Artificial Intelligence 2

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Today program

Bayesian networks

- an efficient way to represent any full joint probability distribution by exploiting conditional independence

Semantics of networks

- representing the full joint probability distribution

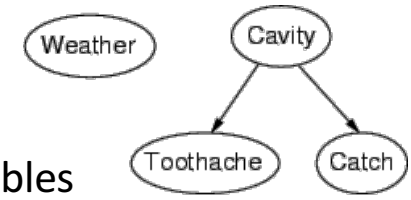
Constructing the networks

Inference in Bayesian networks

- exact inference
  - enumeration, variable elimination
- approximate inference
  - sampling methods (direct, likelihood weighting, MCMC)

Bayesian network

specifies **conditional independence relationships** among random variables



a directed acyclic graph (DAG)

- nodes correspond to random variables
- predecessors of nodes are called parents
- each node X has a conditional probability distribution **P**(X | Parents(X))

alternative names

- belief network, probabilistic network, causal network, knowledge map

An example (burglary detection)

We have a **burglar alarm** installed at home.
It is fairly reliable at detecting a **burglary**,
but occasionally responds to minor **earthquakes**.

Our neighbors Mary and John promised to **call us** when they hear the alarm.

- John nearly always calls when he hears alarm, but sometimes confuses the telephone ringing with the alarm
- Mary likes loud music and often misses the alarm altogether

We would like to **estimate the probability of a burglary given the evidence of who has or has not called**.

Other assumptions:

- neighbors do not perceive burglary directly and they do not notice minor earthquakes
- neighbors do not confer (they are independent)

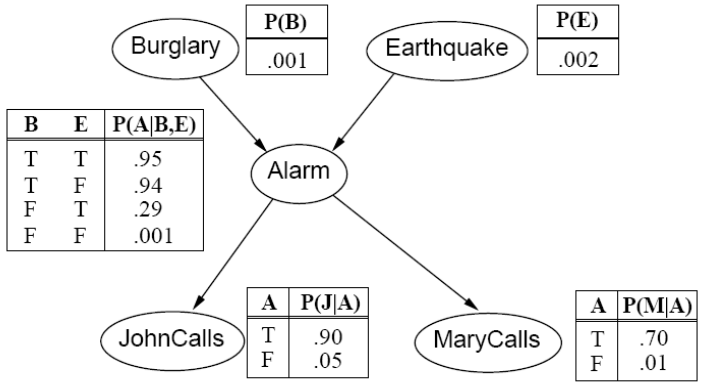
Bayesian network for burglary detection

Random Boolean variables represent possible events.

- some events (the telephone ringing, passing helicopter, alarm failure, …) are ignored

Conditional probability tables (CPTs) describe the conditional probability distributions

- recall that we can keep probability only for values true



The semantics of Bayesian networks

The Bayesian network represents the full joint probability distribution.

$$P(x_1,\ldots,x_n) = \Pi_i\, P(x_i \mid \text{parents}(X_i))$$

Tables **P**(X | Parents(X)) correspond to conditional probability defined by the underlying full joint probability distribution.

Because the full joint probability distribution can be used answer any query (in its domain) we can calculate the same answer using the Bayesian network (via marginalization).

How to build a Bayesian network?

We already have a clue:

$$P(x_1,\ldots,x_n) = \Pi_i P(x_i \mid \text{parents}(X_i))$$

Let us decompose $P(x_1,\ldots,x_n)$ using the chain rule

$$P(x_1,\ldots,x_n) = \Pi_i P(x_i \mid x_{i-1},\ldots,x_1).$$

Then we will get

$$\mathbf{P}(X_i \mid \text{Parents}(X_i)) = \mathbf{P}(X_i \mid X_{i-1},\ldots,X_1)$$

under the condition $\text{Parents}(X_i) \subseteq \{X_{i-1},\ldots,X_1\}$, which is satisfied by numbering the nodes in a way that is consistent with the partial order implicit in the graph structure.

