Reasoning and Decision-Making under Uncertainty

7. Session:
Inference Algorithms for Bayesian Networks

Prof. Dr.-Ing. Stefan Kopp
Center of Excellence „Cognitive Interaction Technology“
AG Sociable Agents

CITEC

Sociable Agents

Recap‘: drawing inferences

Drawing common inferences through applying four basic queries:

- probability of evidence: $Pr(\mathbf{e})=?$
- prior/posterior marginals: $Pr(x_1,\ldots,x_m|\mathbf{e})=?$
- most probable explanation (MPE): $\mathbf{x}=?$ with $Pr(x_1,\ldots,x_n|\mathbf{e})=max$
- maximum a posteriori hypothesis (MAP): $\mathbf{x}=?$ *with* $Pr(x_1,\ldots,x_m|\mathbf{e})=max$

What algorithms are needed?

- optimize some or all node values (variable) so as to maximize probabilities in the network
- <u>calculate probabilities of some or all nodes in the network given values of certain nodes (evidence)</u>

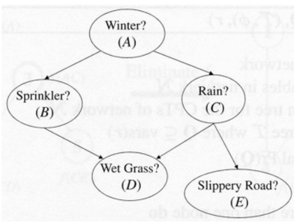
Recap‘: drawing inferences

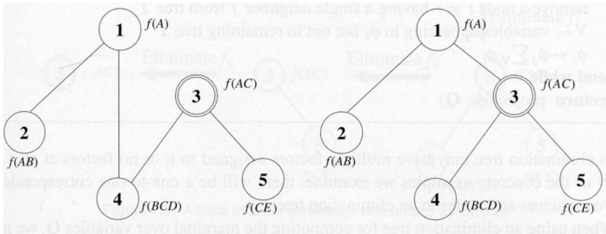
Variable elimination

- combination of summing out variables & multiplying factors (~chain rule)
- order can be arbitrary, but differences regarding costs (~number of variables in biggest factor created during elimination)

Factor elimination

- eliminate all factors except one that contains the query variable(s)
- elimination orders become elimination trees (why?)
- any elimination trees is valid, but differences regarding costs

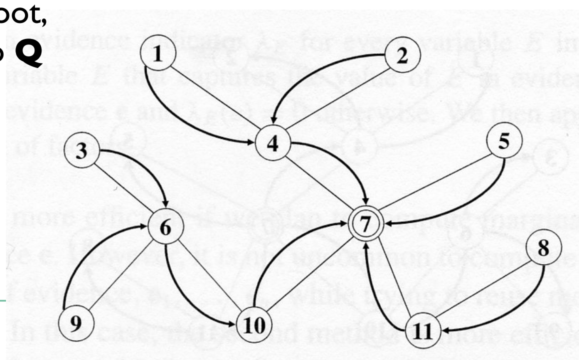




Factor elimination as message passing

Using elimination trees for computing prior marginal over ***Q****:*

- pick one node *r* with **Q** ⊆ *vars(r)* as root node
- eliminate a factor Θ_i if all its neighbors, except the one closer to the *root*, have been eliminated
- when a node *i* is about to be eliminated, it will have a single neighbor *j* and *i*‘s factor is projected and multiplied into factor of
  - viewed as „passing a message from *i* to *j*“
- push messages toward the root
- when all messages are available in root, multiply with factor *r* and project to **Q**



Jointree algorithm

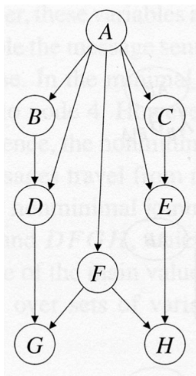
Definition: A jointree $(T, \mathbf{C})$ for a DAG G is a tree T in which each node has a label $\mathbf{C}_i$ (called cluster), satisfying the properties:

- each cluster is a set of nodes from G
- each *family** in G appears in some cluster (**node along with its parents)*
- if a node appears in two clusters $\mathbf{C}_i, \mathbf{C}_j$, it must appear in *every* cluster on the path connecting nodes *i* and *j* („jointree property“)

Further notes and definitions:

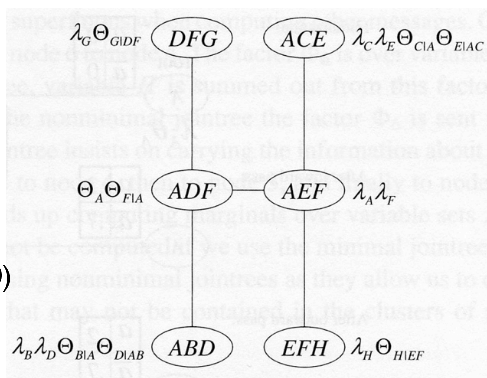
- the separator of edge *i-j* is defined as $S_{ij} := C_i \cap C_j$
- the width of a jointree is the size of its largest cluster minus one
- also known as junction trees, clique trees, Markov trees, hypertrees
- evidence indicator is a factor over variable *X* that captures the value of *X* in evidence **e**: $\lambda_X(x) = 1 \text{ if x consistent with } \mathbf{e}, 0 \text{ otherwise}$

Jointree algorithm



Algorithm:

