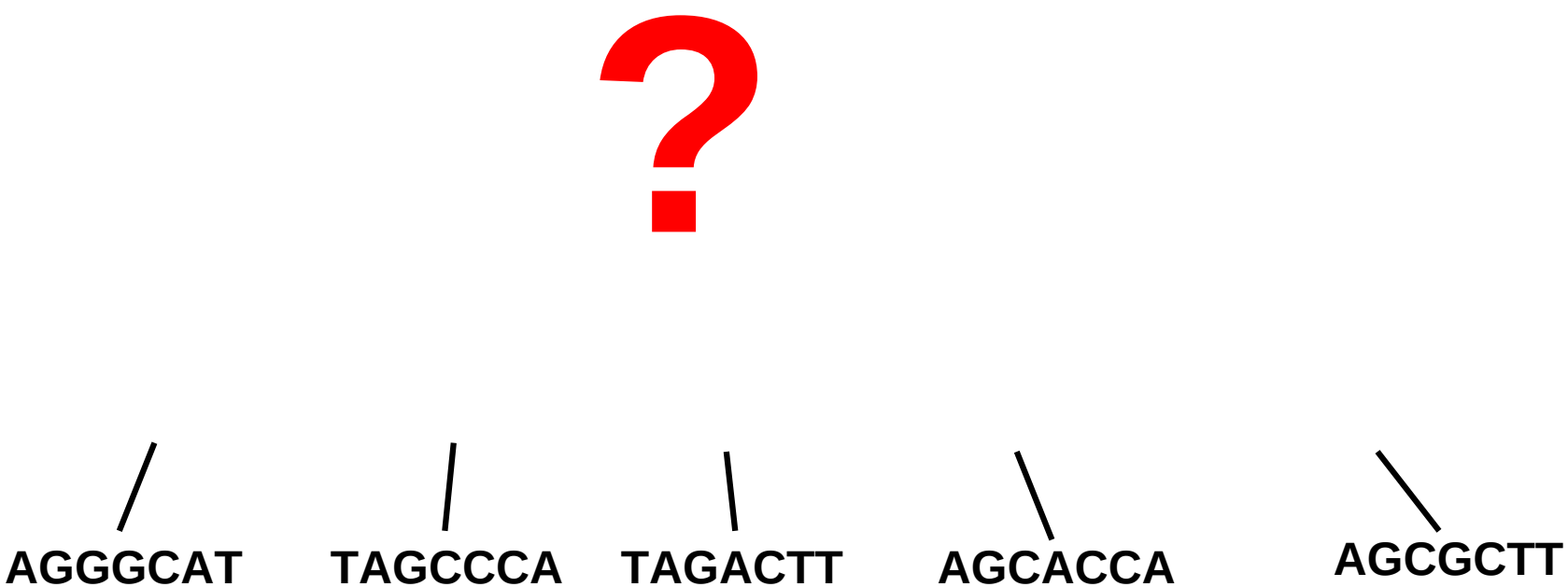
- Example: DNA sequences of length ℓ correspond to ℓ characters and a common state set $\{A, C, G, T\}$ of cardinality 4.

Evolution is a process that changes character states.

Phylogeny on Sequences



Phylogeny on Sequences



Phylogenetic Analysis

- **Step 1:** Collect sequence data for a set of species (and compute a multiple alignment if appropriate).
- **Step 2:** Use phylogenetic inference method(s) to reconstruct tree(s) from the data.
- **Step 3:** Use consensus methods to derive a reliable tree (possibly on a subset of species) from the set of trees computed in Step 2.

Inference Methods

- **Maximum parsimony (MP)**: find a tree that needs the fewest evolutionary changes
- **Maximum likelihood (ML)**: fix a stochastic model of evolution and identify the tree that makes the observed data the most likely (i.e., maximises the conditional probability $\Pr(\text{data} \mid \text{tree})$).
- **Distance-based methods:** estimate evolutionary distances between species and then construct a matching tree with edge lengths

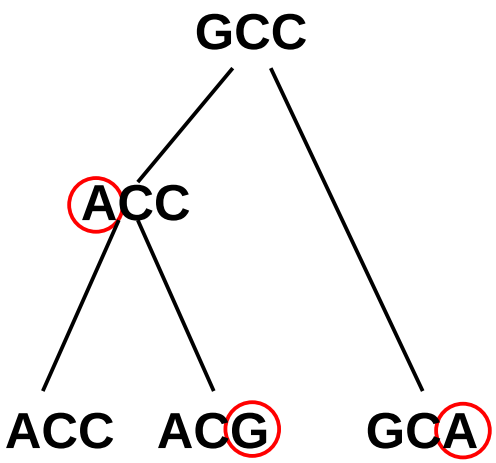
MP and ML are the most popular methods, but they are computationally complex and it is a challenge to get results for large data sets.

Maximum Parsimony

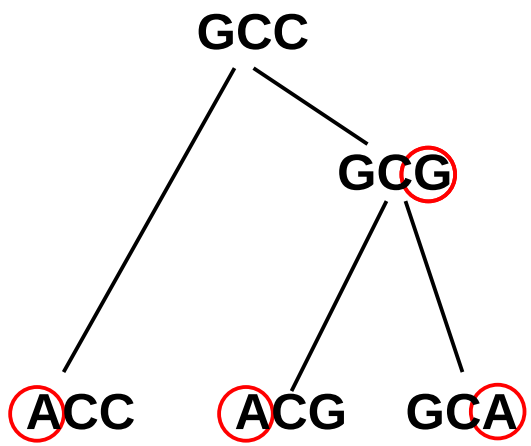
- Idea: A phylogeny is "good" if it explains evolution with the fewest evolutionary changes.

sequences:

ACC
ACG
GCA



3 changes



4 changes

Note: Parsimony score of a tree is independent of the root.

Parsimony Problems

The parsimony score of a tree $T = (V, E)$ with sequence s_v at node v is:

$$p(T) = \sum_{\{u,v\} \in E} H(s_u, s_v),$$

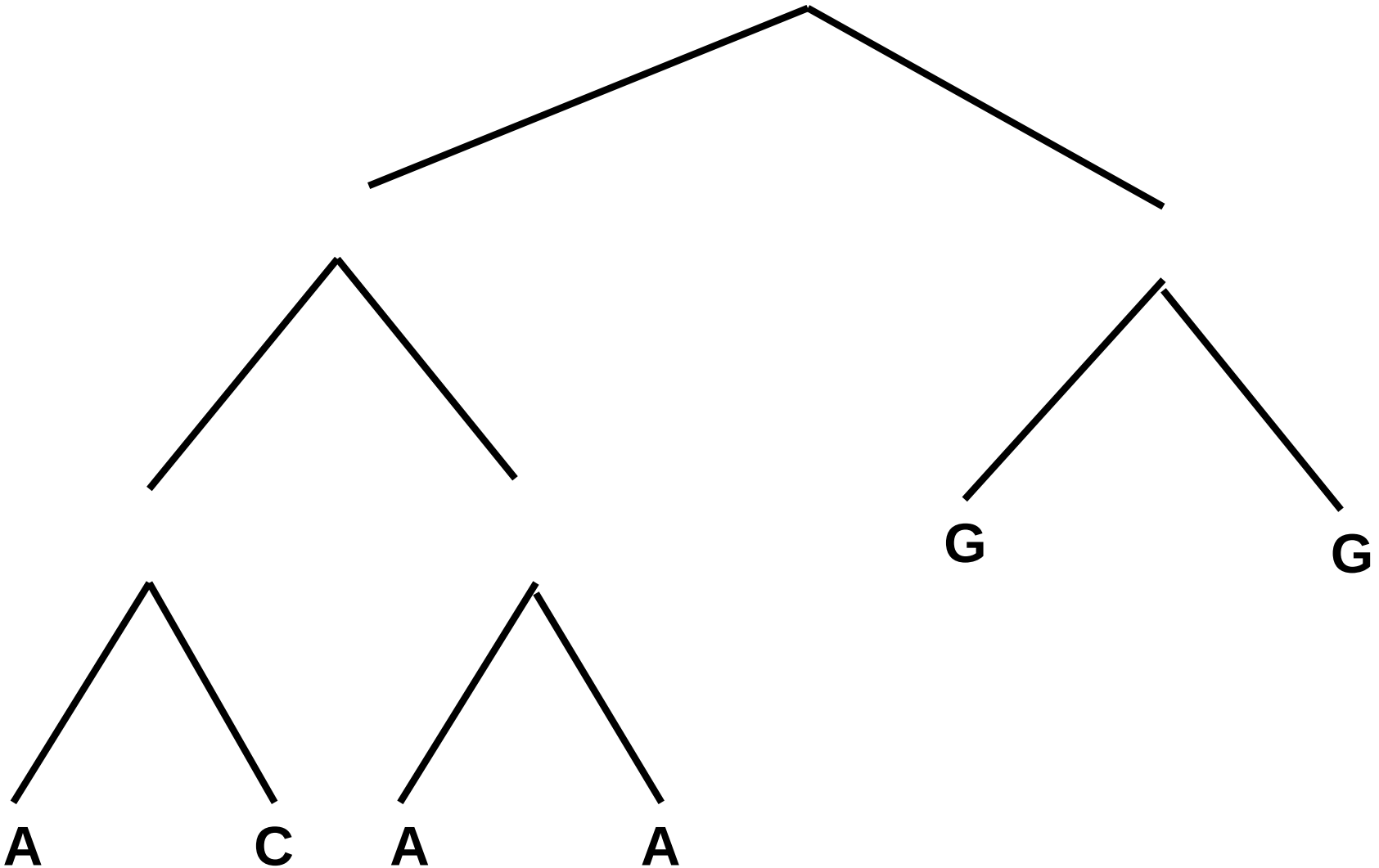
where $H(x, y)$ is the Hamming-distance between sequences x and y.

- **Small Parsimony Problem**: Given a tree T with sequences at the leaves, label the internal nodes with sequences so that $p(T)$ is minimised.
- **Large Parsimony Problem**: Given sequence data for a set of species, find a tree T (among all possible trees with the species as leaves) that leads to a minimum score $p(T)$ after labelling the internal nodes optimally.