Nodes:
determine the set of random variables that are required to model the domain and order them

- any order will work, but the resulting networks will be different
- a recommended order is such that causes precede effects

Arcs:
choose variables X_i in a given order from 1 to n

- in the set $\{X_1,\ldots,X_{i-1}\}$ choose a minimal set of parents for X_i, such that $\mathbf{P}(X_i \mid \text{Parents}(X_i)) = \mathbf{P}(X_i \mid X_{i-1},\ldots,X_1)$ holds
- for each parent insert a link from the parent to X_i
- write down the conditional probability table $\mathbf{P}(X_i \mid \text{Parents}(X_i))$

Some properties:

- the construction method guarantees that the network is acyclic
- the network does not contain no redundant probability values an so it is always consistent (satisfies the axioms of probability)

Constructing Bayesian networks (notes)

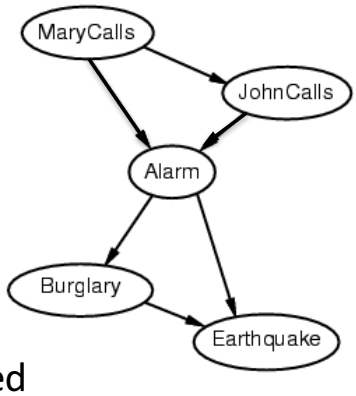
A Bayesian networks can often be far **more compact** than the full joint probability distribution (provided that the network is sparse).

- random variables are often influenced by a few other variables
- assume that each random variable is directly influenced by at most k other variables (and we have n such variables); then the space of representation is
  - $n.2^k$ for Bayesian network
  - 2^n for full joint distribution
- We can also **ignore some slight dependencies**, which makes the network smaller in exchange for less accuracy
  - for example, we assumed that call from Mary and John is driven by the alarm sound only but not for example by earthquake
- naturally we will get a compact Bayesian network only if we **choose the node ordering right**

Constructing Bayesian networks (an example)

Let us use the following order of random variables: MarryCalls, JohnCalls, Alarm, Burglary, Earthquake

- MarryCalls has no parents
- if Marry calls then the alarm is probably active which would make it more likely that John calls
- alarm is probably active if Marry or John calls
- if we know the alarm state then the calls from Marry and John do not influence whether the burglary happened



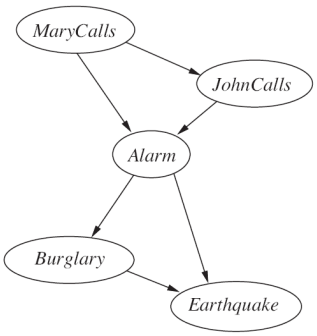
P(Burglary | Alarm, JohnCalls, MarryCalls) = P(Burglary | Alarm)

- the alarm is an earthquake detector of sorts, but if there was a burglary then it explains the alarm and the probability of an earthquake is only slightly above normal

Other orderings of nodes

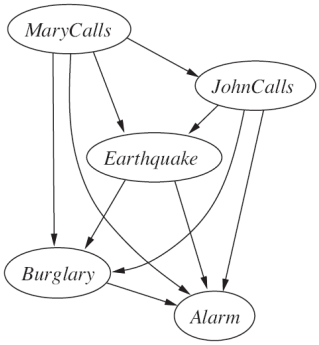
Only two more arcs (in comparison with the previous network) but the problem is how to fill in the CPTs.

- The same problem as using either causal or diagnostic direction.
- It is better to follow the causal direction (causes before effects).
  - leads to smaller networks and easier -to-fill CPTs



When using a wrong ordering we may get big networks, where nothing is saved in comparison to full joint probability distribution.

