Junction Trees
And Belief Propagation

(Slides from Pedro Domingos)

Junction Trees: Motivation

- What if we want to compute all marginals, not just one?
- Doing variable elimination for each one in turn is inefficient
- Solution: Junction trees (a.k.a. join trees, clique trees)

Junction Trees: Basic Idea

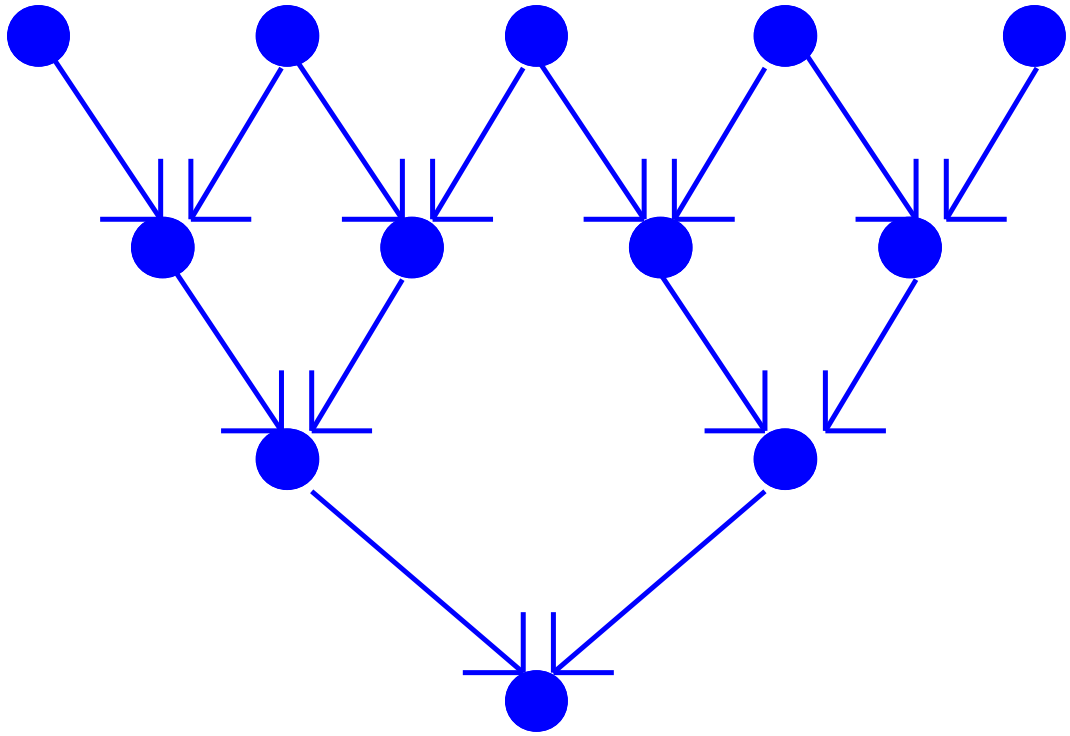
- In HMMs, we efficiently computed all marginals using dynamic programming
- An HMM is a linear chain, but the same method applies if the graph is a tree
- If the graph is not a tree, reduce it to one by clustering variables

The Junction Tree Algorithm

1. Moralize graph (if Bayes net)
2. Remove arrows (if Bayes net)
3. Triangulate graph
4. Build clique graph
5. Build junction tree
6. Choose root
7. Populate cliques
8. Do belief propagation

