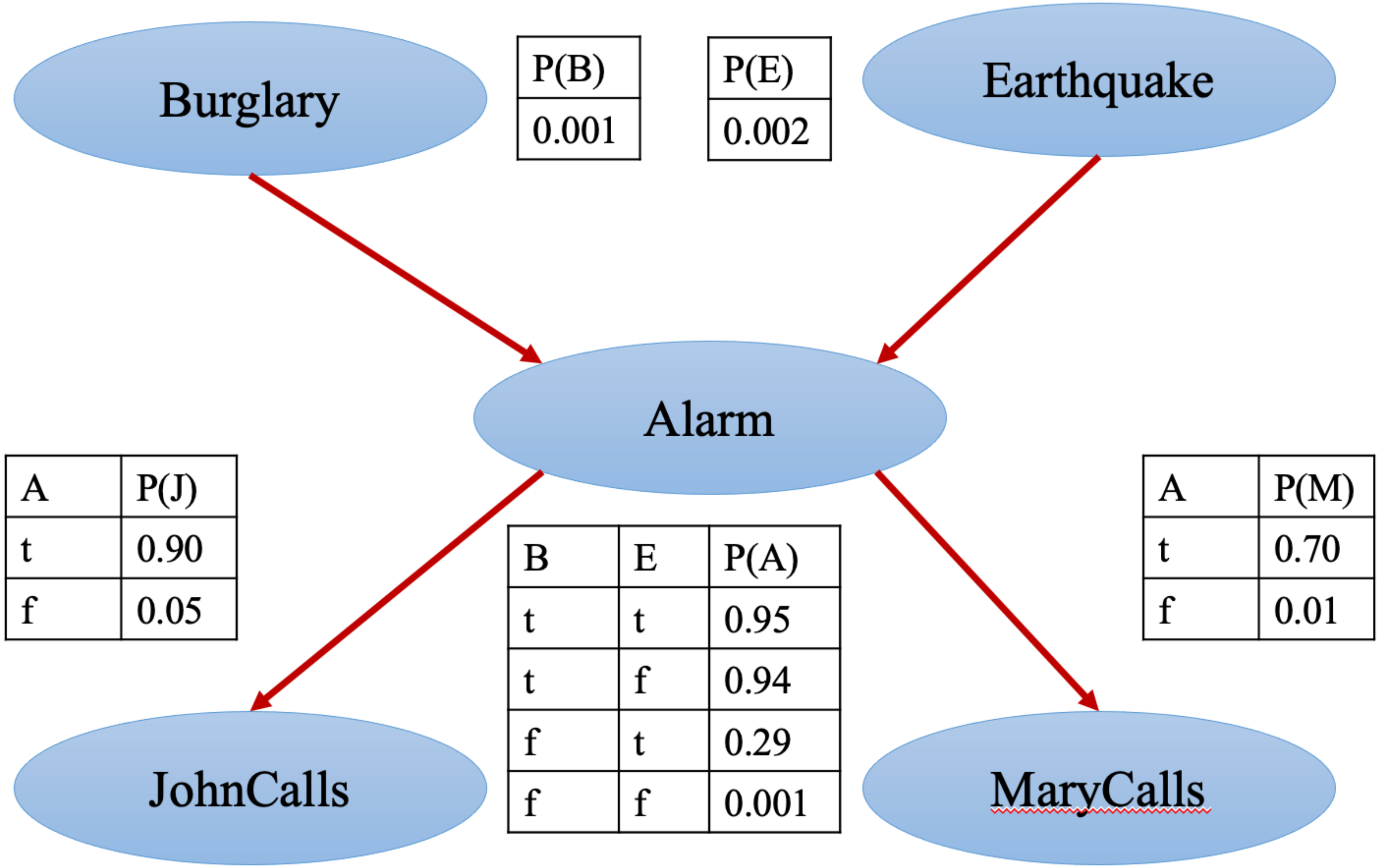
# AIFA: Bayesian Network Inference

19/03/2024

**Koustav Rudra**

# Bayesian Network




| P(B) |
|---|
| 0.001 |