- MaryCalls, JohnCalls, Earthquake, Burglary, Alarm



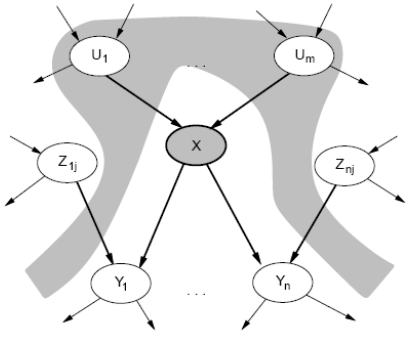
Conditional independence in Bayesian networks

So far we looked at Bayesian networks in terms of the representation of the full joint distribution.

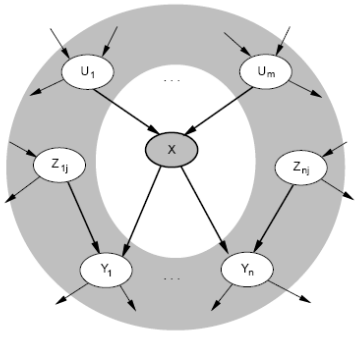
- useful to derive a method for constructing networks

Let us explore topological semantics of networks

a node is conditionally independent of its non-descendants given its parents



a node is conditionally independent of all other nodes given its parents, children, and children's parents (**Markov blanket**)



Inference by enumeration

We introduced the Bayesian networks to **do inference** – to deduce posterior probability of some variable(s) **X** from the query given the values **e** of observed variables (evidence), while having the other variables **Y** hidden.

$$\mathbf{P}(X|\mathbf{e}) = \alpha\, \mathbf{P}(X,\mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X,\mathbf{e},\mathbf{y})$$

the distribution $\mathbf{P}(X,\mathbf{e},\mathbf{y})$ can be computed as follows

$$P(x_1,\ldots,x_n) = \prod_i P(x_i \mid \text{parents}(X_i))$$

We can do some arithmetic tricks by moving some terms $P(x_i \mid \text{parents}(X_i))$ outside the summation.

Inference by enumeration (example)

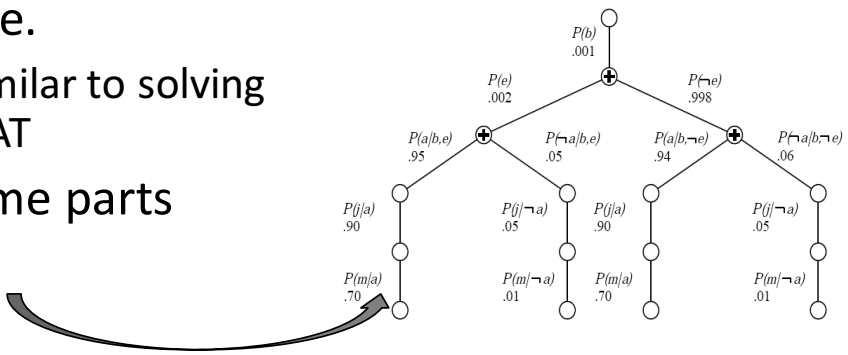
Assume a query about the probability of burglary when both Marry and John calls

$$P(b \mid j,m)$$
$$= \alpha \sum_e \sum_a P(b)\, P(e)\, P(a|b,e)\, P(j|a)\, P(m|a)$$
$$= \alpha\, P(b) \sum_e P(e) \sum_a P(a|b,e)\, P(j|a)\, P(m|a)$$

The structure of computation can be describes using a tree structure.

- it is very similar to solving CSPs and SAT

Notice that some parts are repeated!



Inference by enumeration

```
function ENUMERATION-ASK(X, e, bn) returns a distribution over X
  inputs: X,  the query variable
          e, observed values for variables E
          bn, a Bayesian network with variables {X} ∪ E ∪ Y
  Q(X) ← a distribution over X, initially empty
  for each value x_i of X do
      extend e with value x_i for X
      Q(x_i) ← ENUMERATE-ALL(VARS[bn], e)
  return NORMALIZE(Q(X))

function ENUMERATE-ALL(vars, e) returns a real number
  if EMPTY?(vars) then return 1.0
  Y ← FIRST(vars)
  if Y has value y in e
      then return P(y | Pa(Y)) × ENUMERATE-ALL(REST(vars), e)
      else return Σ_y P(y | Pa(Y)) × ENUMERATE-ALL(REST(vars), e_y)
          where e_y is e extended with Y = y
```

Variable elimination

Enumeration repeats the same parts of the computation.

