Chapter 14: Bayesian networks

A Bayesian network consists of:

- A set of random variables (nodes)
- A directed, acyclic graph over variables.
  - An edge A -> B ≈ "A directly influences B"
- A conditional distribution for each node given its parents.

$$P(X_i|\text{Parents}(X_i))$$

Noise
Midterm
A2 due tonight

Bayes nets are a way to represent conditional dependence.
Recall that full joint probability tables are not useful in practice because: (1) they are too big, (2) they are not intuitive.
A Bayesian network is a way to represent a probability distribution using just conditional probability tables.

Benefits
- easier repr
- Faster inference. Full distribution is exponential.

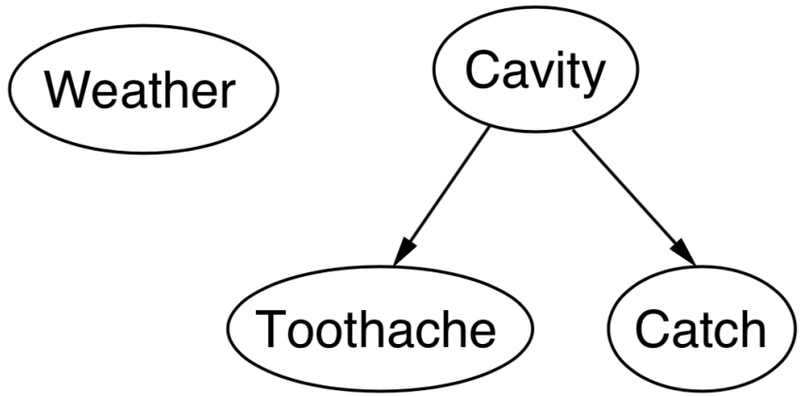
A Bayesian network defines a full joint distribution

$$P(x_1 \dots x_n) = \prod_i P(x_i | \text{Parents}(X_i))$$

What is the probability of j, m, a, -b, -e?
$0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 \approx 0.00063$

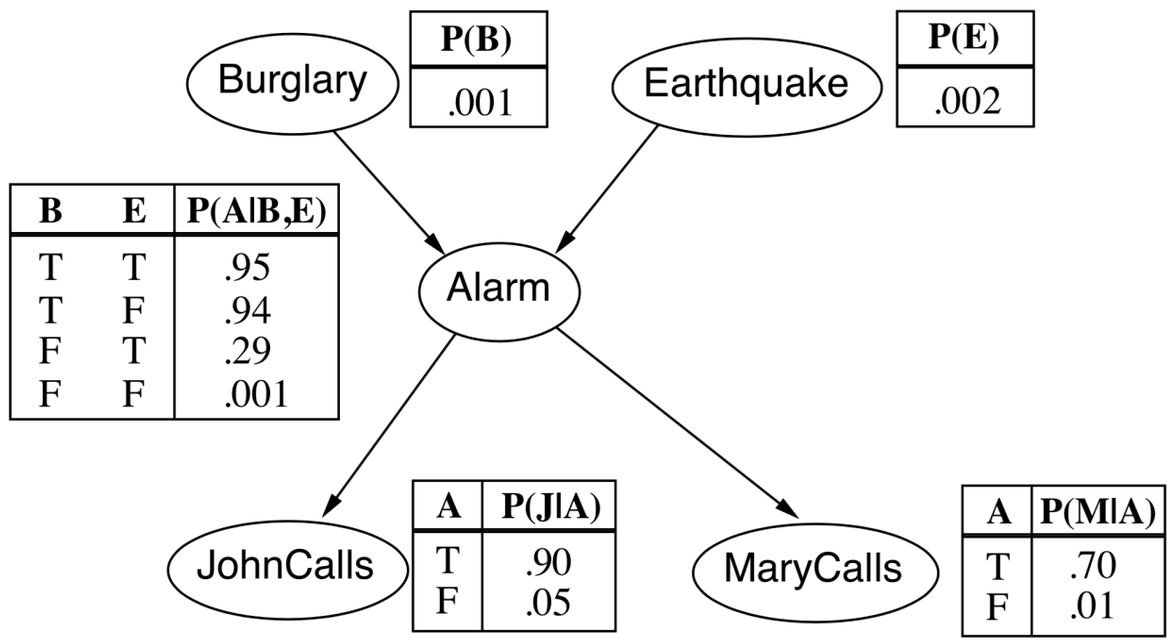
Topological order: Order that puts parents before children.
There is always a topological order for any directed acyclic graph (i.e. any Bayes net)

