PHYLOGENETIC TREES

Mia Persson

Algorithms for Molecular Biology
Autumn 2004
Lund University

Biological Background

- Consider the problem of constructing a **phylogenetic tree** of a set of objects.
- A **phylogenetic tree** (or shorter **phylogenies**) tells us the evolutionary history, or evolutionary relationship, among a set of objects.
- Example of objects are biological species, categories of species, proteins, nucleic acids, languages, or ...

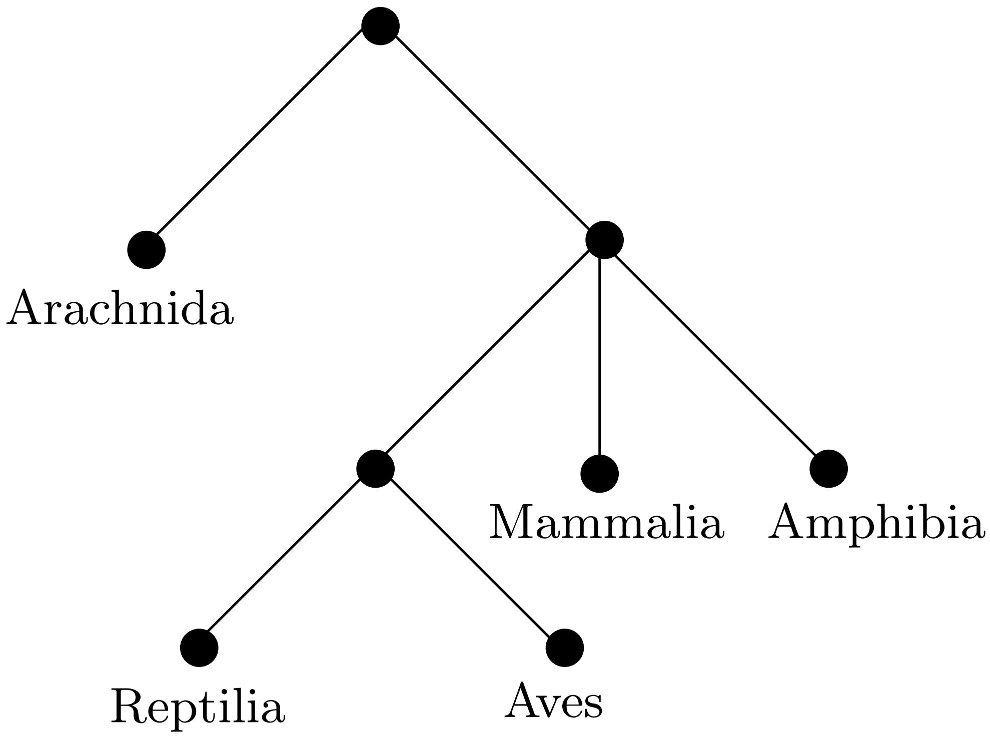
Definition - Phylogenetic Tree

Definitions:

A **tree** is an undirected acyclic connected graph. The set of exterior nodes are called **leaves**. Leaves have degree one, whereas the **interior nodes** have degree greater than one.

A **phylogenetic tree** is an unordered, rooted/unrooted tree with weighted/unweighted edges. The leaves contain the set of objects we want to study. A leaf may contain one object or a set of objects.

A Phylogenetic Tree Example



Some Methods for Phylogenetic Tree Construction

- Character state methods - Part 1
- Distance-based methods - Part 2

Part 1: Character State Methods

Data:

- For each object there is a set of *discrete characters* associated to it.
- Example of discrete characters are the numbers of fingers, presence or absence of a molecular restriction site, etc.
- Each character can have a finite number of *states*.

The data is placed in a **character state matrix**. See example...