We can remember the result and reuse it later.

$$\mathbf{P}(B \mid j,m)$$
$$= \alpha\, \mathbf{P}(B) \sum_e P(e) \sum_a \mathbf{P}(a|B,e) P(j|a) P(m|a)$$
$$= \alpha\, \mathbf{f}_1(B) \sum_e \mathbf{f}_2(E) \sum_a \mathbf{f}_3(A,B,E)\, \mathbf{f}_4(A)\, \mathbf{f}_5(A)$$

Factors $\mathbf{f}_i$ are matrices (tables) corresponding to CPTs.

Evaluation will be done **from right to left**.

- **the product of factors** corresponds to the pointwise product (it is not a multiplication of matrices)
- **summing out a variable** is done by adding up the sub-matrices formed by fixing the variable to each of its values in turn

Operations on factors

The pointwise **product** of two factors yields a new factor whose variables are the union of the variables from the original factors.

$f(X_1,\ldots,X_j,Y_1,\ldots,Y_k,Z_1,\ldots Z_l) = f(X_1,\ldots,X_j,Y_1,\ldots,Y_k) \cdot f(Y_1,\ldots,Y_k,Z_1,\ldots Z_l)$

A	B	f_1(A,B)
T	T	0.3
T	F	0.7
F	T	0.9
F	F	0.1

*

B	C	f_2(B,C)
T	T	0.2
T	F	0.8
F	T	0.6
F	F	0.4

=

A	B	C	f_3(A,B,C)
T	T	T	0.06 = 0.3*0.2
T	T	F	0.24 = 0.3*0.8
T	F	T	0.42 = 0.7*0.6
T	F	F	0.28 = 0.7*0.4
F	T	T	0.18 = 0.9*0.2
F	T	F	0.72 = 0.9*0.8
F	F	T	0.06 = 0.1*0.6
F	F	F	0.04 = 0.1*0.4

