Part 8: Bayesian Networks

8.a Representation and Semantics

Univ.-Prof. Dr. Gerhard Widmer
Department of Computational Perception
Johannes Kepler University Linz

gerhard.widmer@jku.at
http://www.cp.jku.at/people/widmer

Reverend Thomas Bayes (1702-1761)

[from the *Encyclopaedia Britannica*:]

Bayes, Thomas (b. 1702, London - d. 1761, Tunbridge Wells, Kent), mathematician who first used probability inductively and established a mathematical basis for probability inference (a means of calculating, from the number of times an event has occurred, the probability that it will occur in future trials).
He set down his findings on probability in "Essay Towards Solving a Problem in the Doctrine of Chances" (1763), published posthumously in the *Philosophical Transactions of the Royal Society of London.*
The only works he is known to have published in his lifetime are *Divine Benevolence, or an Attempt to Prove That the Principal End of the Divine Providence and Government is the Happiness of His Creatures* (1731) and *An Introduction to the Doctrine of Fluxions, and a Defence of the Mathematicians Against the Objections of the Author of the Analyst* (1736) which countered attacks by Bishop Berkeley on the logical foundations of Newton's calculus.

Overview

Probabilistic Queries

Remember:

- Every probabilistic query of the form $\mathbf{P}(\mathbf{X} \mid \mathbf{E}=\mathbf{e})$ can be answered from the *full joint probability distribution* over all domain variables (via Inference by Enumeration)
- BUT (1): Size of full joint is exponential in number of domain variables
- BUT (2): Difficult for system designer to specify probabilities for an exponential number of highly specific atomic events
- Luckily: complexity can be reduced dramatically by independence and conditional independence relationships

In this lecture: Bayesian Networks

- Effective, principled way of
  - representing (in)dependencies among variables
  - reconstructing the full joint probability distribution
  - answering arbitrary probabilistic queries

  ➔ The BN is our agent's **probabilistic knowledge base!** (BNs are a "*factorised representation*" of the full joint distribution)

Bayesian Networks

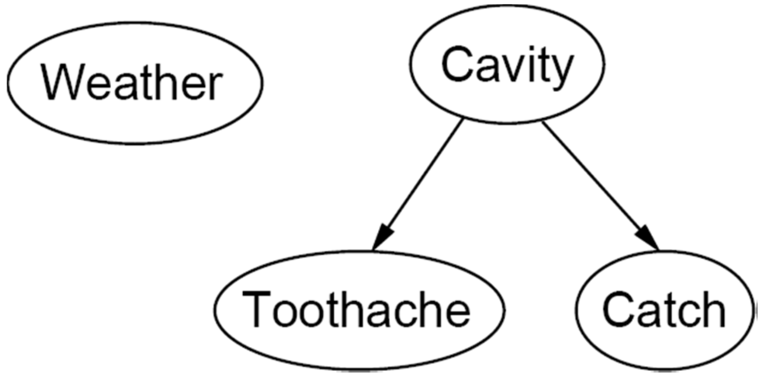
A Bayesian Network

- is a directed acyclic graph (DAG)
- Each node is a random variable
- Some nodes are connected via directed links (arrows)
- If there is an arrow from node X to node Y, we say that X *is a parent of* Y
- Each node X_i has a conditional probability distribution $\mathbf{P}(X_i \mid Parents(X_i))$ associated with it
  (in the simplest case: in the form of a *conditional probability table*)
- The graph has no directed cycles

Intuitive interpretation:

- An arrow from X to Y means that X has a direct influence on the probability of Y (or that knowing X gives us information about Y)

A Simple Example



- *Cavity* ~~causes~~ influences the probability of both *Toothache* and *Catch*
- *Toothache* and *Catch* are *conditionally* independent, given *Cavity*
- *Weather* is independent of all the other variables

A Slightly More Complex Example

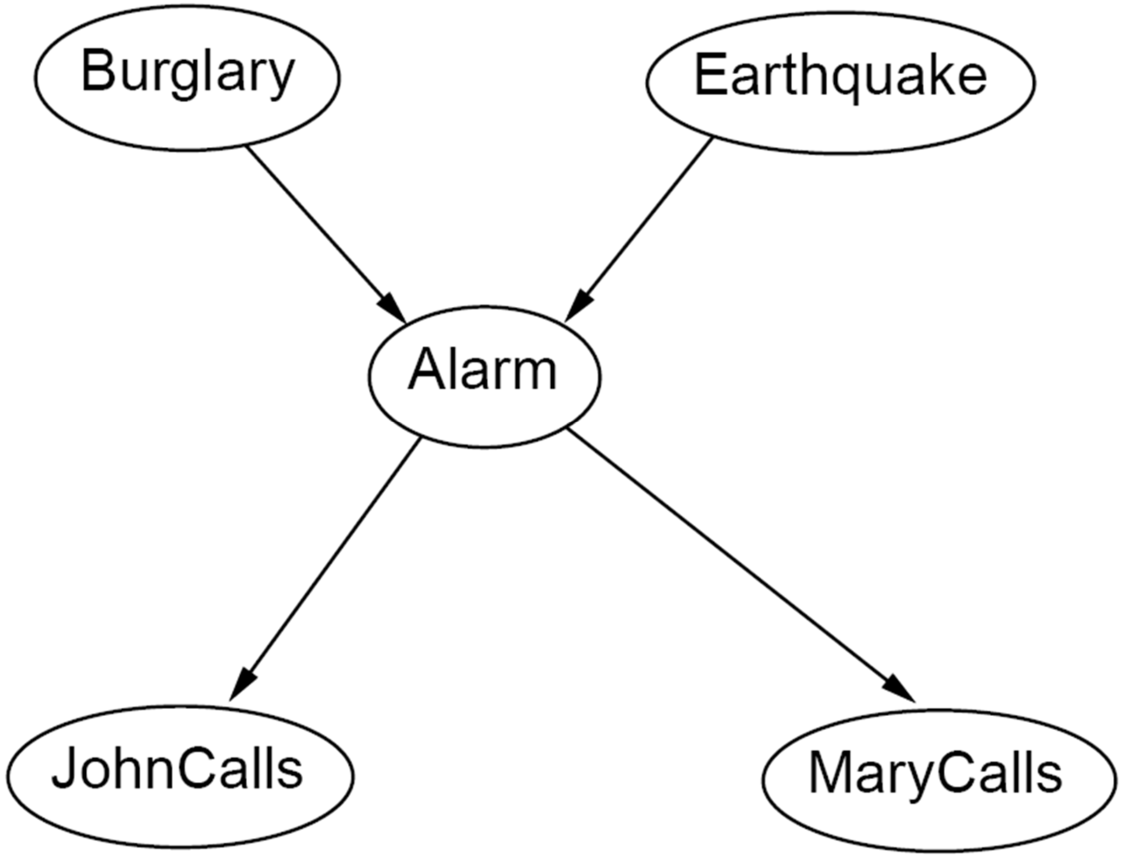
"You have a new **burglar alarm** installed at home. It is fairly reliable at detecting a **burglary**, but also sometimes responds to minor **earthquakes**. You also have two neighbours, **John** and **Mary**, who have promised to call you at work when they hear the alarm.
John always calls when he hears the alarm, but sometimes confuses the telephone ringing with the alarm and calls then, too.
Mary, on the other hand, likes rather loud music and sometimes misses the alarm."

(*Russell & Norvig, ch. 14*)

Typical queries that might be of interest:

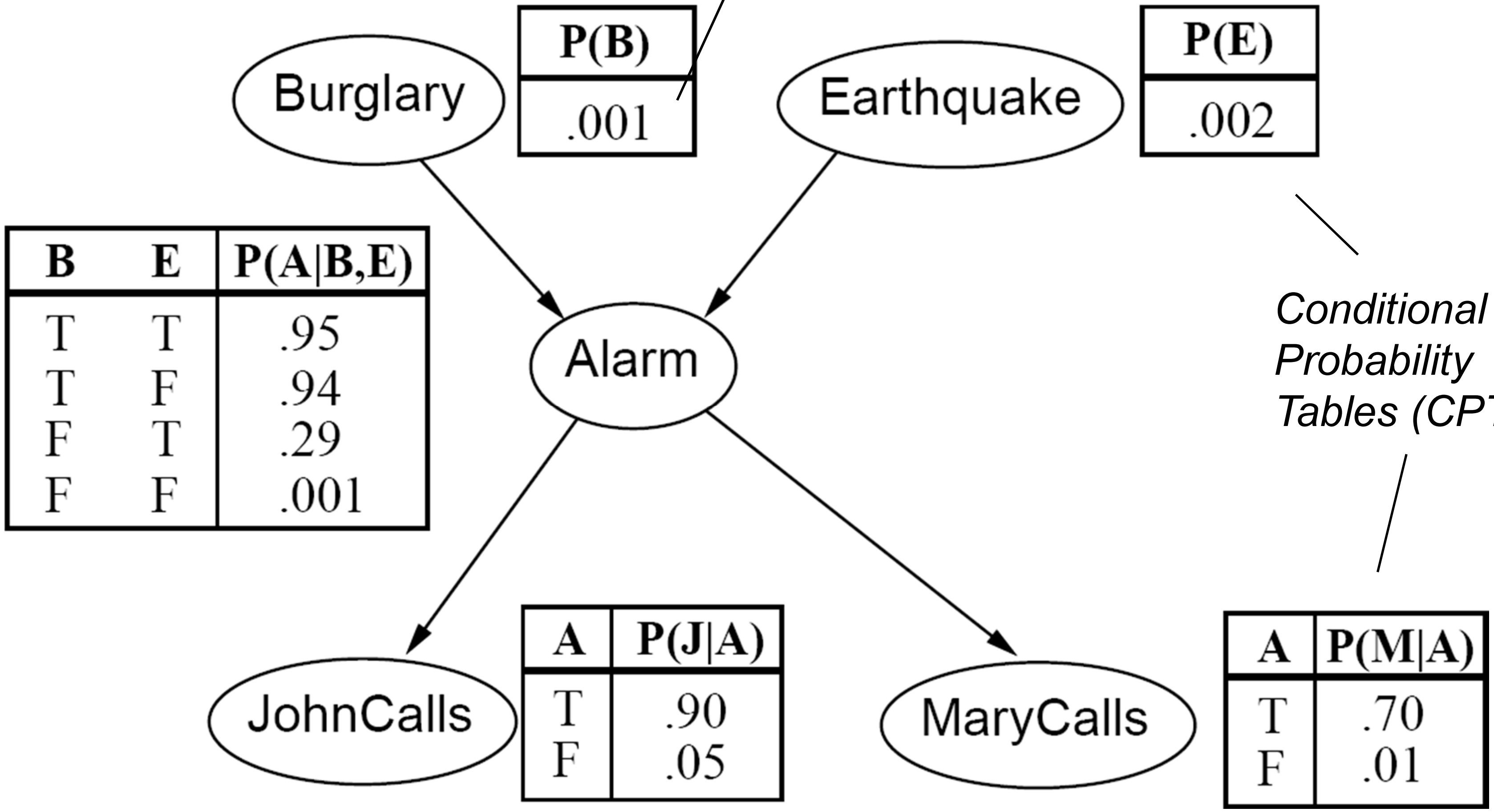
- Given that John and/or Mary has / has not called, what is the probability of a burglary, or of an earthquake?
- Given that the alarm goes off, what is the probability that both John and Mary call and that no burglary happened?
- Given a burglary, what is the probability of the alarm going off?
- Given that Mary has just called, how likely is it that John will call, too?

Modeling the Burglary World as a Bayesian Network



Modeling the Burglary World as a Bayesian Network

= $P(B\text{=true}) = P(b)$
➔ $P(\neg b) = 1\text{-}P(b)$



P(B)
.001

P(E)
.002

B	E	P(A|B,E)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

A	P(J|A)
T	.90
F	.05

A	P(M|A)
T	.70
F	.01

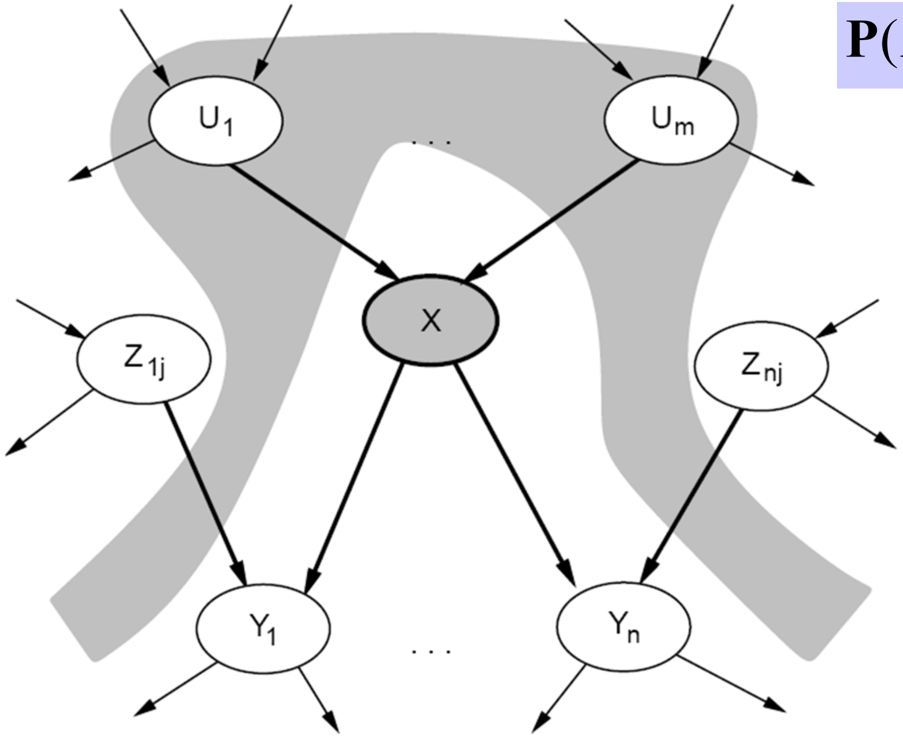
*Conditional Probability Tables (CPTs)**

* **Note:** *Each row in a CPT is a* ***Conditional Probability Distribution***

Important slide:
Make sure you understand this!