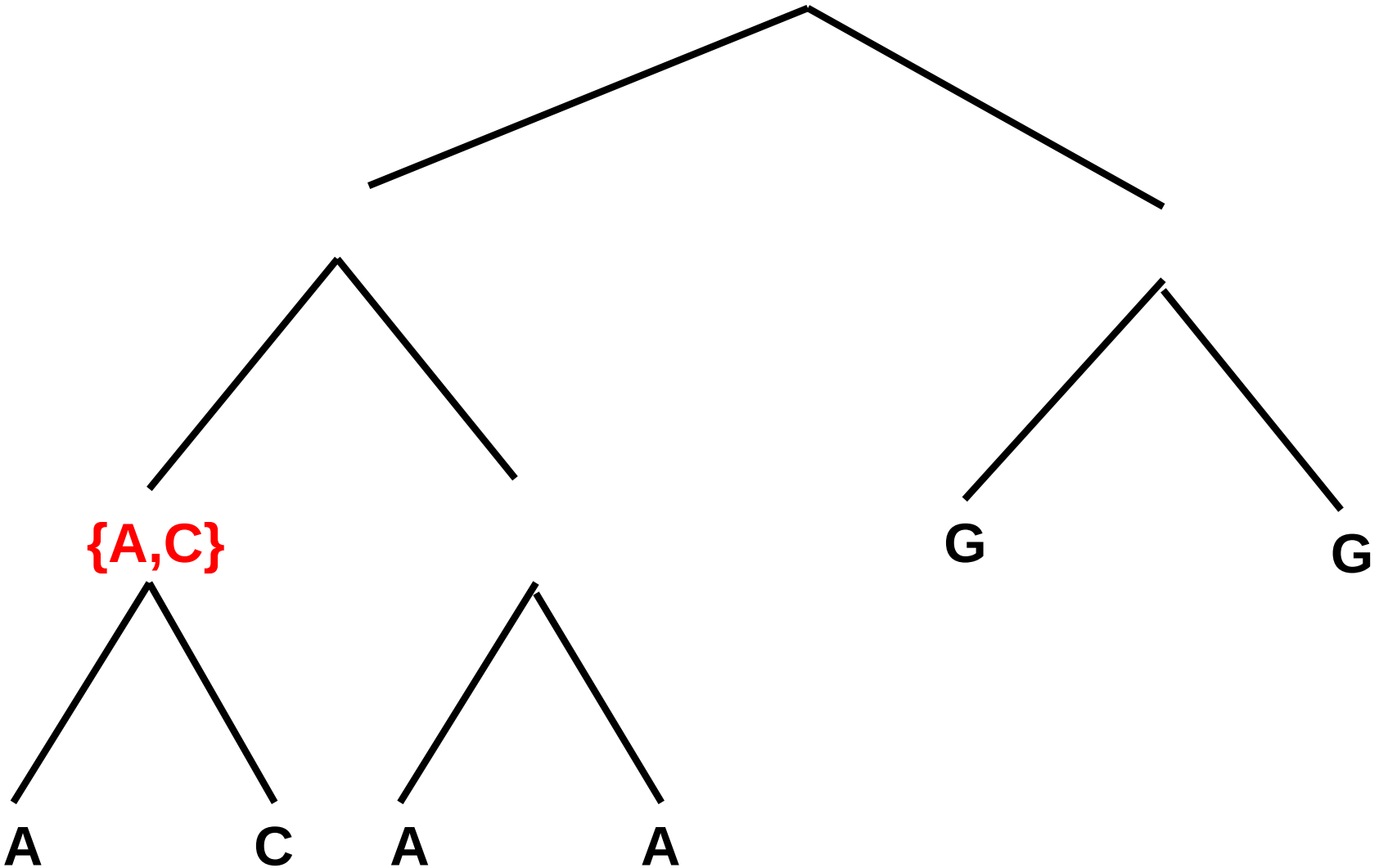
Small Parsimony Problem

- Solved for each site separately
- Optimal algorithm (Fitch, 1971):
  - traverse tree in post-order
  - label each leaf ℓ with the set S_ℓ containing only the state at that site
  - for internal node m, compute
    $$S_m = \begin{cases} S_l \cup S_r & \text{if } S_l \cap S_r \text{ is empty} \\ S_l \cap S_r & \text{otherwise} \end{cases}$$
    where l and r are the children of m.
  - in the end, process the nodes top-down and label each v with the same state as its parent (if contained in S_v), else with an arbitrary state in S_v

