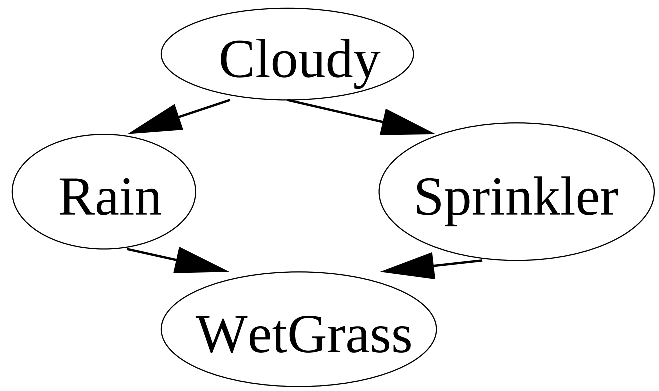
Lecture 6: Inference

- Variable elimination revisited
- More efficient inference
- Coping with loops
  - Clustering
  - Cutset conditioning
- Approximate inference

Recall from last time: Variable elimination

- Variable elimination is a general algorithm for exact inference in Bayes networks
- It is a *dynamic programming algorithm*: it avoid re-computing by storing intermediate results (called *factors*)
- It can be viewed as performing the summation needed to compute a likelihood in an efficient way
- General variable elimination is NP-hard, and the performance depends on the ordering of the nodes
- Good heuristics for ordering nodes exist
- Variable elimination is efficient in polytrees
- Today we look at methods for dealing efficiently with networks that are not polytrees.

Example: Sprinkler network

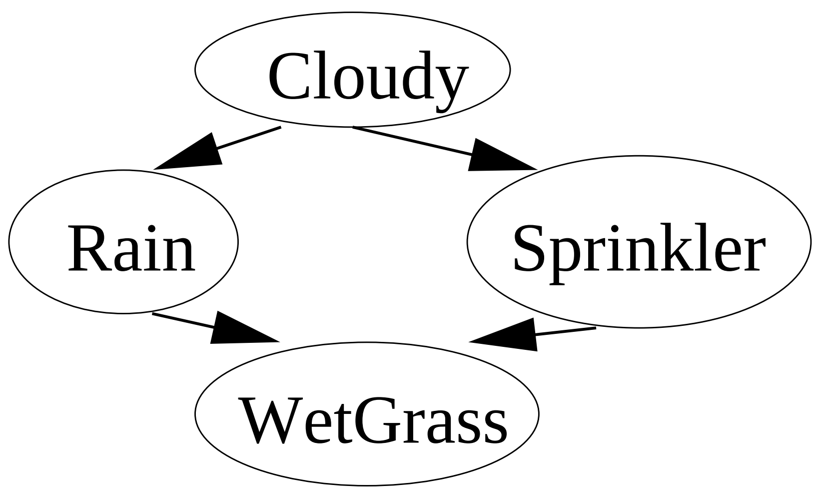


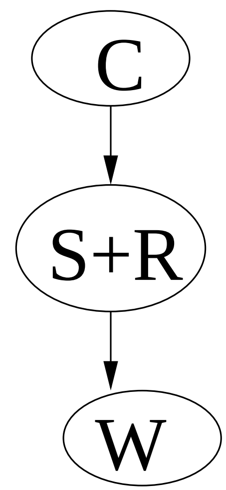
How do we make inference with this network more efficient?

- If it is not a polytree, make it one!
- Pretend the network is indeed a polytree, and use Pearl's belief propagation algorithm (loopy belief propagation)
  Not well-founded theoretically, but VERY successful in practice (e.g. turbo-codes for transmitting information over a noisy channel)
- Approximate the probabilities rather than computing exactly

Creating a polytree

Main idea: take nodes and collapse them together





Ideas for creating polytrees

- Obviously, every network can be collapsed into a polytree with one node, corresponding to the joint distribution

  We would like something more efficient!
- Trivial improvement: we leave the “leaf” nodes alone, collapse all other nodes

  Still can lead to huge tables
- Construct a **join tree** (junction tree, clique tree)

Example: Variable elimination

The initial set of factors is: $P(C), P(R|C), P(S|C), P(W|R,S)$

Suppose that we want to compute $P(W)$ and choose the ordering $\{C, R, S\}$ to eliminate variables

1. Eliminate C: $f_1(R,S) = \sum_C P(C)P(R|C)P(S|C)$

   Now the set of factors remains $P(W|R,S)$, $f_1(R,S)$ (the other factors were used)