Conditional Independence in Bayesian Networks

Central property of a Bayesian Network:
Given the values of its parents, each node X is conditionally independent of its non-descendants ("non-descendants": not the children or their children or …)



$$\mathbf{P}(X \mid U_1,\ldots,U_m) = \mathbf{P}(X \mid U_1,\ldots,U_m, Z_{1j},\ldots,Z_{nj})$$

➔ Parents U_i separate X from irrelevant variables Z_{ij}

But note: X is **not** independent of its children and their descendants!

$$\mathbf{P}(X \mid U_1,\ldots,U_m) \neq \mathbf{P}(X \mid U_1,\ldots,U_m, Y_{1j},\ldots,Y_n)$$

(because knowing the values of the children permits me to better guess the value of X, even if I know the values of the parents of X)

… consider $P(a \mid \neg b, \neg e)$ vs. $P(a \mid \neg b, \neg e, j, m)$ …

Bayesian Networks and the Full Joint Distribution

Remember the full joint distribution: $\mathbf{P}(X_1, \ldots, X_n)$ specifies the probability of every possible atomic event $P(x_1, \ldots, x_n)$ in the agent's world

Remember chain rule:

$$\underline{\mathbf{P}(X_1,\ldots,X_n)} = \mathbf{P}(X_1 \mid X_2,\ldots,X_n)\mathbf{P}(X_2 \mid X_3,\ldots,X_n)\cdots\mathbf{P}(X_n) = \underline{\prod_{i=1}^{n}\mathbf{P}(X_i \mid X_{i+1},\ldots,X_n)}$$

Remember central property of Bayesian Networks (previous slide):

$$\mathbf{P}(X_i \mid Parents(X_i), Z_1,\ldots,Z_n) = \mathbf{P}(X_i \mid Parents(X_i))$$

(where the Z_j are all other non-descendants of X_i)

Now: assume variables are ordered according to the graph structure of the BN, "from bottom to top"
➔ all the children and descendants of node X_i appear before X_i in the ordering, all the parents of X_i appear after X_i, and any other variables after X_i are non-descendants of X_i (is always possible):

$$\underline{\mathbf{P}(X_1,\ldots,X_n)} = \prod_{i=1}^{n}\mathbf{P}(X_i \mid \underbrace{X_{i+1},\ldots,X_n}_{\textit{parents and non-descendants}}) = \underline{\prod_{i=1}^{n}\mathbf{P}(X_i \mid Parents(X_i))}$$

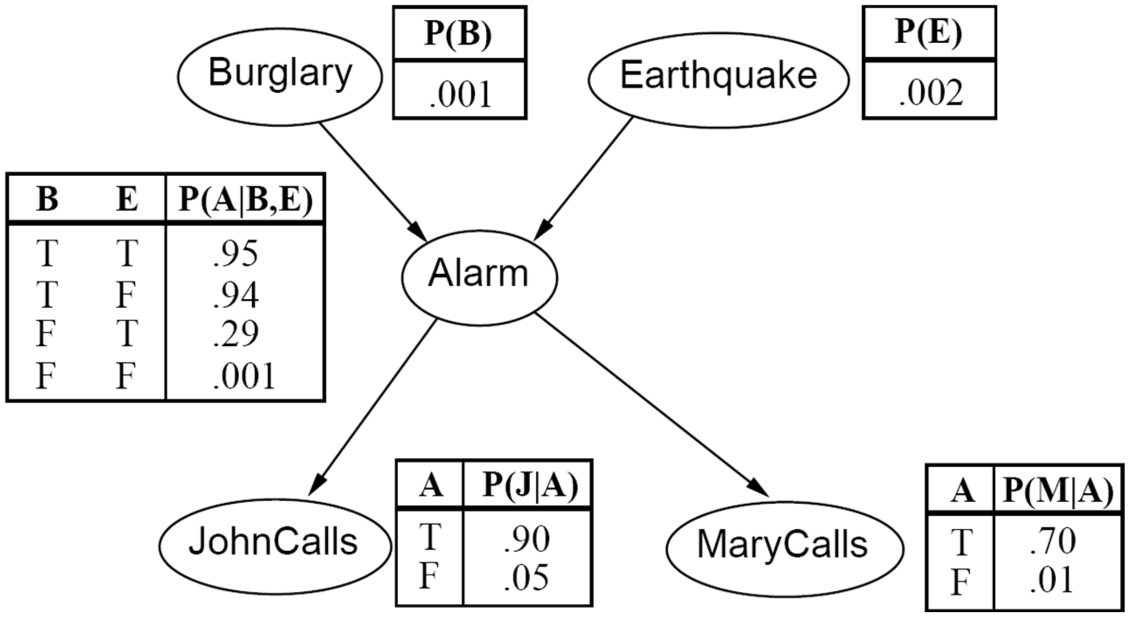
Bayesian Networks and the Full Joint Distribution

$$\mathbf{P}(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} \underbrace{\mathbf{P}(X_i \mid Parents(X_i))}_{\textit{given in CPTs}}$$

In words:

- Any entry in the full joint distribution can be computed as a product of the conditional probabilities of each variable, given its respective parent values
- Any entry in the full joint distribution can be computed as the product of the appropriate elements of the conditional probability tables (CPTs) in the Bayesian network.
- ***A Bayesian network is a structured, compact, complete, and non-redundant representation of the full joint distribution***, where the compaction is made possible by the explicit knowledge about conditional independencies between variables

An Example



P(B)
.001

P(E)
.002

B	E	P(A|B,E)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

A	P(J|A)
T	.90
F	.05

A	P(M|A)
T	.70
F	.01

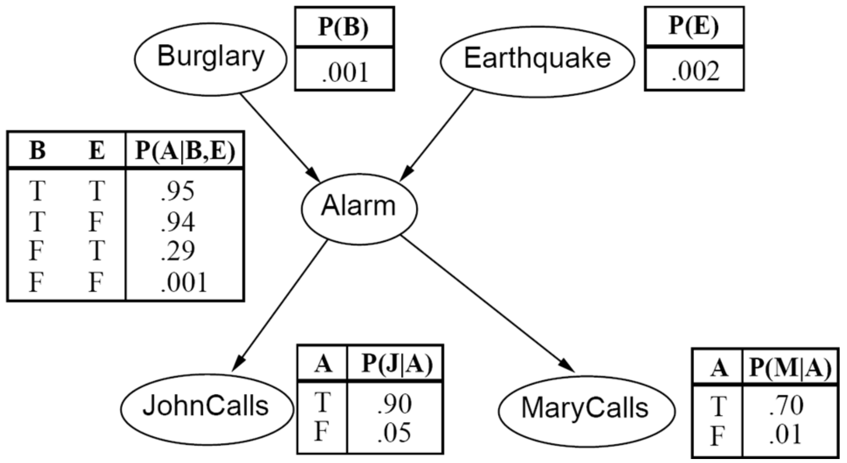
What is the probability that an alarm has sounded, both John and Mary call, but neither a burglary nor an earthquake has happened? (an atomic event …)

$$P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) =$$

$$= P(j \mid a) P(m \mid a) P(a \mid \neg b \wedge \neg e) P(\neg b) P(\neg e) =$$

$$= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = 0.00062$$

Compactness of the Representation



- A CPT for a Boolean variable X_i with k Boolean parents has 2^k rows (one for each combination of parent values)
- Each row requires one number p, for $X_i = true$ (the number for $X_i = false$ is just 1-p)
- If each variable has **at most k** parents, the complete network requires $O(n \cdot 2^k)$ numbers

➔ For a fixed k, complexity grows **linearly** with n (number of variables = size of the network), not exponentially ($O(2^n)$) as with the full joint distribution!

- Burglary network: $1 + 1 + 4 + 2 + 2 = 10$ numbers (vs. $2^5 = 32$)
- Network with n=30 and k=5: requires 960 numbers (vs. $2^{30} > 1{,}000{,}000{,}000$!)

Constructing Bayesian Networks

To construct a Bayesian network, we need to do three things:

1. Decide on the random variables that define the agent's world
2. Specify the structure of the network
3. Specify the conditional probability tables for each node

Step 1: must always be done by the system designer

Step 2: is often done by the system designer
(or an expert in the particular application domain);
but there are also algorithms for ***automatically learning*** the dependency structures (➔ machine learning)
– too complex to be discussed in this class …

Step 3: is usually done via ***learning*** from example observations
(estimation of probabilities from a set of observed events)

Informal Method for Constructing a Bayesian Network Step 2: Specify the Structure of the Network

```
1.  Choose some fixed ordering of the variables $X_1, \ldots, X_n$
2.  For $i = 1$ to $n$ do:
        add $X_i$ to the network;
        select parents from $X_1, \ldots, X_{i-1}$ such that $\mathbf{P}(X_i \mid Parents(X_i)) = \mathbf{P}(X_i \mid X_1, \ldots, X_{i-1})$
```

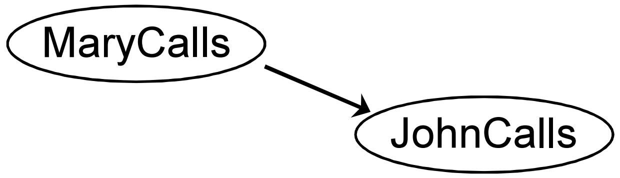
IMPORTANT!
Must choose all variables as $Parents(X_i)$ that somehow influence my estimate of the probability of X_i, or whose prob. somehow correlates with the prob. of X_i
➔ Guarantees that resulting network really represents the true joint distribution

- **Incremental procedure** – add variables/nodes one by one
- **Optimally:** Choose an ordering such that all the variables that should "logically" be the parents of X (i.e., those that "cause" X) are added before X

  ➔ Would guarantee maximal compactness of resulting network

Node Ordering and Compactness: Example

Suppose we choose the ordering M, J, A, B, E (a poor ordering …)



Is *JohnCalls* independent of *MaryCalls*?, i.e., $\mathbf{P}(J \mid M) = \mathbf{P}(J)$
(if we don't know anything else)?

Well … if Mary calls, it is likely that the alarm has gone off, which makes it more likely that John calls (compared to if the alarm had not gone off).

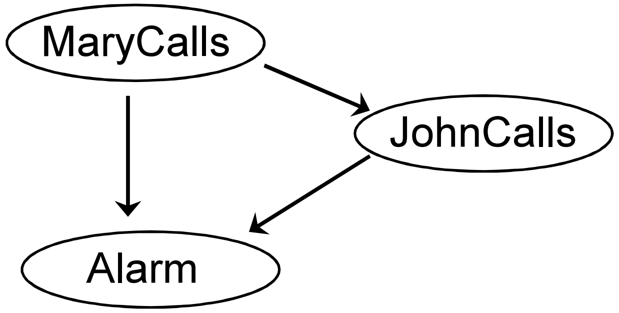
➔ The probabilities of *MaryCalls* and *JohnCalls* are somehow related (through a common cause: the alarm)

➔ $\mathbf{P}(J \mid M) \neq \mathbf{P}(J)$

➔ *JohnCalls* needs *MaryCalls* as a parent (or vice versa)

Node Ordering and Compactness: Example

Suppose we choose the ordering M, J, A, B, E

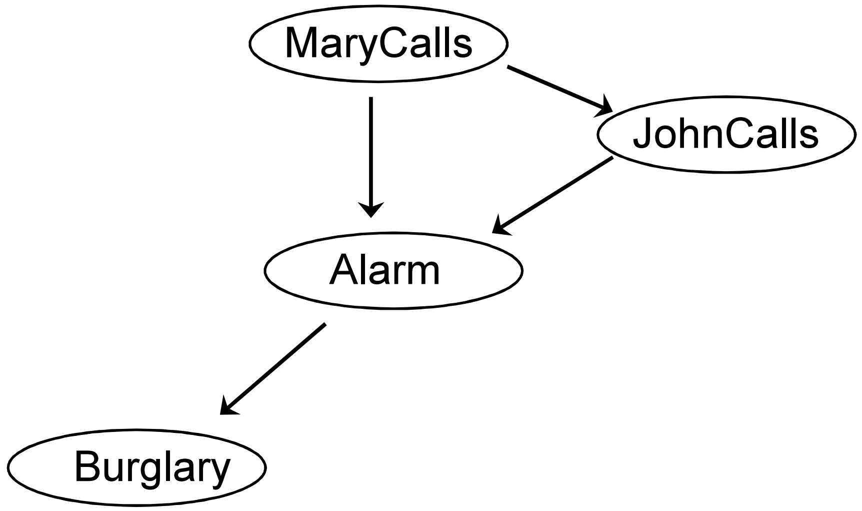


$\mathbf{P}(J \mid M) = \mathbf{P}(J)$? **NO**

$\mathbf{P}(A \mid J) = \mathbf{P}(A)$? **NO** $\mathbf{P}(A \mid J, M) = \mathbf{P}(A \mid J)$? **NO**

Node Ordering and Compactness: Example

Suppose we choose the ordering M, J, A, B, E



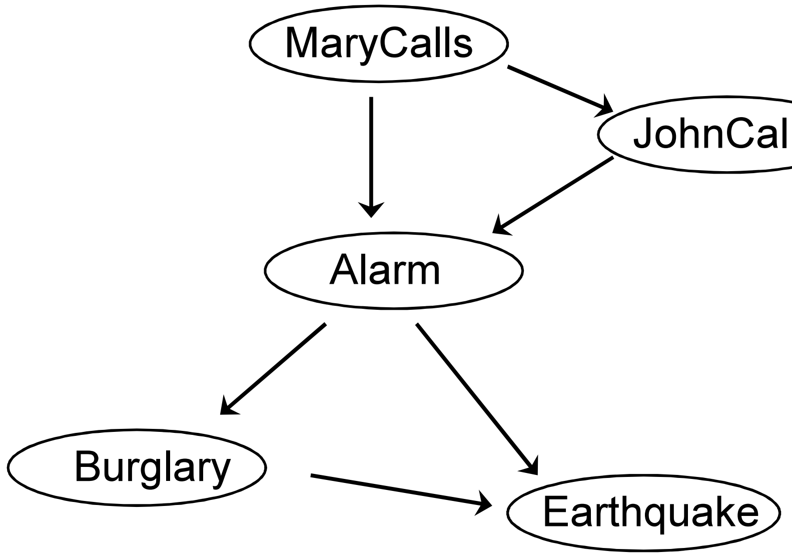
$\mathbf{P}(J \mid M) = \mathbf{P}(J)$? NO

$\mathbf{P}(A \mid J) = \mathbf{P}(A)$? NO $\quad$ $\mathbf{P}(A \mid J, M) = \mathbf{P}(A \mid J)$? NO

$\mathbf{P}(B \mid A) = \mathbf{P}(B)$? NO $\quad$ $\mathbf{P}(B \mid A, J, M) = \mathbf{P}(B \mid A)$? YES