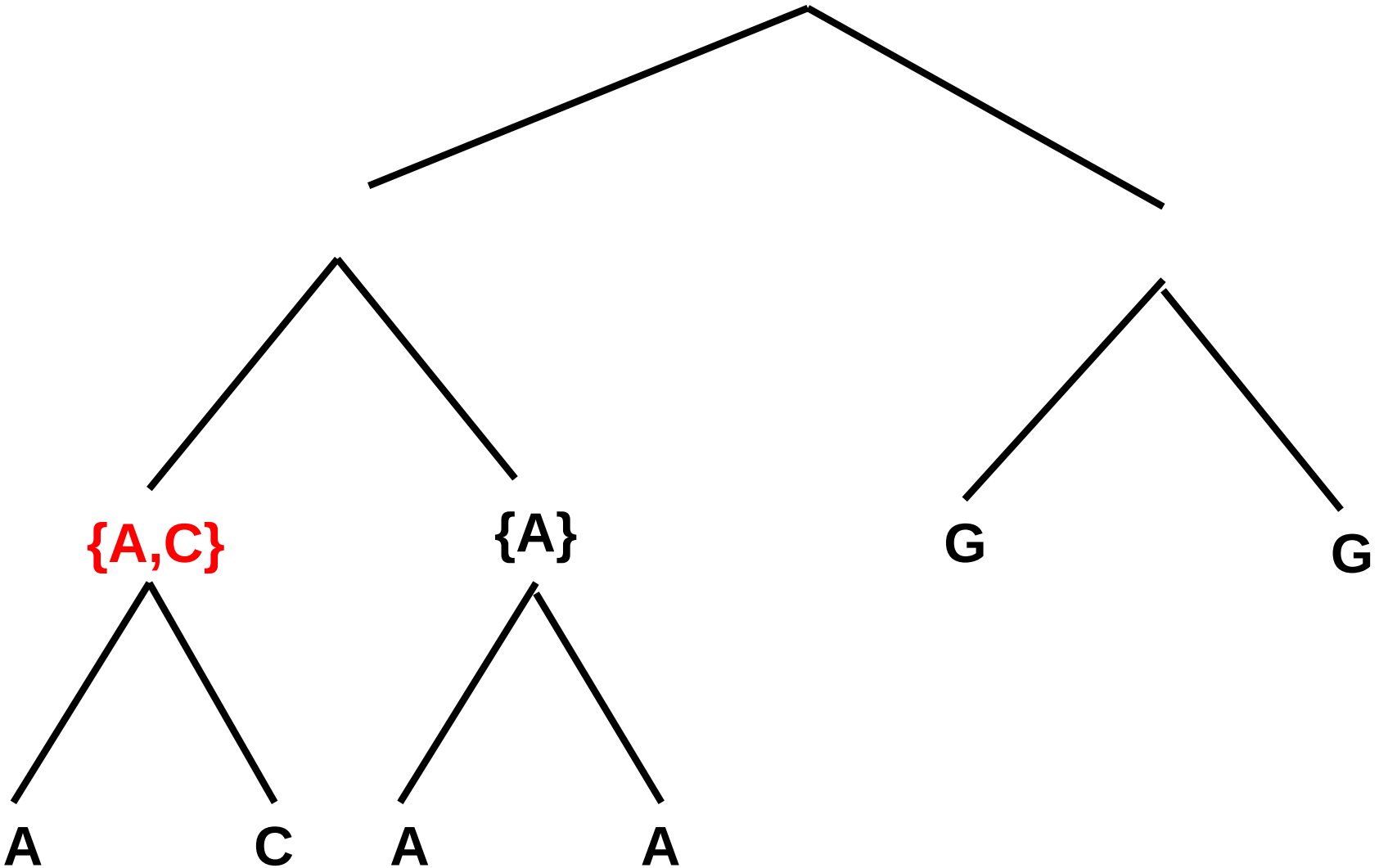
Example of Fitch's algorithm



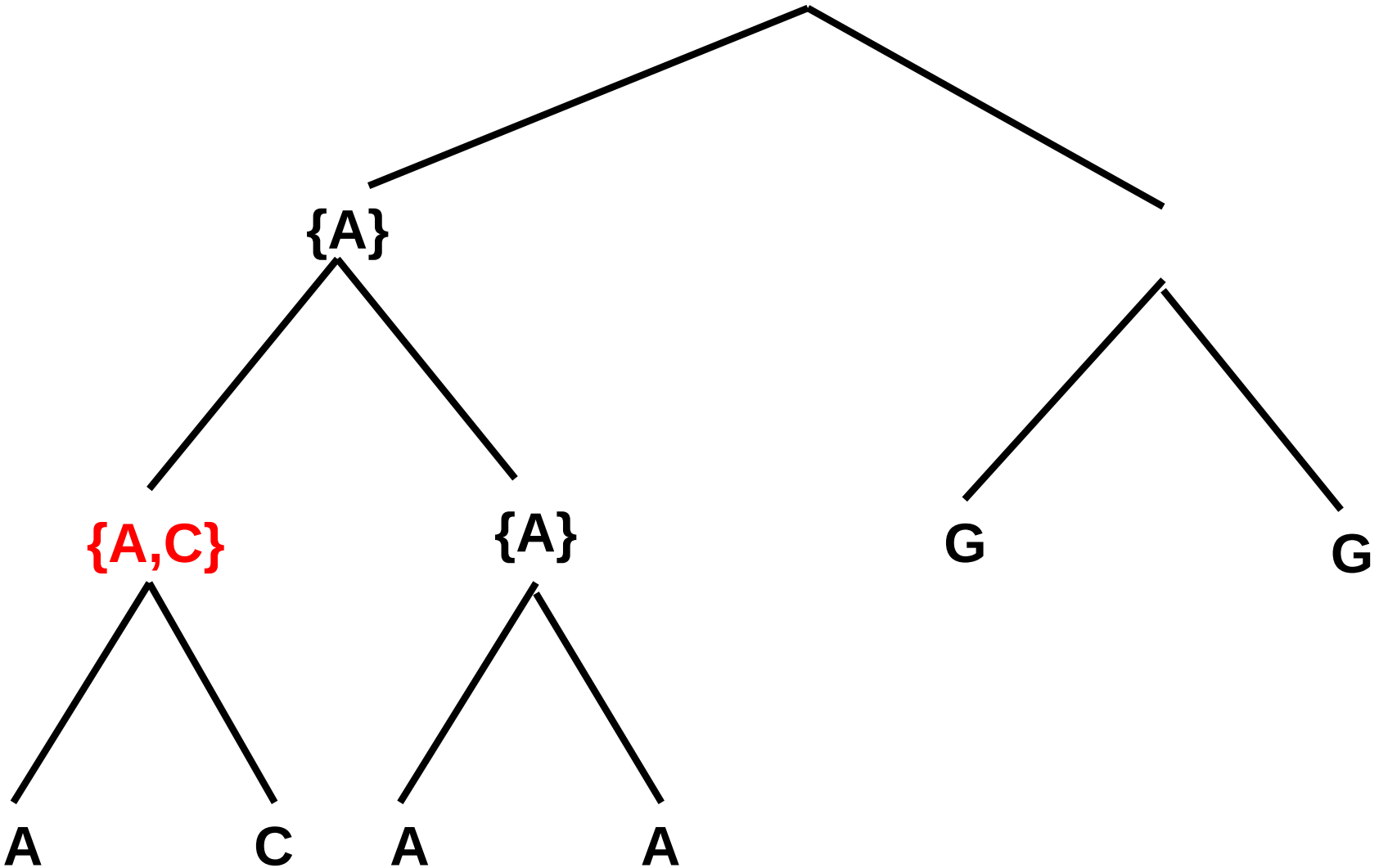
Example of Fitch's algorithm



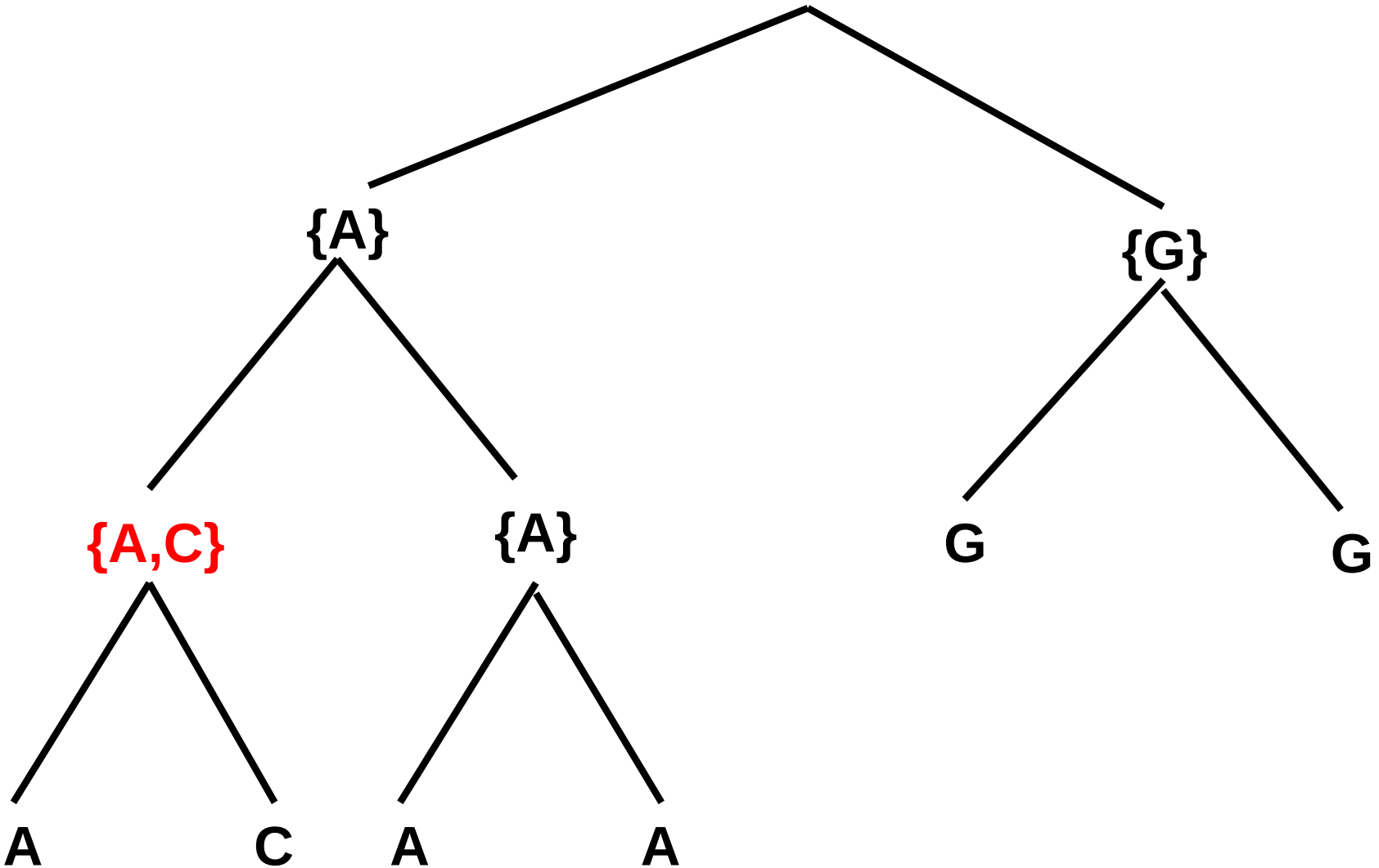
Example of Fitch's algorithm



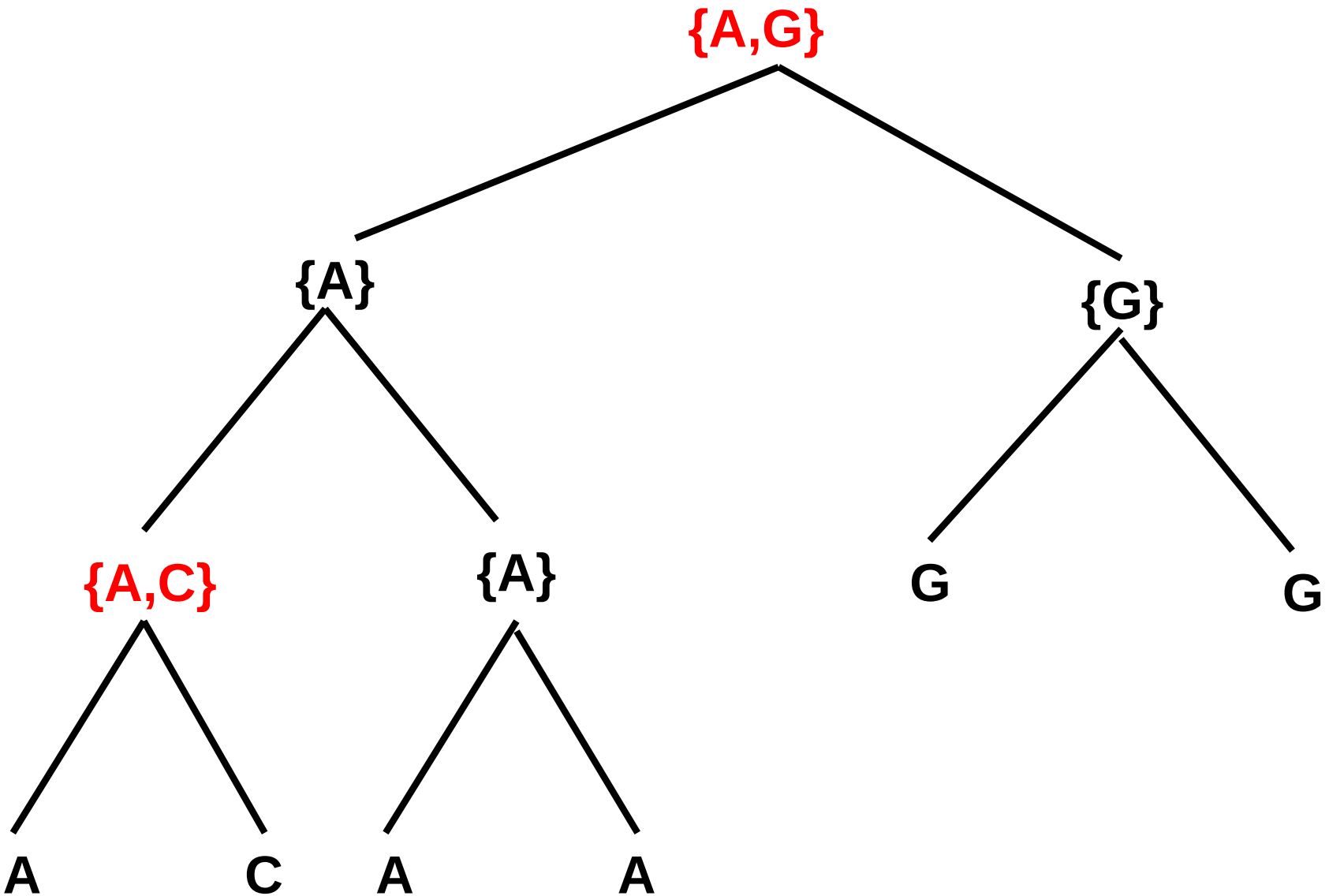
Example of Fitch's algorithm



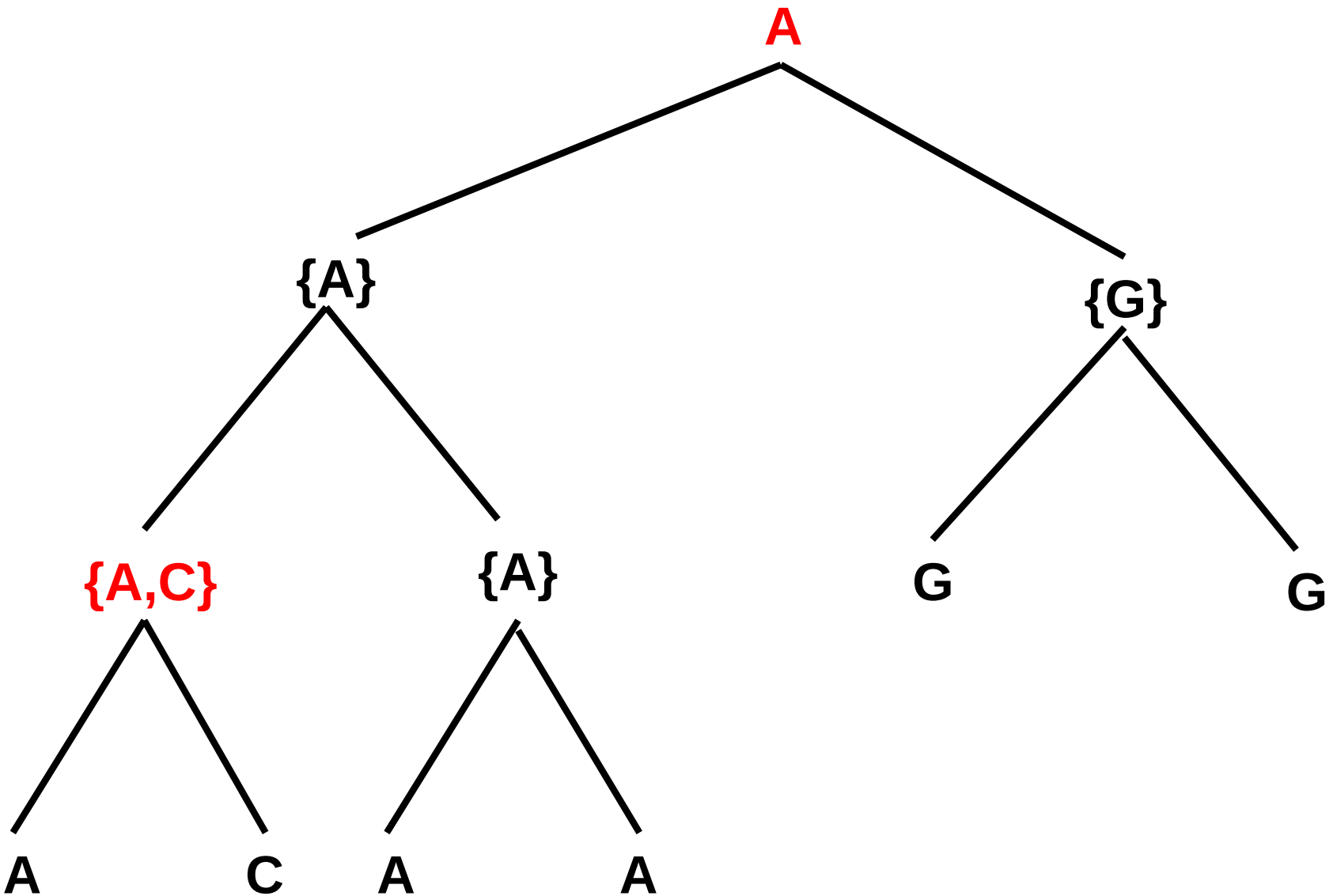
Example of Fitch's algorithm



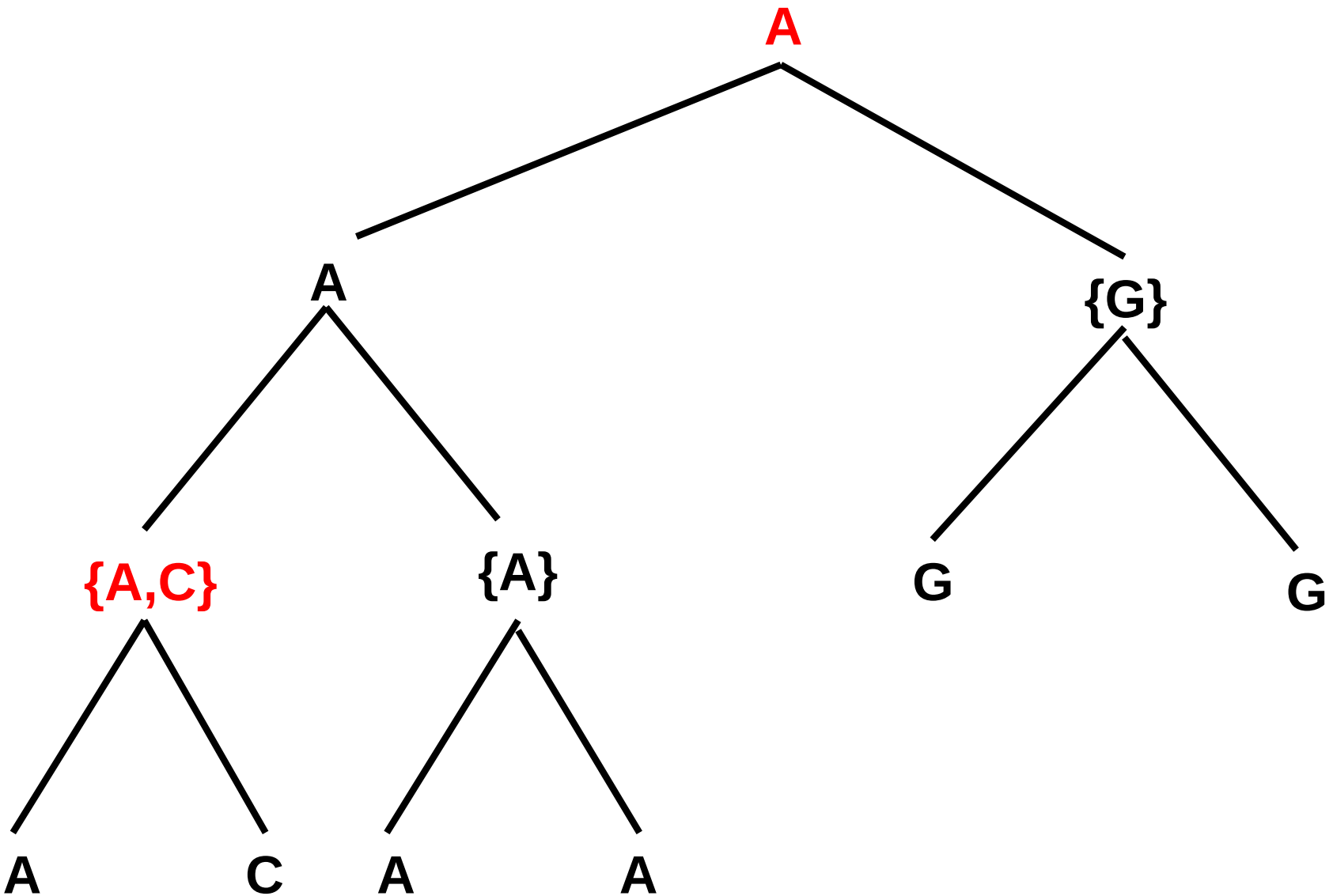
Example of Fitch's algorithm



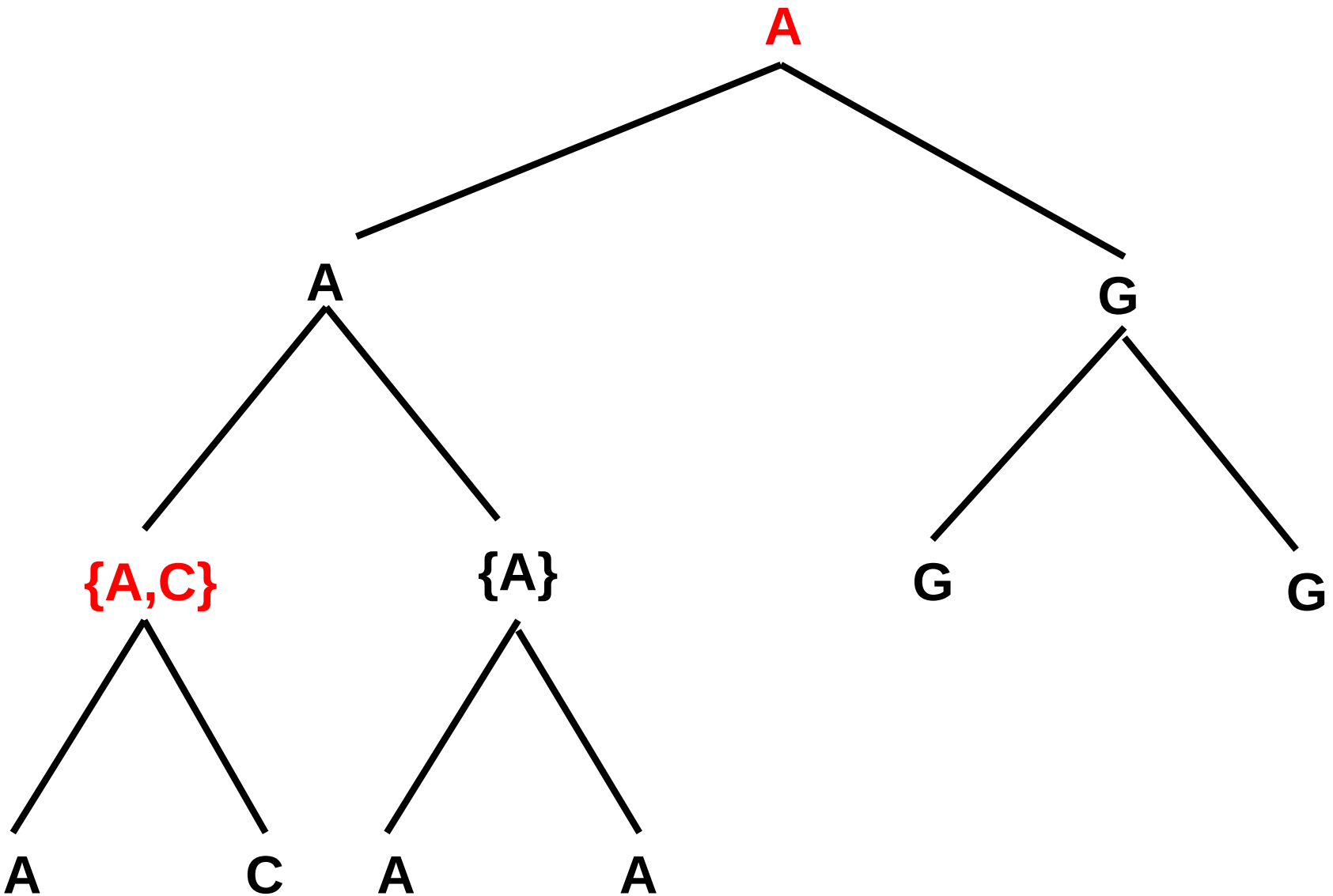
Example of Fitch's algorithm



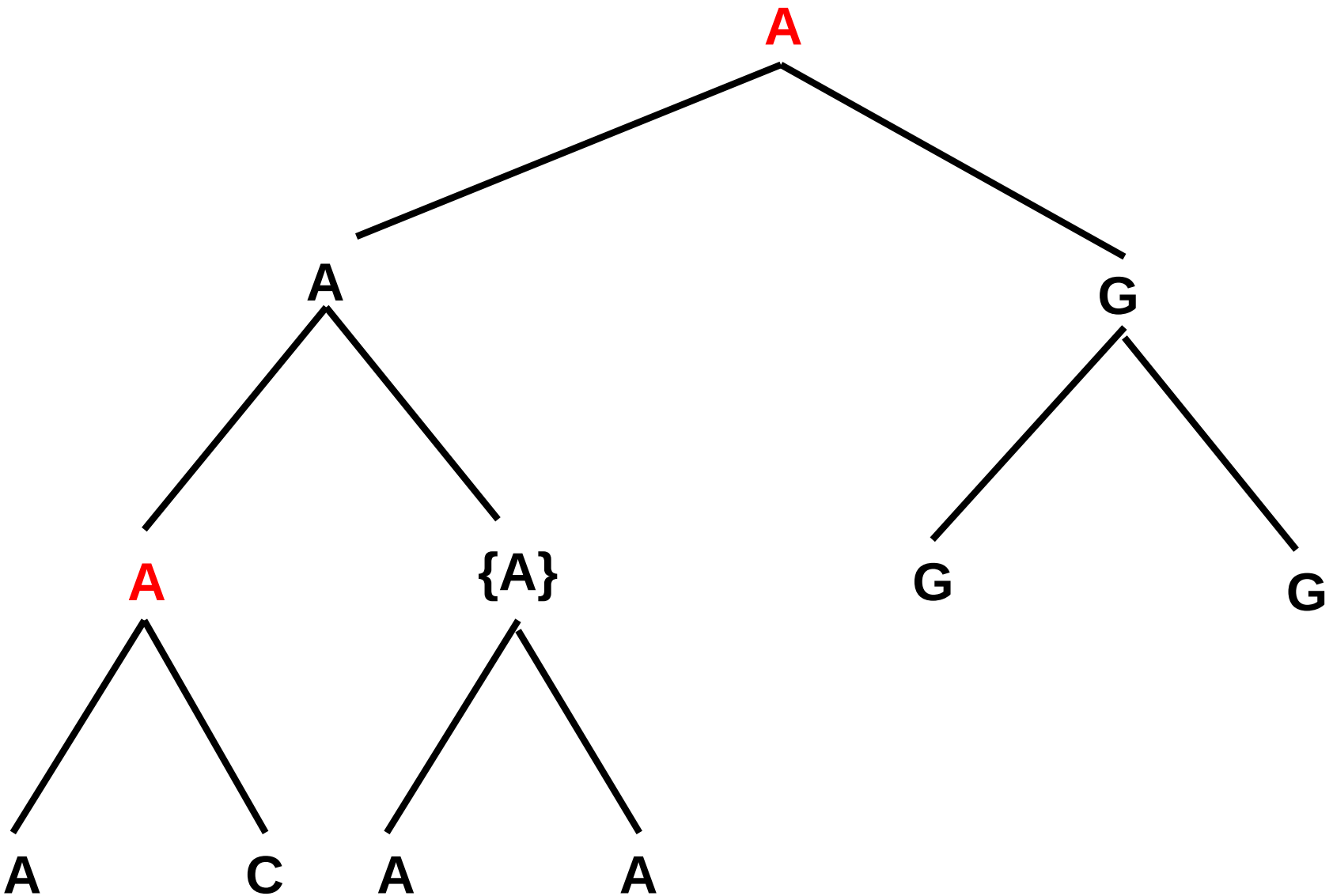
Example of Fitch's algorithm



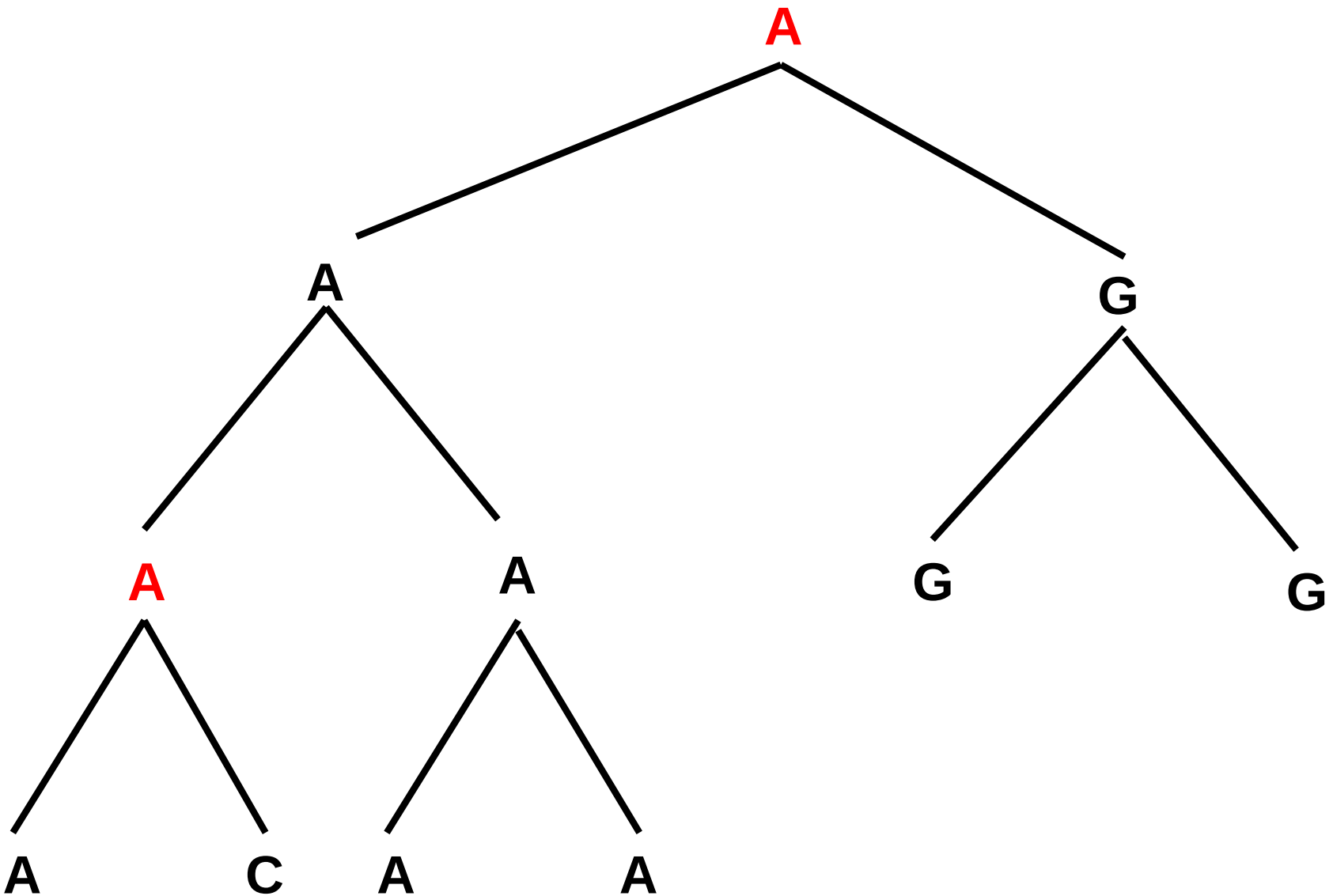
Example of Fitch's algorithm



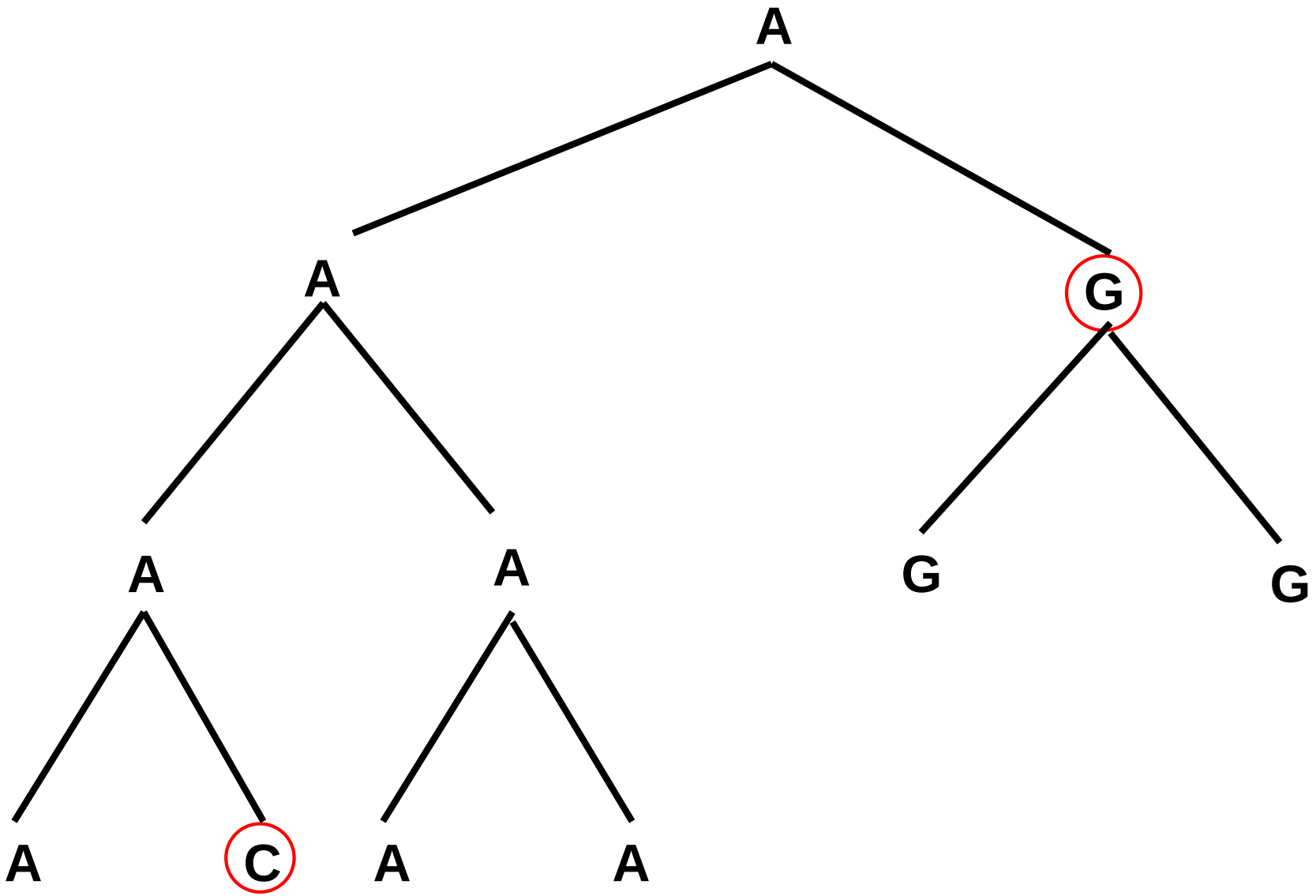
Example of Fitch's algorithm



Example of Fitch's algorithm



Example of Fitch's algorithm



- Final score in this example: $p(T) = 2$

Weighted Small Parsimony

- Generalised model where state changes have different costs, given by cost matrix $C = (c_{ij})$, where c_{ij} is the cost for the evolutionary change from state i to j.