Then we **sum out** a variable to eliminate it: $\Sigma_a f(A,B,C) = f(B,C)$

A	B	C	f_3(A=T,B,C)
T	T	T	0.06
T	T	F	0.24
T	F	T	0.42
T	F	F	0.28

+

A	B	C	f_3(A=F,B,C)
F	T	T	0.18
F	T	F	0.72
F	F	T	0.06
F	F	F	0.04

=

B	C	f_4(B,C)
T	T	0.24
T	F	0.96
F	T	0.48
F	F	0.31

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 belief network specifying joint distribution P(X_1, ..., X_n)
  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))
```

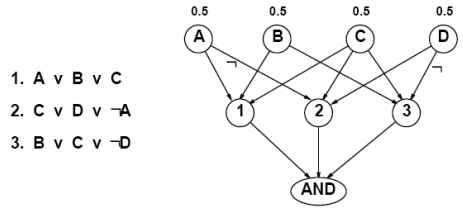
- The algorithm works for any ordering of variables.
- The complexity is given by the size of the largest factor constructed during the operation of the algorithm.
- Eliminate whichever variable minimizes the size of the next factor to be constructed (heuristic).

The complexity of exact inference

If the Bayesian network is a **poly-tree** (there is at most one undirected path between any two nodes in the network), then the time and space complexity is linear in the size of the network that is defined as the number of CPT entries **$O(n.d^k)$**.

For **multiply connected networks**, the complexity is larger:

- 3SAT can be reduced to inference in the Bayesian networks so inference in Bayesian networks is NP-hard



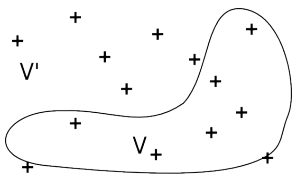
- the problem is as hard as that of computing the number of satisfying assignments for a propositional logic formula, that s **#P-hard**

Approximate inference

Exact inference is intractable for large, multiply connected networks so we may need to consider approximate inference methods based on **Monte Carlo** algorithms.

Monte Carlo algorithms are used to estimate quantities that are difficult to calculate exactly.

- generate many samples
- use statistics to estimate the quantity
- more samples = more accuracy



For **Bayesian networks** we describe two families of algorithms

- direct sampling
- Markov chain sampling

Direct sampling

A **sample** corresponds to an instantiation of random variables.

Each sample should should be generated from a known probability distribution (given by CPTs in the Bayesian network).

- nodes (variables) are taken in topological order
- the probability distribution is conditioned on the values already assigned to parents
- generate a sample value based on this distribution

Let N be the number of samples and $N(x_1,\ldots,x_n)$ be the number of occurrences of event $x_1,\ldots,x_n$, then

$P(x_1,\ldots,x_n) = \lim_{N\to\infty}(N(x_1,\ldots,x_n)/N)$

```
function PRIOR-SAMPLE(bn) returns an event sampled from bn
  inputs: bn, a belief network specifying joint distribution P(X1, ..., Xn)
  x ← an event with n elements
  for i = 1 to n do
      xi ← a random sample from P(Xi | parents(Xi))
           given the values of Parents(Xi) in x
  return x
```

Direct sampling (an example)

Generate a value for Cloudy from the distribution
P(Cloudy) = ⟨0.5, 0.5⟩

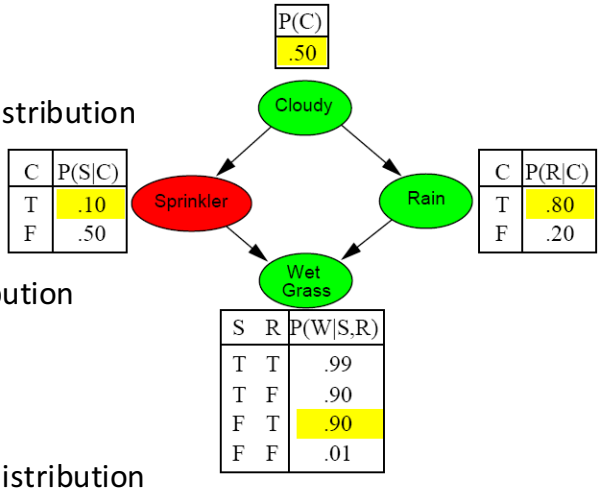
let **true** be selected

Generate a value for Sprinkler from the distribution
P(Sprinkler | Cloudy=true) = ⟨0.1, 0.9⟩

let it be **false**

Generate a value for Rain from the distribution
P(Rain | Cloudy=true) = ⟨0.8, 0.2⟩

let it be **true**



Generate a value for WetGrass from the distribution
P(WetGrass | Sprinkler=false, Rain=true) = ⟨0.9, 0.1⟩

let it be **true**

We have a sample Cloudy=true, Sprinkler=false, Rain=true, WetGrass=true

The probability of obtaining that sample is 0.5 * 0.9 * 0.8 * 0.9 = 0.324

Rejection sampling

However, we are looking for **P**(X | e)!
From all the generated samples, we will select only those consistent with the evidence e (other samples are rejected).

P(X | e) ≈ **N**(X,e) / N(e)