Character State Matrix - An Example

	# of wheels	Has engine
Bike	2	N
Tricycle	3	N
Car	4	Y
Pickup truck	4	Y
Skateboard	4	N

Rows $\Leftrightarrow$ Objects

Columns $\Leftrightarrow$ Characters

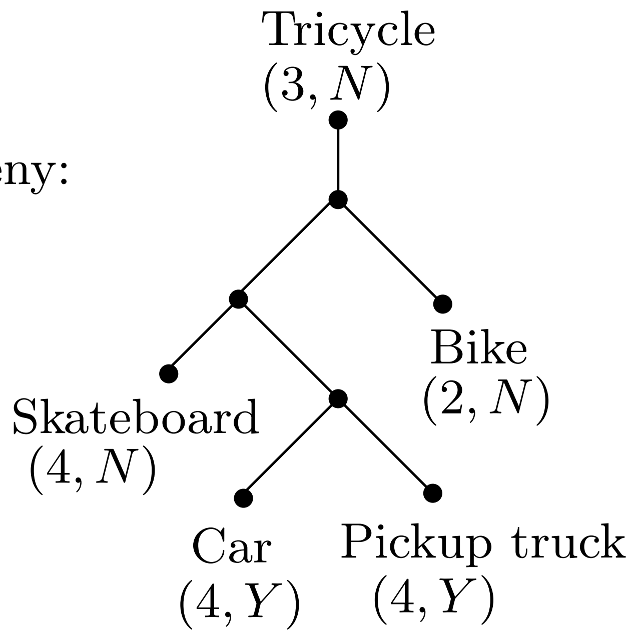
Each character has a set of possible states.

Perfect Phylogeny Problem

What we want:

A tree in which each state of each character induces a connected subgraph. **"Perfect Phylogeny"**.

A perfect phylogeny:



Perfect Phylogeny Problem (decision version)

Given a set O with n objects, a set C of m characters, each character having at most r states (n, m, r positive integers).

Perfect Phylogeny Problem (PP):
Is there a perfect phylogeny for O?

Also important: **Construction version of the above**

Combinatorics

Question: How many different leaf-labeled unrooted binary trees for $n \geq 3$ objects can we build?

Answer: $\prod_{i=3}^{n}(2i-5)$ different trees.

Proof by induction.

$$\begin{cases} T_3 & = 1 \\ T_{n+1} & = T_n \cdot \#edges = T_n \cdot (2n-3) \end{cases}$$

Growth is superexponential in n. Therefore, exhaustive search over all possible trees not practical.

Perfect Phylogeny - Complexity

The Perfect Phylogeny problem is **NP**-complete in the general case, but solvable in polynomial time for certain variants:

- Ordered characters
- Unordered characters, fixed number of states
- Unordered characters, fixed number of characters

Binary Character States

All entries of the state character matrix are 0 or 1. Then the Perfect Phylogeny problem becomes solvable in $O(mn)$ time.

Algorithm PP:

Phase 1: Decide if the input matrix M admits a perfect phylogeny.

Phase 2: If yes, then construct one.

Algorithm PP - Phase 1

Let M be a binary matrix with n rows (objects) and m columns (characters). Let O_j denote the set of objects with a 1 in column j.

Lemma: M admits a perfect phylogeny (PP) iff for every pair of columns i and j, either O_i and O_j are disjoint or one contains the other.

Proof:
$\Rightarrow$) Suppose $A, B \in O_i$, $C \notin O_i$ and $A \notin O_j$, $B, C \in O_j$. Contradiction.

$\Leftarrow$) By induction on the number of characters.

Lemma immediately gives an $O(m^2 n)$ time algorithm for phase 1, i.e., to decide if M admits a PP. But we can do better...

Algorithm PP - Phase 1 (cont' d)

Faster method: Use an auxiliary matrix L.

Algorithm FAST

1. Consider each column of M as a binary number, radix sort into decreasing order, place largest number in column 1.
2. Remove duplicate columns. Call the resulting matrix M'.
3. For each element $M'_{i,j}$:
   If $M'_{i,j} = 0$ then let $L_{i,j} = 0$.
   If $M'_{i,j} = 1$, set $L_{i,j}$ equal to the largest index $k < j$ such that $M'_{i,k} = 1$; if no such index exists, let $L_{i,j} = -1$.
4. If there is a column j for which $L_{i,j} \neq L_{l,j}$ for some i, l and $L_{i,j}, L_{l,j}$ are nonzero, then return FALSE; else return TRUE.

Algorithm PP - Phase 1 (cont' d)

Running time for FAST: $(O(mn))$

Algorithm PP - Phase 1 (cont' d)

Correctness of FAST:

- If the algorithm answers TRUE:
  Consider an arbitrary column j with $L_{i,j} \neq 0$ for some i.
  If $L_{i,j} < p < j$ then $O_j \cap O_p = \emptyset$ (ok, by Lemma)

- If the algorithm answers FALSE:
  Suppose M' has a perfect phylogeny.
  $L_{i,j} = k$ and $L_{l,j} = k' < k$ for some i, j, k, k', l.
  $M'_{l,k} = 0$ but $M'_{i,k} = 1$, so $O_k \cap O_j \neq \emptyset$.
  $O_j \not\subseteq O_k$ since $M'_{l,k} = 0$.
  $O_k \not\subseteq O_j$ since column k is to the left of column j.
  Contradicts the Lemma, so M' has no perfect phylogeny.