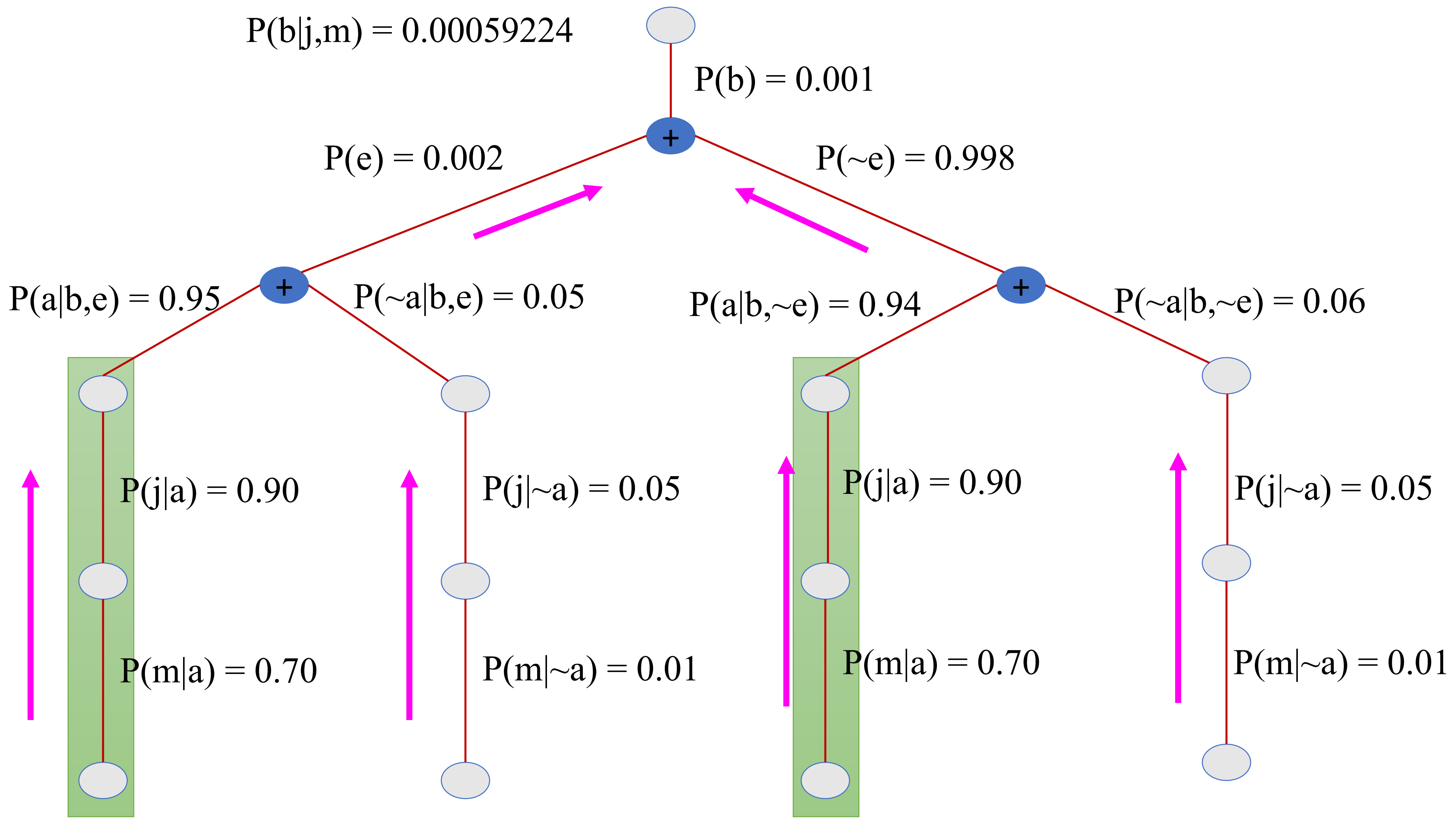
| P(E) |
|---|
| 0.002 |

| B | E | P(A) |
|---|---|---|
| t | t | 0.95 |
| t | f | 0.94 |
| f | t | 0.29 |
| f | f | 0.001 |