Node Ordering and Compactness: Example

Suppose we choose the ordering M, J, A, B, E



$\mathbf{P}(J \mid M) = \mathbf{P}(J)$? **NO**

$\mathbf{P}(A \mid J) = \mathbf{P}(A)$? **NO** $\mathbf{P}(A \mid J, M) = \mathbf{P}(A \mid J)$? **NO**

$\mathbf{P}(B \mid A) = \mathbf{P}(B)$? **NO** $\mathbf{P}(B \mid A, J, M) = \mathbf{P}(B \mid A)$? **YES**

$\mathbf{P}(E \mid A) = \mathbf{P}(E)$? **NO** $\mathbf{P}(E \mid A, B) = \mathbf{P}(E \mid A)$? **NO(!)**

$\mathbf{P}(E \mid A, B, J, M) = \mathbf{P}(E \mid A, B)$? **YES**

➔ Resulting network is more complex and unintuitive than our previous one … *Why?*

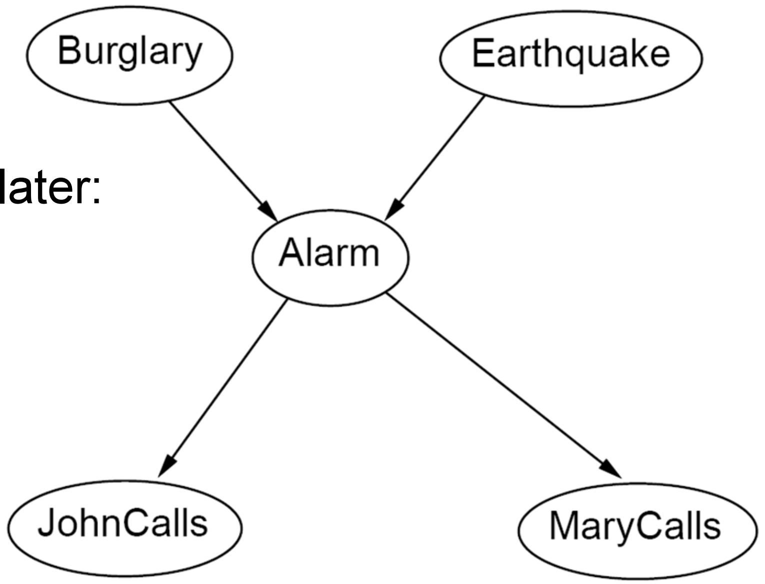
Node Ordering and Compactness

Insights:

- Deciding conditional independence in “non-causal” directions is hard
- Guessing conditional probabilities in non-causal directions is even harder …
- Networks whose nodes are not ordered according to causality tend to be complex (because they need to explicitly encode indirect correlations between variables which are really due to some common cause ..)

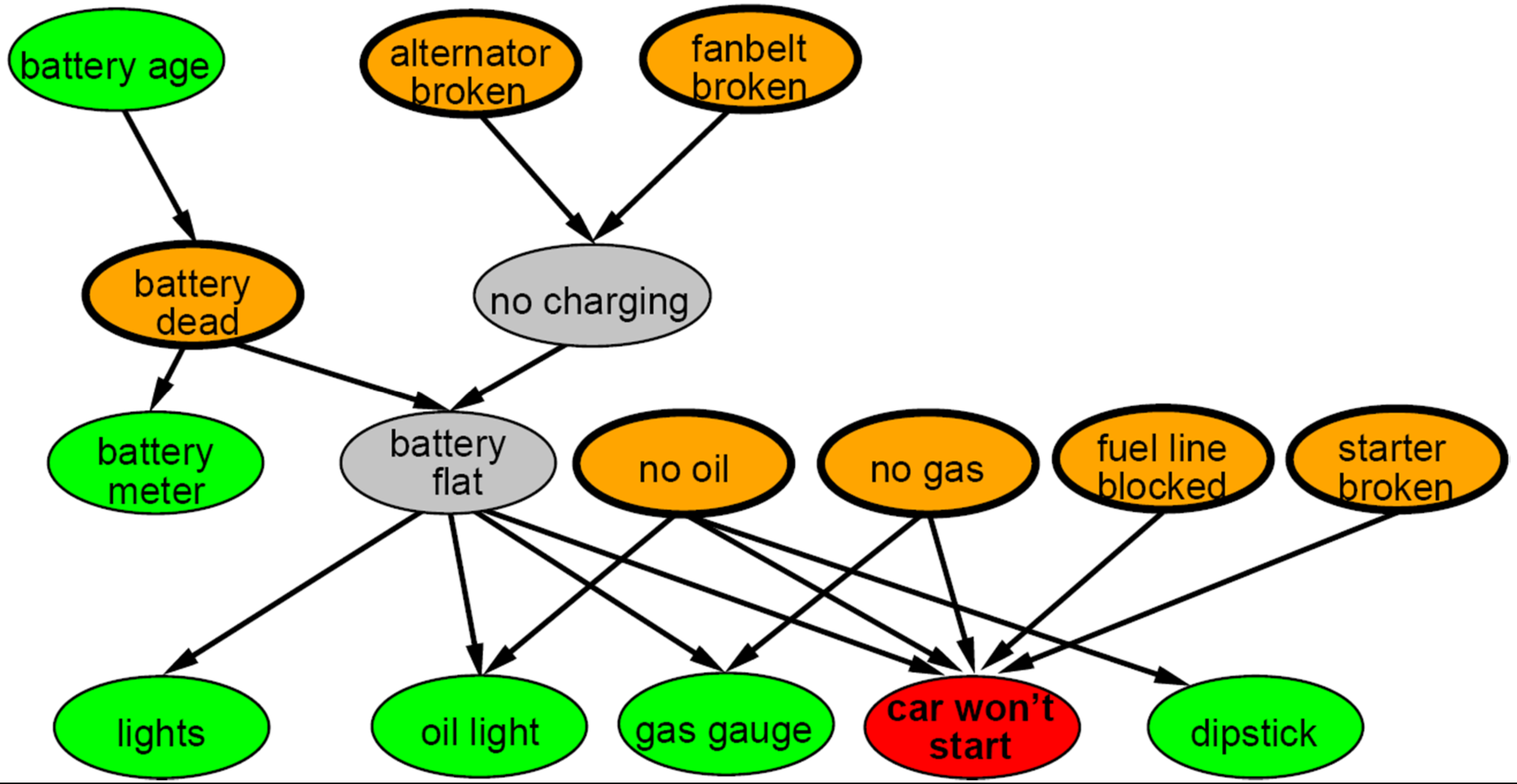
Consequence:

- Add variables in “order of causation” – primary causes first, consequences later:



Example: Car Diagnosis

Initial evidence: car won’t start
green: observable (testable) variables
orange: possible diagnoses / causes (their probabilities will be determined via probabilistic queries, given the values of some of the testable variables)
grey: hidden variables – ensure sparse structure, reduce number of parameters



Part 8: Bayesian Networks

8.b Inference ~~and (briefly) Learning~~

Univ.-Prof. Dr. Gerhard Widmer
Department of Computational Perception
Johannes Kepler University Linz

gerhard.widmer@jku.at
http://www.cp.jku.at/people/widmer

Overview

Exact inference by enumeration

~~**Efficient implementation: The Variable Elimination Algorithm**~~

Approximate inference by stochastic sampling:

- Rejection Sampling
- Likelihood Weighting
- Gibbs Sampling
  (special case of Markov Chain Monte Carlo (MCMC) sampling)

~~**Learning Bayesian Networks from Observations (briefly)**~~

➔ *If you are interested in the omitted topics (and more), consider attending my class "Probabilistic Models" (2 h / 3 ECTS; WS)*

➔ http://www.cp.jku.at/teaching/current/prob_models_introduction_veryshort.pdf

Probabilistic Inference in Bayesian Networks

Basic task of probabilistic inference:

- **Given** the known/observed concrete values of some set of **evidence variables**)
- **Compute** the posterior probability distribution for a set of **query variables**

Notation:

- $\mathbf{X}$ ... query variables
- $\mathbf{E} = \{E_1, \ldots, E_n\}$ ... evidence variables
- $\mathbf{e} = \{e_1, \ldots, e_n\}$ ... values of the evidence variables in the observed event
- $\mathbf{Y} = \{Y_1, \ldots, Y_m\}$ ... all the other variables ("hidden variables")
- $\mathbf{V} = \mathbf{X} + \mathbf{E} + \mathbf{Y}$ ... all variables

➔ Probabilistic query: $\mathbf{P}(\mathbf{X} \mid \mathbf{E}=\mathbf{e})$ = ?

Example: $\mathbf{P}(\, Burglary \mid JohnCalls = true,\, MaryCalls = true)$?

Exact Inference by Enumeration

Remember: Any conditional probability can be computed from the full joint distribution by summing over all hidden variables ("inference by enumeration"):

$$\mathbf{P}(\mathbf{X} \mid \mathbf{E}=\mathbf{e}) = \alpha\, \mathbf{P}(\mathbf{X},\mathbf{E}=\mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(\mathbf{X},\mathbf{E}=\mathbf{e},\mathbf{Y}=\mathbf{y})$$

	toothache		*¬ toothache*	
	catch	*¬ catch*	*catch*	*¬ catch*
cavity	.108	.012	.072	.008
¬ cavity	.016	.064	.144	.576

$$P(cavity \mid toothache) =$$

$$= \frac{P(cavity \wedge toothache)}{P(toothache)} = \boxed{(.108 + .012)} \,/\, \boxed{(.108 + .012 + .016 + .064)} = 0.6$$

$\boxed{1/\alpha}$

Exact Inference by Enumeration in Bayesian Networks

Remember:

Bayesian network implicitly represents complete joint distribution

$$\mathbf{P}(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} \mathbf{P}(X_i \mid Parents(X_i))$$

➔ The probability of any atomic event (i.e., any term of the form $P(x_1, \ldots, x_n)$) can be computed as a product of conditional probabilities from the network

➔ Any query $\mathbf{P}(\mathbf{X} \mid \mathbf{E}=\mathbf{e})$ can be answered from a Bayesian network by computing **sums** (over all value combinations of the unseen variables) **of products** of conditional probabilities (with the values for the $\mathbf{E}$ variables fixed)

➔ Can sum out over variables from the joint distribution without actually constructing its explicit table representation (which would be exponential in the number of variables)

Exact Inference by Enumeration in Bayesian Networks: An Example

Query: $\mathbf{P}(\,Burglary \mid JohnCalls = true,\ MaryCalls = true) = \mathbf{?}$

- Query variable X = $Burglary$
- Evidence $(\mathbf{E} = \mathbf{e}) = \{JohnCalls = true,\ MaryCalls = true\}$ (or $\{j,m\}$, in our shorthand notation)
- Hidden variables $\mathbf{Y} = \{Earthquake,\ Alarm\}$
- Probabilistic query: $\mathbf{P}(X \mid \mathbf{E}{=}\mathbf{e})$ = ?

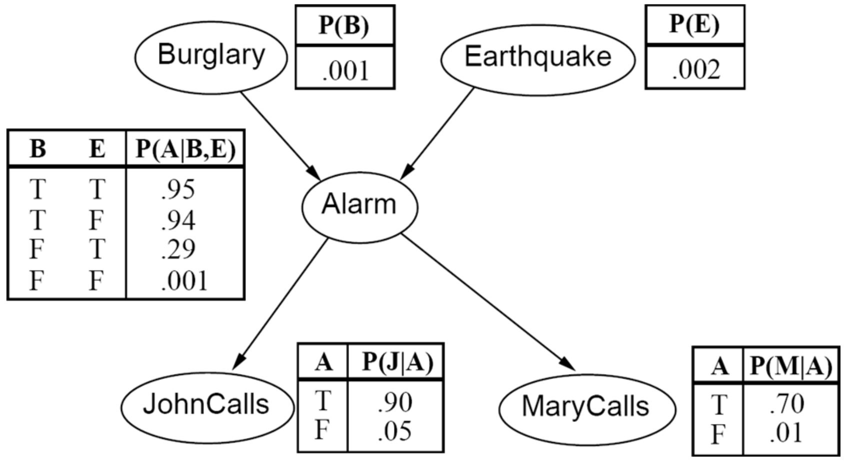
➔ 2 hidden variables
➔ Query can be computed via double sum

$$\mathbf{P}(B \mid j,m) = \alpha\,\mathbf{P}(B,j,m) = \alpha \sum_{x\in\{e,\neg e\}} \sum_{y\in\{a,\neg a\}} \underbrace{\mathbf{P}(B,j,m,E=x,A=y)}$$

these can be computed from the CPTs of the Bayesian Network

Exact Inference by Enumeration

Example computation for *Burglary = true*:



$$
\begin{aligned}
&P(b \mid j,m) \\
&= \alpha \sum_{x\in\{e,\neg e\}} \sum_{y\in\{a,\neg a\}} P(b,j,m,E=x,A=y) \\
&= \alpha\,[P(b,j,m,e,a) \\
&\quad + P(b,j,m,e,\neg a) \\
&\quad + P(b,j,m,\neg e,a) \\
&\quad + P(b,j,m,\neg e,\neg a)]
\end{aligned}
$$

$$
\begin{aligned}
&= \alpha\,[P(b)P(e)P(a\mid b,e)P(j\mid a)P(m\mid a) \\
&\quad + P(b)P(e)P(\neg a\mid b,e)P(j\mid \neg a)P(m\mid \neg a) \\
&\quad + P(b)P(\neg e)P(a\mid b,\neg e)P(j\mid a)P(m\mid a) \\
&\quad + P(b)P(\neg e)P(\neg a\mid b,\neg e)P(j\mid \neg a)P(m\mid \neg a)]
\end{aligned}
$$

➔ $$\left.\begin{aligned} P(b\mid j,m) &= \alpha\times 0.00059224 \\ P(\neg b\mid j,m) &= \alpha\times 0.0014919 \end{aligned}\right\} \quad \alpha = 1/(0.00059224+0.0014919) = 479.8142159$$

$$\underline{\mathbf{P}(B\mid j,m)} = \alpha\times[0.00059224,\ 0.0014919] \approx \underline{[0.284,\ 0.716]}$$

➔ **Probability of burglary, given that both neighbours call, is about 28 %**

This part not treated in class – ignore for exam!