Algorithm PP, Phase 1 - An Example

M	c_1	c_2	c_3	c_4	c_5	c_6
A	0	0	0	1	1	0
B	1	1	0	0	0	0
C	0	0	0	1	1	1
D	1	0	1	0	0	0
E	0	0	0	1	0	0

Construct M':

M'	c_1'	c_2'	c_3'	c_4'	c_5'	c_6'
A	1	1	0	0	0	0
B	0	0	1	1	0	0
C	1	1	0	0	1	0
D	0	0	1	0	0	1
E	1	0	0	0	0	0

Construct L:

L	1	2	3	4	5	6
A	-1	1	0	0	0	0
B	0	0	-1	3	0	0
C	-1	1	0	0	2	0
D	0	0	-1	0	0	3
E	-1	0	0	0	0	0

In each column of L: All nonzero entries are equal.

Thus, M has a perfect phylogeny.

Algorithm PP - Phase 2.

Create *root*

for $i := 1$ **to** n **do**

  $curNode := root$

  for $j := 1$ **to** m **do**

    if $M'_{i,j} = 1$ **then**

      if $\exists$ edge ($curNode$,u) labeled j **then**

        $curNode := u$

      else

        Create node u

        Create edge ($curNode$,u) labeled j

        $curNode := u$

  Place i in *curNode*

for each node u except *root* **do**

  Create as many leaves linked to u as there are objects in u

Algorithm PP - Phase 2 (cont' d)

The algorithm above constructs a Perfect Phylogeny (if one exists for M) in time $O(mn)$.

Character State Matrix - Two Characters

Another special case of the Perfect Phylogeny Problem.

Can also be solved by a polynomial-time algorithm.

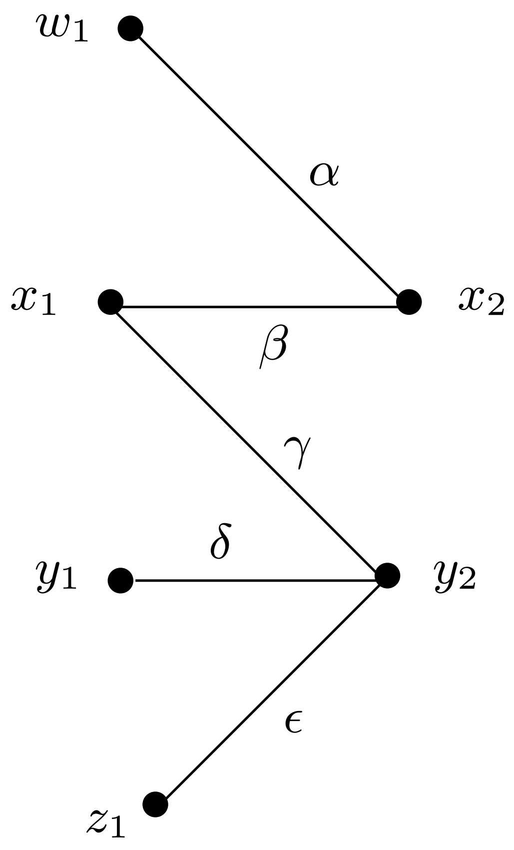
State intersection graph (SIG) for character state matrix M:

- Each state of each character in M corresponds to a vertex v in the SIG.
- Connect vertices i and j if at least one object has both states i and j.

Example:

	c_1	c_2
A	x_1	x_2
B	y_1	y_2
C	x_1	x_2
D	y_1	y_2
E	w_1	x_2
F	x_1	x_2
G	z_1	y_2
H	x_1	y_2

$\Rightarrow$ **SIG**:



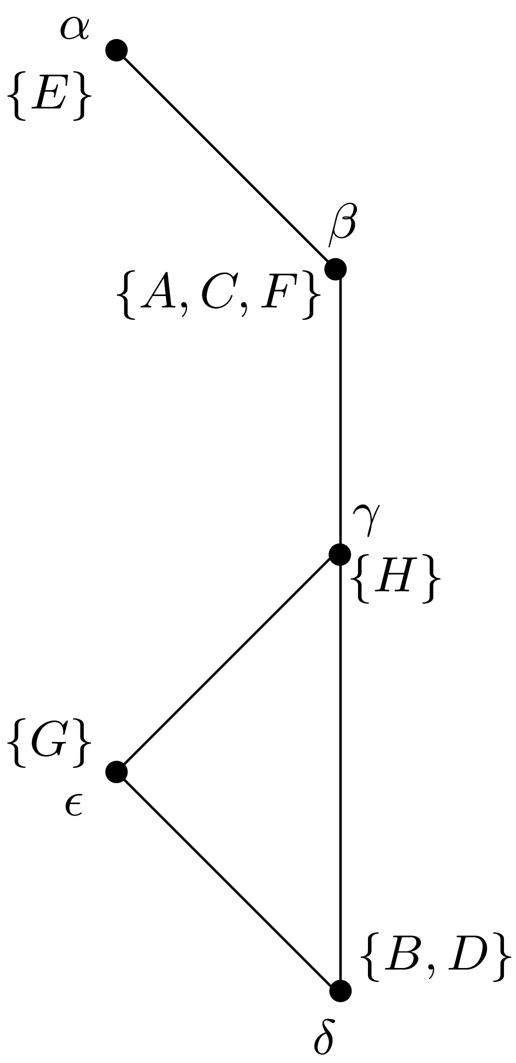
Character State Matrix - Two Characters (cont' d)

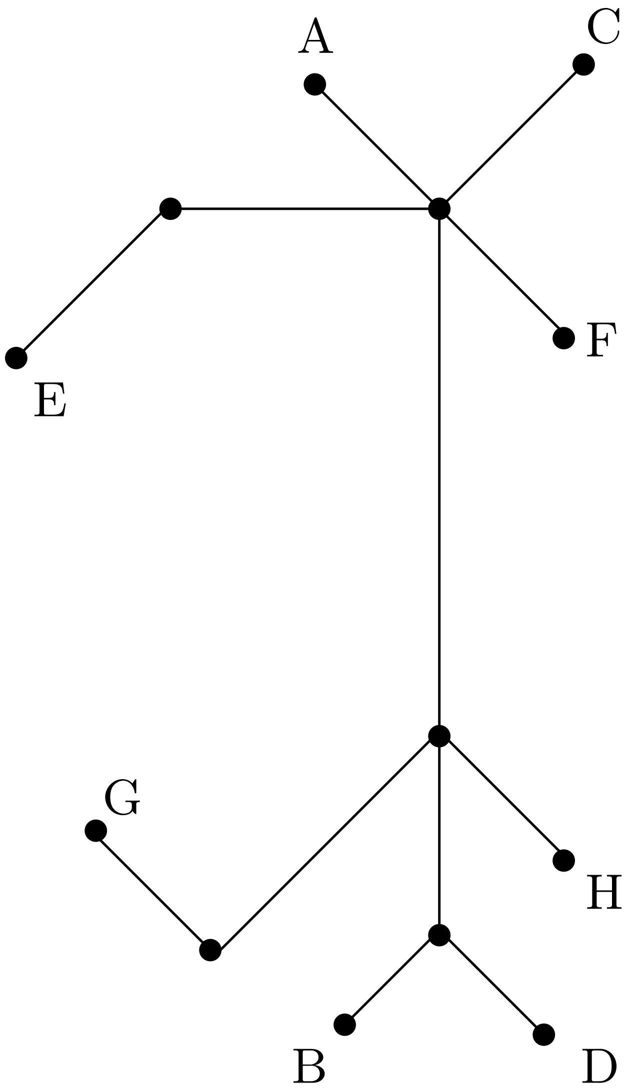
Theorem: Character state matrix M with two characters admits a perfect phylogeny iff its SIG is acyclic.

Yields an $O(n)$ time algorithm for the decision problem.

To solve the **construction problem**:
Create auxiliary graph G whose vertices correspond to edges in the SIG, compute a spanning tree for G, and attach leaves.

Example:





Parsimony and Compatibility

Sometimes the data does not admit a perfect phylogeny.

What to do?

Strategy 1: *The parsimony criterion*

Allow errors, but minimize the number of edges in the final tree.

Strategy 2: *The compatibility criterion*

Exclude as few characters as possible to get a perfect phylogeny.

Bad news: Both strategies lead to **NP**-complete problems.

Good news: Branch-and-bound methods based on clustering and existing heuristics for the Maximum Clique problem can be used.

Part 2: Distance-Based Methods

Consider the problem of reconstructing a tree based on comparative numerical data between n objects.

Input:
Distance-matrix = (n, n)-matrix M (*metric space*) with the following properties:

- $M_{i,j} > 0$ for $i \neq j$
- $M_{i,j} = 0$ for $i = j$
- $M_{i,j} = M_{j,i}$ for all i, j
- $M_{i,j} \leq M_{i,k} + M_{k,j}$ for all i, j, k

Distance-Based Methods (cont' d)

Given a metric space distance matrix M ((n,n)-matrix).