2. Eliminate R: $f_2(S,W) = \sum_R f_1(R,S)P(W|R,S)$

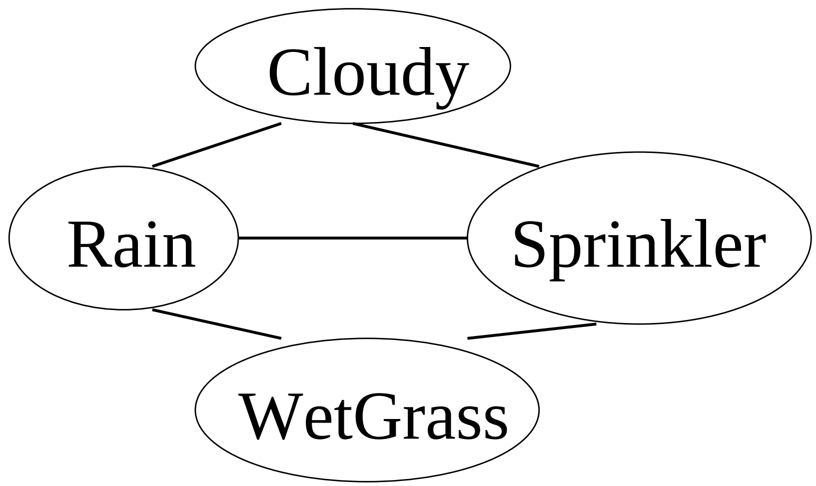
   The set of factors is now $f_2(S,W)$.

3. Eliminate S: $f_3(W) = \sum_S f_2(S,W)$

Convince yourselves that this indeed gives us the correct answer!

What is the induced graph of the network?

Example: Induced graph



In the induced graph, we connect all variables that appear in the same factor in variable elimination. This means that we get at least all edges in the moral graph, and potentially more

The inference process is exponential in the size of the largest clique in the graph (which is 3, in our case).

Cluster tree

Going back to the variable elimination process, consider what happens before we eliminate a variable:

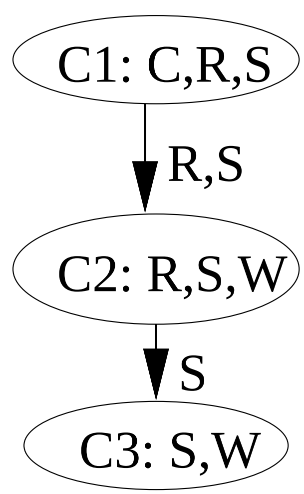
- Each factor contributing to the computation is in some table.
- The “ensemble” of tables is a data structure, associated with a **cluster** of variables.

  E.g. to compute f_1 we need tables involving C, R, S
- Computing a factor involves information from another factor E.g. f_2 uses f_1

Cluster tree

A **cluster tree** (resulting from variable elimination) is a tree where:

- Each node corresponds to a factor from variable elimination.
- We draw an edge from cluster C_i to cluster C_j if factor f_i is used to compute factor f_j. We annotated the edge by the variables present in f_i.



Properties of the cluster tree

- It is **faithful** to the Bayes net, i.e. for every variable X, $\{X\} \cup Parents(X)$ appears as a subset of some cluster
- **Running intersection property**: If X appears in clusters C_i and C_j, it also appears in every cluster on the path between C_i and C_j

There can be several trees with this property! These are called **join trees**, **junction trees** or **clique trees**

A join tree or clique tree can be computed outside variable elimination, just by looking at the graph structure of the network

Constructing a join tree

1. Moralize the graph
2. Triangulate the graph (Pearl, Sec. 3.2.4)
   (a) Find an ordering of the nodes, e.g. through maximum cardinality search
   (b) Starting from n to 1, fill in edges between any two neighbors of the current node that have lower rank

   Finding an optimal triangulation is NP-hard.
3. Find all the cliques in the resulting graph; these will be the vertices of the tree
4. Draw the edges between the vertices in such a way as to enforce the running intersection property

There are efficient algorithms for doing inference on clique trees

Clique trees vs. Variable elimination

- Both use the same kinds of computation
- The overall complexity is the same
- Clique trees are computed **ahead of time** and then just used when we need to do inference

  Hence they require more space, but then we can re-use them a lot
- Inference using clique trees can be **incremental** and **lazy**
- The inference algorithm specific to clique trees is designed to be very efficient on queries involving multiple variables.