1. construct jointree for a given Bayesian network
2. assign each CPT $\Theta_{X|\mathbf{U}}$ to a cluster that contains X and **U**
3. assign each evidence indicator λ_X to a cluster that contains *X*
4. select a root node that contains the query **Q**
5. start eliminating factors (using projecting and multiplying) inwards/outwards*
6. finally project cluster in the root node onto **Q**



**different propagation strategies with different space and time complexities*

- Shenoy-Shafer architecture (Shenoy & Shafer 1990)
- Hugin architecture (Jensen et al. 1990)

Some remarks on elimination algorithms

Performance is strongly influenced by the network structure (e.g. number of parents per node, loops, paths between nodes)

- thus also called structure-based algorithms

But, the network structure may be simplified

- _goal_: compute *Pr*(**Q,e**) for query variables **Q**, evidence **e**
- complexity of inference affected by number and location of query and evidence var's in the network ➔ query structure
- network pruning by removing nodes and edges not relevant for (**Q,e**)

Recursive conditioning

Idea: simplify a problem by solving a number of cases and combining the results to a solution to the original problem (case analysis)

$$Pr(x) = \sum_c Pr(x, c)$$

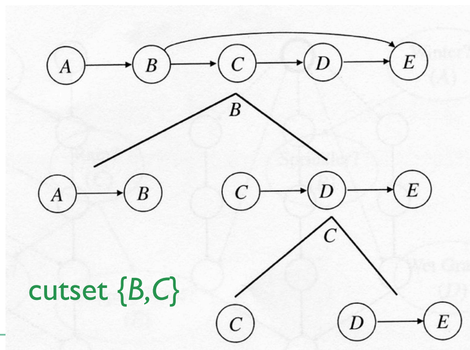
Approach: reduce query on a network into a queries on simpler networks

- if var *E* given as evidence, the network can be pruned
- in general, any query *Pr*(**q,e**) leads to decomposition into networks N_e^r and N_e^l such that

„cutset" **e**

$$Pr(q) = \sum_e Pr(q, e)$$

$$= \sum_e Pr_e^l(q^l, e^l) Pr_e^r(q^r, e^r)$$



Recursive conditioning

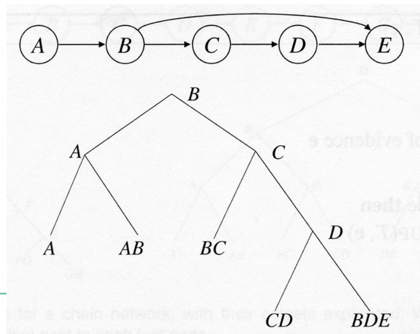
Recursive conditioning algorithm:

- decompose network in a divide-and-conquer fashion, following an appropriate cutset
- when at leaf node, look up the conditioned CPT
- propagate value back according to $\sum_{e} Pr_e^l(q^l, e^l) Pr_e^r(q^r, e^r)$

Question (again): what is an appropriate cutset order?

Answer (again): all are valid, some lead to less work

- need to minimize total number of considered cases
- use decomposition trees: full binary trees, leaves are CPTs in the network
- useful to employ caching techniques (Darwiche, chapt. 8)

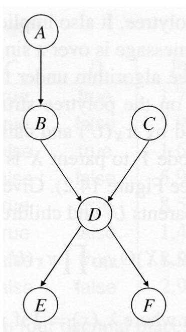


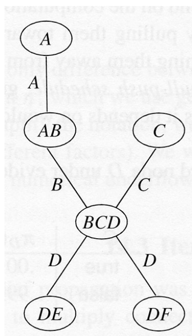
Belief propagation

- Proposed as exact inference in polytree* networks, later generalized to *approximative* solution for arbitrary networks
- trade-off quality with computational costs

*Belief propagation algorithm for computing joint marginals Pr(X,**e**):*

- identical to (exact) jointree algorithm for jointrees that coincide with the polytree network structure
- *Example*:



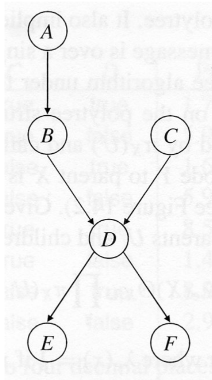


**polytree = network with only one path between any two nodes*

Belief propagation

*Computing joint marginals Pr(X,**e**):*

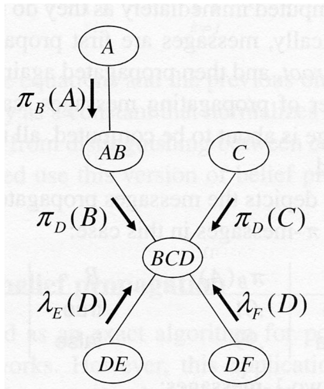
- node *i* in jointree has „cluster“ $C_i = X\boldsymbol{U}$ (with $\boldsymbol{U}$ parents of *X*)
- edge *i-j* in jointree corresp. to edge *X-Y* in network has „separator“ $\boldsymbol{S}_{ij}=X$
- „messages“ to eliminate factors:
  - from U to X: causal support $\pi_X(U)$
  - from Y to parent X: diagnostic support $\lambda_Y(X)$
- messages are sent by a node, when it has received messages from all other nodes
  - start with those that do not depend on others



Example:

- Belief propagation toward node *D*, evidence *E=true*

$$Pr(BCD, \mathbf{e}) = \Theta_{D|BC}\pi_D(B)\pi_D(C)\lambda_E(D)\lambda_F(D)$$



Belief propagation

problem: can lead to „deadlocks“ in *non-polytree* networks when messages are dependent on each other

solution: iterative belief propagation

- assume initial values to each message in the network
- propagate beliefs and re-iterate
- converge to a „fixed point“ solution
  - may generally have multiple fixed points on a given network
  - may oscillate on some networks, loop forever

Stochastic sampling

Idea: simulate an event according to some probability of occurrence, estimate the prob. of this event from its frequency in these simulations

Simulating a Bayesian network:

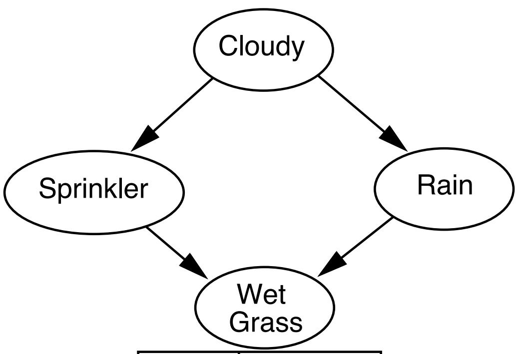
A Bayesian network induces a distribution *Pr(**X**)*

Basic algorithm:

- visit each node in topological order
- generate value for each node according to *Pr(x|**u**)*
- end with a sample $\{\mathbf{x}^1,....,\mathbf{x}^n\}$ of *n* events
- estimate probability ^*Pr(**x**)* of value ***x*** from its frequency in this sample
- show that ^*Pr(**x**)* converges against *Pr(**x**)* with increasing *n*