Additive Matrix Problem (decision version):
Is M *additive*, i.e., does there exists a weighted, unrooted, binary, phylogenetic tree T for M in which the total distance between leaves i and j equals $M_{i,j}$ for all i, j?

Solvable in polynomial time using the Four Point Condition:

Lemma. [Buneman 1971] M is additive iff any four objects can be labeled i, j, k, l such that $M_{i,j} + M_{k,l} = M_{i,k} + M_{j,l} \geq M_{i,l} + M_{j,k}$ holds.

Distance-Based Methods (cont' d)

If we know that M is additive then the **construction version** of the problem is also interesting.

The following algorithm for the construction version of the Additive Matrix Problem runs in $O(n^2)$ time.

Additive Matrix Algorithm

Insert any two objects

while objects still remaining **do**

  Choose a remaining object z and two objects x, y already in the tree

   repeat

    ok := True

    Calculate where on the path(x, y) to insert an internal node c with

    leaf z

    if placement coincides with an existing internal node I **then**

     if first time for this z that placement coincides with I **then**

      let y be an object in the subtree rooted at I

       ok := False

   until ok

  Insert c and z

Additive Matrix Problem - An Algorithm (cont' d)

How to calculate where on path(x, y) the internal node c should be placed:

Let $d_{i,j}$ be the distance between i and j.

We have:

$$
\begin{aligned}
M_{x,z} &= d_{x,c} + d_{z,c} && (1)\\
M_{y,z} &= d_{y,c} + d_{z,c} && (2)\\
d_{y,c} &= M_{x,y} - d_{x,c} && (3)
\end{aligned}
$$

Additive Matrix Problem - An Algorithm (cont' d)

Subtract (2) from (1), and use (3):

$$d_{x,c} \quad = \quad \frac{M_{x,y} + M_{x,z} - M_{y,z}}{2}$$

Proceed similarly for the other two unknowns and get:

$$\begin{aligned} d_{y,c} \quad &= \quad \frac{M_{x,y} + M_{y,z} - M_{x,z}}{2} \\ d_{z,c} \quad &= \quad \frac{M_{x,z} + M_{y,z} - M_{x,y}}{2} \end{aligned}$$