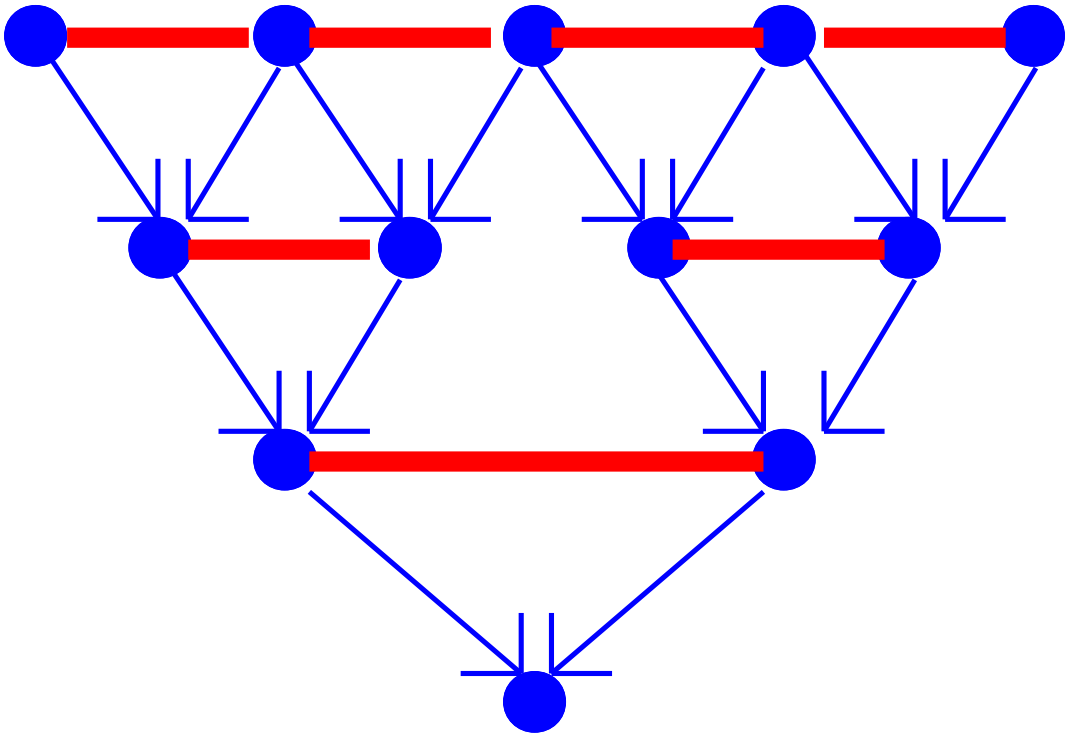
Example

Imagine we start with a Bayes Net having the following structure

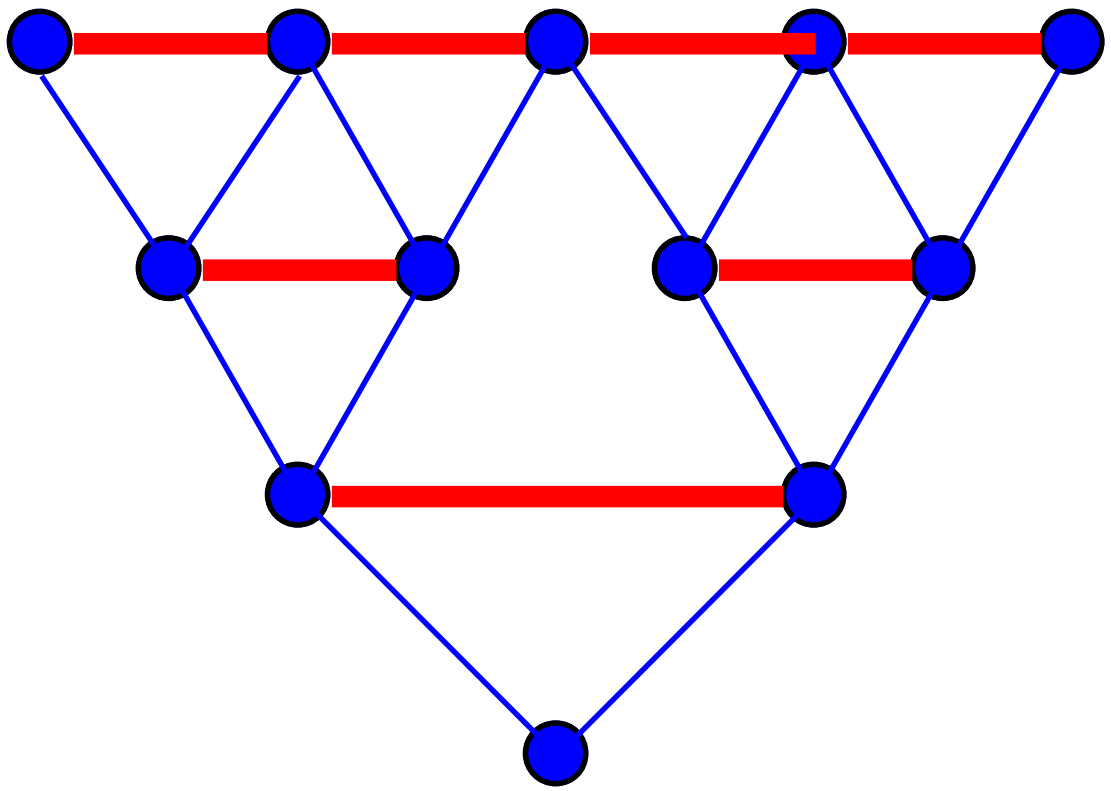


Step 1: Moralize the Graph

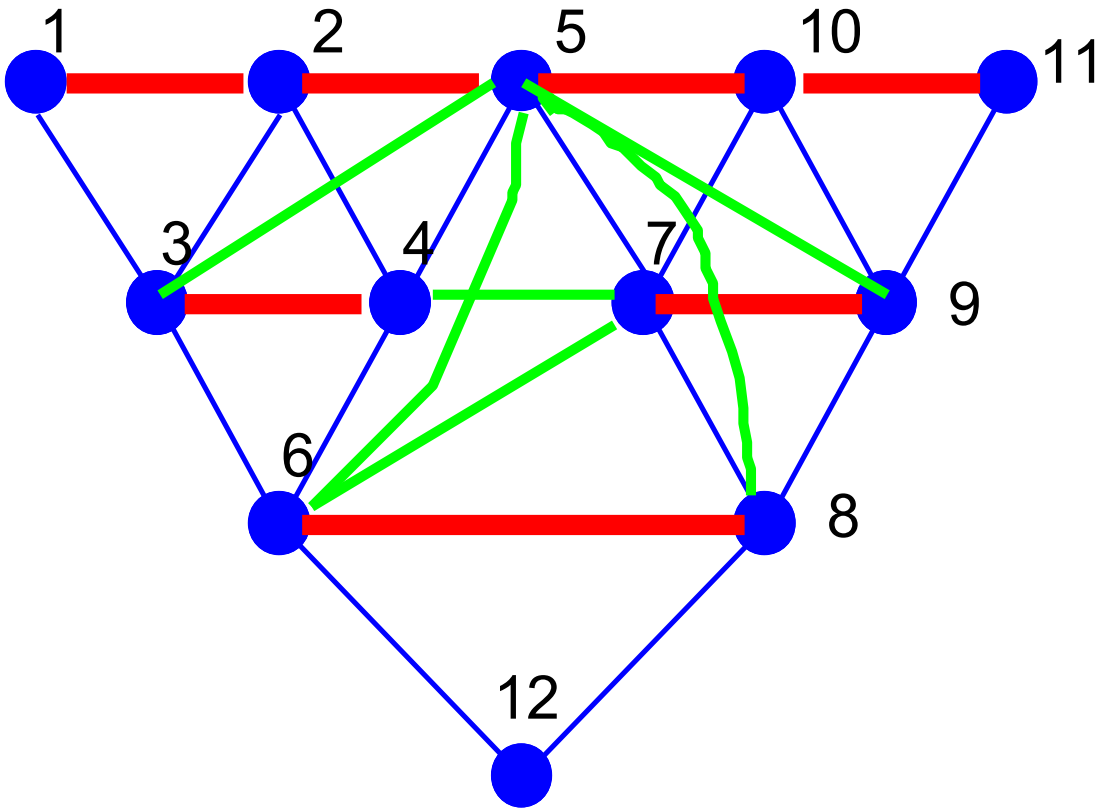
Add an edge between non-adjacent (unmarried) parents of the same child.