Exact Inference by Enumeration: Optimising the Computation

$$P(b \mid j,m)$$

$$= \alpha \sum_x \sum_y P(b, j, m, E = x, A = y)$$

$$\begin{aligned}
= \alpha\ [&P(b,j,m,e,a) & = \alpha\ [&P(b)P(e)P(a \mid b,e)\boxed{P(j \mid a)P(m \mid a)} \\
+&P(b,j,m,e,\neg a) & +&P(b)P(e)P(\neg a \mid b,e)\boxed{P(j \mid \neg a)P(m \mid \neg a)} \\
+&P(b,j,m,\neg e,a) & +&P(b)P(\neg e)P(a \mid b,\neg e)\boxed{P(j \mid a)P(m \mid a)} \\
+&P(b,j,m,\neg e,\neg a)] & +&P(b)P(\neg e)P(\neg a \mid b,\neg e)\boxed{P(j \mid \neg a)P(m \mid \neg a)}\ ]
\end{aligned}$$

Note: The product $\boxed{P(j|a)P(m|a)}$ must be computed 2x (for each value of E); and then again 2x, if we also want to compute $P(\neg b \mid j,m)$ …
The same goes for $\boxed{P(j|\neg a)P(m|\neg a)}$

Generally: In more complex cases, many products must be computed *(exponentially!)* many times

➔ **Basic Idea:** Try to avoid repeated calculations (by doing the calculation only ***once*** and ***storing the result*** for later use)

➔ A kind of **"dynamic programming"**

➔ “Variable Elimination” Algorithm

Basic algorithm:

1. Write query computation expression as a sum of products
2. Choose an (arbitrary) ordering $\boldsymbol{L}$ of the “hidden” variables (i.e., those to be summed out)
3. for each variable X in $\boldsymbol{L}$:
   - A. “Push in” the sum over X as far as possible
   - B. in this innermost sum:
     - a. Compute product of involved terms (CPT entries or stored ‘factors’) for all possible value combinations of the involved variables
     - b. Sum out the variable to be summed over (X)
     - c. Store resulting factor as a (multi-dimentional) table (‘factor’)
     - d. Replace the sum by the new factor

This part not treated in class – ignore for exam!

Variable Elimination: Example

Example: $\mathbf{P}(B \mid j,m) = ?$

1. Write query computation expression as a sum of products

$$\mathbf{P}(B \mid j,m) = \alpha \sum_{x=e,\neg e} \sum_{y=a,\neg a} \mathbf{P}(B)P(E=x)P(A=y \mid B,E=x)P(j \mid A=y)P(m \mid A=y)$$

2. Choose an (arbitrary) ordering ***L*** of the “hidden” variables – e.g.,

E, A

3.A. [1] “Push in” the sum over E as far as possible

$$\mathbf{P}(B \mid j,m) = \alpha \sum_{y=a,\neg a} \mathbf{P}(B)\,P(j \mid A=y)P(m \mid A=y) \sum_{x=e,\neg e} P(E=x)P(A=y \mid B,E=x)$$

This part not treated in class – ignore for exam!

Variable Elimination: Example

$$\mathbf{P}(B \mid j, m) = \alpha\, \mathbf{P}(B) \sum_{y=a,\neg a} P(j \mid A=y) P(m \mid A=y) \underbrace{\sum_{x=e,\neg e} P(E=x) P(A=y \mid B, E=x)}$$

3.B. in this innermost sum:

a. Compute product of involved terms (CPT entries or stored ‘factors’) for all possible value combinations of the involved variables

➔ compute $P(E)P(A|B,E)$ for all values of E, A, B (entries from CPTs)

➔ gives an intermediate table $\mathbf{T}$ of dimensionality $|E|\times|A|\times|B|$

b. Sum out over E :

➔ for each pair of values of A and B, sum up $\mathbf{T}(e,A,B) + \mathbf{T}(\neg e,A,B)$

c. Store resulting factor as a (multi-dimentional) table

➔ gives a table $\mathbf{F}_E(A,B)$ of dimensionality $|A|\times|B|$ (subscript E indicates that E has been summed out)

This part not treated in class – ignore for exam!

Variable Elimination: Example

d. Replace the sum by the new factor

$$\mathbf{P}(B \mid j, m) =$$

$$= \alpha \sum_{y=a,\neg a} \mathbf{P}(B) P(j \mid A=y) P(m \mid A=y) \underbrace{\sum_{x=e,\neg e} P(E=x) P(A=y \mid B, E=x)}_{\mathbf{F}_E(A,B)} =$$

$$= \alpha \sum_{y=a,\neg a} \mathbf{P}(B) P(j \mid A=y) P(m \mid A=y) \mathbf{F}_E(A=y, B)$$

Variable Elimination: Example

$$\mathbf{P}(B \mid j,m) = \alpha \sum_{y=a,\neg a} \mathbf{P}(B) P(j \mid A=y) P(m \mid A=y)\, \mathbf{F}_E(A=y,B)$$

Next iteration:

3.A. [2] "Push in" the sum over A as far as possible

$$\mathbf{P}(B \mid j,m) = \alpha\ \mathbf{P}(B)\ \underbrace{\sum_{y=a,\neg a} P(j \mid A=y) P(m \mid A=y)\, \mathbf{F}_E(A=y,B)}$$

3.B. in this innermost sum:

a. Compute products of involved terms (CPT entries or stored 'factors')

➔ compute $P(j|A)\ P(m|A)\ \mathbf{F}_E(A,B)$ for all values of A, B

➔ gives an intermediate table $\mathbf{T}$ of dimensionality $|A|\times|B|$

b. Sum out over A :

➔ for each value of B, sum up $\mathbf{T}(a,B) + \mathbf{T}(\neg a,B)$

c. Store resulting factor as table (vector) $\mathbf{F}_A(B)$ of dimensionality $|B|$

This part not treated in class – ignore for exam!

Variable Elimination: Example

d. replace the sum by the new factor

$$\mathbf{P}(B \mid j,m) =$$

$$= \alpha\ \mathbf{P}(B) \underbrace{\sum_{y=a,\neg a} P(j \mid A=y) P(m \mid A=y) \mathbf{F}_E(A=y,B)}_{\mathbf{F}_A(B)}$$

$$= \alpha\ \mathbf{P}(B)\mathbf{F}_A(B)$$

➔ Final Step:

Compute the result for each value of B
(i.e., take point-wise product of $\mathbf{P}(B)$ and $\mathbf{F}_A(B)$):

$$\mathbf{P}(B \mid j,m) = \begin{bmatrix} P(b \mid j,m) \\ P(\neg b \mid j,m) \end{bmatrix} = \alpha\ \mathbf{P}(B) \cdot \mathbf{F}_A(B) = \alpha \begin{bmatrix} P(b)\mathbf{F}_A(b) \\ P(\neg b)\mathbf{F}_A(\neg b) \end{bmatrix}$$

The Complexity of Exact Inference in Bayesian Networks

Consider: $$\mathbf{P}(B \mid j,m) = \alpha\,\mathbf{P}(B,j,m) = \alpha \sum_{x\in\{e,\neg e\}} \sum_{y\in\{a,\neg a\}} \mathbf{P}(B,j,m,E=x,A=y)$$

➔ Need to sum over $2 \text{ x } 2 = 2^2 = 4$ value combinations of hidden variables E, A

➔ In the worst case, given n variables, almost all of these may be hidden and thus must be summed out:

➔ Worst-case complexity of exact inference in Bayesian Networks is $O(2^n)$

Can be dramatically improved via clever variable elimination algorithms (see previous slide), *if*

- network is sparsely connected (there are many conditional independencies)
- we happen to pick (guess) a good variable elimination ordering

➔ Still: in the *worst case*, no improvement (asymptotically) over full enumeration of joint distribution
(and intermediate 'factors' in variable elimination can grow exponentially …)

➔ Need for ***approximate*** algorithms

Approximate Inference in Bayesian Networks: Stochastic Sampling

Basic idea:

- Randomly generate atomic events (according to the probability distribution defined by the network)
- The frequency with which a certain event occurs gives an estimate of its true probability
- The relative frequencies of all atomic events give an ***estimate of the full joint distribution***

➔ Will estimate $\mathbf{P}(\mathbf{X} \mid \mathbf{e})$ via a constant (large) number of random samples instead of an exponential summation process

Outline of the next slides:

1. Estimating the prior (unconditional) probability of events:
   - Sampling without evidence
2. Approximately solving probabilistic queries $\mathbf{P}(\mathbf{X} \mid \mathbf{e})$:
   - Rejection Sampling
   - Likelihood Weighting
   - Gibbs Sampling
   - more generally: Markov Chain Monte Carlo (MCMC) sampling

Estimating the Probability of Events: Sampling without Evidence (“Sampling from the Prior Distribution”)

Algorithm for randomly generating an atomic event according to the probability distribution represented by the network:

- Sample variables in topological order in the network (from ‘top’ to ‘bottom’)
- “Sampling a variable” =
  randomly choosing a value for the variable according to the variable’s Conditional Probability Table (CPT), **taking into account the values of the variable’s parents (which have already been assigned)**

➔ *Exercise:* Devise an algorithm for choosing values from a set according to a discrete probability distribution
(assuming that you have a pseudo-random number generator that returns a random real number uniformly distributed in the interval [0:1])

Estimating the Probability of Events: Sampling without Evidence ("Sampling from the Prior Distribution")

Algorithm for randomly generating an atomic event according to the probability distribution represented by the network:

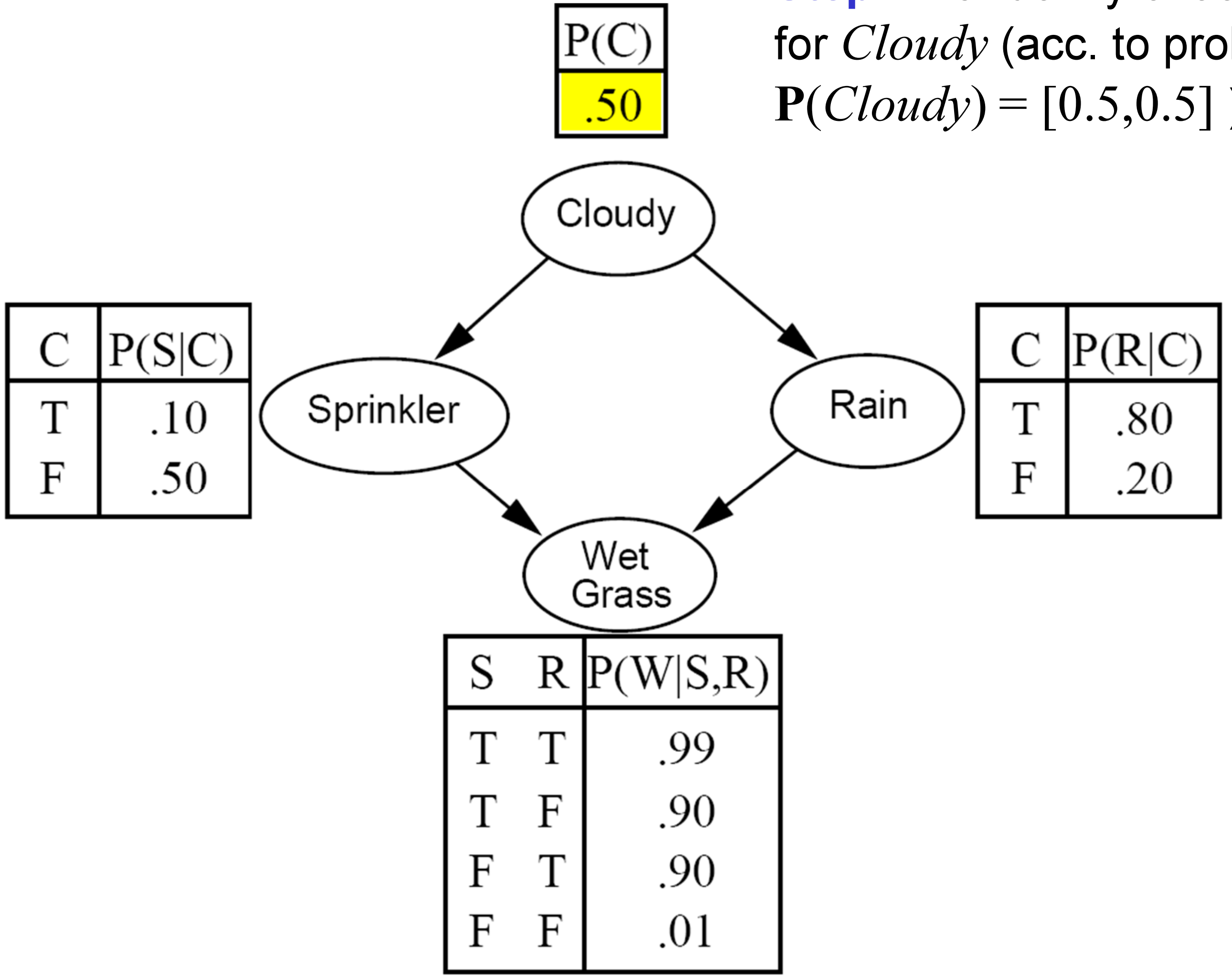
```
function PRIOR-SAMPLE($bn$) returns an event sampled from $bn$
    inputs: $bn$, a belief network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

    $\mathbf{x} \leftarrow$ an event with $n$ elements
    for $i = 1$ to $n$ do
        $x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$        *)
            given the values of $Parents(X_i)$ in $\mathbf{x}$
    return $\mathbf{x}$
```

*) *Assumption: Variables X_i are sorted in topological order ("from top to bottom")*