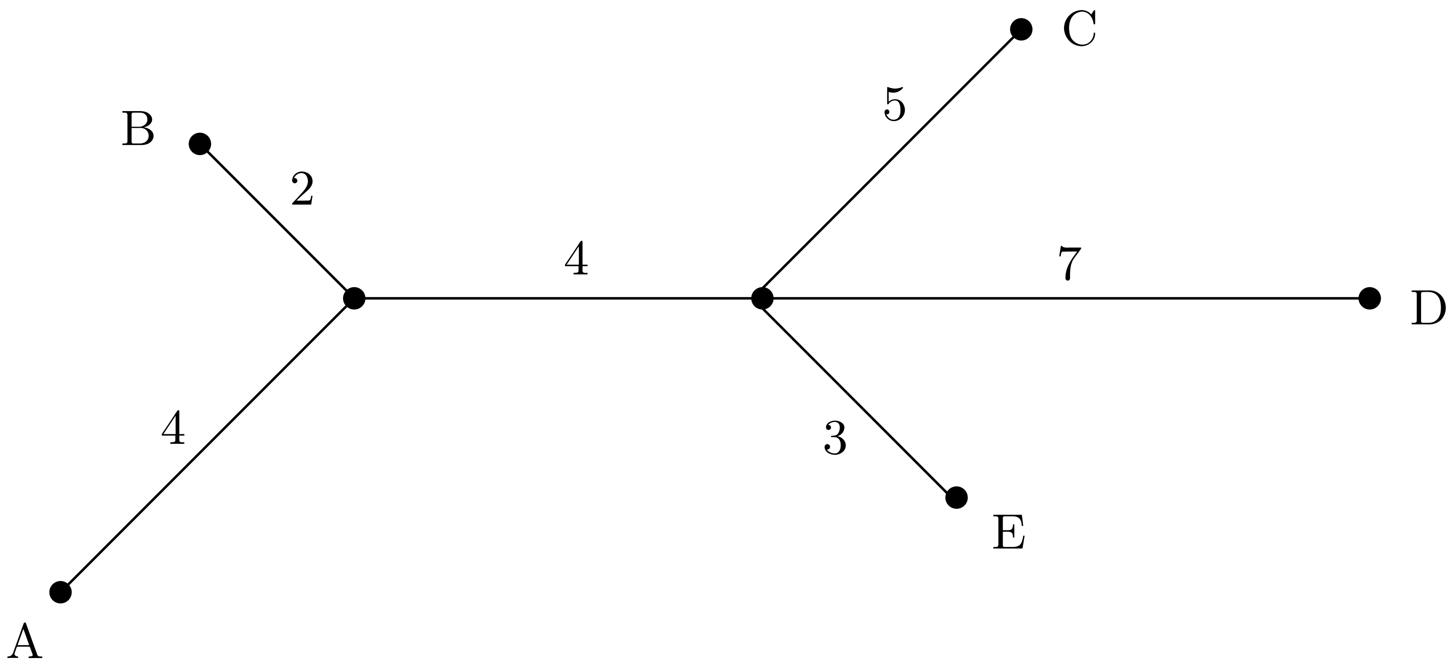
Additive Matrix Problem - An Example

Construct an additive tree for the following distance matrix:

	A	B	C	D	E
A	0	6	13	15	11
B		0	11	13	9
C			0	12	8
D				0	10
E					0

Additive Matrix Problem - An Example (cont' d)

Result:



Additive Matrix Algorithm - Correctness

Lemma. The algorithm for the Additive Matrix Problem constructs an unique additive tree (if one exists) for M.

Proof. By induction on the number of objects in M.

Ultrametric Trees

Problem: Real-life distance matrices are rarely additive since data often contains errors or there may occur multiple changes.

Therefore, we want to find a tree that is "almost" additive.

Idea: Let each pairwise distance be specified as an interval, and look for an *ultrametric tree*.

Definition: An **ultrametric tree** is an additive tree which can be rooted so that all paths from the root to a leaf have the same length.

Ultrametric Trees (cont' d)

Given two distance matrices M^l and M^h.

The Sandwich Tree Problem:
Construct an ultrametric tree with $M^l_{i,j} \le d_{i,j} \le M^h_{i,j}$ for all i, j (if one exists).

Definitions:

G^h = The complete graph corresponding to M^h

$(a, b)^T_{max}$ = The largest-weight edge on the unique path from a to b in T

Cut-weight for an edge e in T: $CW(e)$ =max$\{M^l_{a,b} \mid e = (a, b)^T_{max}\}$

Ultrametric Trees (cont' d)

Algorithm for the Sandwich Tree problem:

1. Construct a MST T for G^h.
2. Sort the edges of T in nondecreasing order of weights.
   Build a binary tree R for which LCA(i, j) contains the value of $(i, j)^T_{max}$.
3. Preprocess R to support efficient LCA queries.
   Use R to determine the cut-weights for all edges in T.
4. Sort the edges of T in nondecreasing order of cut-weights.
   Construct a binary ultrametric tree U for the objects.

Ultrametric Trees - An Example

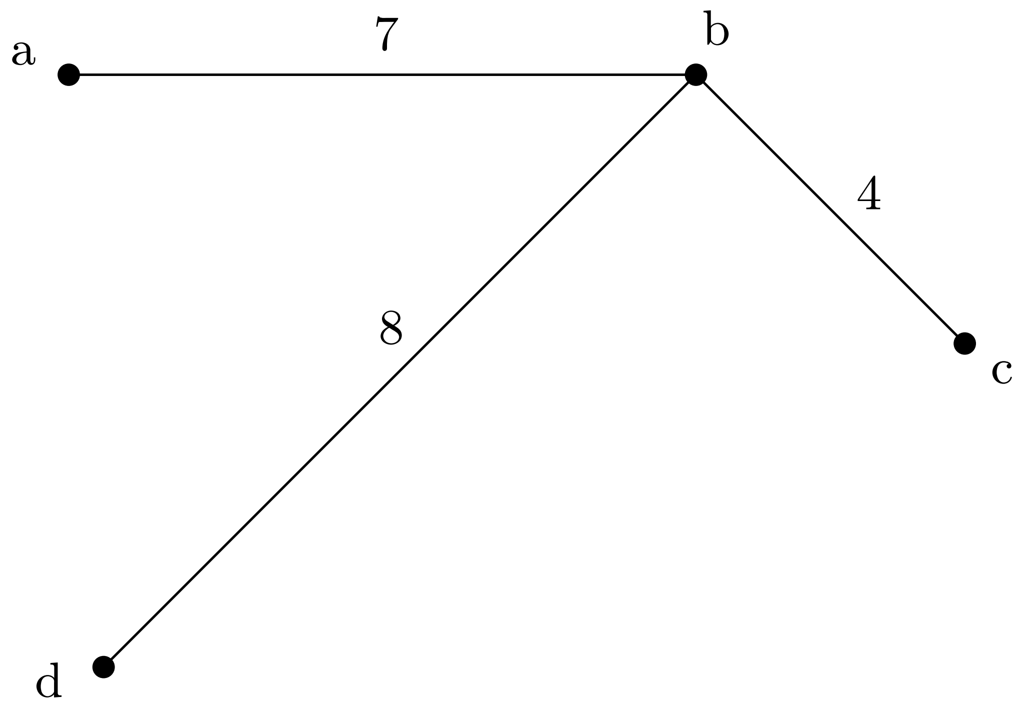
Construct an ultrametric sandwich tree for matrices M^l and M^h:

M^l	a	b	c	d
a	0	5	5	7
b		0	2	4
c			0	8
d				0

M^h	a	b	c	d
a	0	7	8	9
b		0	4	8
c			0	10
d				0

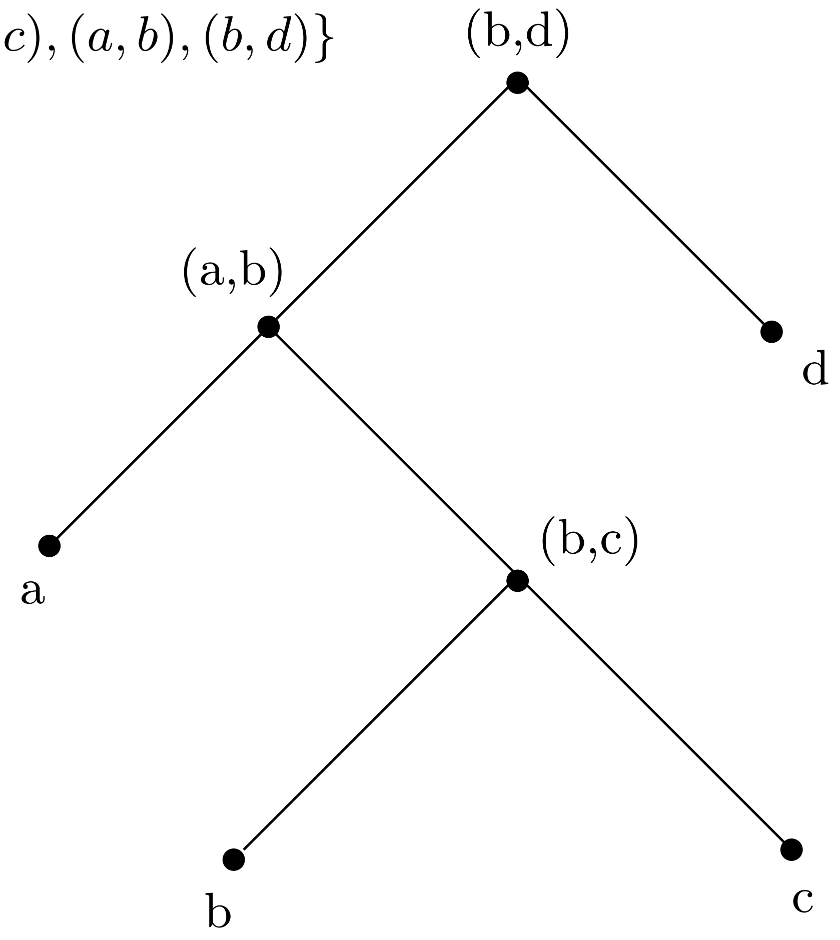
Ultrametric Trees - An Example (cont' d)

1. MST T for G^h:



Ultrametric Trees - An Example (cont' d)

2. Sort edges of T: $\{(b, c), (a, b), (b, d)\}$
   Build tree R:



Ultrametric Trees - An Example (cont' d)