Step 2: Remove Arrows

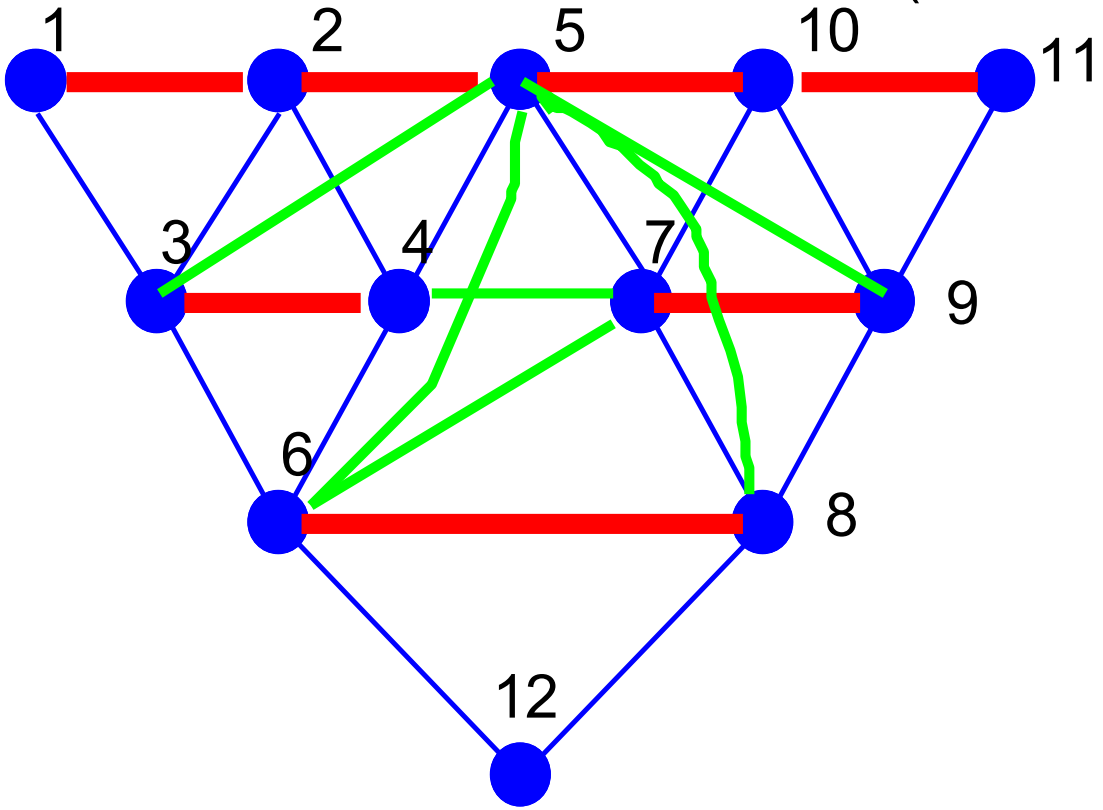


Step 3: Triangulate the Graph



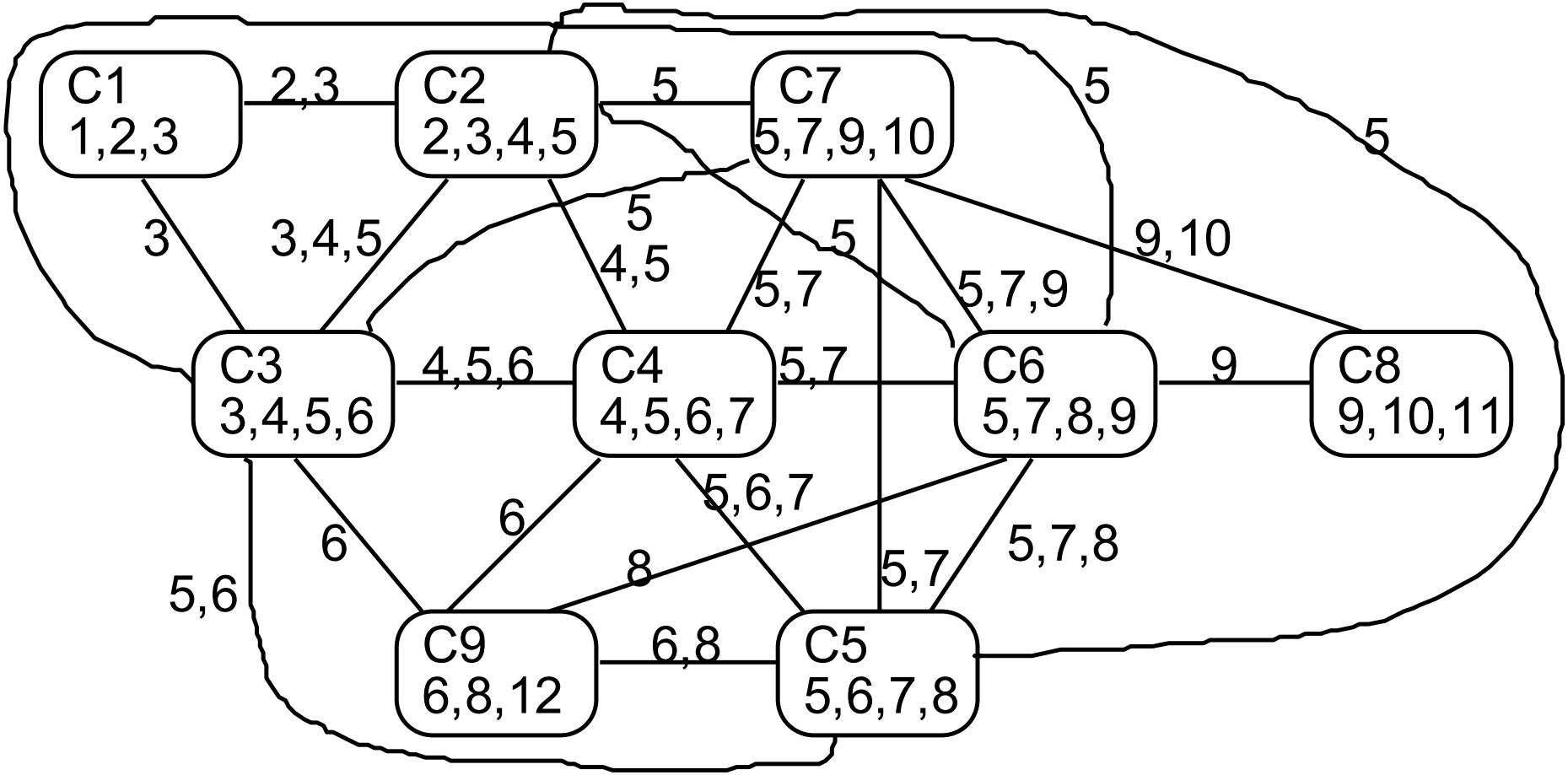
Step 4: Build Clique Graph

Find all cliques in the moralized, triangulated graph. A clique becomes a node in the clique graph. If two cliques intersect below, they are joined in the clique graph by an edge labeled with their intersection from below (shared nodes).