Sampling from a Simple Example Network

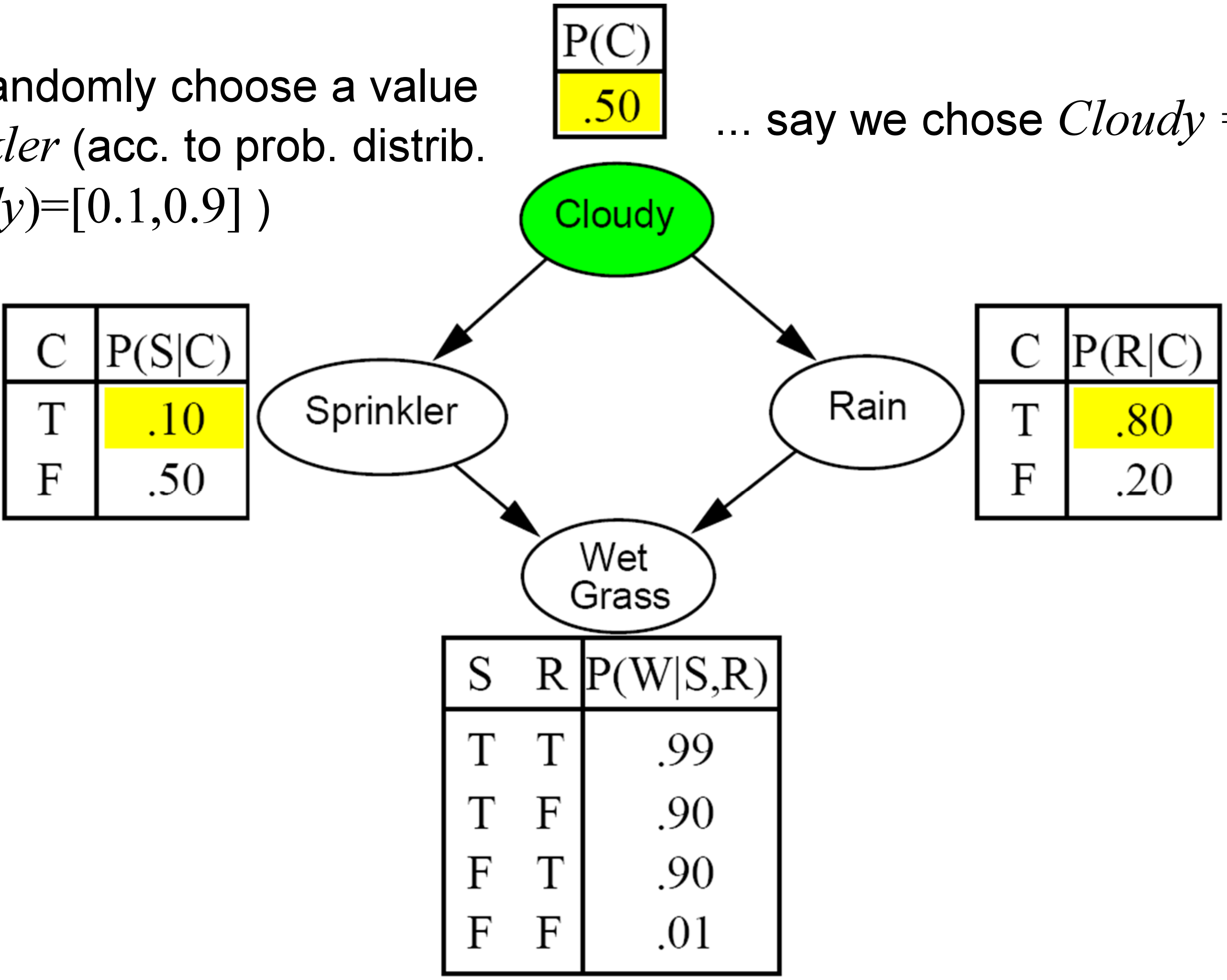
Step 1: randomly choose a value for *Cloudy* (acc. to prob. distribution $\mathbf{P}(Cloudy) = [0.5, 0.5]$)



Sampling from a Simple Example Network

Step 2: randomly choose a value for *Sprinkler* (acc. to prob. distrib. $\mathbf{P}(S|cloudy)=[0.1,0.9]$)

... say we chose $Cloudy = true$ ...



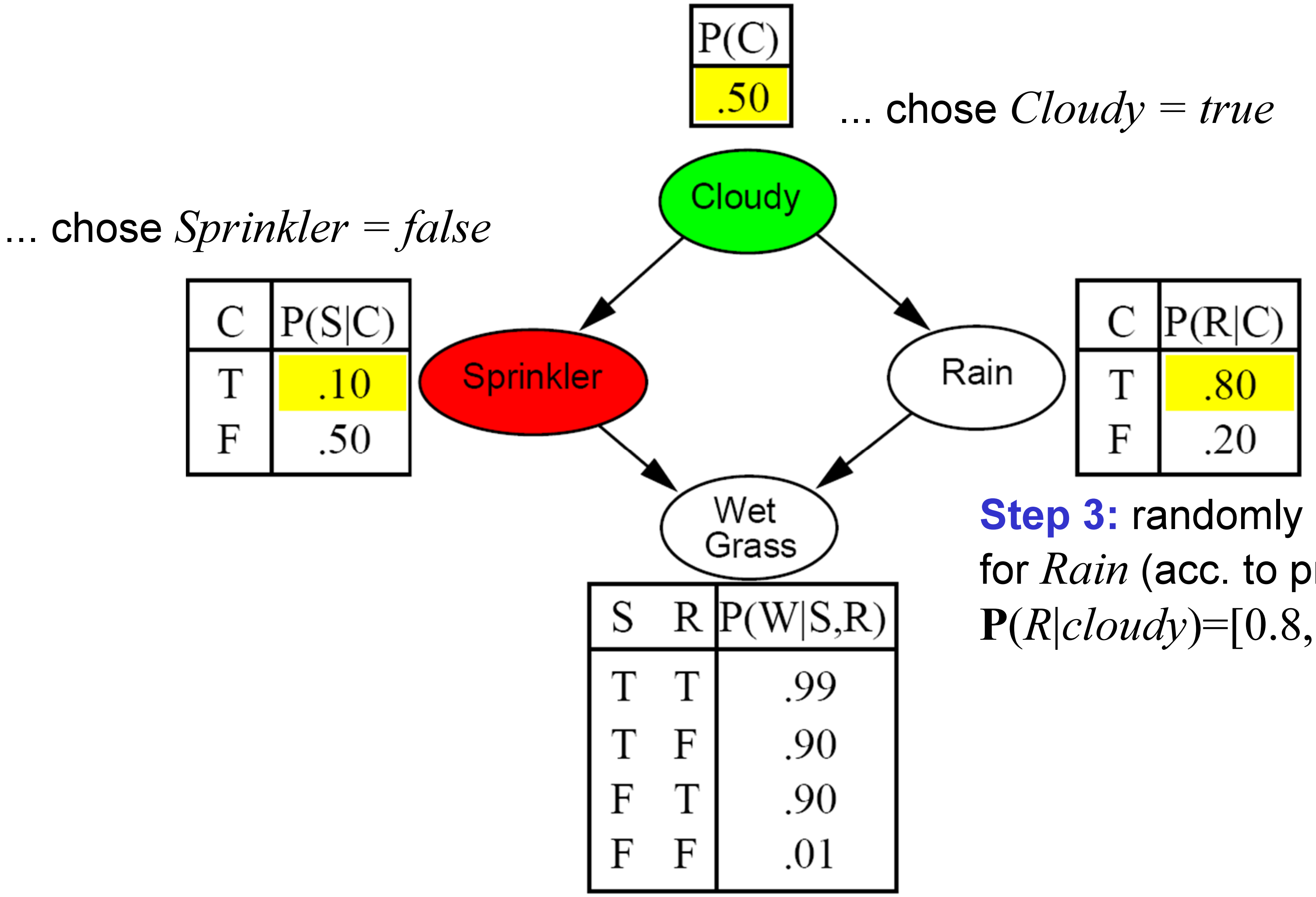
P(C)
.50

C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

Sampling from a Simple Example Network



P(C)
.50

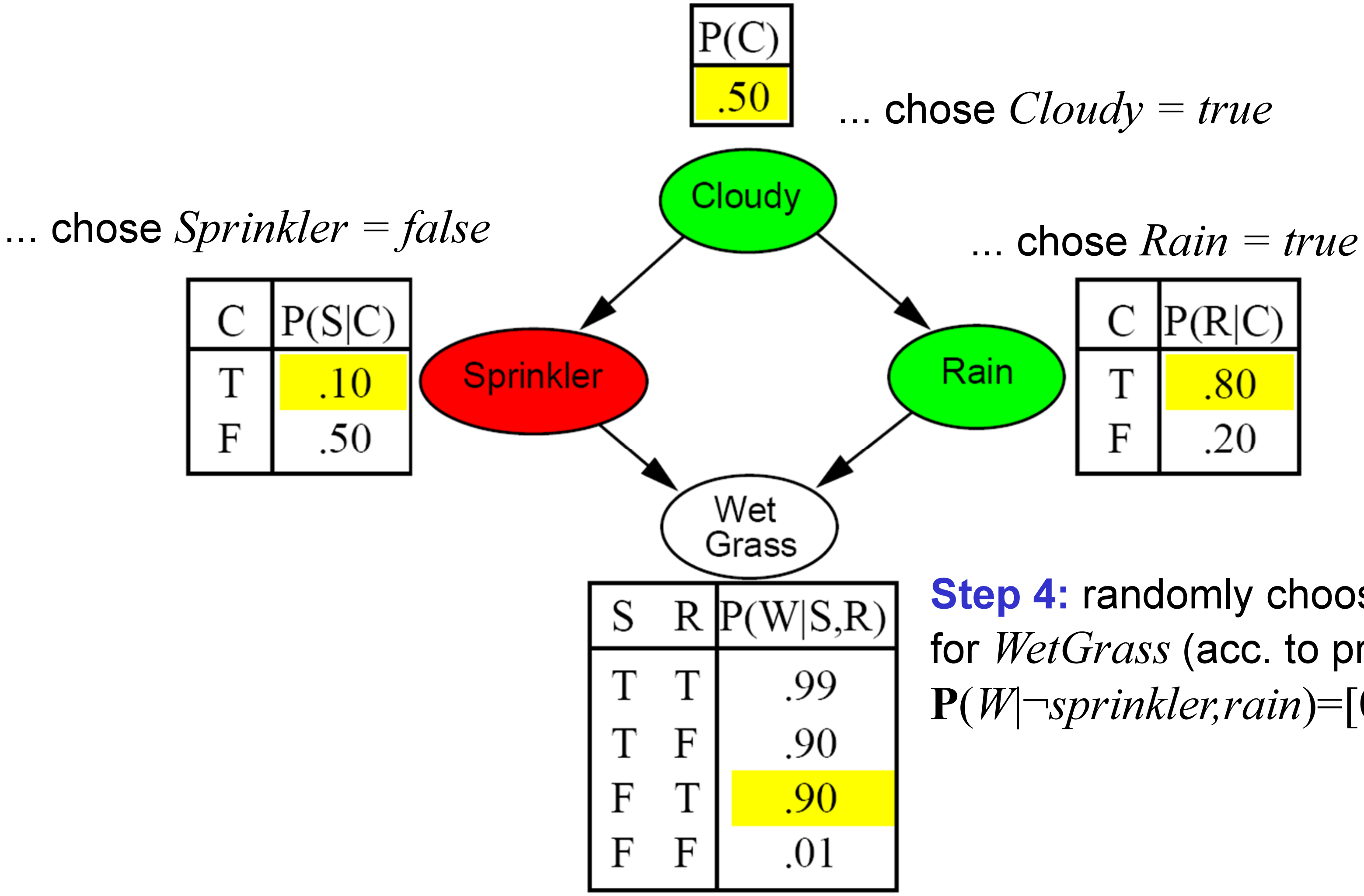
C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

Step 3: randomly choose a value for *Rain* (acc. to prob. distrib. $\mathbf{P}(R|cloudy)=[0.8,0.2]$)

Sampling from a Simple Example Network


P(C)
.50
... chose *Cloudy* = *true*
Cloudy
... chose *Sprinkler* = *false*
... chose *Rain* = *true*
Sprinkler
Rain
Wet Grass

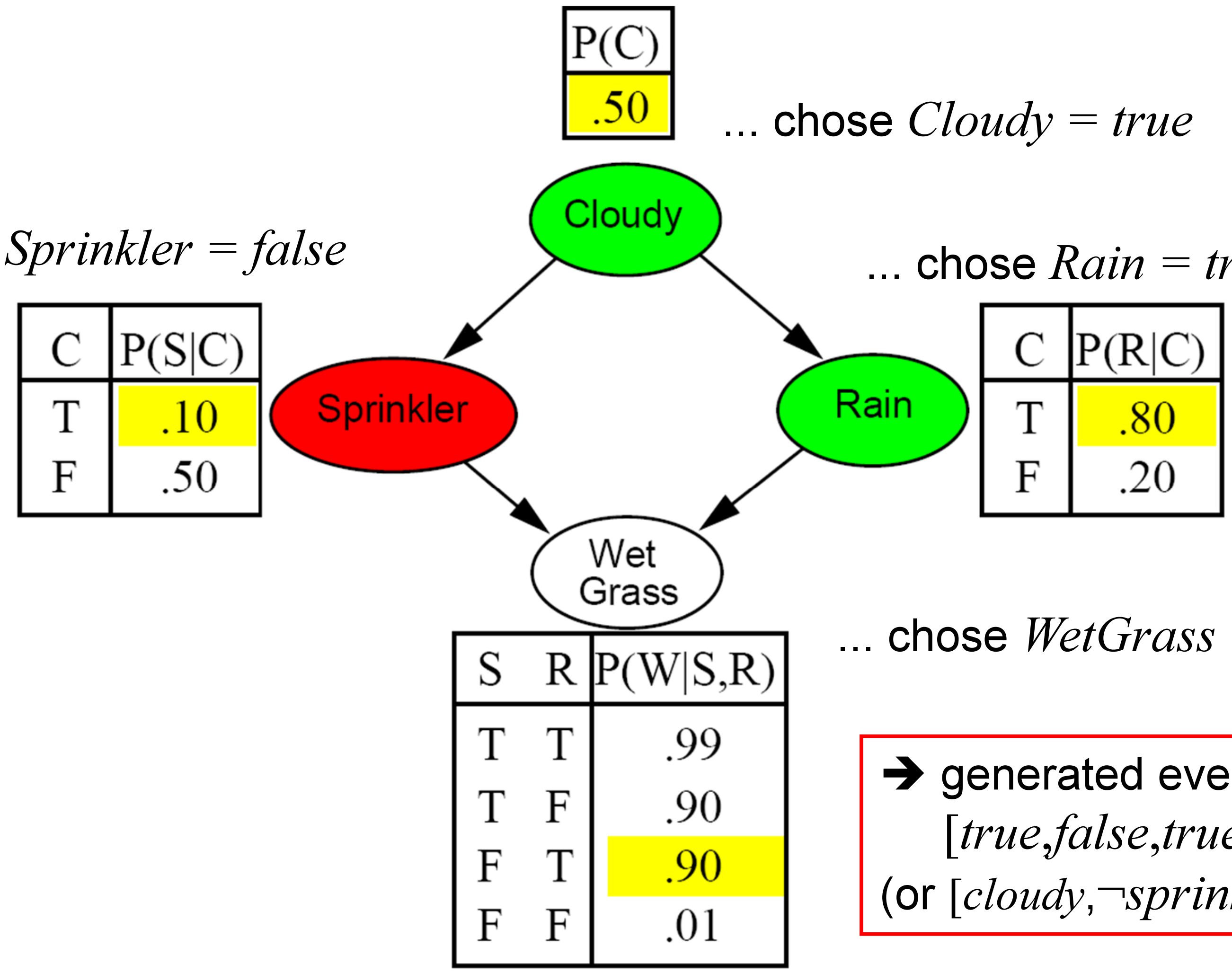
C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