```
function REJECTION-SAMPLING(X, e, bn, N) returns an estimate of P(X|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])
```

Assume that we generated 100 samples, but only for 27 samples we have Sprinkler= true and from them in 8 samples we have Rain = true while in 19 samples we have Rain = false. Then

P(Rain | Sprinkler=true) ≈ Normalize(⟨8, 19⟩) = ⟨0.296, 0.704⟩

The major weakness is **rejecting too many samples**!

Likelihood weighting

Instead of rejecting inconsistent samples it seems more efficient to generate only samples consistent with evidence e.

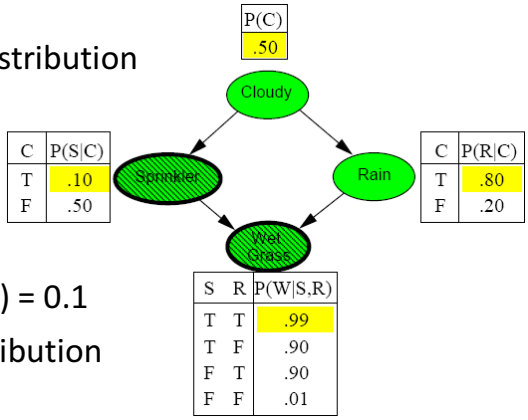
- fix the values for the evidence variables E and sample only the non-evidence variables
- The probability of obtaining a sample is $P(z,e) = \Pi_i P(z_i \mid parents(z_i))$
- But this is not what we want! We miss $w(z,e) = \Pi_j P(e_j \mid parents(e_j))$.
- Hence each sample is **weighted** as follows:

  P(X | e) ≈ α **N**(X,e) w(X,e)

Likelihood weighting (an example)

Let the query be **P**(Rain | Sprinkler=true,WetGrass=true))

- The initial weight is w = 1.0
- Generate a value for Cloudy from the distribution **P**(Cloudy) = ⟨0.5, 0.5⟩
  let it be **true**
- The value Sprinkler=**true** is known, but we modify the weight
  w ← w * P(Sprinker=true | Cloudy=true) = 0.1
- Generate a value for Rain from the distribution **P**(Rain | Cloudy=true) = ⟨0.8, 0.2⟩
  let it be **true**
- The value WetGrass=**true** is known, but we modify the weight
  w ← w * P(WetGrass=true | Sprinker=true,Rain=true) = 0.099



We obtained a sample
Cloudy=true, Sprinkler=false, Rain=true, WetGrass=true,
with the weight 0.099

Likelihood weighting (algorithm)

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

---

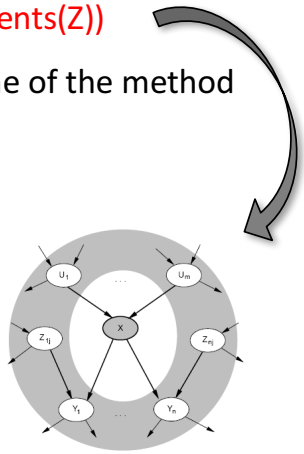
function WEIGHTED-SAMPLE($bn, \mathbf{e}$) **returns** an event and a weight
  $\mathbf{x} \leftarrow$ an event with n elements; $w \leftarrow 1$
  for $i = 1$ **to** n **do**
    if X_i has a value x_i in $\mathbf{e}$
      then $w \leftarrow w \times P(X_i = x_i \mid parents(X_i))$
      else $x_i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

Markov Chain Monte Carlo (MCMC)

Direct sampling generates each sample from scratch.

We can obtain a sample differently:

- start with a randomly generated sample consistent with evidence e
- for a selected variable X (outside evidence E) select a new value conditioned on the the values of the variables in the Markov blanket
  $P(x \mid mb(X)) = P(x \mid parents(X)) \prod_{Z \in Children(X)} P(z \mid parents(Z))$
- we will get a so called **Markov chain**, hence the name of the method **Markov Chain Monte Carlo** (MCMC)
- samples are processed as in direct sampling