- Small parsimony problem can still be solved optimally in polynomial time using a dynamic programming algorithm due to Sankoff and Cedergren (1983).
- Basic idea: calculate cost s_i^v for state i at node v with children l and r as:

$$s_i^v = \min_j(c_{ij} + s_j^l) + \min_n(c_{in} + s_n^r)$$

- Optimal cost in the end is given by $\min_i s_i^a$, where a is the root node of the tree.

Large Parsimony Problem

- Finding the optimal tree with respect to parsimony score for a given set of species is an NP-hard problem, therefore exact polynomial-time algorithms are very unlikely.
- Enumerating all possible trees is prohibitively expensive for larger data sets (there are $1 \cdot 3 \cdot 5 \ldots (2n-3)$ rooted binary trees with n labelled leaves and $n-1$ unlabelled internal nodes)
- For example, on $n = 20$ leaves there are $\approx 10^{21}$ trees.
- Many different heuristics and exact approaches have been proposed: stochastic search, branch and bound, nearest neighbour interchange, subtree pruning and recrafting, tree bisection and reconnection (TBR), ...

A Heuristic Approach

- **Incremental tree building heuristic:**
  - Start with a small number of species and compute an optimal tree for them.
  - Process the remaining species one by one; add each one to the tree so that the resulting tree has the smallest possible parsimony score.
- Does not guarantee optimal trees.
- Different processing orders lead to different trees.

An Exact Approach

- **Branch-and-bound algorithm:**
  - Enumerate all possible trees: Start with a tree on two species, then try all possibilities for adding the third species, the fourth species, and so on.
  - Record the best parsimony score obtained so far.
  - If the current (partial) tree already has a parsimony score worse than the best tree so far, do not explore this possibility further.
- Guaranteed to compute an optimal tree.
- Works because adding species to a tree can only make the parsimony score worse.
- Leads to better running-time than complete enumeration, but is still too slow for large data sets.