Example: Variable elimination with evidence

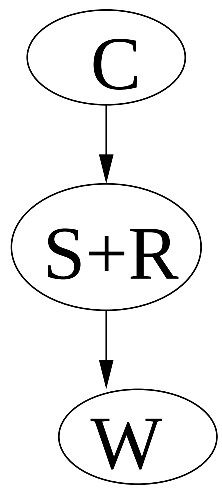
Suppose that we want to compute $P(W|R = t)$. This involves computing $P(R = t, W)$ and then normalizing. So we compute $P(R = t, W)$ using variable elimination. The initial set of factors is:

$$P(C), P(R = t|C), P(S|C), P(W|R = t, S)$$

We choose the ordering $\{C, S\}$ to eliminate variables

1. Eliminate C: $f_1(S) = \sum_C P(C)P(R = t|C)P(S|C)$
   Now the set of factors remains $P(W|R = t, S), f_1(S)$ (the other factors were used)
2. Eliminate S: $f_2(W) = \sum_S P(W|R = t, S)f_1(S)$

Example: Using evidence with the joint tree



We plug in $R = t$ everywhere where R appears

Example: Bad ordering

The initial set of factors is: $P(C), P(R|C), P(S|C), P(W|R,S)$

Suppose that we want to compute $P(W)$ and choose the ordering $\{R, S, C\}$ to eliminate variables

1. Eliminate R: $f'_1(C,S,W) = \sum_R P(R|C)P(W|R,S)$

   Now the set of factors remains $P(C), P(S|C), f'_1(C,S,W)$

   (the other factors were used)
2. Eliminate S: $f'_2(C,W) = \sum_S f'_1(C,S,W)P(S|C)$

   The set of factors is now $f'_2(C,W), P(C)$.
3. Eliminate C: $f_3(W) = \sum_C f'_2(C,W)$

Convince yourselves that this indeed gives us the correct answer again!

What is the induced graph of the network?

Tree induced by bad ordering

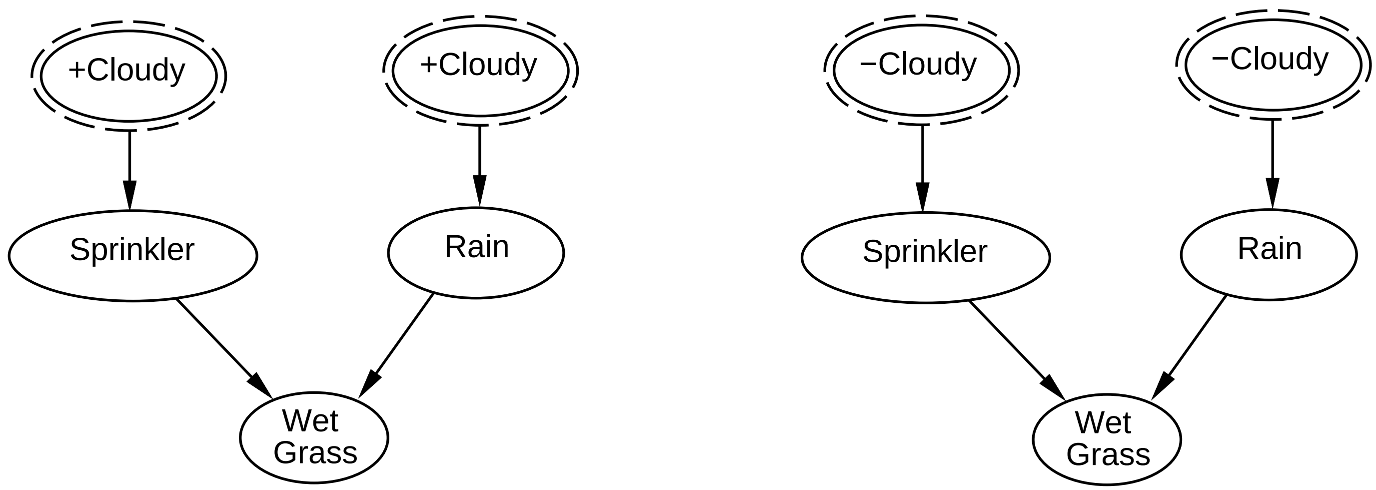
Some of the nodes are really big. This can happen in some networks with good orderings too!

Cutset conditioning

Main idea: Instead of building one polytree with big nodes, we build a few polytrees, but each one is simple (not more complex than the original network)

Cutset conditioning:

- Pick a set of variables that would break the cycles in the network. These form the **cutset**
- Substitute all possible values for each variable
- For each value combination, we get a polytree
- When we have a query, compute the answer in each polytree, then add up the numbers!



Problem: Number of trees is exponential in the size of the cutset!

Bounded cutset conditioning

- Order the trees in the decreasing order of their likelihood
- Evaluate only one of the trees, until satisfied with the results

This is the approach used in the Hugin system (www.hugin.com). It is worth checking it out!

This is an approximation algorithm

Approximation algorithms

- Most of the time, we do not need to know exact probabilities, just rough values
  E.g. When the probabilities are just an intermediate step to making some decision