Example

P(C)
.50



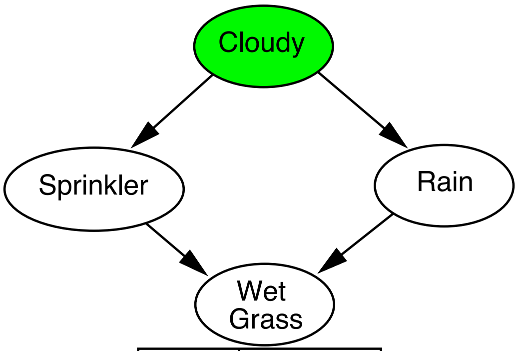
C	P(S|C)
T	.10
F	.50

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

Example

P(C)
.50



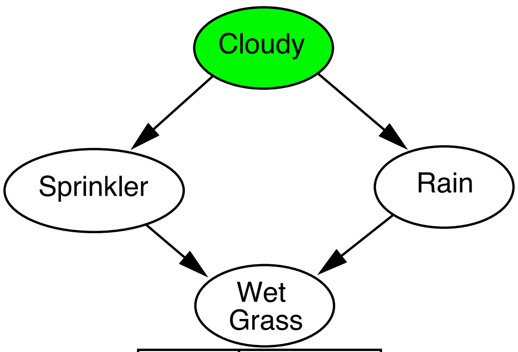
C	P(S|C)
T	.10
F	.50

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

Example

P(C)
.50



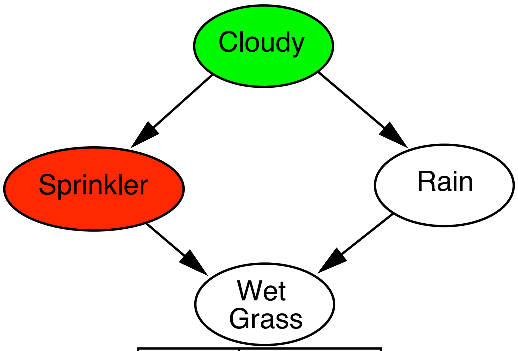
C	P(S|C)
T	.10
F	.50

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

Example

P(C)
.50



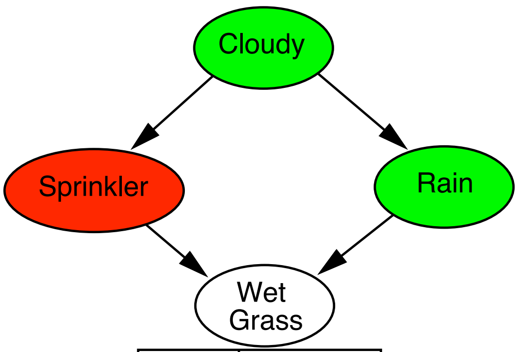
C	P(S|C)
T	.10
F	.50

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

Example

P(C)
.50



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

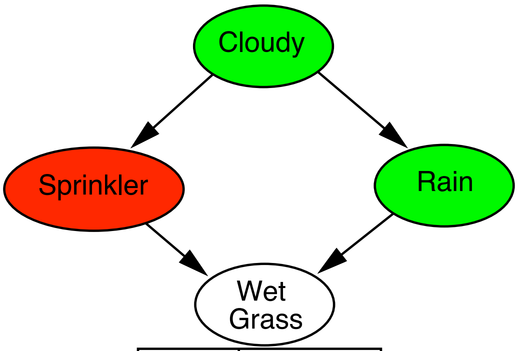
C	P(R|C)
T	.80
F	.20

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

Example

P(C)
.50

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



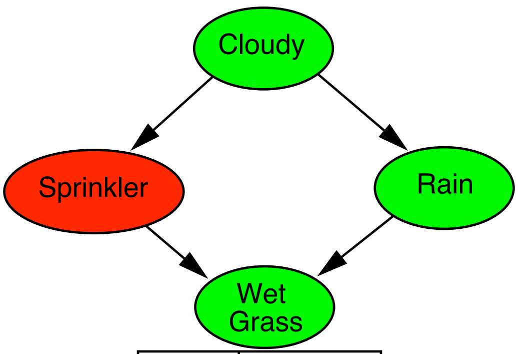
C	P(R|C)
T	.80
F	.20

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

Example

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

returned event:
$\mathbf{x}^i = <T,F,T,T>$

Stochastic sampling

Sampling relies on taking probability as *expectation* about a function

- expectation value of a function *f(**X**)*: $Ex(f) := \sum_x f(x) \cdot Pr(x)$
  μ
- variance of a function *f(**X**)*: $Var(f) := \sum_x (f(x) - Ex(f))^2 \cdot Pr(x)$
  σ^2

Direct sampling function:

- let $\hat{\alpha}(x) := 1$ if α true at $\mathbf{x}$, 0 otherwise
- then: $Ex(\hat{\alpha}) = Pr(\alpha)$

$$Var(\hat{\alpha}) = Pr(\alpha)Pr(\neg\alpha) = Pr(\alpha) - Pr(\alpha)^2$$

That is, approximating *Pr* boils down to estimating the expectation
How?

Monte Carlo simulation

Principle:

- simulate random sample ***x***^{1}, ..., ***x***^{n} from sampling distribution *Pr(**X**)*
- evaluate function at each instantiation *f(**x**^{1}), ..., f(**x**^{n})*
- compute arithmetic average of attained values: sample mean

$$Av_n(f) := \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}^i)$$

Works because of *law of large numbers*: for function *f* with expectation μ and every ϵ >0:

$$\lim_{n \to \inf} P(|Av_n(f) - \mu| \leq \epsilon) = 1$$

Monte Carlo simulation using $\hat{\alpha}(x)$ gives direct sampling:

- simulate sample **x**^{1},.., **x**^{n} from Bayesian network
- compute values $\hat{\alpha}(x^1), \ldots, \hat{\alpha}(x^n)$
- estimate $Pr(\alpha)$ using sample mean $Av_n(\hat{\alpha})$

Rejection sampling

Calculate conditional prob. *Pr(a|b)* with *Pr(.)* induced by network

Idea:

- calculate estimate for *Pr(a∧b)* and *Pr(b)*: $Av_n(\hat{\gamma}), Av_n(\hat{\beta})$ with $\gamma = \alpha \wedge \beta$
- take ratio as estimate for *Pr(a|b)*: $Av_n(\hat{\gamma})/Av_n(\hat{\beta})$
  - c_1=#samples with $a \wedge b$=true, c_2=#samples with b=true ➔ $(c_1/n)/(c_2/n)=c_1/c_2$
- reject all samples in which *b* is false: rejection sampling

Example:

estimate *P(Rain|Sprinkler=true)* from 100 samples; 27 have *Sprinkler=true*, of these 8 have *Rain=true*, 19 have *Rain=false*

$$\hat{\mathbf{P}}(Rain|Sprinkler = true) = \text{NORMALIZE}(\langle 8, 19 \rangle) = \langle 0.296, 0.704 \rangle$$

True answer: <0.3,0.7>

Importance sampling

Idea: reduce variance due to rare events by sampling from an importance distribution *Pr'* emphasizing instantiations consistent with rare event

Monte Carlo simulation using the importance sampling function:

$$\tilde{\alpha}(x) = Pr(x)/Pr'(x) \text{ if } \alpha \text{ true at instantiation } x, 0 \text{ otherwise}$$

Improves on direct sampling only when *Pr'* emphasizes important events no less than *Pr*

Problem: Finding ideal distribution generally not feasible, but some other weaker conditions can be ensured easier and still improve on variance

Likelihood weighting

Given evidence **e**, what is *Pr(***x***|***e***)*?

Idea: generate only samples consistent with **e** by sampling non-evidence var‘s and weighting samples by likelihood they accord with **e**

- consistent estimate, but performance drops with growing evidence

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($\mathbf{W}[X]$)

---

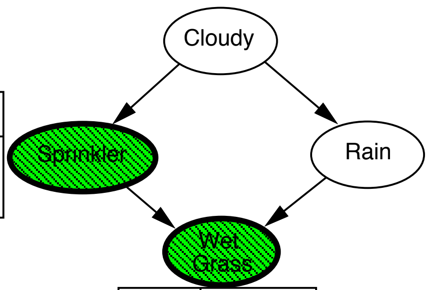
function WEIGHTED-SAMPLE($bn, \mathbf{e}$) **returns** an event and a weight

  $\mathbf{x} \leftarrow$ 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

Example

Query: **P**(Rain | Sprinkler=true, WetGrass=true) = ??