Step 4: randomly choose a value for *WetGrass* (acc. to prob. distrib. $\mathbf{P}(W|\neg sprinkler, rain)=[0.9,0.1]$)

Sampling from a Simple Example Network



P(C)
.50

... chose *Cloudy* = *true*

... chose *Sprinkler* = *false*

C	P(S|C)
T	.10
F	.50

... chose *Rain* = *true*

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

... chose *WetGrass* = *true*

➔ generated event
[*true*,*false*,*true*,*true*]
(or [*cloudy*,¬*sprinkler*,*rain*,*wet*])

Sampling Distribution and Full Joint Distribution

Need to prove that the sampling distribution converges to the true distribution:

Variables are sampled independently (given values of their parents)

➔ Probability that the sampling algorithm Prior-Sample (PS) generates an event $(x_1,\ldots,x_n)$ is the product of the probabilities of generating each x_i :

$$P_{PS}(x_1,\ldots,x_n) = \prod_{i=1}^{n} P_{PS}(x_i) = \prod_{i=1}^{n} P(x_i \mid parents(X_i))$$

(because each sampling step depends only on the parent values)

➔ As the number of sampled events goes to infinity, the *sampling distribution* $\mathbf{P}_{PS}$ perfectly approximates the full joint distribution $\mathbf{P}$ represented by the network

➔ Estimates based on this sampling process are said to be *consistent*

➔ Estimating the Probability of an Atomic Event

Estimating the probability of an atomic (i.e., fully specified) event $(x_1,\ldots,x_n)$ **:**

- Generate N random samples from the network
- Count the number of times K that the specific event appears in these N samples
- Take ratio $\hat{P} = \frac{K}{N}$ as an estimate of the probability $P(x_1,\ldots,x_n)$

Example:

$$\begin{aligned} P(cloudy, \neg sprinkler, rain, wet) &= P(true, false, true, true) = \\ &= 0.5 \times 0.9 \times 0.8 \times 0.9 = 0.324 \end{aligned}$$

➔ If we take a large number of samples, we can expect around 32.4 % of the samples to be equal to $(true, false, true, true)$

Estimating the Probability of a *Partially Specified* Event

Estimating the probability of a partially specified event $(x_1,\ldots,x_m)$, with $m < n$ **:**

- Take N random samples (according to the net)
- Count the number of times K of occurrences of atomic events among these samples that **match** $(x_1,\ldots,x_m)$
  (i.e., that have these specific values for the variables $X_1,\ldots,X_m$)
- Take ratio $\hat{P} = \frac{K}{N}$ as an estimate of the probability $P(x_1,\ldots,x_m)$

Example:

- Assume we generate 1000 samples from the sprinkler network, and 511 of them have $Rain = true$, then the estimated probability of rain

$$\hat{P}(Rain = true) = \frac{511}{1000} = 0.511$$

Sampling from the Posterior Distribution – Answering Probabilistic Queries via Sampling (1): Rejection Sampling

Goal: compute approximate solution to probabilistic query $\mathbf{P}(\mathbf{X} \mid \mathbf{E}=\mathbf{e})$ via sampling

Idea: use only those samples for estimating $\mathbf{P}(\mathbf{X})$ that match the given evidence $\mathbf{E}=\mathbf{e}$

Simple Algorithm:

- Generate N samples from the network
- Keep only those N' samples that match the evidence $\mathbf{e}$ (i.e., reject those samples with $\mathbf{E} \neq \mathbf{e}$)
- Count K = number of times that $\mathbf{X}=\mathbf{x}$ appears in the remaining N' samples
- Take ratio $\hat{P} = \dfrac{K}{N'}$ as estimate of probability $P(\mathbf{X}=\mathbf{x} \mid \mathbf{E}=\mathbf{e})$

Easy to show: as $N \rightarrow \infty$, the estimate converges to the true probability

Sampling from the Posterior Distribution – Answering Probabilistic Queries via Sampling (1): Rejection Sampling

Goal: compute approximate solution to probabilistic query $\mathbf{P}(\mathbf{X} \mid \mathbf{E}=\mathbf{e})$ via sampling

Idea: use only those samples for estimating $\mathbf{P}(\mathbf{X})$ that match the given evidence $\mathbf{E}=\mathbf{e}$

Simple Algorithm:

```
function REJECTION-SAMPLING($X$, e, $bn$, $N$) returns an estimate of $P(X|\mathrm{e})$
    local variables: N, a vector of counts over $X$, initially zero

    for $j = 1$ to $N$ do
        x ← PRIOR-SAMPLE($bn$)
        if x is consistent with e then
            N[$x$] ← N[$x$]+1 where $x$ is the value of $X$ in x
    return NORMALIZE(N[$X$])
```

Easy to show: as $N \rightarrow \infty$, the estimate converges to the true probability

Rejection Sampling: Example

Probabilistic query: $\mathbf{P}(Rain \mid Sprinkler = true)$

Assume that

- we sample $N = 100$ events
- 73 of these have $Sprinkler = false$ ➔ are rejected
- 27 have $Sprinkler = true$ ➔ $N' = 27$
- of these 27 samples, 8 have $Rain = true$ and 19 have $Rain = false$

➔ $$\hat{P}(Rain = true \mid Sprinkler = true) = \frac{8}{27} = 0.296$$

$$\hat{P}(Rain = false \mid Sprinkler = true) = \frac{19}{27} = 0.704$$

$$\hat{\mathbf{P}}(Rain \mid Sprinkler = true) = [0.296, 0.704]$$

(True answer computed from the network is $\mathbf{P}(Rain \mid Sprinkler = true) = [0.3, 0.7]$)

Rejection Sampling

Advantage of Rejection Sampling:

- Low computational complexity:
  linear in number of samples and size of network

Problem with Rejection Sampling:

- Rejects many examples:
  proportion of samples consistent with evidence $\mathbf{E}=\mathbf{e}$ drops exponentially with number of evidence variables $\mathbf{E}$!

Note: Rejection sampling is very similar to estimating conditional probabilities by observing the real world:

- Would estimate $\mathbf{P}(\ Rain \mid RedSkyAtNight=true)$ by observing the sky every evening and counting how often it rains after a red sky was observed the previous evening, ignoring those evenings when the sky is not red
  ➔ the world itself is the sample generation algorithm
- Could take a long time if the sky is red only very rarely …

Answering Probabilistic Queries via Sampling (2): Likelihood Weighting

Idea:

- Avoid inefficiency of rejection sampling by **generating only events that are consistent with the evidence** $\mathbf{e}$

Basic algorithm:

- Fix the values of the evidence variables $\mathbf{E} = \mathbf{e}$
- Sample only remaining variables $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$
  ➔ guarantees that each generated event is consistent with the evidence
- When collecting the event counts for the query variables $\mathbf{X}$, **weight** each event by **the probability that in an event like this, the evidence variables would actually have these values:**
  Weight = product of the conditional probabilities of the evidence variables, given their parents' values

Intuition: Events in which the actual evidence would be unlikely should be given less weight

Answering Probabilistic Queries via Sampling (2): Likelihood Weighting

```
function LIKELIHOOD-WEIGHTING($X$, $\mathbf{e}$, $bn$, $N$) returns an estimate of $P(X|\mathbf{e})$
    local variables: $\mathbf{W}$, a vector of weighted counts over $X$, initially zero

    for $j = 1$ to $N$ do
        $\mathbf{x}, w \leftarrow$ WEIGHTED-SAMPLE($bn$)
        $\mathbf{W}[x] \leftarrow \mathbf{W}[x] + w$ where $x$ is the value of $X$ in $\mathbf{x}$
    return NORMALIZE($\mathbf{W}[X]$)
```

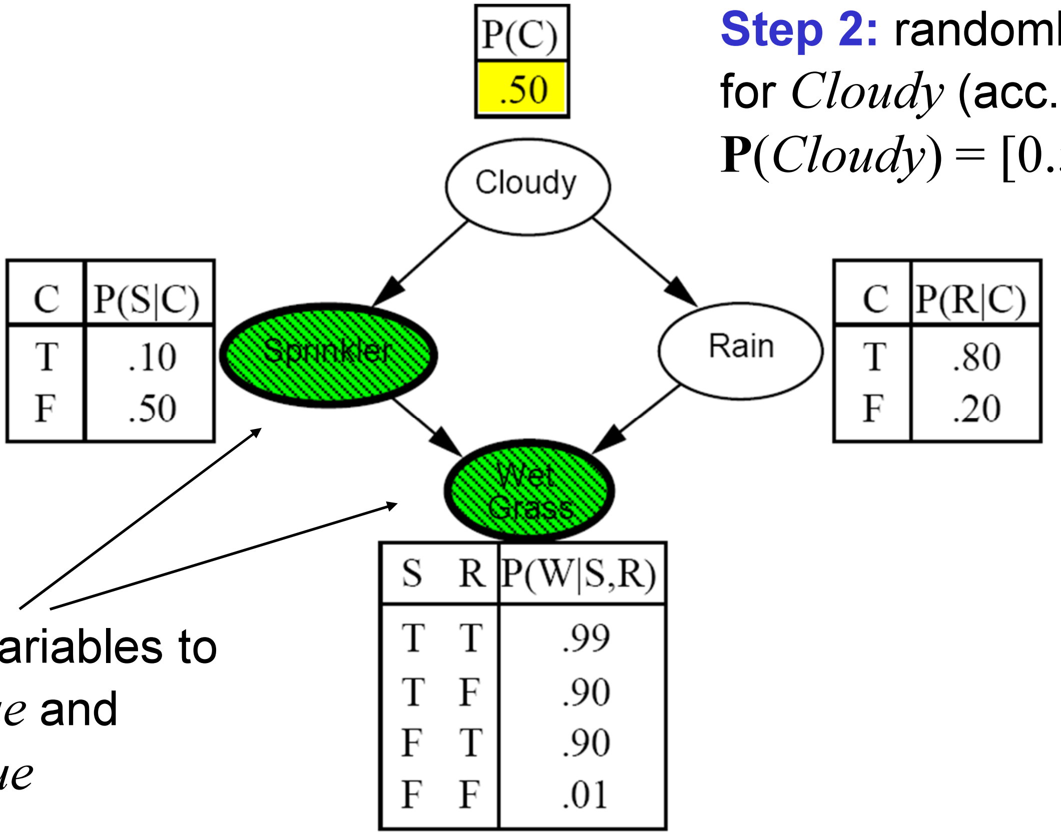
```
function WEIGHTED-SAMPLE($bn$, $\mathbf{e}$) returns an event and a weight

    $\mathbf{x} \leftarrow$ an event with $n$ elements; $w \leftarrow 1$
    for $i = 1$ to $n$ do
        if $X_i$ has a value $x_i$ in $\mathbf{e}$
            then $w \leftarrow w \times P(X_i = x_i \mid parents(X_i))$
            else $x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
    return $\mathbf{x}$, $w$
```

Likelihood Weighting: Example

Query: $\mathbf{P}(\,Rain \mid Sprinkler = true,\ WetGrass = true) = \mathbf{?}$

Step 2: randomly choose a value for *Cloudy* (acc. to prob. distribution $\mathbf{P}(Cloudy) = [0.5, 0.5]$)



P(C)
.50

C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

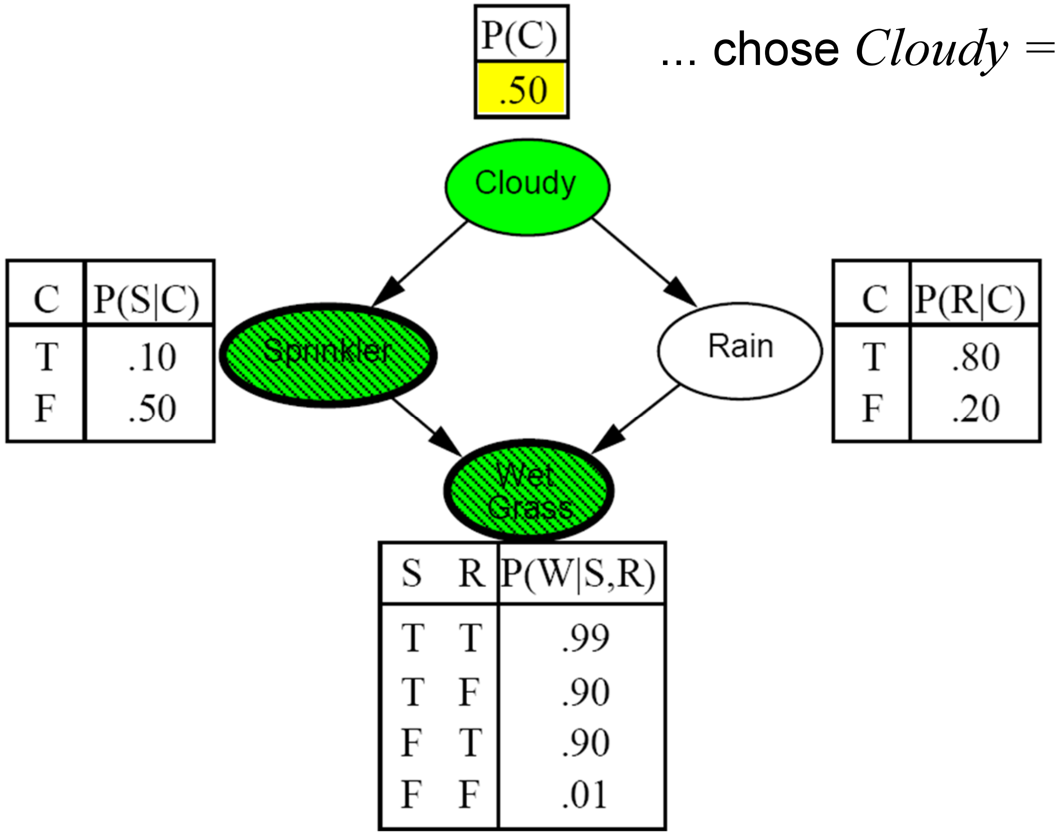
Step 0:
fix evidence variables to *Sprinkler=true* and *WetGrass=true*