| A | P(J) |
|---|---|
| t | 0.90 |
| f | 0.05 |

| A | P(M) |
|---|---|
| t | 0.70 |
| f | 0.01 |

# Bayesian Network: Inference by Enumeration

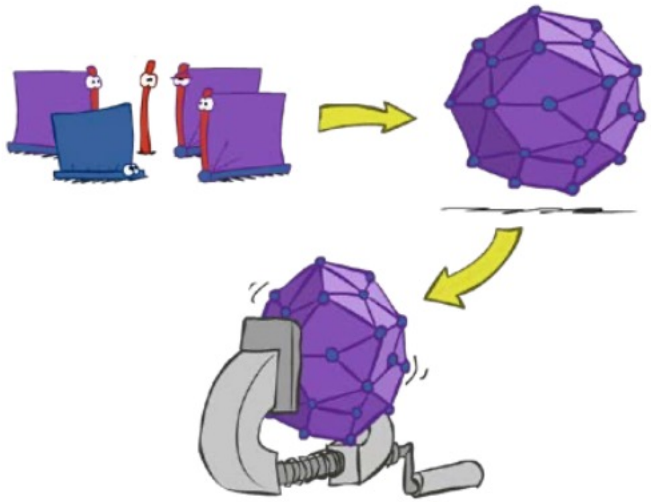


# Inference by Variable Elimination

# Inference by Elimination

- Why is inference by enumeration so slow?

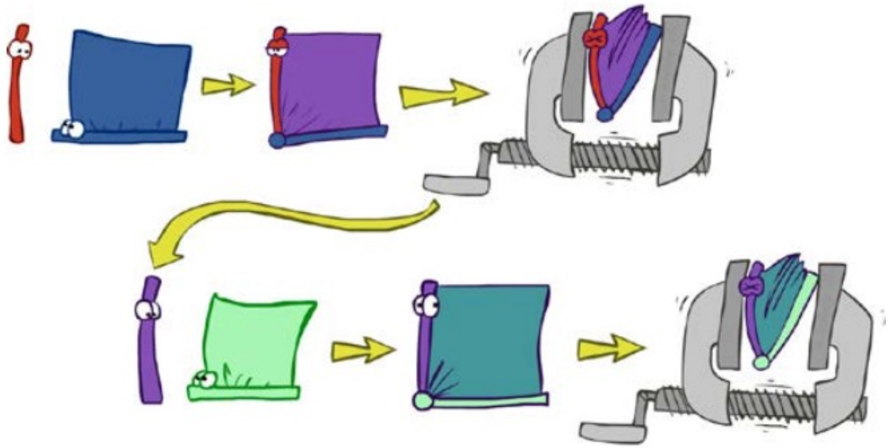
- We join up the whole joint distribution before you sum out the hidden variables



# Variable Elimination

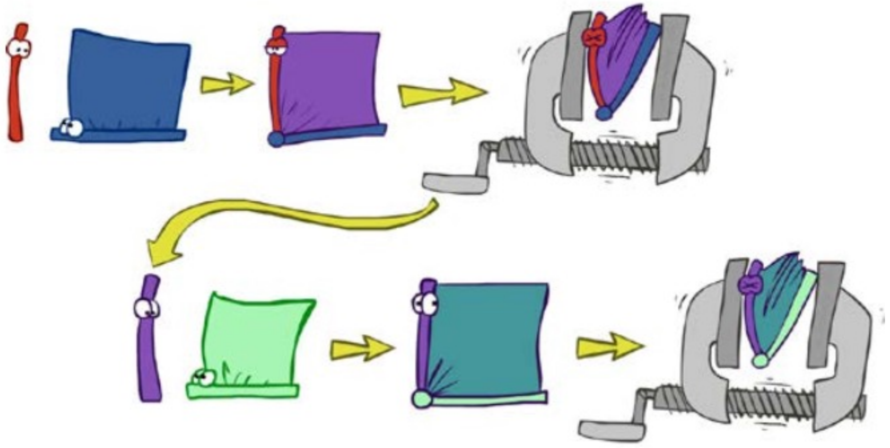
- Idea: interleave joining and marginalizing