Final Comments about Parsimony

- In simulation experiments with realistic models of evolution, trees with optimal parsimony scores yield "good" phylogenies.
- Experimental evidence shows that only trees with parsimony scores less than 0.01% above the optimal score yield sufficiently good trees.
- **Challenges:**
  - More effective heuristics
  - Good lower bounds
  - Faster branch-and-bound or other exact algorithms
  - Statistical performance issues

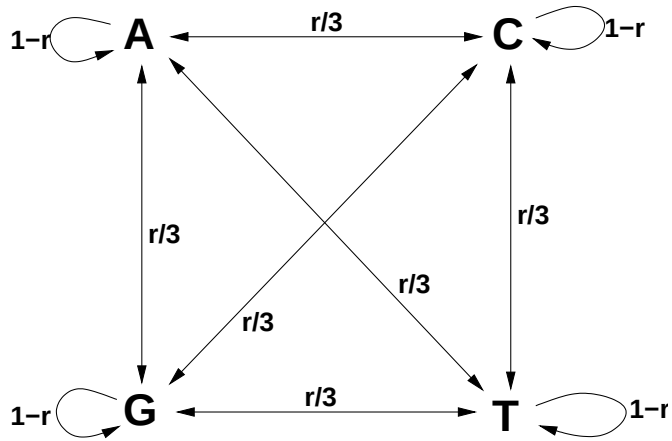
Maximum Likelihood

Maximum Likelihood (ML)

- Define a stochastic model of evolution.
- For every possible tree T with the species as leaves, there is a certain probability $\Pr(\text{data} \mid T)$ that the tree produces the given sequences at the leaves.
- **Idea of ML:**
  The best tree is the one that maximises $\Pr(\text{data} \mid T)$.

Jukes Cantor Model

- Model of DNA evolution introduced by T. Jukes and C. Cantor (1969)
- Each site changes independently according to the following Markov chain:

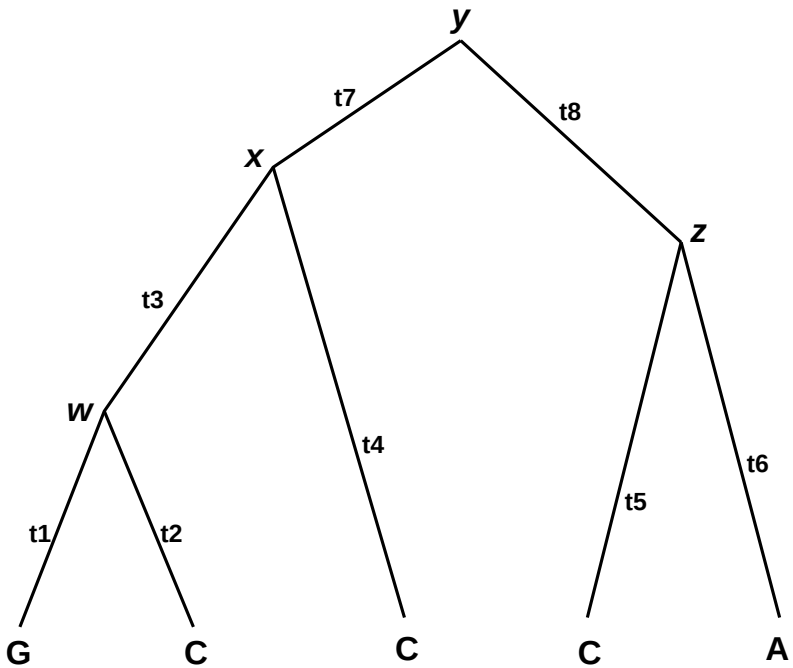


- Continuous limit leads to $\Pr(q_j \mid q_i, t) = \frac{1}{4}\left(1 - e^{-\frac{4}{3}rt}\right)$ for the probability of ending up in state q_j after time t if starting in state $q_i \neq q_j$.

Applying the Jukes Cantor Model

- Molecular clock assumption: rate of change r is the same on all edges of the tree
- As all sites are assumed to change independently, the probability of a sequence changing from s_1 to s_2 is the product of the probabilities for the individual sites.
- If every edge of the tree is associated with a length (time), we can compute the probability that the given sequences appear at the leaves.
- We want to determine edge lengths that maximise this probability.

Example



$$\Pr(\text{data} \mid T, t_1, t_2, \ldots, t_8) = \sum_w \sum_x \sum_y \sum_z \Pr(y) \Pr(x \mid y, t_7) \times$$
$$\Pr(w \mid x, t_3) \Pr(G \mid w, t_1) \Pr(C \mid w, t_2) \Pr(C \mid x, t_4) \times$$
$$\Pr(z \mid y, t_8) \Pr(C \mid z, t_5) \Pr(A \mid z, t_6)$$

(efficiently computable by dynamic programming)

ML Computation Approach

- Enumerate all trees with the given sequences at the leaves.
- For each tree, estimate the edge lengths and compute the likelihood score, possibly using iterative adjustment of edge lengths (or some other edge length optimisation methods) to improve the score.
- Output the tree with the best likelihood score.

Comments about ML

- Even the evaluation of a single tree is computationally expensive, as it is difficult to determine the edge lengths. Current methods compute only local optima; none of them guarantees to find optimal edge lengths.
- Similar to MP, ML has the problem of needing to search the huge space of all possible trees efficiently.
- Branch-and-bound techniques are used for small data sets.
- Phylogeny estimation using ML methods depends on the stochastic model used.
- ML can be proved to be "statistically consistent" for a general class of stochastic models.

Phylogeny Estimation Trends

- Analysis of phylogeny estimation methods in stochastic models of evolution (concepts of "statistical consistency" and "fast convergence")
- Speeding up the search through the space of all trees for MP and ML
- Speeding up the ML evaluation of a fixed model tree topology (assigning edge lengths)
- Non-tree models
- Phylogeny estimation based on gene order

Further Sources

- Tandy Warnow: Computational methods in phylogenetic analysis. Tutorial at Computational Systems Bioinformatics Conference (CSB), Stanford, 2004.
- Notes of Joe Felsenstein's lectures in the course "Algorithms in Molecular Biology," organised by R. Karp and L. Ruzzo at the University of Washington, Winter 1998.

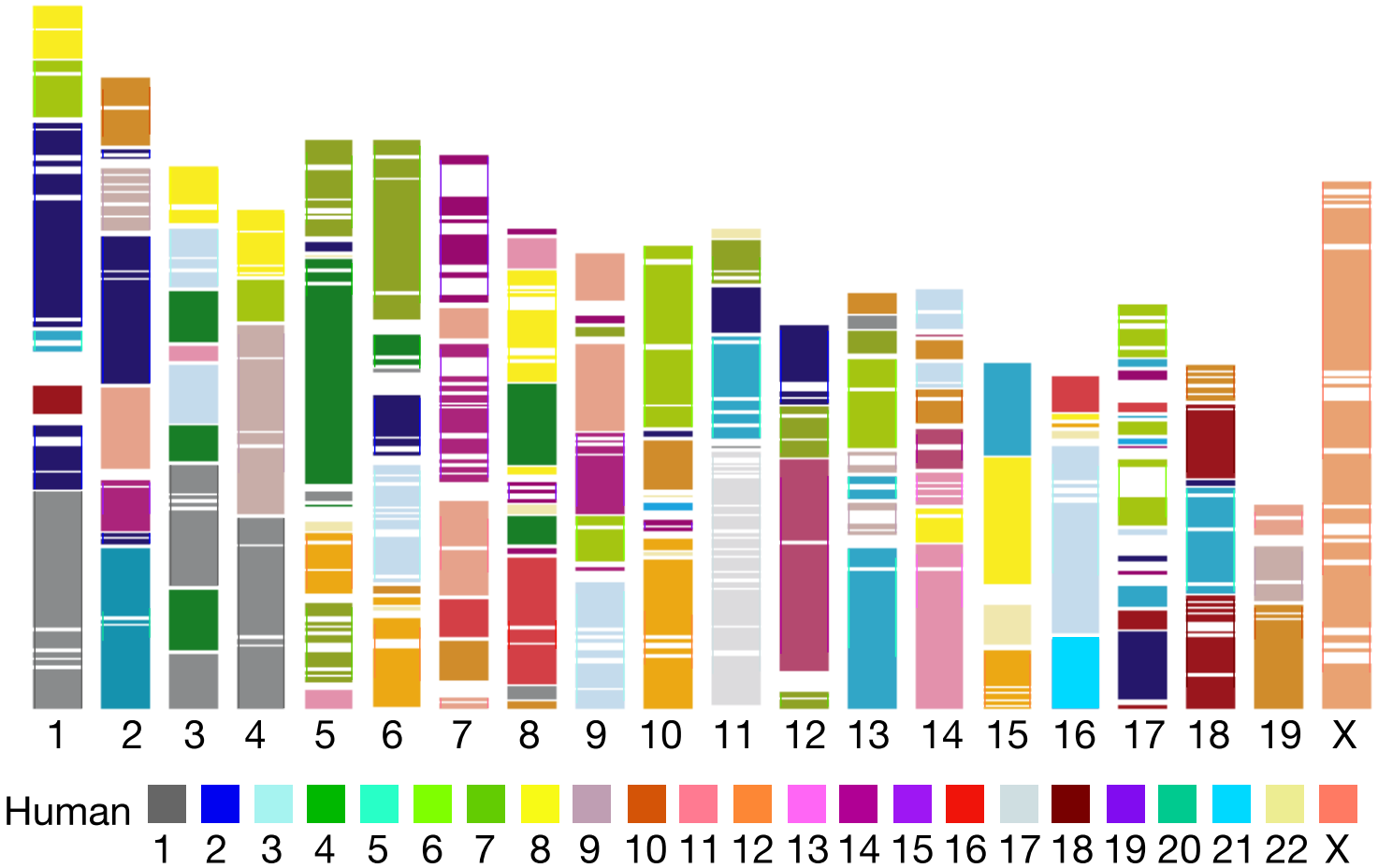
Genome Rearrangement

(mainly from Chapter 5 of [JP])

Motivation

- Genome: genes arranged on chromosomes.
- Genome rearrangement: The order and location of genes on chromosomes changes over time (evolution).
- Genome rearrangement events happen much less frequently than single-letter mutations in DNA.
- The number of rearrangement steps required to transform one genome into another is a good measure for evolutionary distance, also for species that are very far from each other.

Mouse Genome / Human Genome



(Waterston et al., Nature, 2002)

Mouse and Human

"In some ways, the human genome is just the mouse genome cut into about 300 large genomic fragments, called *synteny blocks*, that have been pasted together in a different order."

(Jones and Pevzner, An Introduction to Bioinformatics Algorithms, 2004)

- Divergence of humans and mice approximately 80 million years ago.
- An estimated 250 rearrangements have occured in the two genomes since then.