The Clique Graph

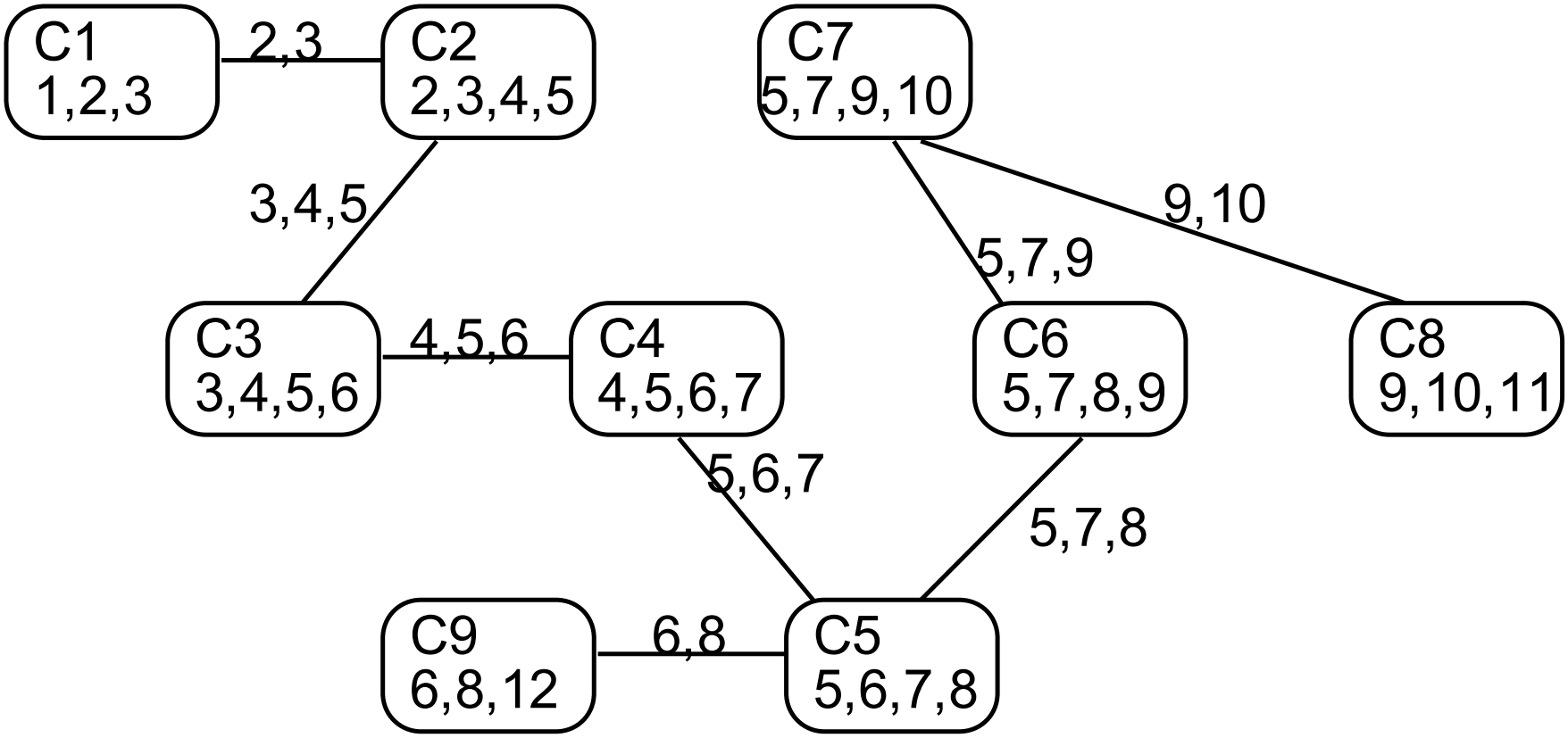
The label of an edge between two cliques is called the separator.