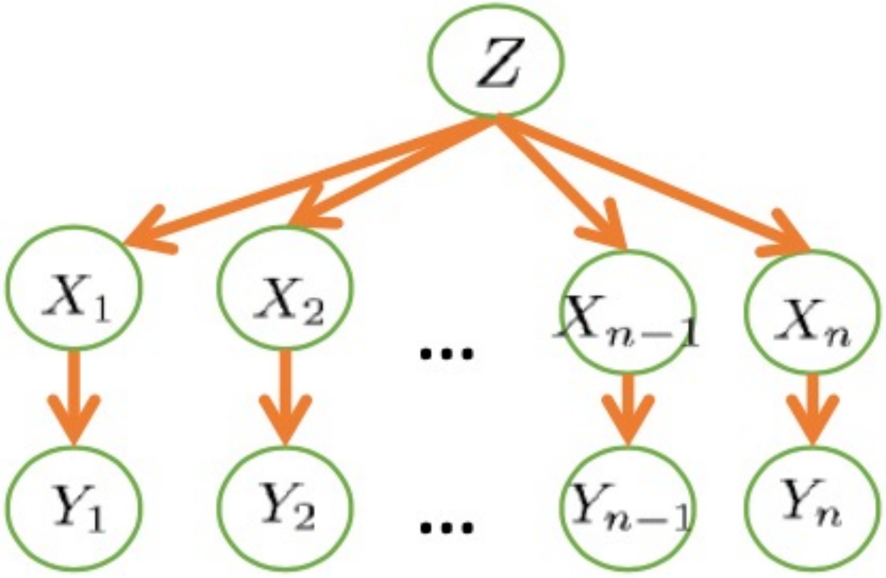
- Called “Variable Elimination”
- Still NP-hard, but usually much faster than inference by enumeration

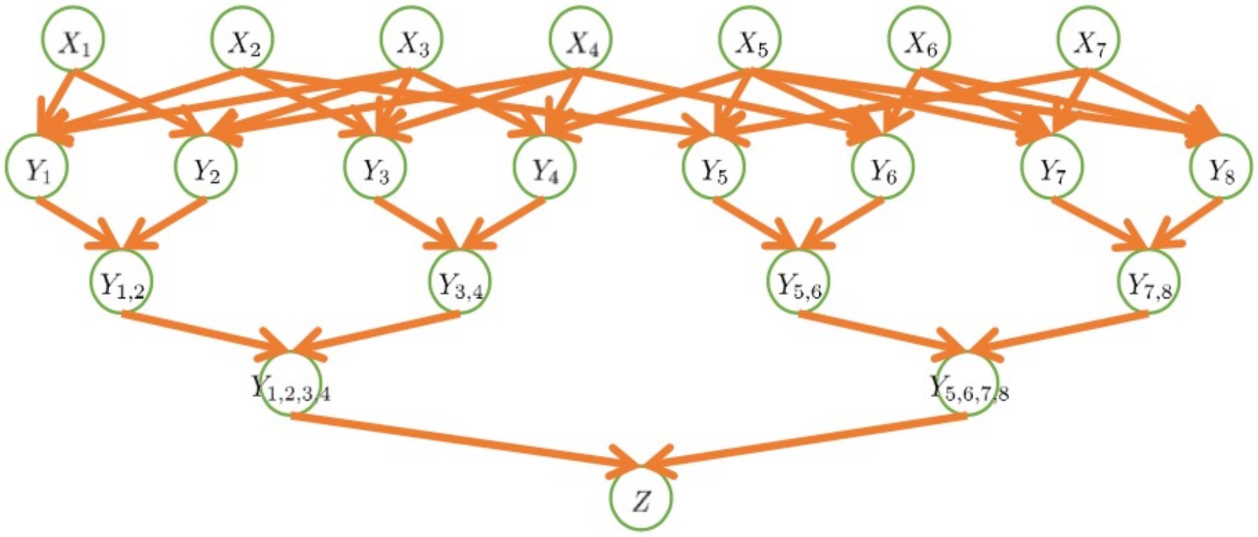


# Variable Elimination

- Interleave joining and marginalizing
- $d^k$ entries computed for a factor over k variables with domain sizes d
- Ordering of elimination of hidden variables can affect size of factors generated
- Worst case: running time exponential in the size of the Bayes' net