Types of Rearrangements

- **For unichromosomal genomes:**
  - **Reversal** (inversion): a contiguous interval of genes is put into the reverse order
- **For multichromosomal genomes:**
  - **Reversal** (as above)
  - **Translocation** (recombining two chromosomes)
  - **Fission** (splitting of a chromosome into two)
  - **Fusion** (merging of two chromosomes into one)

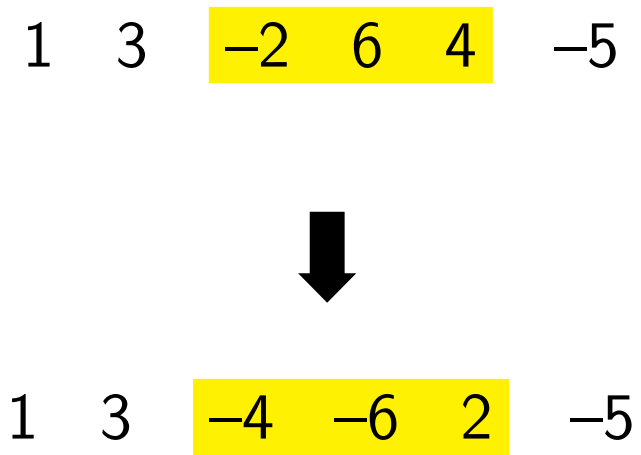
Modelling of Genome

- Set of n genes (or genomic fragments, syntenies), represented by $\{1, 2, \ldots, n\}$.
- **Signed model** (reversal of a, b, c yields $-c, -b, -a$) vs. **unsigned model** (reversal of a, b, c yields c, b, a). Choice of model depends on available data.
- Example of unichromosomal genome with $n = 6$ genes:

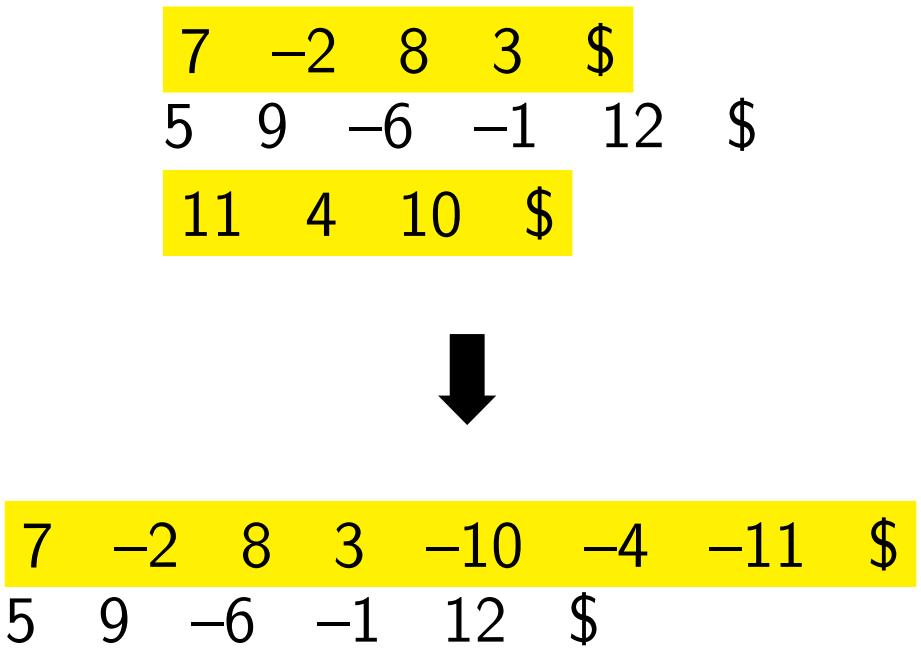
  1 3 –2 6 4 –5
- Example of multichromosomal genome with 10 genes:

  1 –3 7 4 $
  –6 –2 10 $
  5 9 –8 $

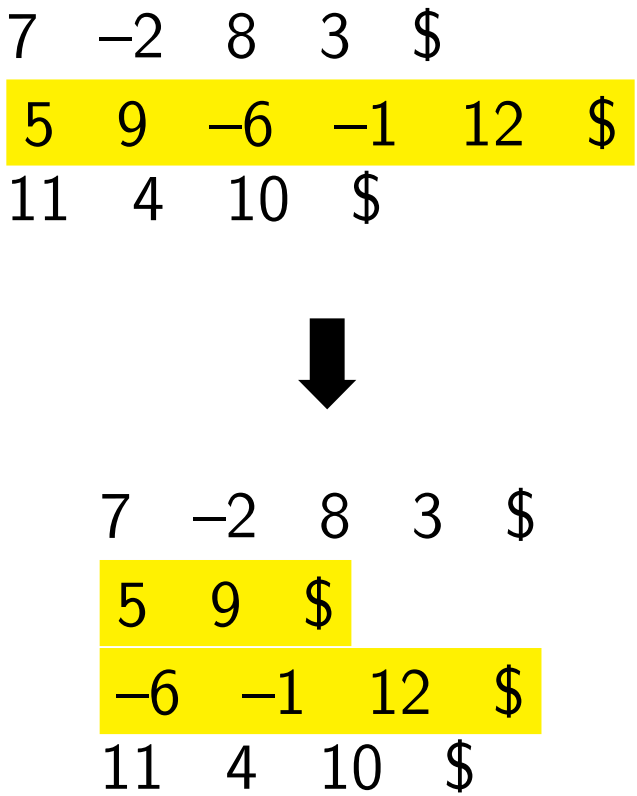
Reversal



Fusion



Fission



Translocation

7 –2 8 3 $
5 9 –6 –1 12 $
11 4 10 $

7 –2 8 –4 –11 $
5 9 –6 –1 12 $
–3 10 $

Rearrangement Problem

Pairwise Genome Rearrangement Problem:

- **Given two genomes A and B, compute an optimal (i.e., shortest) sequence of rearrangement events transforming A into B.**

We will consider only **unichromosomal genomes** and rearrangement by **reversals** in the remainder of this lecture.

Reversal distance

Definition. Given two permutations $\pi = (\pi_1, \ldots, \pi_n)$ and $\sigma = (\sigma_1, \ldots, \sigma_n)$ of $\{1, 2, \ldots, n\}$, the **reversal distance** $d(\pi, \sigma)$ is the smallest number of reversals that will take π to σ.

Remark. Without loss of generality, we can assume that one of the two permutations is the identity, $(1, 2, \ldots, n)$, and consider only $d(\pi)$, the reversal distance between π and the identity. ➠ Sorting by reversals

To distinguish signed and unsigned reversal distance, we can also write $d_s(\pi)$ and $d_u(\pi)$, respectively.

Signed versus Unsigned

What is the reversal distance of $\pi = (3, 4, 1, 2)$?

Answer for unsigned permutations:

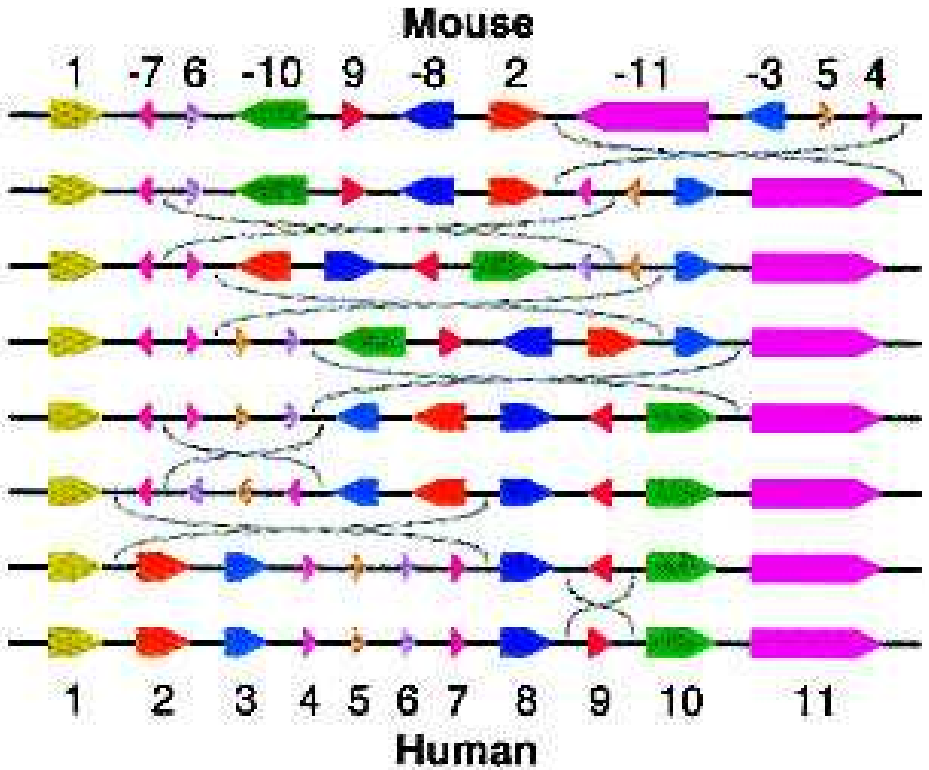
- $d_u(\pi) = 2$: 3 4 1 2
  1 4 3 2
  1 2 3 4

Answer for signed permutations:

- $d_s(\pi) = 3$: 3 4 1 2
  –4 –3 1 2
  –4 –3 –2 –1
  1 2 3 4

Mouse and Human X Chromosome

Mouse X: 1 7 6 10 9 8 2 11 3 5 4
Human X: 1 2 3 4 5 6 7 8 9 10 11



(Pevzner and Tesler, 2003)

Sorting by Reversals

—

Unsigned Permutations

Unsigned Reversal Distance

Known Results

- Determining the unsigned reversal distance is an *NP*-hard problem [Caprara, 1997].
- Interest in approximation algorithms, i.e., polynomial-time algorithms that are guaranteed to approximate the reversal distance within a small factor (called the approximation ratio).
- History of approximation results:
  - 2-approximation [Kececioglu and Sankoff, 1993]
  - 1.75-approximation [Bafna and Pevzner, 1993]
  - 1.5-approximation [Christie, 1995]
  - 1.375-approximation [Berman, Hannenhalli, Karpinski, 2002]

A First Attempt

- Naive greedy algorithm:

  for $i = 1$ **to** n **do**
  if $\pi_i \neq i$ **then**
  perform a reversal that brings i into position i.

A First Attempt

- Naive greedy algorithm:

  for $i = 1$ **to** n **do**
  if $\pi_i \neq i$ **then**
  perform a reversal that brings i into position i.

- **Example:**

  6 1 2 3 4 5

A First Attempt

- Naive greedy algorithm:

  for $i = 1$ **to** n **do**
  if $\pi_i \neq i$ **then**
  perform a reversal that brings i into position i.

- **Example:**

6	1	2	3	4	5
1	6	2	3	4	5

A First Attempt

- Naive greedy algorithm:

  for $i = 1$ **to** n **do**
  if $\pi_i \neq i$ **then**
  perform a reversal that brings i into position i.

- **Example:**

6	1	2	3	4	5
1	6	2	3	4	5
1	2	6	3	4	5

A First Attempt

- Naive greedy algorithm:

  for $i = 1$ **to** n **do**
  if $\pi_i \neq i$ **then**
  perform a reversal that brings i into position i.

- **Example:**

6	1	2	3	4	5
1	6	2	3	4	5
1	2	6	3	4	5
1	2	3	6	4	5

A First Attempt

- Naive greedy algorithm:

  for $i = 1$ **to** n **do**
  if $\pi_i \neq i$ **then**
  perform a reversal that brings i into position i.