P(C)
.50

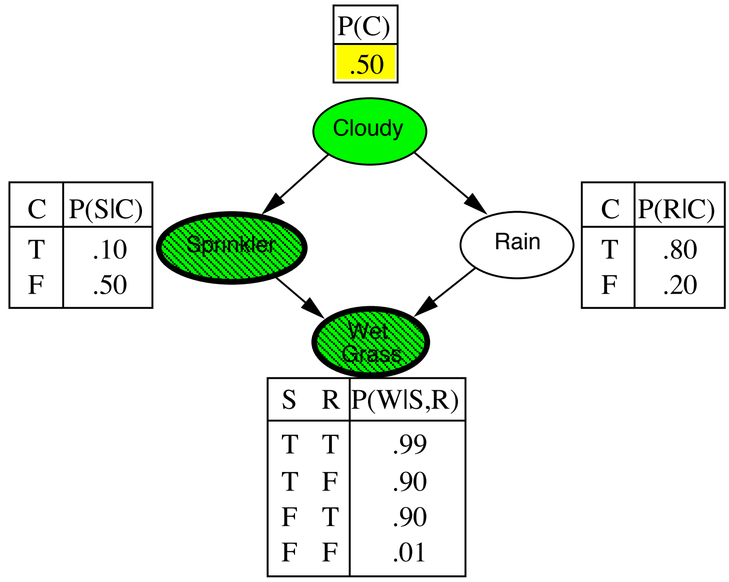


C	P(S|C)
T	.10
F	.50

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

$w = 1.0$



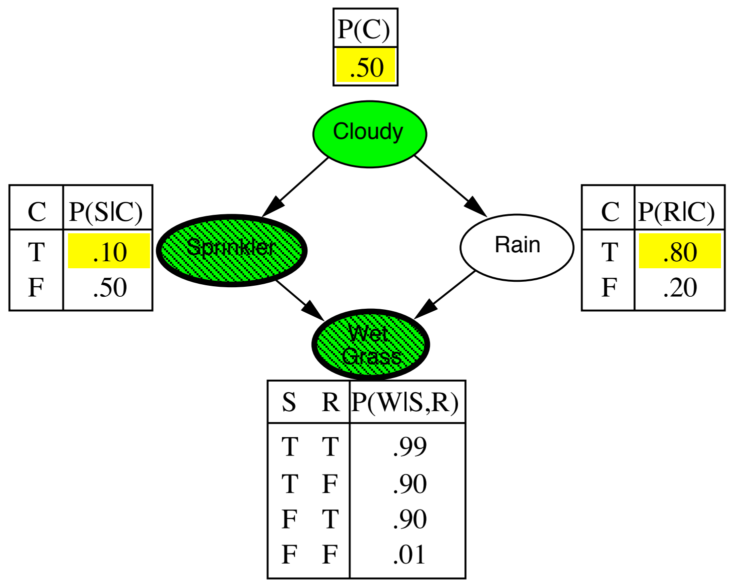
P(C)
.50

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

$w = 1.0$



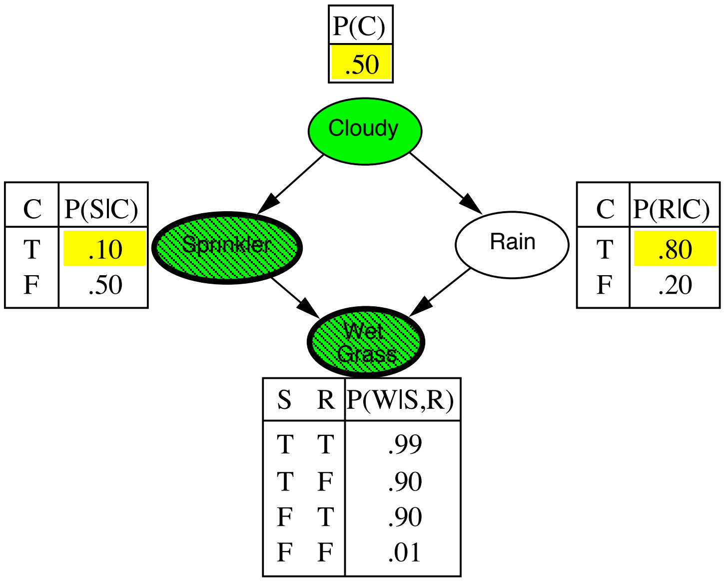
P(C)
.50

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

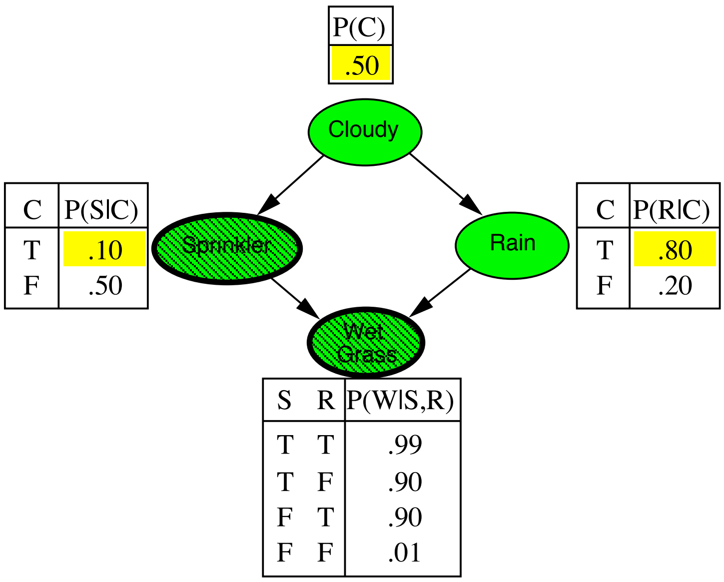
C	P(R|C)
T	.80
F	.20

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

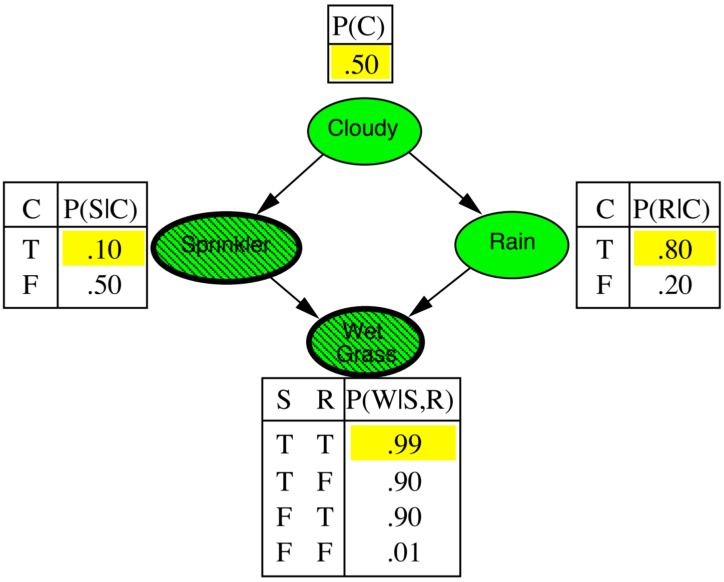
$w = 1.0$



$w = 1.0 \times 0.1$



$w = 1.0 \times 0.1$



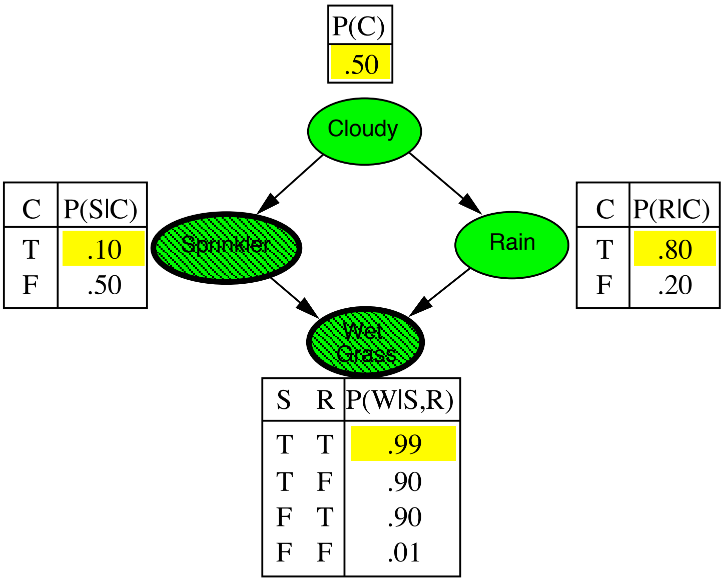
P(C)
.50

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

$w = 1.0 \times 0.1$



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