Step 1: set weight $w = 1.0$

Likelihood Weighting: Example

Query: $\mathbf{P}(Rain \mid Sprinkler = true, WetGrass = true) = \mathbf{?}$

... chose $Cloudy = true$



P(C)
.50

C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

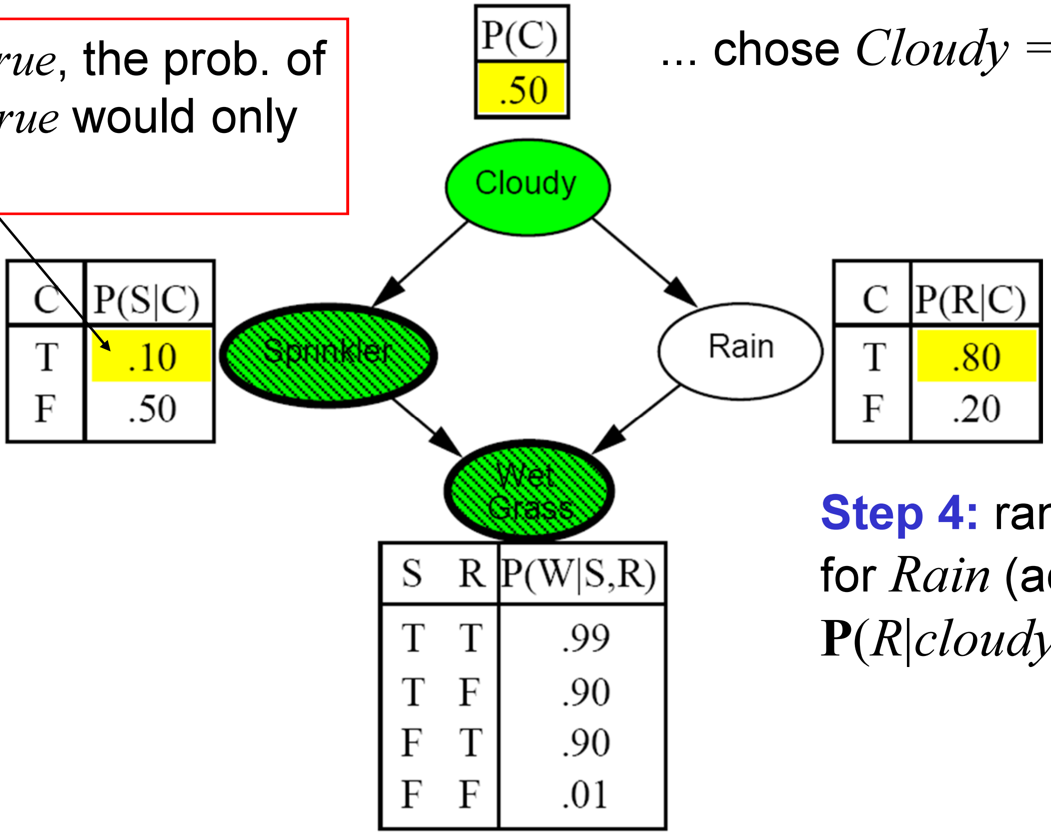
weight $w = 1.0$

Likelihood Weighting: Example

Query: $\mathbf{P}(Rain \mid Sprinkler = true, WetGrass = true) = ?$

if $Cloudy = true$, the prob. of $Sprinkler = true$ would only be 0.1 !

... chose $Cloudy = true$



C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

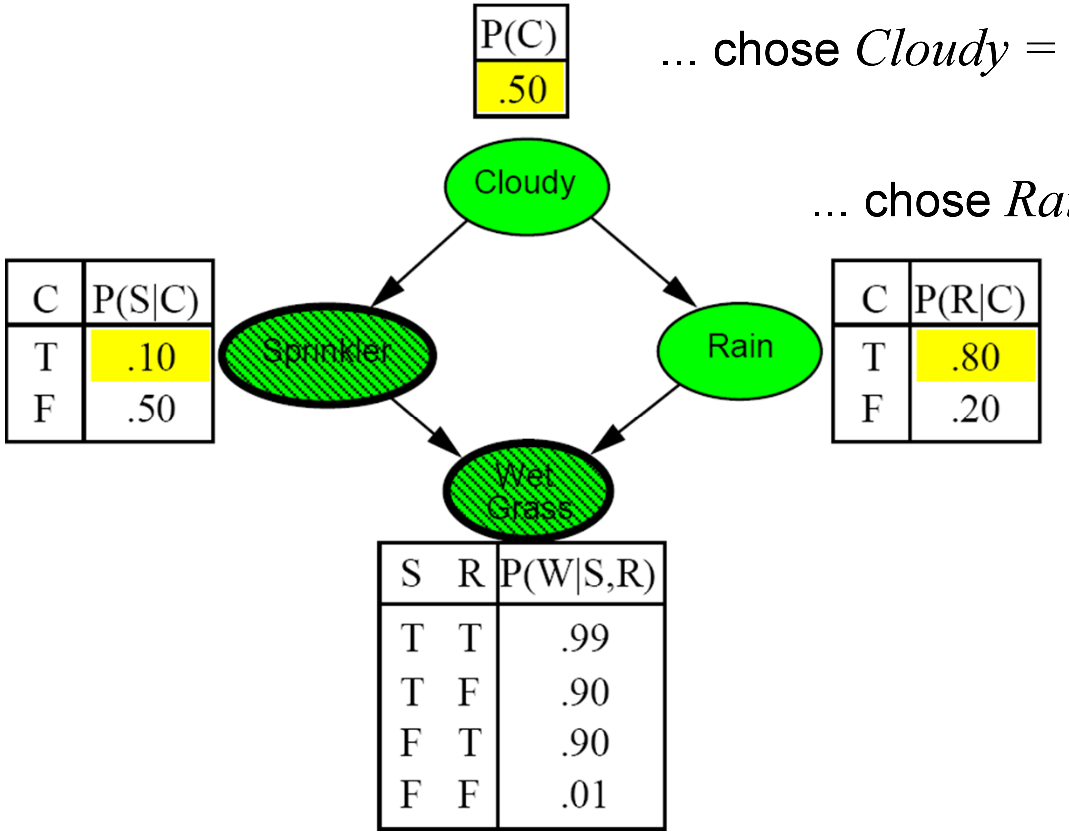
S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

Step 4: randomly choose a value for $Rain$ (acc. to prob. distrib. $\mathbf{P}(R|cloudy)=[0.8,0.2]$)

➔ **Step 3:** set weight $w = w \times 0.1 = 0.1$

Likelihood Weighting: Example

Query: $\mathbf{P}(Rain \mid Sprinkler = true, WetGrass = true) = \mathbf{?}$



P(C)
.50

C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

... chose $Cloudy = true$

... chose $Rain = true$

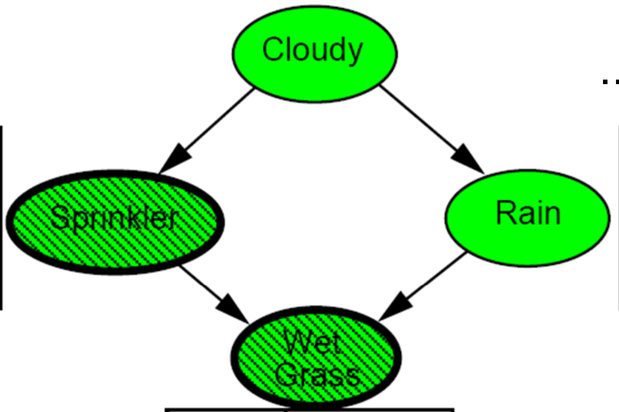
weight $w = 0.1$

Likelihood Weighting: Example

Query: $\mathbf{P}(Rain|\ Sprinkler = true,\ WetGrass = true) = \mathbf{?}$

P(C)
.50

... chose $Cloudy = true$



... chose $Rain = true$

C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

if $Sprinkler = true$ and $Rain = true$, the prob. of $WetGrass = true$ is 0.99

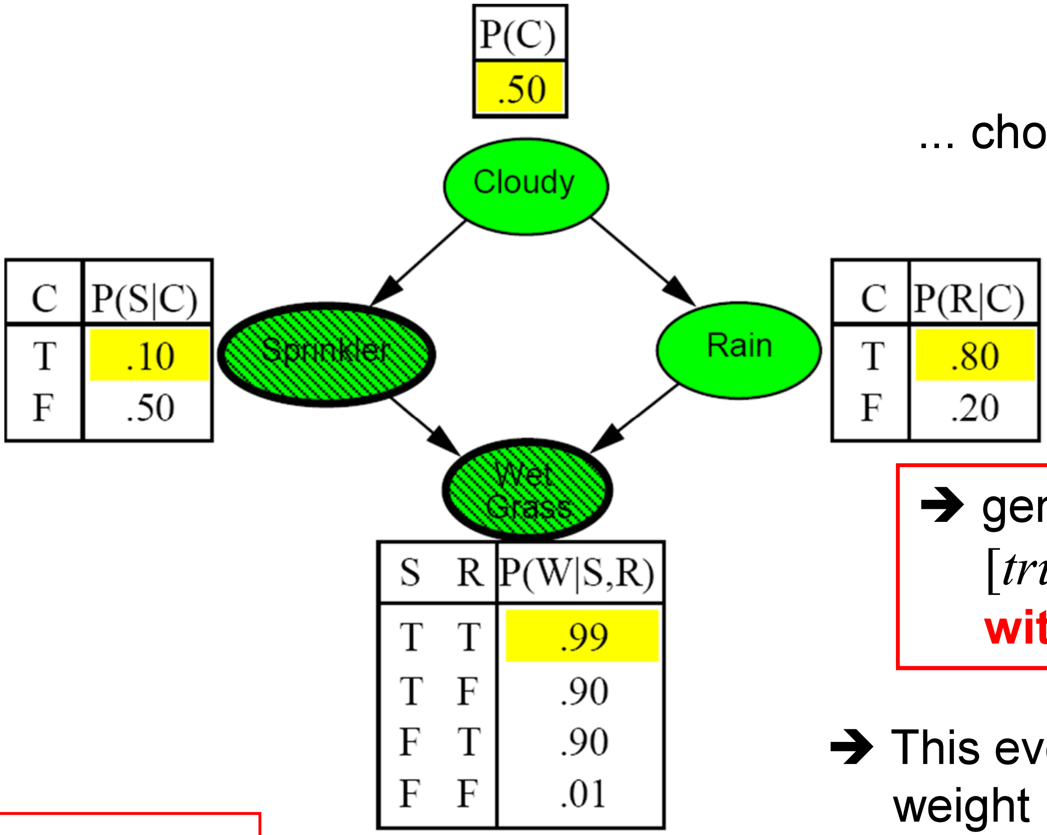
➔ **Step 5:** set weight $w = w \times 0.99 = 0.099$

Likelihood Weighting: Example

Query: $\mathbf{P}(Rain|\ Sprinkler = true,\ WetGrass = true) = \mathbf{?}$

... chose $Cloudy = true$

... chose $Rain = true$



➔ generated event $[true,true,true,true]$ **with weight 0.099**

➔ This event will contribute a weight of 0.099 to $\hat{P}(Rain = true \mid Sprinkler = true, WetGrass = true)$

weight $w = 0.099$

Answering Probabilistic Queries via Sampling (2): Likelihood Weighting

Advantage of Likelihood Weighting:

- Generates only samples consistent with given evidence
  ➔ not as wasteful as rejection sampling

Problem with Likelihood Weighting:

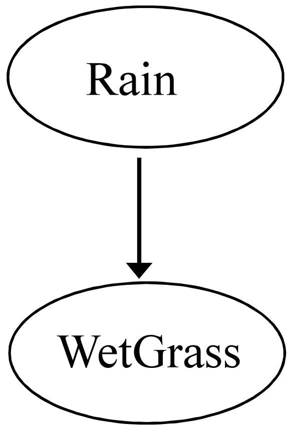
- Evidence guides the sampling decisions only if it appears among the parents/ancestors of a variable
- Evidence that is "downstream" influences result only through the weight
- Weights decay exponentially with number of downstream evidence variables

➔ When a lot of the evidence in the net is "downstream", most of the generated samples will have extremely small weights (i.e., are highly non-representative) ➔ need to sample for a long time until we happen to generate a high-weight sample that is typical of the evidence

➔ Essentially, we sample from a distribution that is closer to the prior $\mathbf{P}(\mathbf{Z})$ than the posterior $\mathbf{P}(\mathbf{Z} \mid \mathbf{E}=\mathbf{e})$ …

The Problem with Likelihood Weighting: An Example

Consider the following model of the climate in Death Valley:



P(rain)
.0001

R	P(wet|R)
T	1.0
F	0.00005

… and the query **P**(R|wet)
(“what is the probability that it rained or did not rain, given that the grass is wet”)

➔ LW would start by sampling variable Rain and might produce thousands of samples representing the less probable situation (Rain=false,Wet=true) (and correctly attach the low weight 0.00005 to each of them) before it happens to generate even *one* sample (Rain=true,Wet=true) (with weight 1.0) that is much more likely given the evidence …

The true ans

* Exercise: Compute the true answer **P**(R|wet)

Answering Probabilistic Queries via Sampling (3): Markov Chain Monte Carlo (MCMC) Sampling

Basic Idea:

Try to “fix” (improve) a (possibly non-representative) sample by repeatedly re-sampling some of the variables that were generated early in the process, taking into account the current values of all the other variables.
➔ iterative process

Simplest variant: “Gibbs Sampling”

- Fix the values of the evidence variables $\mathbf{E} = \mathbf{e}$
- Sample only remaining variables $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$
  ➔ guarantees that each generated event is consistent with the evidence
- Then: iterate N times:
  - for each Z_i in $\mathbf{Z}$ do: *)
    re-sample value of Z_i , assuming all other values in the net as given
- Return the final sample after N re-sampling iterations

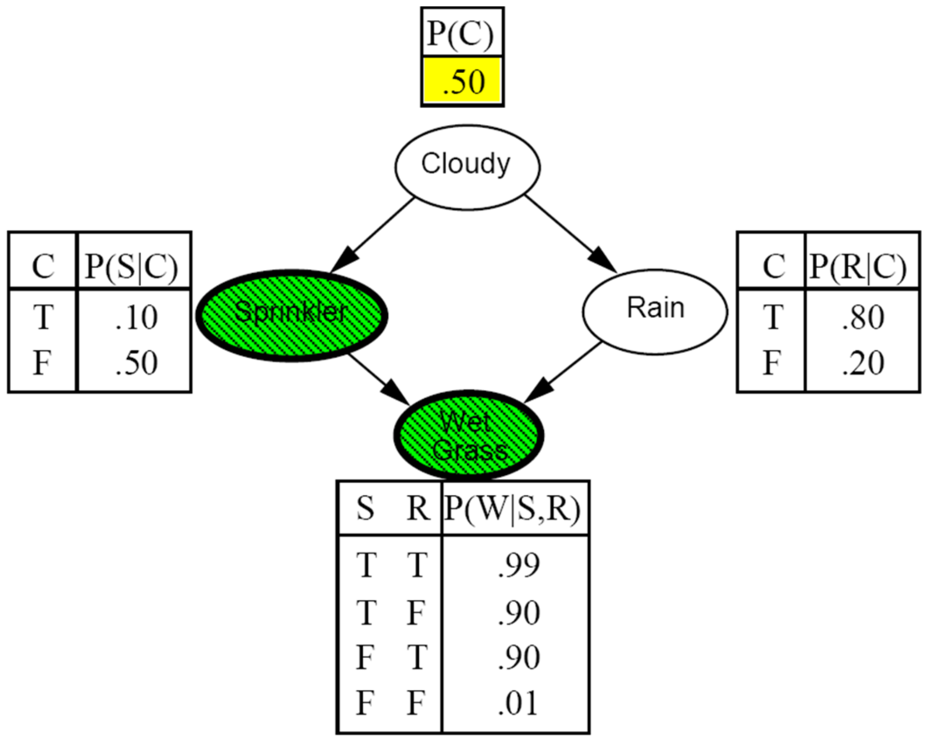
*) *Variables can be resampled in an arbitrary (possibly random) order*

This part not treated in class – ignore for exam!