# Variable Elimination Algorithm

- Evaluate expressions from right to left
- Summations over each variable are done only for those portions of the expression that depend on the variable
- $P(B|j,m) = \alpha \overset{B}{\boxed{P(B)}} \sum_e \overset{E}{\boxed{P(e)}} \sum_a \overset{A}{\boxed{P(a|B,e)}} \overset{J}{\boxed{P(j|a)}} \overset{M}{\boxed{P(m|a)}}$
- Each part of the expression is annotated with the name of the associated variable
  - These parts are called factors
- $f_M(A) = \begin{pmatrix} P(m|a) \\ P(m|{\sim}a) \end{pmatrix}$
- $f_J(A) = \begin{pmatrix} P(j|a) \\ P(j|{\sim}a) \end{pmatrix}$

# Variable Elimination Algorithm

B E A J M

- $P(B|j,m) = \alpha P(B) \sum_e P(e) \sum_a P(a|B,e)P(j|a)P(m|a)$

- Sum out A from the product of three factors
- $f_{\bar{A}JM}(B,E) = \sum_a f_A(a,B,E) \times f_J(a) \times f_M(a)$
- $f_{\bar{A}JM}(B,E) = f_A(a,B,E) \times f_J(a) \times f_M(a) + f_A(\sim a,B,E) \times f_J(\sim a) \times f_M(\sim a)$
- Pointwise product

- $f_{\bar{E}\bar{A}JM}(B) = f_E(e) \times f_{\bar{A}JM}(B,e) + f_E(\sim e) \times f_{\bar{A}JM}(B,\sim e)$

- $P(B|j,m) = \alpha f_B(B) \times f_{\bar{E}\bar{A}JM}(B)$

# Variable Elimination Algorithm

- Pointwise product of a pair of factors
- Summing out a variable from a product of factors

- Pointwise product of two factors f1 and f2 yield a new factor f
  - Variables of f are union of variables in f1 and f2

| A | B | f1(A,B) |
|---|---|---|
| T | T | 0.3 |
| T | F | 0.7 |
| F | T | 0.9 |
| F | F | 0.1 |

| B | C | f1(B,C) |
|---|---|---|
| T | T | 0.2 |
| T | F | 0.8 |
| F | T | 0.6 |
| F | F | 0.4 |

| A | B | C | f3(A,B, C) |
|---|---|---|---|
| T | T | T | 0.3X0.2 |
| T | T | F | 0.3X0.8 |
| T | F | T | 0.7X0.6 |
| T | F | F | 0.7X0.4 |
| F | T | T | 0.9X0.2 |
| F | T | F | 0.9X0.8 |
| F | F | T | 0.1X0.6 |
| F | F | F | 0.1X0.4 |

# Variable Elimination Algorithm

- Summing out a variable is straight forward
  - Any factor that does not depend on the variable to be summed out can be moved outside the summation process
- $\sum_e f_E(e) \times f_A(A, B, e) \times f_J(A) \times f_M(A) = f_J(A) \times f_M(A) \times \sum_e f_E(e) \times f_A(A, B, e)$

- Pointwise product is computed within the summation
- Variable is summed out of the resulting matrix
- Matrices are not multiplied until we need to sum out a variable

# Variable Elimination Algorithm

- $P(J|b) = \alpha P(b) \sum_e P(e) \sum_a P(a|b,e) P(J|a) \sum_m P(m|a)$

- $\sum_m P(m|a) = 1$

- Every variable that is not an ancestor of a query variable or evidence variable is irrelevant to the query

- Variable elimination algorithm can remove all these variables before evaluating the query