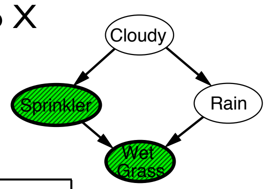
$w = 1.0 \times 0.1 \times 0.99 = 0.099$ = weight for event <t,t,t,t>

Markov Chain Monte Carlo (MCMC)

Network is in a state = current assignment to variables

Next state: sample *non-evidence variable* X given its Markov blanket
= variables that, when known, make other variables irrelevant to X

- Markov blanket of *Cloudy* is *Sprinkler* and *Rain*
- Markov blanket of *Rain* is *Sprinkler*, *Cloudy*, *WetGrass*



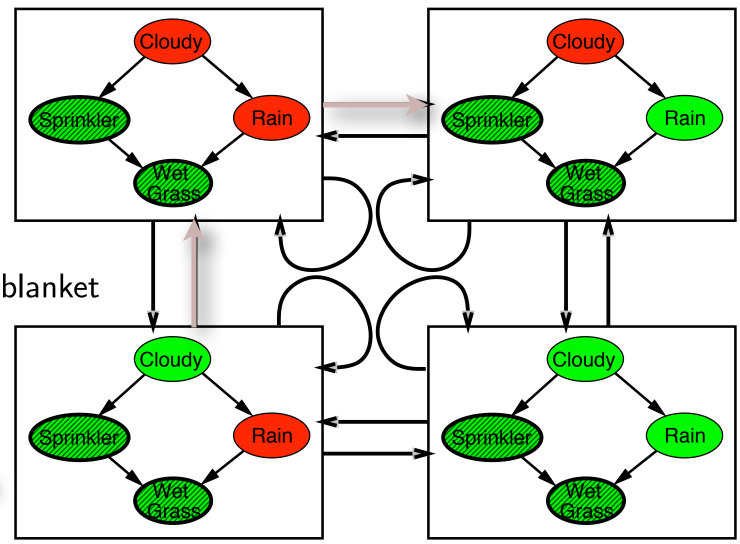
function MCMC-ASK(X, $\mathbf{e}$, bn, N) **returns** an estimate of $P(X|\mathbf{e})$
  local variables: $\mathbf{N}[X]$, a vector of counts over X, initially zero
    $\mathbf{Z}$, the nonevidence variables in bn
    $\mathbf{x}$, the current state of the network, initially copied from $\mathbf{e}$