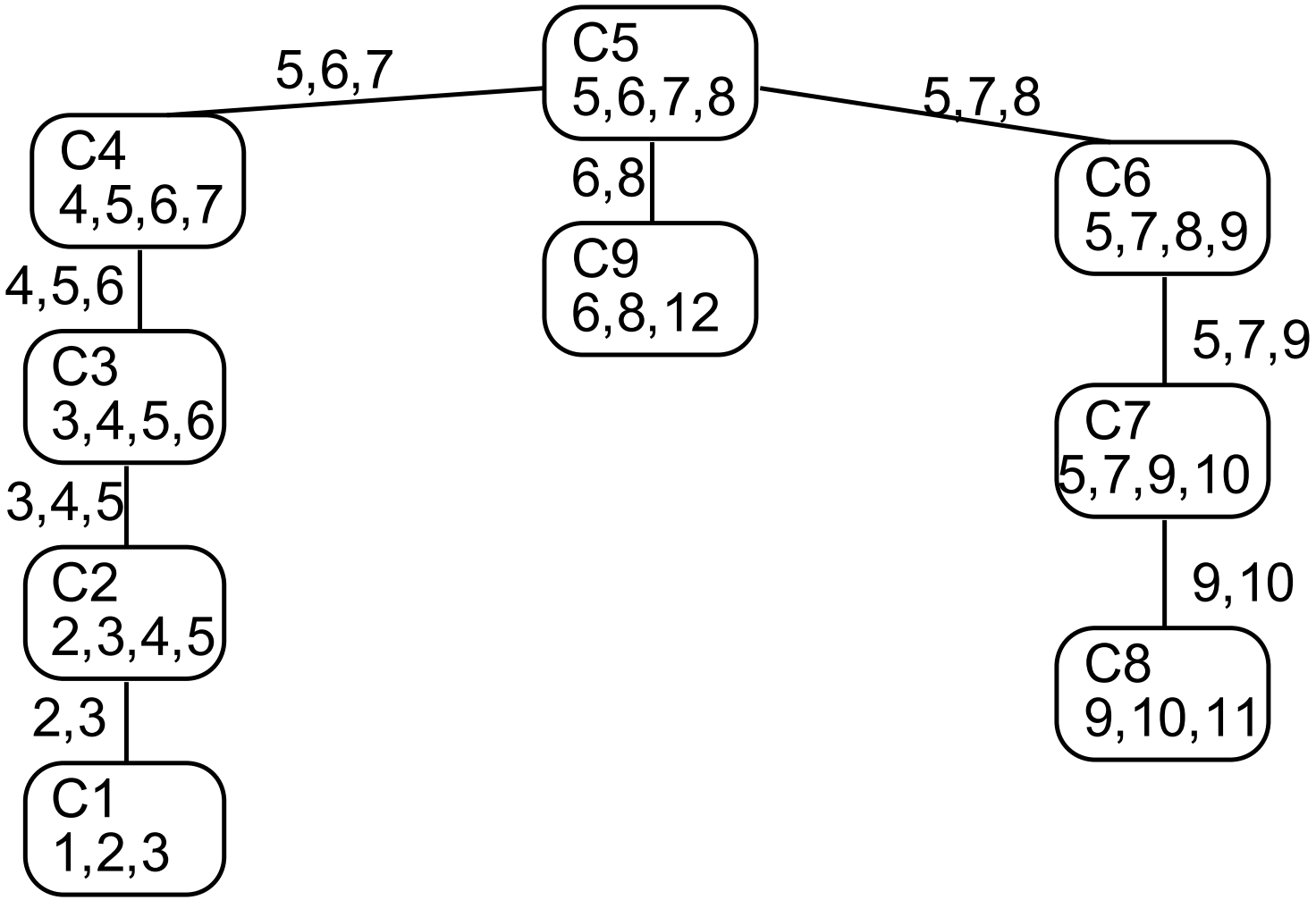
Junction Trees

- A junction tree is a subgraph of the clique graph that:
  1. Is a tree
  2. Contains all the nodes of the clique graph
  3. Satisfies the *running intersection property*.
- *Running intersection property*:
  For each pair *U*, *V* of cliques with intersection *S*, all cliques on the path between *U* and *V* contain *S*.

Step 5: Build the Junction Tree



Step 6: Choose a Root

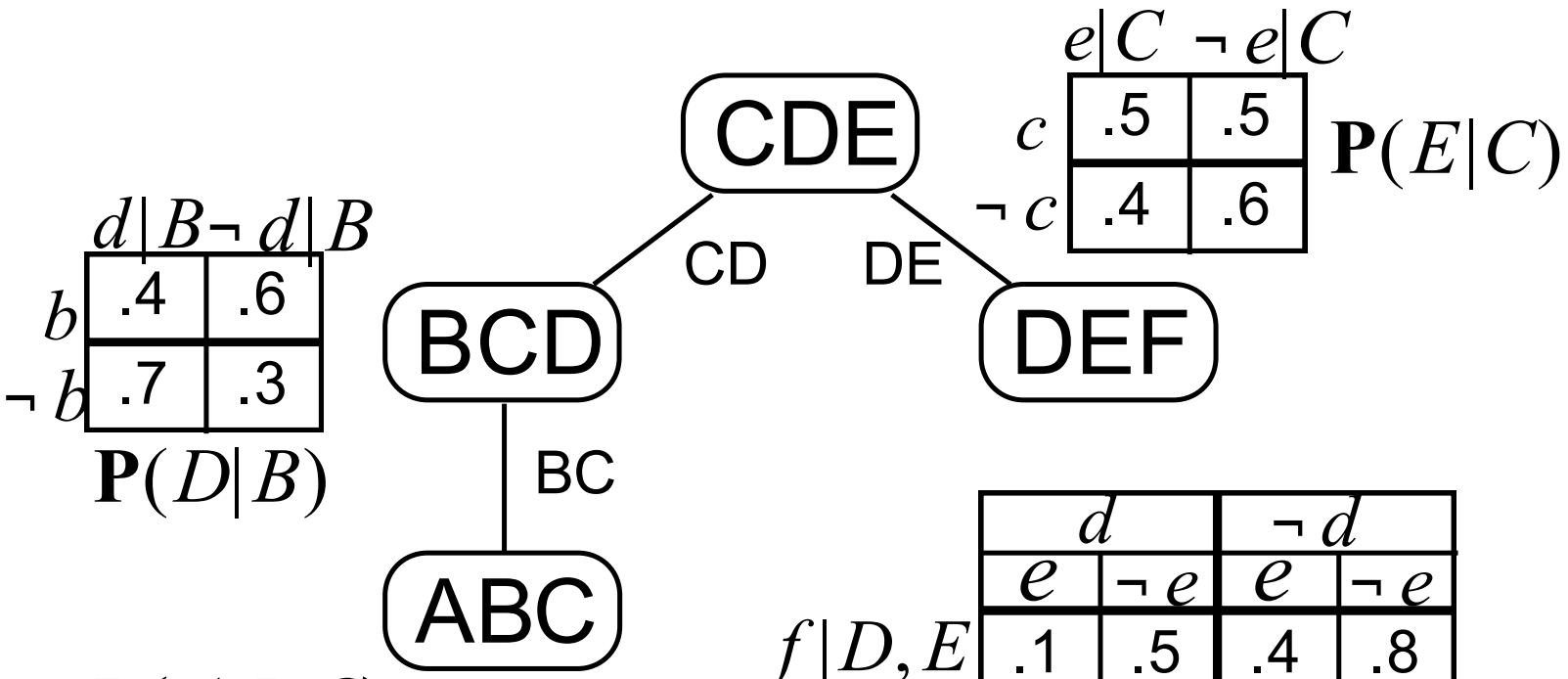


Step 7: Populate the Cliques

- Place each potential from the original network in a clique containing all the variables it references

- For each clique node, form the product of the distributions in it (as in variable elimination).

Step 7: Populate the Cliques



	$e \mid C$	$\neg e \mid C$
c	.5	.5
$\neg c$	.4	.6

$\mathbf{P}(E \mid C)$

	$d \mid B$	$\neg d \mid B$
b	.4	.6
$\neg b$	.7	.3

$\mathbf{P}(D \mid B)$

	d		$\neg d$	
	e	$\neg e$	e	$\neg e$
$f \mid D, E$	.1	.5	.4	.8
$\neg f \mid D, E$	.9	.5	.6	.2

$\mathbf{P}(F \mid D, E)$

$\mathbf{P}(A, B, C)$

	b		$\neg b$	
	c	$\neg c$	c	$\neg c$
a	.007	.003	.063	.027
$\neg a$	.162	.648	.018	.072

Step 8: Belief Propagation

1. Incorporate evidence

2. Upward pass:
   Send messages toward root

3. Downward pass:
   Send messages toward leaves

Step 8.1: Incorporate Evidence

- For each evidence variable, go to *one* table that includes that variable.

- Set to *0* all entries in that table that disagree with the evidence.