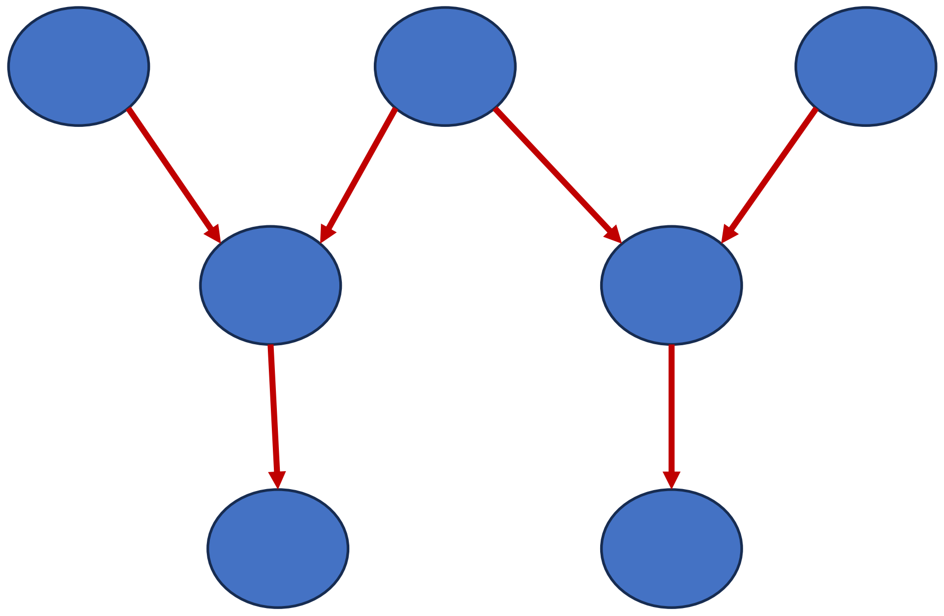
# Variable Elimination Algorithm

- function ELIMINATION-ASK(X,e,bn) returns a distribution over X
  - inputs: X, the query variable
    - e, evidence specified as an event
    - bn, a Bayesian network specifying joint distribution P(X1,X2,…,Xn)

  - factors←[]; vars ←REVERSE(VARS[bn])
  - for each var in vars do
    - factors ←[MAKE-FACTOR(var,e)|factors]
    - if var is a hidden variable then factors ←SUM-OUT(var,factors)
  - return NORMALIZE(POINTWISE-PRODUCT(factors))

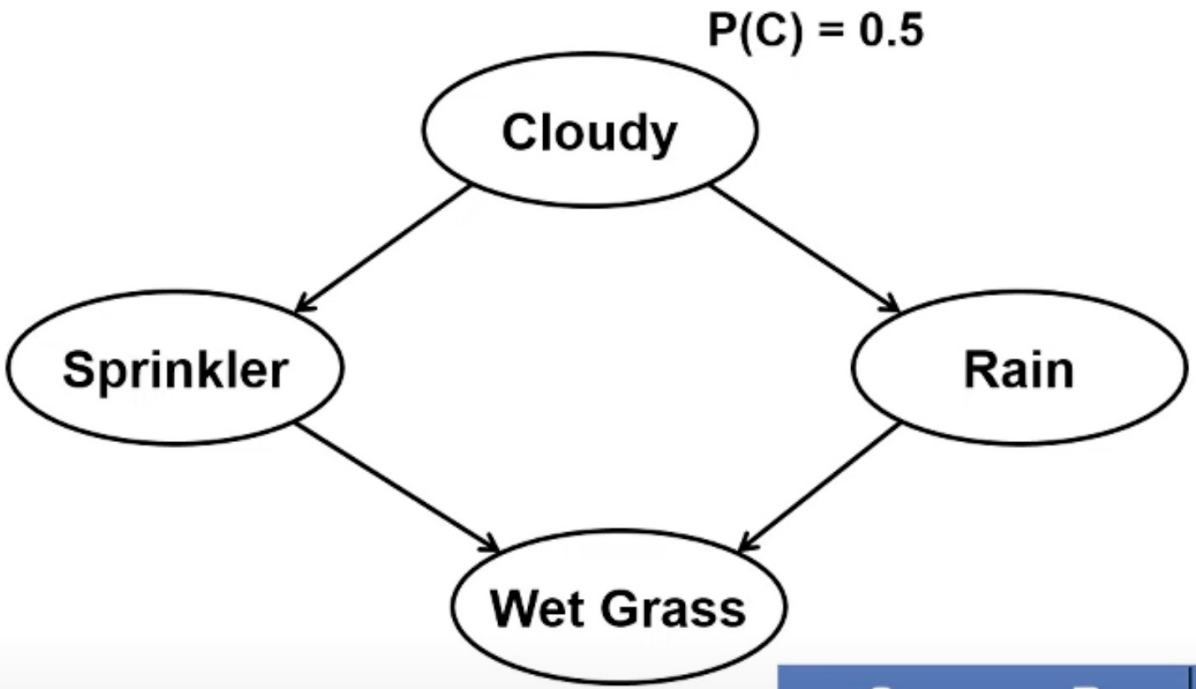
# Answering Queries

- We consider cases where the belief network is a poly tree
  - There is atmost one undirected path between any two nodes