Example: Cavity



What is the dependence structure of the cavity problem?

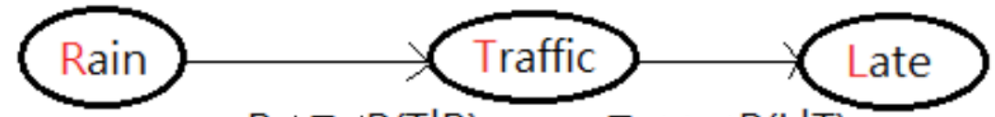
Example: Earthquake



What is the dependence structure of the alarm problem?

Why do the probabilities not add up to 1?

Example: Traffic



R	P(R)
t	0.1
f	0.9

R	T	P(T|R)
t	t	0.8
t	f	0.2
f	t	0.1
f	f	0.9

T	L	P(L|T)
t	t	0.3
t	f	0.7
f	t	0.1
f	f	0.9

What is the dependence structure of the traffic problem?

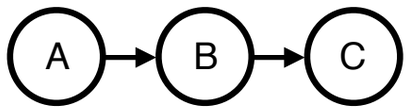
Compactness

n Boolean variables
Each variable has <= k parents.
How big is the biggest conditional probability table? 2^(k+1) - 1
Bayes net: Requires O(n * 2^k) parameters. Grows linearly in n.
Versus O(2^k) for the full joint distribution.

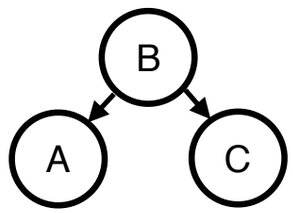
Parameters to specify the whole joint distribution:
Cavity: 5
Alarm: 10
Traffic: 5

Conditional independence

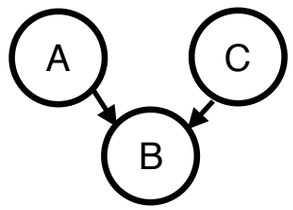
$A \not\perp C$



$A \not\perp C$



$A \perp C$



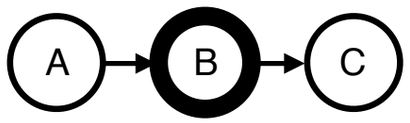
We can read off conditional dependence/independence relationships from the network.

Note: Variables can be independent even if they're connected in the graph. We can only know for sure that variables are independent, not dependent.

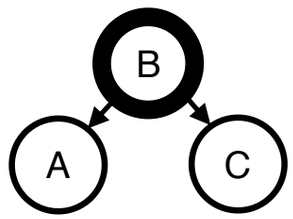
Types of connections:
Direct path: a -> b -> c
Upside down V: b <- a -> c
V shape: b -> a <- c

Conditional independence

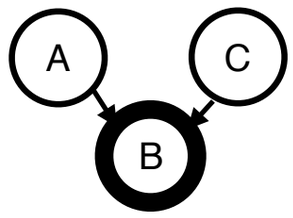
$A \perp C \mid B$



$A \perp C \mid B$



$A \not\perp C \mid B$



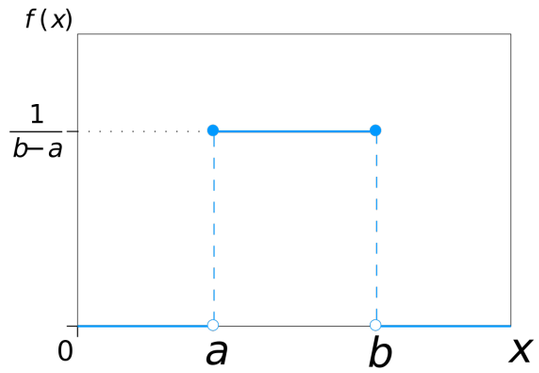
We can read off conditional dependence/independence relationships from the network.