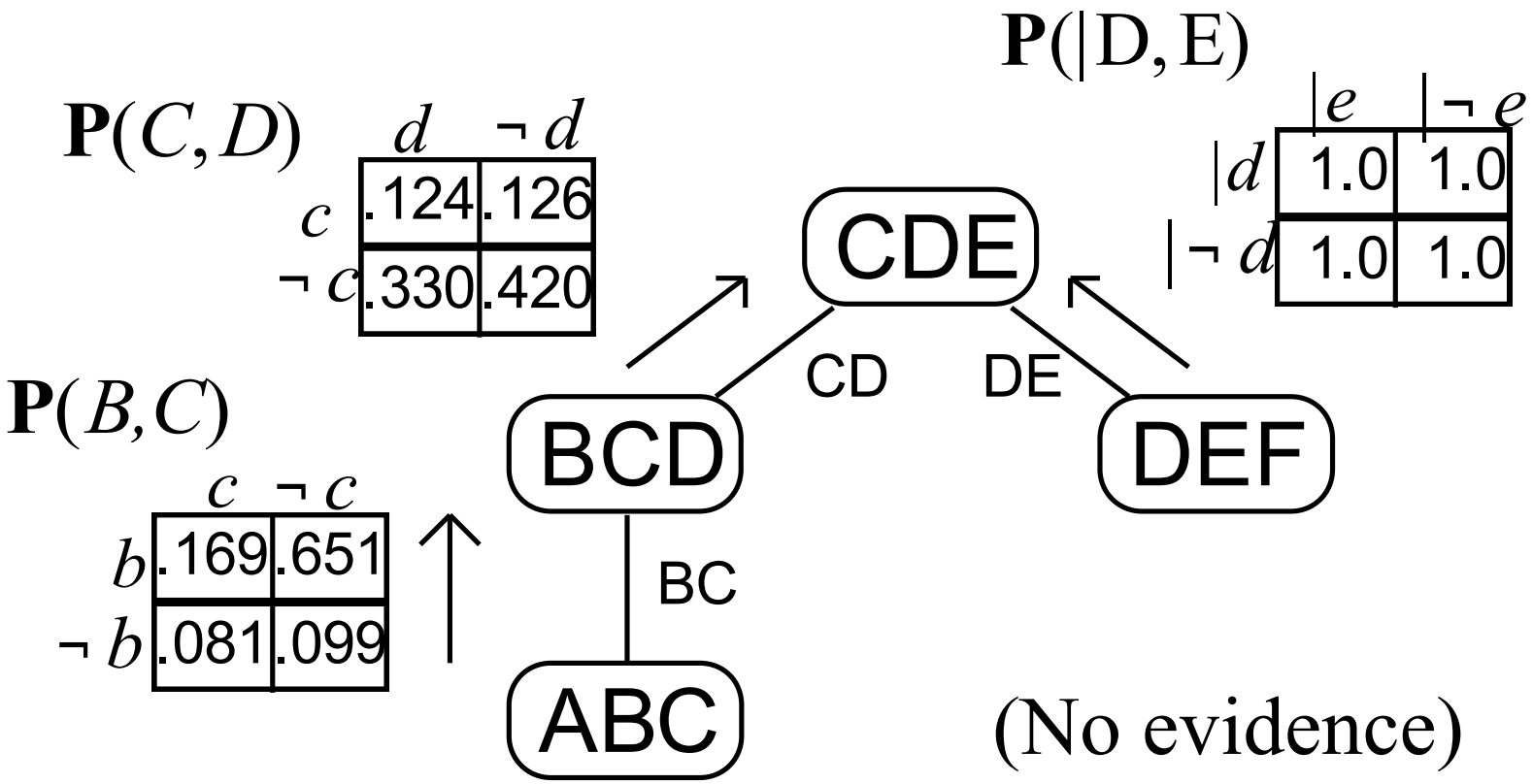
Step 8.2: Upward Pass

- For each leaf in the junction tree, send a message to its parent. The message is the marginal of its table, summing out any variable not in the separator.
- When a parent receives a message from a child, it multiplies its table by the message table to obtain its new table.
- When a parent receives messages from all its children, it repeats the process (acts as a leaf).
- This process continues until the root receives messages from all its children.