- The time and space complexity of exact inference in polytrees is linear in the size of the network

# Inference in multiply connected Belief Networks



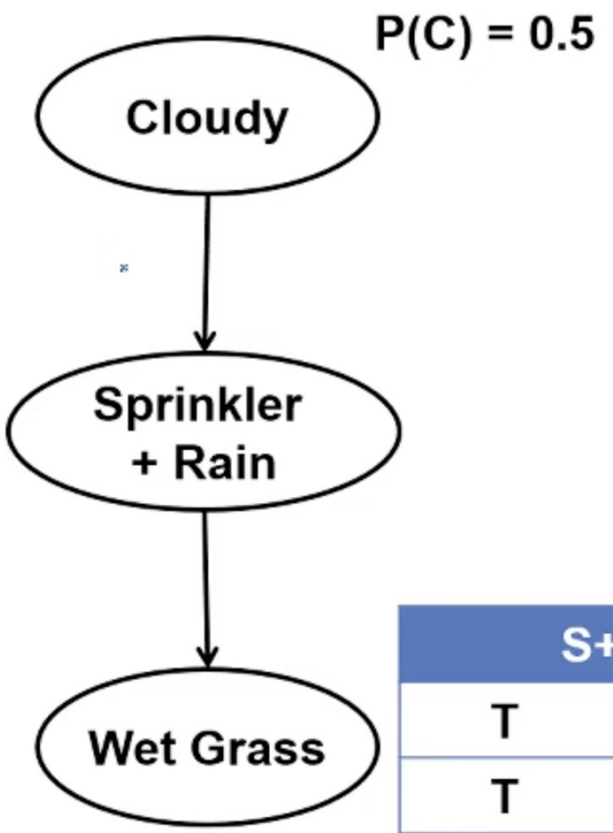

| C | P(S) |
|---|---|
| T | 0.10 |
| F | 0.50 |

| C | P(R) |
|---|---|
| T | 0.80 |
| F | 0.20 |

| S | R | P(W) |
|---|---|---|
| T | T | 0.99 |
| T | F | 0.90 |
| F | T | 0.90 |
| F | F | 0.00 |

# Clustering Methods

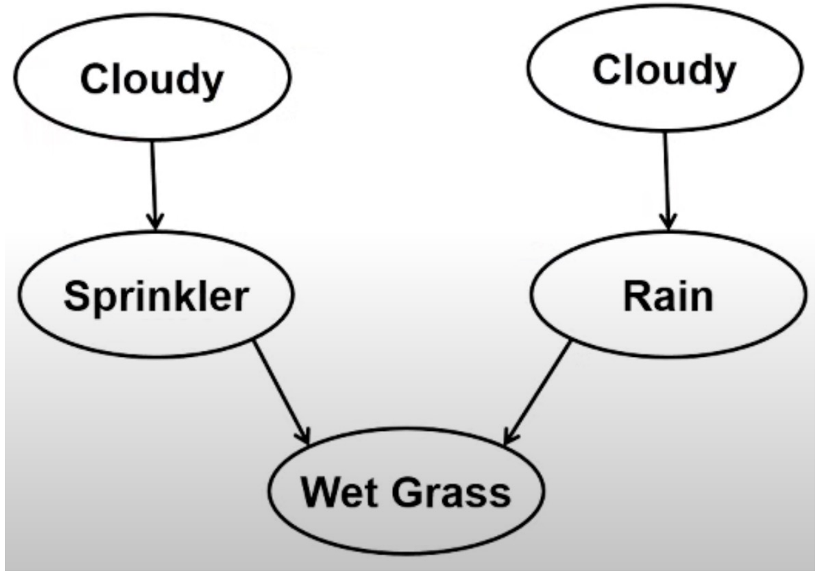
| C | P(S+R = x) | | | |
|---|---|---|---|---|
| | TT | TF | FT | FF |
| T | 0.08 | 0.02 | 0.72 | 0.18 |
| F | 0.40 | 0.10 | 0.40 | 0.10 |