Gibbs Sampling

Query:

$\mathbf{P}(Rain \mid Sprinkler = true, WetGrass = true) = ?$



P(C)
.50

C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

Assume we sampled the following event in a first pass: $[cloudy, sprinkler, \neg rain, wet]$

(rather unlikely given the network:
$P(sprinkler \mid cloudy) = 0.1$,
$P(wet \mid sprinkler, \neg rain) = 0.9$
➔ would get weight 0.09 in likelihood weighting)

Gibbs re-sampling step 1:
re-sample (say) *Rain*, given fixed values $[cloudy, sprinkler, wet]$ (see next page):
➔ might produce $Rain = true$

Gibbs re-sampling step 2:
re-sample *Cloudy*, given fixed values $[rain, sprinkler, wet]$
➔ might produce $Cloudy = false$

➔ new event $[\neg cloudy, sprinkler, rain, wet]$ is much more likely given the evidence (would get a weight of $0.5 \times 0.99 = 0.495$ in likelihood weighting)

This part not treated in class – ignore for exam!

How to re-sample a variable, given fixed values for all others?

Example: re-sample *Rain*, given fixed values [*cloudy*, *sprinkler*, *wet*] :

$$\boxed{\mathbf{P}(R\,|\,cl,spr,w) =}$$

$$= \frac{\mathbf{P}(R\wedge(cl,spr,w))}{P(cl,spr,w)} =$$

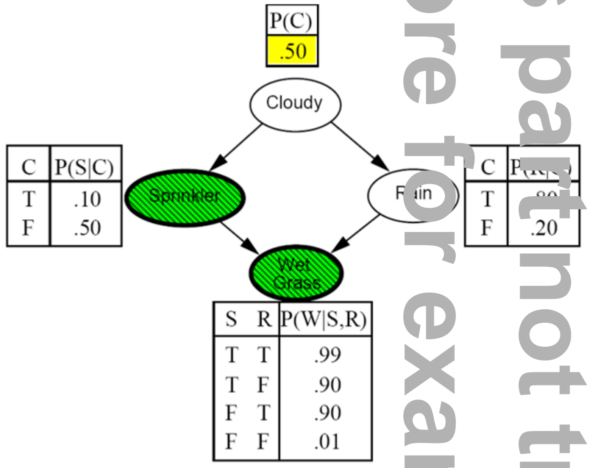
$$= \frac{\mathbf{P}(R,cl,spr,w)}{\sum_{r^\pm} P(r^\pm,cl,spr,w)} =$$

$$= \frac{P(cl)P(spr\,|\,cl)\mathbf{P}(R\,|\,cl)\mathbf{P}(w\,|\,spr,R)}{\sum_{r^\pm} P(cl)P(spr\,|\,cl)P(r^\pm\,|\,cl)P(w\,|\,spr,r^\pm)} =$$

$$= \frac{P(cl)P(spr\,|\,cl)\mathbf{P}(R\,|\,cl)\mathbf{P}(w\,|\,spr,R)}{P(cl)P(spr\,|\,cl)\times\sum_{r^\pm} P(r^\pm\,|\,cl)P(w\,|\,spr,r^\pm)} =$$

$$= \frac{\mathbf{P}(R\,|\,cl)\mathbf{P}(w\,|\,spr,R)}{\sum_{r^\pm} P(r^\pm\,|\,cl)P(w\,|\,spr,r^\pm)} =$$

$$\boxed{=\alpha\times\mathbf{P}(R\,|\,cl)\mathbf{P}(w\,|\,spr,R)}$$



➔ To re-sample *Rain*: compute $P(R|cl)P(w|spr,R)$ for all possible values of *Rain* (i.e., *true*, *false*) and probabilistically sample a new value for *Rain* acc. to the resulting distribution

In our case: $\mathbf{P}(R|cl)\mathbf{P}(w|spr,R) = [0.815, 0.185]$

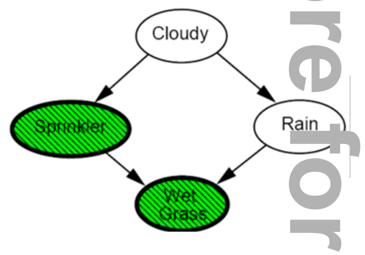
➔ choose value [*true*,*false*] probabilistically acc. to ratio $815 : 185$

➔ might now choose *Rain*=*true*

This part not treated in class – ignore for exam!

How to re-sample a variable, given fixed values for all others?

Example: re-sample $Rain$, given fixed values $[cloudy, \underline{sprinkler}, \underline{wet}]$:



$$\boxed{\mathbf{P}(R \mid cl, spr, w) =}$$

$$= \frac{\mathbf{P}(R \wedge (cl, spr, w))}{P(cl, spr, w)} =$$

$$= \frac{\mathbf{P}(R, cl, spr, w)}{\sum_{r^\pm} P(r^\pm, cl, spr, w)} =$$

$$= \frac{P(cl)P(spr \mid cl)\mathbf{P}(R \mid cl)\mathbf{P}(w \mid spr, R)}{\sum_{r^\pm} P(cl)P(spr \mid cl)P(r^\pm \mid cl)P(w \mid spr, r^\pm)} =$$

$$= \frac{P(cl)P(spr \mid cl)\mathbf{P}(R \mid cl)\mathbf{P}(w \mid spr, R)}{P(cl)P(spr \mid cl) \times \sum_{r^\pm} P(r^\pm \mid cl)P(w \mid spr, r^\pm)} =$$

$$= \frac{\mathbf{P}(R \mid cl)\mathbf{P}(w \mid spr, R)}{\sum_{r^\pm} P(r^\pm \mid cl)P(w \mid spr, r^\pm)} =$$

$$\boxed{= \alpha \times \mathbf{P}(R \mid cl)\mathbf{P}(w \mid spr, R)}$$

Insight 1:
- to compute new re-sampling distribution for some variable X, we need only CPTs that contain X ;
- the computation depends only on the variables in X's "Markov Blanket": X, its parents, its children, & their parents

➔ **efficiently computable!**

Insight 2:
- (downstream) evidence influences re-sampling (e.g., new value for R is influenced by spr and w)

➔ propagation of evidence information

➔ **new distribution we sample from is closer to posterior distribution**

Why Gibbs Sampling Works

Intuitive Explanation: See *Insight 2* on previous page

Formal Explanation:

- Gibbs Sampling is a special case of ***Markov Chain Monte Carlo (MCMC) Sampling***
- Sequence of Gibbs sampling steps forms a *Markov Process* or *Markov Chain**), where each state corresponds to a complete assignment of values to all the variables, and the transition probability is $\mathbf{P}(V \mid \text{all other variables})$
- It can be proven (exercise ☺) that the chain converges to a "stationary distribution" that is equivalent to the posterior distribution $\mathbf{P}(\mathbf{X}|\mathbf{e})$ that we ideally wish to sample from

➔ **Procedure:** to obtain a set of samples, first run the Gibbs Sampler for a while (to permit the chain to (nearly) converge to the true posterior distribution), then start collecting samples. Take these as directly representing the distribution $\mathbf{P}(\mathbf{X} \mid \mathbf{E}=\mathbf{e})$ [➔ no weighting necessary as in Likelihood Weighting]

**) [simplified:] A (first-order) Markov chain describes a discrete time process where (the probability of) the system's state at time t depends only on the previous state.*
(More about this in "Machine Learning & Pattern Classification" in SS, "Probabilistic Models" in WS)

This part not treated in class – ignore for exam!

Learning Bayesian Networks (just briefly; too complex for this class)

This part not treated in class – ignore for exam!

Basic problem:

- For many tasks, it is impossible to specify an appropriate BN “by hand”:
  - experts not available or too expensive
  - required knowledge not available (problem not well enough understood)
  - complex networks need large numbers of parameters

➔ **Goal:**

- Agent should learn a good network from experience / observations
- *Observations (= training examples):* specific events observed by the agent (value assignments to random variables that occur at the same time)

Assumptions:

- The random variables used to describe the agent’s world are pre-defined by the system designer
- Training examples are *“i.i.d.”* (*i*ndependent from each other, and *i*dentically *d*istributed [i.e., produced by the same (hidden) process])

Learning Bayesian Networks

This part not treated in class – ignore for exam

Learning Tasks:

1. ***Structure Learning:*** given a sequence of observations (events) in terms of a set of random variables, infer both the (causal) structure of the domain (i.e., the optimal structure of the BN = factorisation of the joint distribution) and the parameters (conditional probability distributions)

2. ***Parameter Learning:*** given a fixed graph structure of a BN (e.g., provided by a domain expert), and a sequence of observations (events), learn the parameters of the model (i.e., all the conditional probabilities in the CPTs). Two scenarios:
   A. Learning *from complete data* (fully observed events – in each observation, the values of all random variables are known)
   B. Learning *from incomplete data* (partially observed events – the values of some of the random variables are not known)

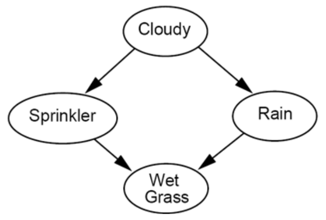
Centeral Question: what is a "good", "correct", "optimal" network???

➔ see next slide: "Maximum Likelihood Principle"

This part not treated in class – ignore for exam!

Parameter Learning (1): Learning from Complete (Fully Observed) Data

Task:



- given the graph structure of a BN N (e.g.,) and a sequence of training observations E (e.g., [¬cloudy,sprinkler,rain,wet], [cloudy, ¬sprinkler, ¬rain, wet], …), learn parameters for N (i.e., conditional probabilities for the CPTs) that "fit" the observations.

Criterion: What is a "good" set of parameters?

- Intuitively: A network N is good if the full joint probability distribution it represents is the same as / similar to the empirical distribution implicit in the training examples
- In other words: A network N is good if the probability that it would generate events like the ones in the training set E is high
- Formally: The ***likelihood*** $L(N : E)$ of a model (network) N relative to a training set E of events is the probability that N would generate E:

$$L(N:E) = P(E \mid N) = (because\, E\; are\; i.i.d.) = \prod_{e \in E} P(e \mid N)$$

➔ Goal of parameter learning is to find the parameters that give the maximally likely model

➔ **Maximum Likelihood Principle**

Parameter Learning (1): Maximising the Likelihood for Complete (Fully Observed) Data

$$L(N:E) = P(E \mid N) = \prod_{e \in E} P(e \mid N) =$$

$$= \prod_{e \in E} \prod_{i} P(x_i[e] \mid Parents(X_i)[e]) =$$

$x_i[e]$ … value of variable X_i in evidence example (observation) e

$$= \prod_{i} \left[\prod_{e \in E} \underbrace{P(x_i[e] \mid Parents(X_i)[e])}_{\text{parameters we wish to estimate (CPTs) …}} \right]$$

Consequences:

- We can maximise the parameters for each random variable X_i separately
- For each random variable X_i , we need to find the parameters (conditional probabilities) that maximise the product of $\mathbf{P}(X_i \mid Parents(X_i))$ in the given set of training observations
- It is easy to prove that for discrete variables, the estimates that maximise this are the *relative frequencies* as observed in the training data (how often each value of X_i co-occurs with each value combination of the parents of X_i)
- In other words: we can estimate all the parameters by *counting* in E

➔ Parameter learning from fully observed data in Bayes Nets is fairly simple

This part not treated in class – ignore for exam!

Parameter Learning (2): Learning from Incomplete (Partially Observed) Data

Task:

- given the graph structure of a BN N and a sequence of *incomplete* training observations E (e.g., [¬cloudy,?,?,wet], [cloudy,?,?,wet], …), learn parameters for N (i.e., conditional probabilities for the CPTs) that "fit" the observations.

Motivation:

- In many applications, only a subset of the variables used to model the world may be observable (e.g., in robot navigation and sensing)
- Measurements can be missing because of failure of measuring sensor

➔ Learning from incomplete observations is a difficult optimisation problem (find parameters that maximise the likelihood relative to incomplete data)

➔ Some general-purpose methods exist (Gradient Ascent search, Expectation Maximisation, ...)

➔ Too complex for this class.

Structure Learning

Task: Given only a sequence of training observations (events) E (e.g., [¬cloudy,sprinkler,rain,wet], [cloudy, ¬sprinkler, ¬rain,wet]), infer both:

- ***the (causal) structure of the domain*** (which variables depend on which others, or should be modelled as depending on which others), and
- **the optimal set of parameters** for this structure, relative to the training set E

➔ **Optimisation / search problem:**

- Define a quality criterion to be optimised (e.g., the likelihood of a model relative to the training data)
- Search the space of possible graph structures and parameter assignments for a model that maximises this quality criterion

➔ huge search space
➔ difficult search problem
➔ fairly effective search/learning algorithms exist

This part not treated in class – ignore for exam!

Summary

Bayesian Networks (BNs) are a natural representation for (causally induced) conditional independence relations

BNs represent a **full joint probability distribution** in a compact way

Exact inference with BNs is possible, but costly

Efficient approximate reasoning with BNs: stochastic sampling algorithms

Automatic learning of BNs: difficult problem, but a variety of learning algorithms exist

BNs are **extensively used in many practical modelling & prediction problems** (computer vision, robotics, system diagnosis, …, …)

There are many **more types of probabilistic networks** for knowledge representation and inference (for both discrete and continuous domains), with a lot of scientific literature ….