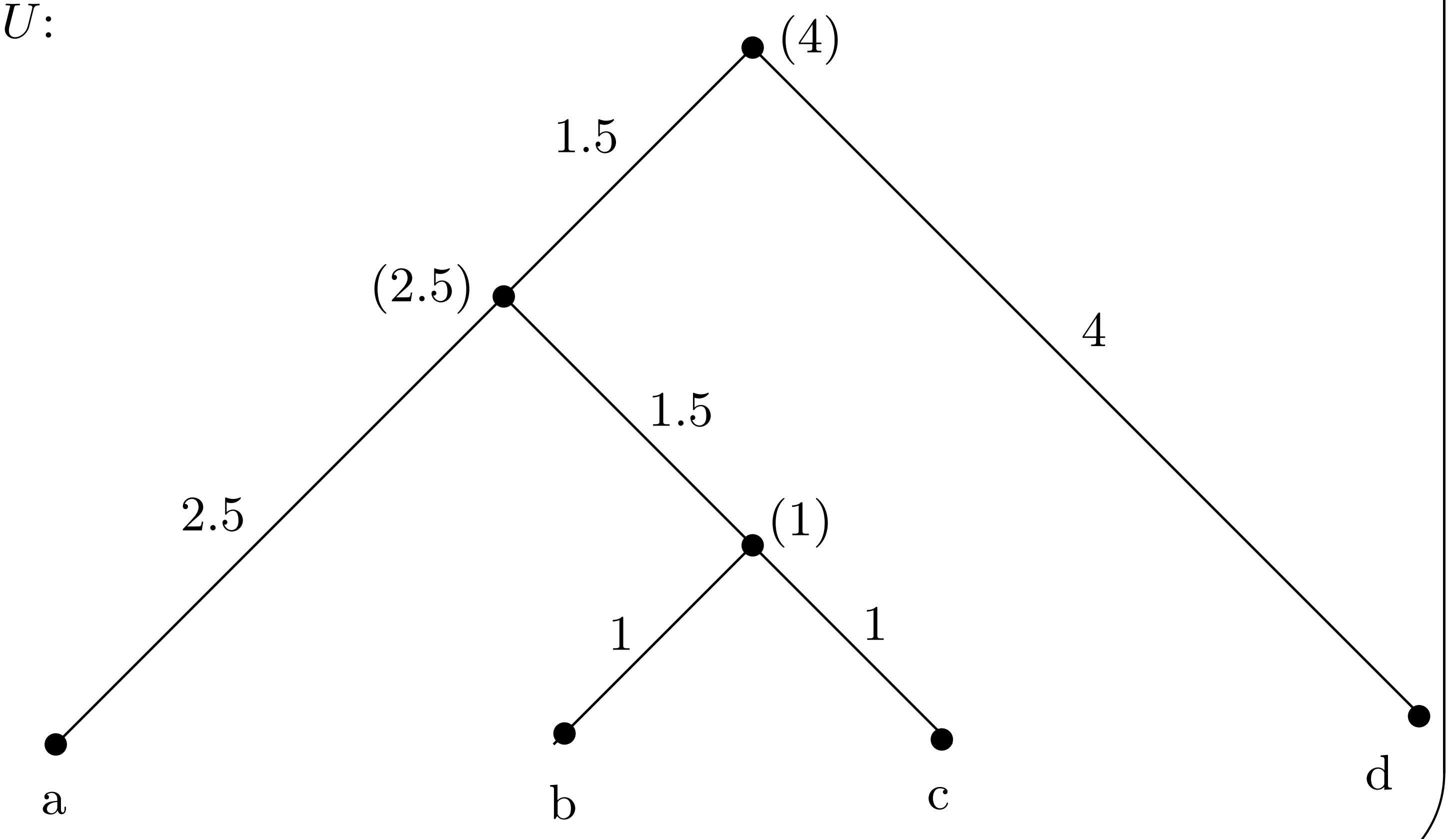
3. Determine cutweights for all edges in T:

(u,v)	$M^l_{u,v}$	$LCA(u, v)$
(a,b)	5	(a,b)
(a,c)	5	(a,b)
(a,d)	7	(b,d)
(b,c)	2	(b,c)
(b,d)	4	(b,d)
(c,d)	8	(b,d)

$$CW(a, b) = 5$$
$$CW(b, c) = 2$$
$$CW(b, d) = 8$$

Ultrametric Trees - An Example (cont' d)

4. Sort edges of T according to CW: $\{(b,c),(a,b),(b,d)\}$
   Final tree U:



Ultrametric Trees (cont' d)

Time analysis of the algorithm:

1. Building T: $O(n^2)$ time (Prim's algorithm with Fibonacci heaps)
2. Sorting: $O(n \log n)$ time since T has n-1 edges.
   Building R: $O(n \cdot \alpha(n, n))$ time (disjoint-set forest data structure)
3. Preprocessing: $O(n)$ time
   Then: $O(n^2)$ time ($O(1)$ time to look up LCA of one pair of objects)
4. Sorting: $O(n \log n)$ time
   Building U: $O(n \log n)$ time

Total running time: $O(n^2)$