- **Example:**

6	1	2	3	4	5
1	6	2	3	4	5
1	2	6	3	4	5
1	2	3	6	4	5
1	2	3	4	6	5

A First Attempt

- Naive greedy algorithm:

  for $i = 1$ **to** n **do**
  if $\pi_i \neq i$ **then**
  perform a reversal that brings i into position i.

- **Example:**

  6 1 2 3 4 5
  1 6 2 3 4 5
  1 2 6 3 4 5
  1 2 3 6 4 5
  1 2 3 4 6 5
  1 2 3 4 5 6

  (using $n - 1 = 5$ reversals)

A First Attempt

- Naive greedy algorithm:

  for $i = 1$ **to** n **do**
  if $\pi_i \neq i$ **then**
  perform a reversal that brings i into position i.

- **Example:**

  6 1 2 3 4 5
  1 6 2 3 4 5
  1 2 6 3 4 5
  1 2 3 6 4 5
  1 2 3 4 6 5
  1 2 3 4 5 6
  (using $n - 1 = 5$ reversals)

Optimum:

6 1 2 3 4 5
5 4 3 2 1 6
1 2 3 4 5 6
(using two reversals)

➠ ratio $\frac{n-1}{2}$ (terrible!)

The Concept of Breakpoints

- A **breakpoint** in π is a pair of elements π_i and π_{i+1} with $|\pi_i - \pi_{i+1}| > 1$.
- For convenience, we add 0 and $n+1$ to the beginning and end of π, respectively.
- Let $b(\pi)$ denote the number of breakpoints of π.

Example:

$\pi = (0, 2, 1, 3, 4, 5, 8, 7, 6, 9)$ has $b(\pi) = 4$ breakpoints

Claim. $\lceil b(\pi)/2 \rceil \leq d_u(\pi) \leq n - 1$

Proof:

- Every reversal "fixes" at most two breakpoints.

Increasing and Decreasing Strips

- Any permutation can be partitioned into increasing strips (overlined) and decreasing strips (underlined).
- **Example:** $\pi = (\overline{0}, \underline{2, 1}, \overline{3, 4, 5}, \underline{8, 7, 6}, \overline{9})$

Observation: If there is at least one decreasing strip, there is a reversal that reduces the number of breakpoints.

Proof: Consider smallest element in any decreasing strip, say, element 5:

Case 1: $\ldots, \underline{7, 6, 5}, 1, \ldots, \overline{3, 4}, 9, \ldots$

Case 2: $\ldots, \overline{3, 4}, 9, \ldots, \underline{7, 6, 5}, 1, \ldots$

Breakpoint Reversal Sort

- Algorithm **Breakpoint Reversal Sort**:

  while $b(\pi) > 0$ **do**
    if π has a decreasing strip **then**
      perform a reversal that decreases $b(\pi)$
    else
      perform a reversal that turns an increasing strip into a decreasing strip;

Breakpoint Reversal Sort

- Algorithm **Breakpoint Reversal Sort**:

  while $b(\pi) > 0$ **do**
    if π has a decreasing strip **then**
      perform a reversal that decreases $b(\pi)$
    else
      perform a reversal that turns an increasing strip into a decreasing strip;

Example: $\overline{0}\ \underline{2}\ \ \underline{8\ 7\ 6\ 5}\ \underline{1}\ \underline{4\ 3}\ \ \overline{9}$ $\qquad b(\pi) = 5$

Breakpoint Reversal Sort

- Algorithm **Breakpoint Reversal Sort**:

  while $b(\pi) > 0$ **do**
    if π has a decreasing strip **then**
      perform a reversal that decreases $b(\pi)$
    else
      perform a reversal that turns an increasing strip into a decreasing strip;

Example: $\overline{0}\ \underline{2}\ \underline{8\ 7\ 6\ 5}\ \underline{1}\ \underline{4\ 3}\ \overline{9}$ $\qquad b(\pi) = 5$

Breakpoint Reversal Sort

- Algorithm **Breakpoint Reversal Sort**:

  while $b(\pi) > 0$ **do**
    if π has a decreasing strip **then**
      perform a reversal that decreases $b(\pi)$
    else
      perform a reversal that turns an increasing strip into a decreasing strip;

Example: $\overline{0}\ \underline{2}\ \underline{8\ 7\ 6\ 5}\ \underline{1}\ \underline{4\ 3}\ \overline{9}$ $\quad b(\pi) = 5$

$\overline{0}\ \ \overline{2\ 3\ 4}\ \underline{1}\ \ \overline{5\ 6\ 7\ 8\ 9}$ $\quad b(\pi) = 3$

Breakpoint Reversal Sort

- Algorithm **Breakpoint Reversal Sort**:

  while $b(\pi) > 0$ **do**
    if π has a decreasing strip **then**
      perform a reversal that decreases $b(\pi)$
    else
      perform a reversal that turns an increasing strip into a decreasing strip;

Example: $\overline{0}\ \underline{2}\ \underline{8\ 7\ 6\ 5}\ \underline{1}\ \underline{4\ 3}\ \overline{9}$ $\quad b(\pi) = 5$

$\overline{0}\ \overline{2\ 3\ 4}\ \underline{1}\ \overline{5\ 6\ 7\ 8\ 9}$ $\quad b(\pi) = 3$

Breakpoint Reversal Sort

- Algorithm **Breakpoint Reversal Sort**:

  while $b(\pi) > 0$ **do**
    if π has a decreasing strip **then**
      perform a reversal that decreases $b(\pi)$
    else
      perform a reversal that turns an increasing strip into a decreasing strip;

Example: $\overline{0}\ 2\ \underline{8\ 7\ 6\ 5}\ \underline{1}\ \underline{4\ 3}\ \overline{9}$ $\quad b(\pi) = 5$

$\overline{0}\ \overline{2\ 3\ 4}\ \underline{1}\ \overline{5\ 6\ 7\ 8\ 9}$ $\quad b(\pi) = 3$

$\overline{0\ 1}\ \underline{4\ 3\ 2}\ \overline{5\ 6\ 7\ 8\ 9}$ $\quad b(\pi) = 2$

Breakpoint Reversal Sort

- Algorithm **Breakpoint Reversal Sort**:

  while $b(\pi) > 0$ **do**
    if π has a decreasing strip **then**
      perform a reversal that decreases $b(\pi)$
    else
      perform a reversal that turns an increasing strip into a decreasing strip;

Example: $\overline{0}\ \underline{2}\ \underline{8\ 7\ 6\ 5}\ \underline{1}\ \underline{4\ 3}\ \overline{9}$ $\quad b(\pi) = 5$

$\overline{0}\ \overline{2\ 3\ 4}\ \underline{1}\ \overline{5\ 6\ 7\ 8\ 9}$ $\quad b(\pi) = 3$

$\overline{0\ 1}\ \underline{4\ 3\ 2}\ \overline{5\ 6\ 7\ 8\ 9}$ $\quad b(\pi) = 2$

Breakpoint Reversal Sort

- Algorithm **Breakpoint Reversal Sort**:

  while $b(\pi) > 0$ **do**
    if π has a decreasing strip **then**
      perform a reversal that decreases $b(\pi)$
    else
      perform a reversal that turns an increasing strip into a decreasing strip;

Example: $\overline{0}\ \underline{2}\ \underline{8\ 7\ 6\ 5}\ \underline{1}\ \underline{4\ 3}\ \overline{9}$ $\quad b(\pi) = 5$

$\overline{0}\ \overline{\underline{2\ 3\ 4}}\ \underline{1}\ \overline{5\ 6\ 7\ 8\ 9}$ $\quad b(\pi) = 3$

$\overline{0\ 1}\ \underline{4\ 3\ 2}\ \overline{5\ 6\ 7\ 8\ 9}$ $\quad b(\pi) = 2$

$\overline{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}$ $\quad b(\pi) = 0$

Theorem. Algorithm Breakpoint Reversal Sort is a 4-approximation algorithm.

Proof.

- Every reversal performed by the algorithm
  - reduces $b(\pi)$ by at least 1, if there is a decreasing strip.
  - creates at least one decreasing strip, otherwise.
- Thus, the algorithm needs at most $2b(\pi)$ reversals.
- Since the optimum is at least $\lceil b(\pi)/2 \rceil$, the claim follows.

Idea for Improvement

- If there is a decreasing strip, the next reversal reduces $b(\pi)$ by one and we are fine.
- The only bad case is when there is no decreasing strip, as then we get a reversal that does not reduce $b(\pi)$.
- If we could always choose a reversal reducing $b(\pi)$ and, at the same time, yielding a permutation that again has at least one decreasing strip, the bad case would never occur.
- One can show: If all reversals on π that reduce $b(\pi)$ create a permutation without decreasing strips, then there exists a reversal that reduces $b(\pi)$ by two! [Kececioglu and Sankoff, 1993]

The Improved Algorithm

while $b(\pi) > 0$ **do**

choose a reversal reducing $b(\pi)$ by the largest amount,
resolving ties among those that reduce $b(\pi)$ by one
in favor of reversals that leave a decreasing strip;

The Improved Algorithm

while $b(\pi) > 0$ **do**

choose a reversal reducing $b(\pi)$ by the largest amount, resolving ties among those that reduce $b(\pi)$ by one in favor of reversals that leave a decreasing strip;

Theorem. This algorithm is a 2-approximation algorithm.

Proof Sketch:

- Whenever the algorithm creates a π without decreasing strip, the previous reversal reduced $b(\pi)$ by two.
- The algorithm needs at most $b(\pi)$ reversals, and the optimum needs at least $\lceil b(\pi)/2 \rceil$.

The Improved Algorithm

while $b(\pi) > 0$ **do**

choose a reversal reducing $b(\pi)$ by the largest amount, resolving ties among those that reduce $b(\pi)$ by one in favor of reversals that leave a decreasing strip;

Theorem. This algorithm is a 2-approximation algorithm.

Proof Sketch:

- Whenever the algorithm creates a π without decreasing strip, the previous reversal reduced $b(\pi)$ by two.
- The algorithm needs at most $b(\pi)$ reversals, and the optimum needs at least $\lceil b(\pi)/2 \rceil$.

Note: Best known approximation ratio is 1.375.

Sorting by Reversals

—

Signed Permutations

Signed Reversal Distance

Known Results

- 1.5-approximation [Bafna and Pevzner, 1993]
- **Polynomial-time optimal algorithm**, running-time $O(n^4)$ [Hannenhalli and Pevzner, 1995].
- Running-time improved to $O(n^2)$ [Kaplan, Shamir and Tarjan, 1997].
- Running-time for computing only the reversal distance (not the reversals themselves) improved to $O(n)$ [Bader, Moret and Yan, 2001].

Main underlying concept: **breakpoint graph**

Summary: Genome Rearrangement

- Sorting by reversals is polynomial for signed permutations, and *NP*-hard and 1.375-approximable for unsigned permutations.
- Algorithms can be adapted to multichromosomal genome rearrangement with reversal, fusion, fission and translocation.
- Available implementations: GRAPPA, GRIMM, ...
- Related topics:
  - Genome rearrangement in the presence of duplicates
  - Sorting by transpositions
  - Pancake flipping