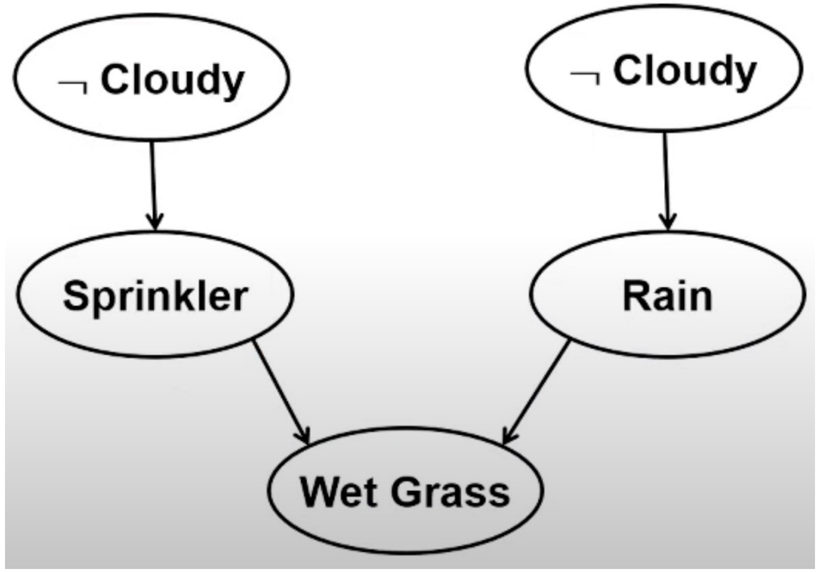



| S+R | | P(W) |
|---|---|---|
| T | T | 0.99 |
| T | F | 0.90 |
| F | T | 0.90 |
| F | F | 0.00 |

# Cutset Conditioning Method

- A set of variables that can be instantiated to yield a poly-tree is called a cutset
- Instantiate the cutset variables to definite values
    - Then evaluate a poly-tree for each possible instantiation



# Stochastic Simulation Methods

- Use the network to generate a large number of concrete models of the domain that are consistent with the network distribution
- They give an approximation of the exact evaluation

# Approximate Inference

# Approximate Inference: Sampling

- Sampling is a lot like repeated simulation
  - Predicting the weather, basketball games, …

- Basic idea
  - Draw N samples from a sampling distribution S
  - Compute an approximate posterior probability
  - Show this converges to the true probability P

Why sample?

- Learning: get samples from a distribution we don't know
- Inference: getting a sample is faster than computing the right answer (e.g. with variable elimination)

# Sampling

- Sampling from given distribution
  - Step 1: Get sample u from uniform distribution over [0, 1)
    - e.g. random() in python
  - Step 2: Convert this sample u into an outcome for the given distribution
    - by having each outcome associated with a sub-interval of [0,1)
    - with sub-interval size equal to probability of the outcome

| C | P(C) |
|---|---|
| red | 0.6 |
| green | 0.1 |
| blue | 0.3 |

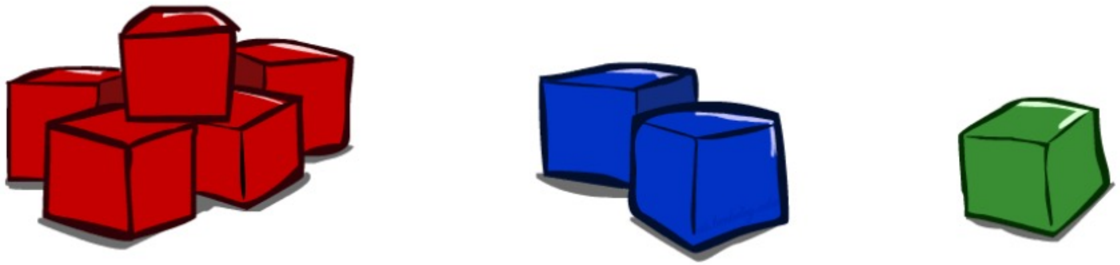
$$0 \leq u < 0.6 \rightarrow C = red$$

$$0.6 \leq u < 0.7 \rightarrow C = blue$$

$$0.7 \leq u < 1 \rightarrow C = red$$

If random() returns u=0.83
Our sample is C = blue

- E.g, after sampling 8 times:



# Thank You