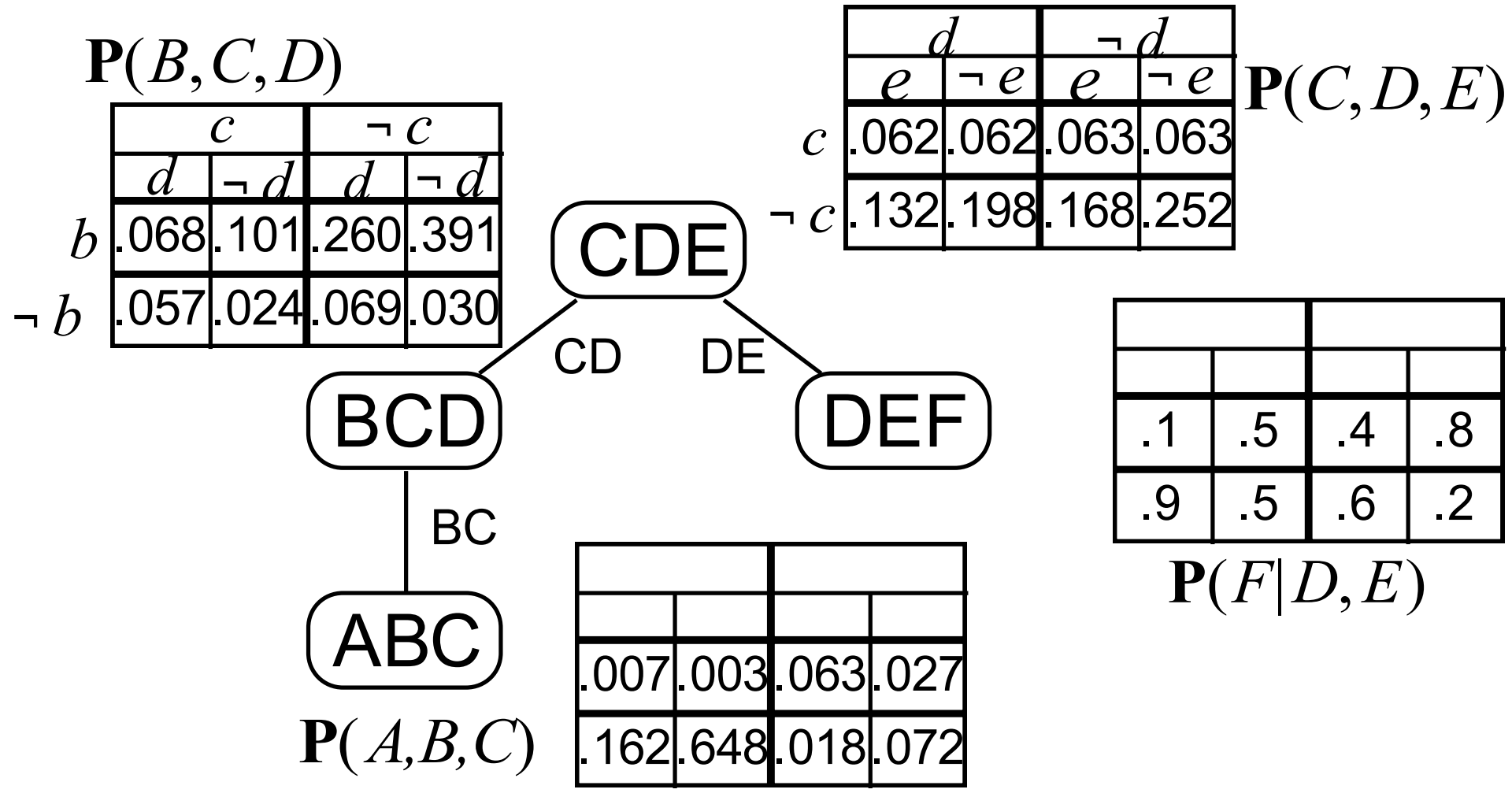
Step 8.3: Downward Pass

- Reverses upward pass, starting at the root.
- The root sends a message to each of its children.
- More specifically, the root divides its current table by the message received from the child, marginalizes the resulting table to the separator, and sends the result to the child.
- Each child multiplies its table by its parent's table and repeats the process (acts as a root) until leaves are reached.
- Table at each clique is joint marginal of its variables; sum out as needed. We're done!