  initialize $\mathbf{x}$ with random values for the variables in $\mathbf{Y}$
  for $j = 1$ to N **do**
    for each Z_i in $\mathbf{Z}$ **do**
      sample the value of Z_i in $\mathbf{x}$ from $\mathbf{P}(Z_i|mb(Z_i))$
        given the values of $MB(Z_i)$ in $\mathbf{x}$
      $\mathbf{N}[x] \leftarrow \mathbf{N}[x] + 1$ where x is the value of X in $\mathbf{x}$
  return NORMALIZE($\mathbf{N}[X]$)

„transition prob.“ of moving into new state

Estimate $\mathbf{P}(Rain|Sprinkler = true, WetGrass = true)$

Sample *Cloudy* given its Markov blanket



Initial state (random)

Count number of times *Rain* is true and false in the samples.

E.g.: visit 100 states: $\hat{\mathbf{P}}(Rain|Sprinkler = true, WetGrass = true)$
$= \text{NORMALIZE}(\langle 31, 69 \rangle) = \langle 0.31, 0.69 \rangle$

MCMC - Markov blanket sampling

Because of the transition probabilities, sampling runs into an equilibrium in which the time spent in each state is proportional to its posterior probability

Transition probability (given the Markov blanket) is:

$$P(x_i'|mb(X_i)) = P(x_i'|parents(X_i))\Pi_{Z_j \in Children(X_i)} P(z_j|parents(Z_j))$$

- easily implemented in parallel systems

Main difficulties:

- difficult to tell if and when convergence has been achieved
- can be wasteful if Markov blanket large, prob‘ doesn‘t change much

Markov process

1856-1922

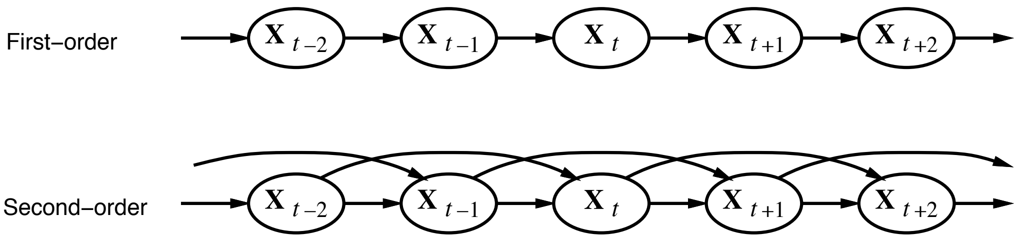
Construct a Bayes net from these variables: parents?

Markov assumption: $\mathbf{X}_t$ depends on **bounded** subset of $\mathbf{X}_{0:t-1}$

First-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$

Second-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-2}, \mathbf{X}_{t-1})$

transition model



Sensor Markov assumption: $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_{0:t}, \mathbf{E}_{0:t-1}) = \mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$

sensor model

Stationary process: transition model $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$ and sensor model $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$ fixed for all t

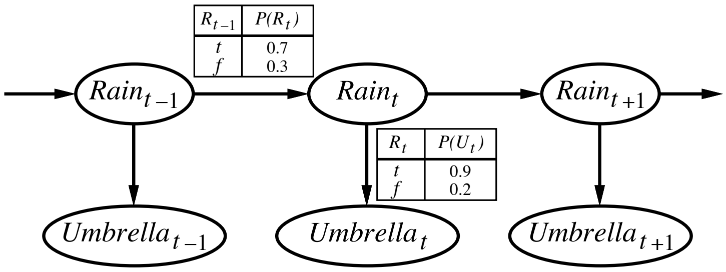
(changes follow fixed law)

Markov process

Need prior probability $P(X_0)$ over states at time 0

Then we have: $P(X_0,X_1,\ldots,X_t,E_1,\ldots,E_t)=P(X_0) \prod_{i=1..t} P(X_i|X_{i-1})P(E_i|X_i)$

Example:



First-order Markov assumption not exactly true in real world!

Possible fixes:

1. **Increase order** of Markov process
2. **Augment state**, e.g., add $Temp_t$, $Pressure_t$

Inference tasks

Filtering: $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$
belief state—input to the decision process of a rational agent

Prediction: $\mathbf{P}(\mathbf{X}_{t+k}|\mathbf{e}_{1:t})$ for $k > 0$
evaluation of possible action sequences;
like filtering without the evidence

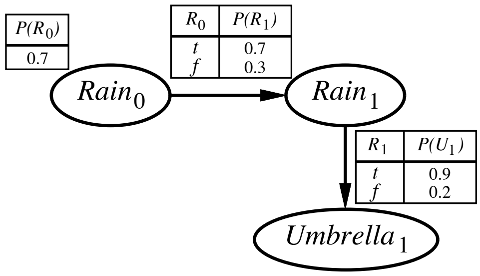
Smoothing: $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ for $0 \leq k < t$
better estimate of past states, essential for learning