Note: Variables can be independent even if they're connected in the graph. We can only know for sure that variables are independent, not dependent.

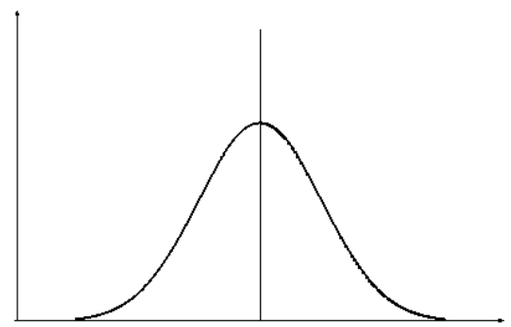
Continuous random variables

$$P(a \leq x < b) = \int_a^b f(x)dx$$

Uniform distribution



Normal distribution



$$f(x) = (1/2\pi)\exp(-x^2/2)$$

What about continuous random variables? Rain -> time to get to work.
Two ways to handle this:

- Discretize. Loses precision.
- Use methods for continuous variables. They are more complicated.

For continuous variables we get a probability density function f(x) rather than a probability mass function P(x).
The probability of any given value (e.g. x = 5.347...) is almost zero.
We need to reason about ranges of values.

Example distributions:

- Uniform
- Gaussian / Normal

Inference in Bayesian networks

Methods

- Construct joint distribution, do naive enumeration.
- Bayes net enumeration
- Variable elmination
- Sampling

Inference by enumeration

```
# Computes P(evidence) = sum_hidden P(evidence, hidden)
def marginalize_enum(bn, vars, evidence):
    if hidden.empty: return 1.0
    y = first(vars) # topological order
    if y in evidence:
        return bn.p(y=evidence[y] | evidence) *
               marginalize_enum(bn, vars-y, evidence)
    else:
        total = 0
        for val in y.domain:
            total += bn.p(y=val | evidence) *
                     marginalize_enum(bn, vars-y,
                         evidence+{y=val})
        return total
```

marginalize_enum() = P(evidence) = Sum_{hidden} P(hidden, evidence)

evidence: dictionary {var: val}
vars: set of variables left to process

Topological order: parents first, then children.

Inference by enumeration

```
def infer_enum(bn, query, evidence):
    dist = probability_table()
    for val in query.domain:
        dist[val] = marginalize_enum(bn, bn.vars,
                                     evidence + {query=val})
```

query = variable
evidence: dictionary {var: val}

Factors

Factor: Generalized probability table, without the sum-to-one requirement.

A factor *f* defined over random variables *A, B* and *C* defines a value for each assignment

f(A=a, B=b, C=c) = <value>

Operations on factors:

- Fix
- Marginalize
- Product

Factor fix and marginalize

Fix: Restrict the domain of a factor. Used to incorporate evidence.

Ex: Fix *A = a:*

$$f_2(B = b) = f_1(A = a, B = b)$$

Marginalize: Sum out a hidden variable.

$$f_2(B = b) = \sum_a f_1(A = a, B = b)$$

Factor product

Input factors f_1 and f_2; output a factor with the union of the two domains.

Pointwise product of factors f_1 and f_2:

$$f_1(x_1, \ldots, x_j, y_1, \ldots, y_k) \times f_2(y_1, \ldots, y_k, z_1, \ldots, z_l)$$
$$= f(x_1, \ldots, x_j, y_1, \ldots, y_k, z_1, \ldots, z_l)$$

E.g., $f_1(a, b) \times f_2(b, c) = f(a, b, c)$

Variable elimination