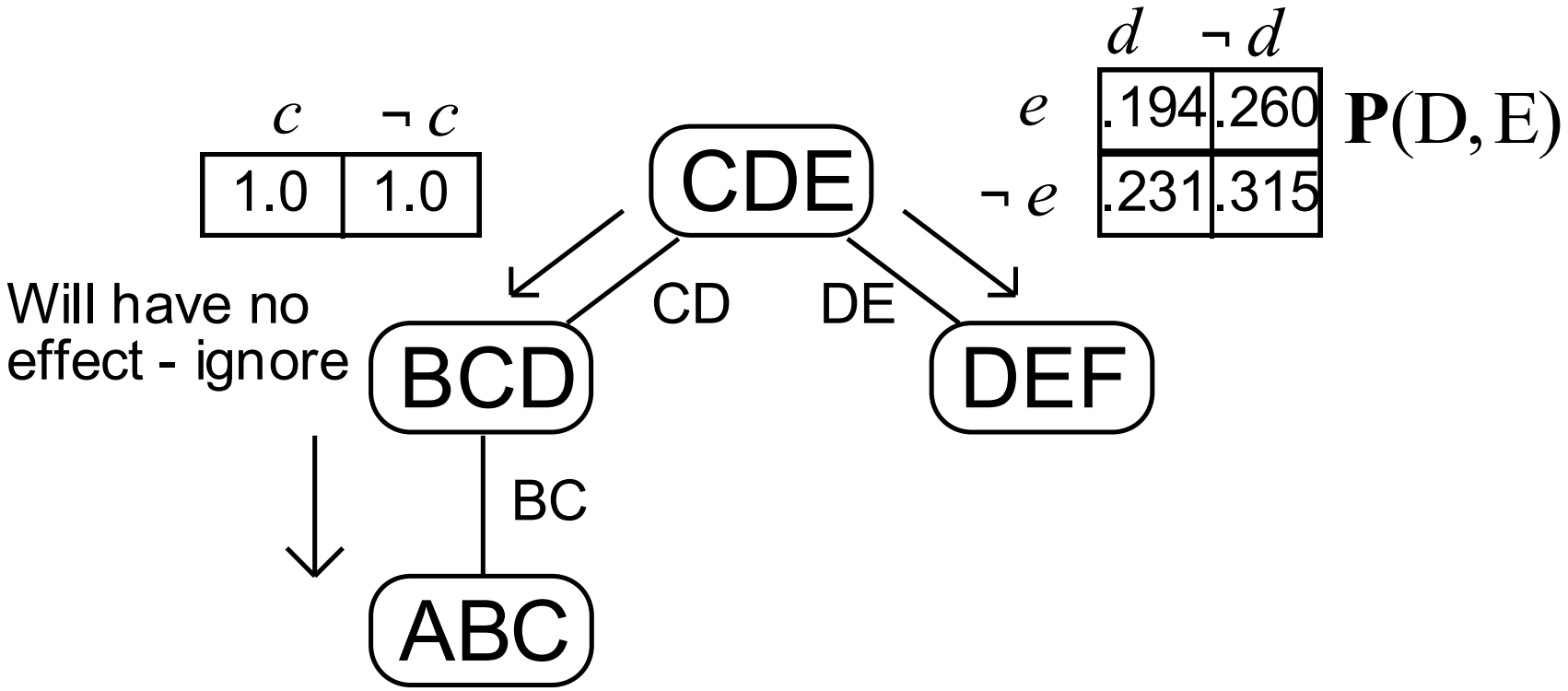
Inference Example: Going Up



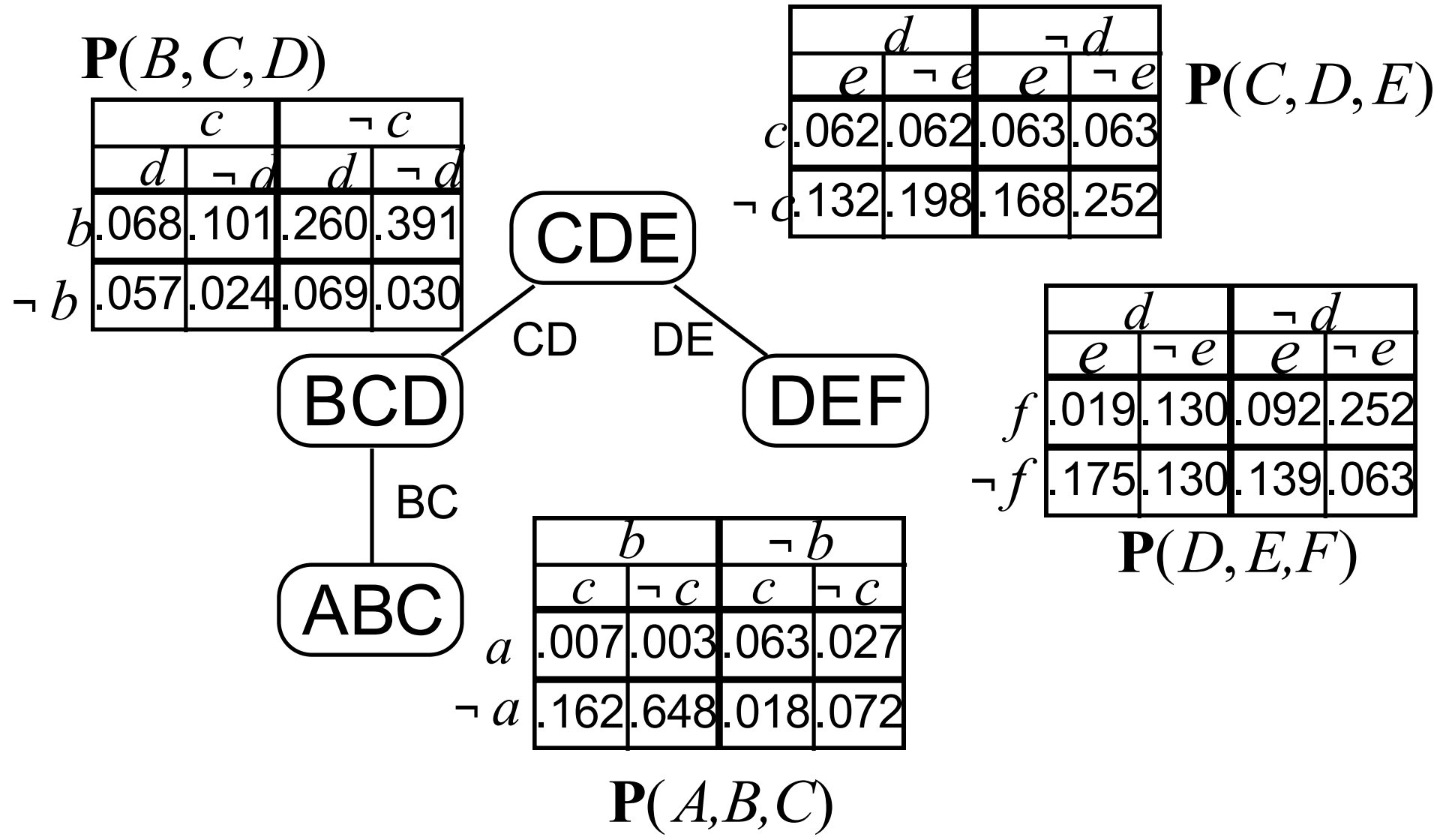
Status After Upward Pass



Going Back Down



Status After Downward Pass



Why Does This Work?

- The junction tree algorithm is just a way to do variable elimination in all directions at once, storing intermediate results at each step.

The Link Between Junction Trees and Variable Elimination

- To eliminate a variable at any step, we combine all remaining tables involving that variable.
- A node in the junction tree corresponds to the variables in one of the tables created during variable elimination (the other variables required to remove a variable).
- An arc in the junction tree shows the flow of data in the elimination computation.

Junction Tree Savings

- Avoids redundancy in repeated variable elimination
- Need to build junction tree only once ever
- Need to repeat belief propagation only when new evidence is received

Exact Inference is Intractable in the worst case

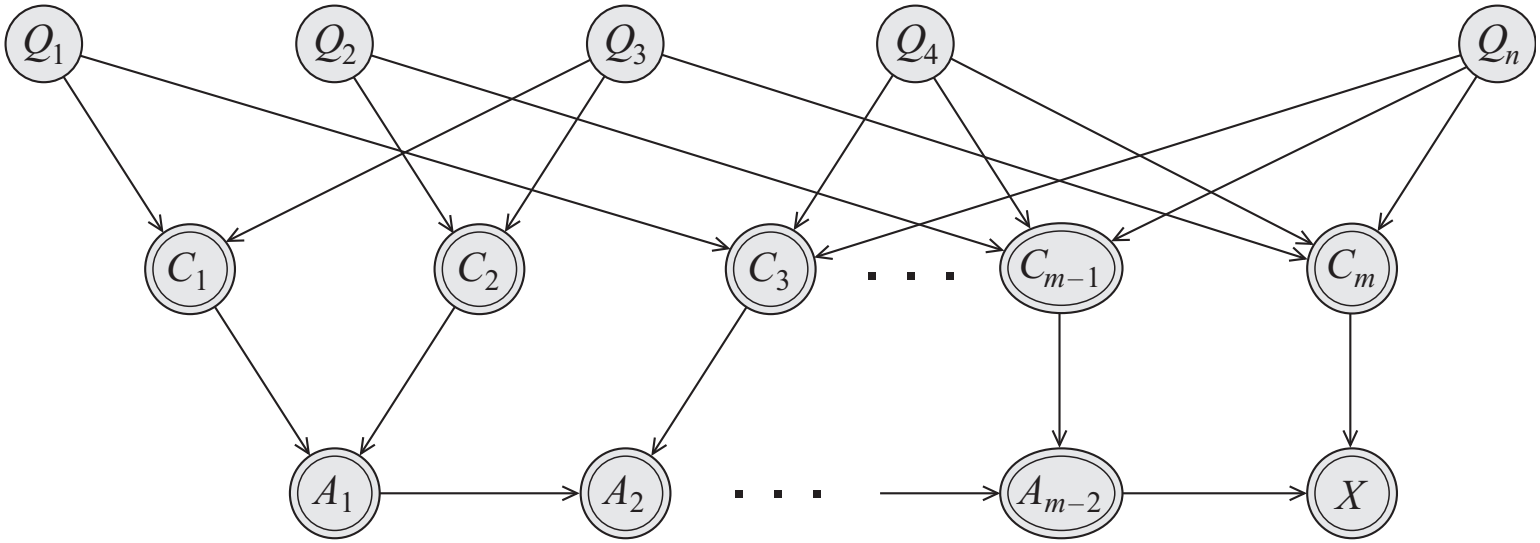
- Exponential in the treewidth of the graph
  - Treewidth can be O(number of nodes) in the worst case…
  - These algorithms can be exponential in the problem size
  - Could there be a better algorithm?

Exact Inference is NP-Hard

- Can encode any 3-SAT problem as a DGM
- Use deterministic CPTs

Exact Inference is NP-Hard (3-SAT)

- Q's are binary random variables
- C's are (deterministic) clauses
- A's are a chain of AND gates



Actually even worse…

- #P complete
- To compute the normalizing constant we have to count the # of satisfying clauses.