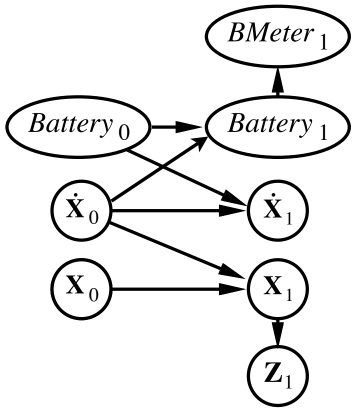
Most likely explanation: $\arg\max_{\mathbf{x}_{1:t}} P(\mathbf{x}_{1:t}|\mathbf{e}_{1:t})$
speech recognition, decoding with a noisy channel
given observations, find sequence of states most likely to have generated them (e.g. Viterbi algorithm)

(see Russell & Norvig, Sect. 15.2 for algorithms)

Dynamic Bayesian networks

$\mathbf{X}_t$, $\mathbf{E}_t$ contain arbitrarily many variables in a replicated Bayes net





Need to give transition and
senor model (stationary)
only for first slice

↓

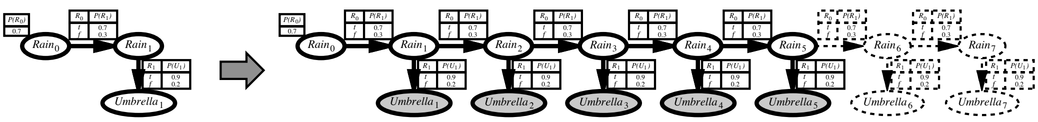
Hidden Markov Model = DBN with a single discrete state variable

Sensor variables Z_t, $BMeter_t$;
state variables X_t, $Xdot_t$, $Battery_t$

Exact inference in DBNs

DBNs are Bayesian networks, i.e., we can use our known algorithms

Exact inference: Unroll network to accommodate all observations and run exact inference algorithm (e.g. variable elimination)



Problem: inference cost for each update grows with t

Rollup filtering: add slice $t+1$, "sum out" slice t using variable elimination

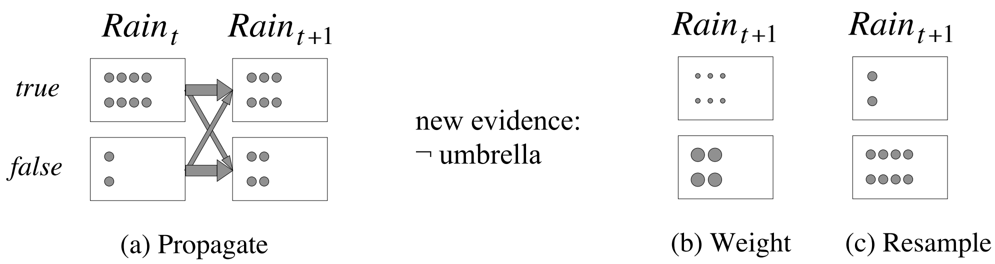
Costs (factor size) almost always exponential in number of state variables

Approximative inference in DBNs

Particle filtering: ensure a population of samples ("particles") that tracks the high-likelihood regions of the state-space

Algorihm: create N samples from prior distribution $P(X_0)$, then cycle...

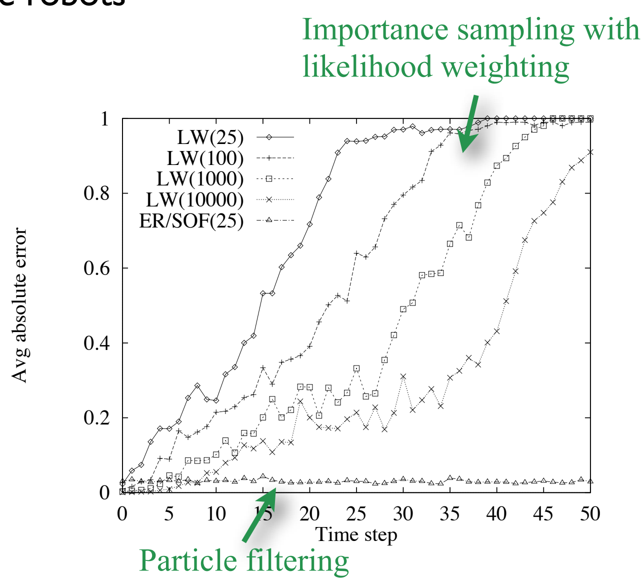
1. *Propagate* by sampling next state x_{t+1}, given x_t and using $P(X_{t+1}|x_t)$
2. *Weight* samples by likelihood it assigns to new evidence $P(e_{t+1}|x_{t+1})$
3. *Resample* new N samples from the current population, probability that sample is replicated proportional to its „weight" (no. samples)



Particle filtering

widely used for tracking nonlinear systems, especially in vision, self-localization, or mapping in mobile robots

- consistent (proof see R&N, Sect. 15.5)
- approximation error remains bounded over time, at least empirically
- in practice efficient, yet no theoretical guarantees (so far)



Bayes nets inference algorithms - summary

Exact algorithms

- Variable Elimination and Factor Elimination
- Jointree algorithm
- Recursive conditioning

Approximative algorithms

- Belief propagation
- Stochastic sampling (Monte Carlo simulation)
  - direct sampling
  - importance sampling, likelihood weighting
- Monte Carlo Markov Chain