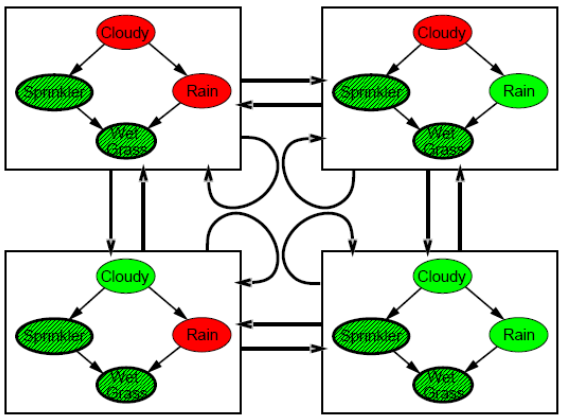
```
function MCMC-Ask(X, e, bn, N) returns an estimate of P(X|e)
  local variables: N[X], a vector of counts over X, initially zero
                   Z, the nonevidence variables in bn
                   x, the current state of the network, initially copied from e
  initialize x with random values for the variables in Z
  for j = 1 to N do
      for each Z_i in Z do
          sample the value of Z_i in x from P(Z_i|mb(Z_i))
              given the values of MB(Z_i) in x
          N[x] ← N[x] + 1 where x is the value of X in x
  return Normalize(N[X])
```

Markov chain (an example)

Assume evidence Sprinkler=true, WetGrass=true

We will get four different states that will be visited by the Markov chain.

We explore 100 states:
for 31 states we have Rain=true
for 69 states we have Rain=false



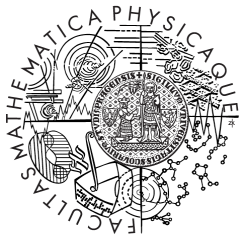
Hence we get:

P(Rain | Sprinkler=true, WetGrass=true)
= Normalize(⟨31, 69⟩) = ⟨0.31, 0.69⟩

The sampling process settles into a dynamic equilibrium in which the long-term fraction of time spent in each state is exactly proportional to its posterior probability.

- let us use the following notation:
  - $q(\mathbf{x} \rightarrow \mathbf{x}')$ for probability of transition from x to x' (this transition probabity defines the Markov chain)
  - $\pi_t(\mathbf{x})$ for probability of being at state x at time t
- in general the following formula holds:
  - $\pi_{t+1}(\mathbf{x}') = \Sigma_{\mathbf{x}} \pi_t(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}')$
- for **stationary distribution** we require
  - $\pi(\mathbf{x}') = \Sigma_{\mathbf{x}} \pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}')$
  - this holds for example when $\pi(\mathbf{x}) q(\mathbf{x} \rightarrow \mathbf{x}') = \pi(\mathbf{x}') q(\mathbf{x}' \rightarrow \mathbf{x})$
- Assume that we changed the value of variable X_i from x_i to x_i', other variables are $\mathbf{Y}_i$ and their values are $\mathbf{y}_i$
  - $q(\mathbf{x} \rightarrow \mathbf{x}') = q((x_i,\mathbf{y}_i) \rightarrow (x_i',\mathbf{y}_i)) = P(x_i' \mid \mathbf{y}_i,\mathbf{e}) = P(x_i' \mid mb(X_i))$
  - This is called **Gibss sampling**
  - $\pi(\mathbf{x})\, q(\mathbf{x} \rightarrow \mathbf{x}') = P(\mathbf{x} \mid \mathbf{e})\, P(x_i' \mid \mathbf{y}_i,e) = P(x_i,\mathbf{y}_i \mid \mathbf{e})\, P(x_i' \mid \mathbf{y}_i,e)$
    $= P(x_i \mid \mathbf{y}_i,\mathbf{e})\, P(\mathbf{y}_i \mid \mathbf{e})\, P(x_i' \mid \mathbf{y}_i,e)$
    $= P(x_i \mid \mathbf{y}_i,\mathbf{e})\, P(x_i',\mathbf{y}_i \mid e) = q(\mathbf{x}' \rightarrow \mathbf{x})\, \pi(\mathbf{x}')$

chain rule



© 2016 Roman Barták
Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz