6.034
Bayes Networks

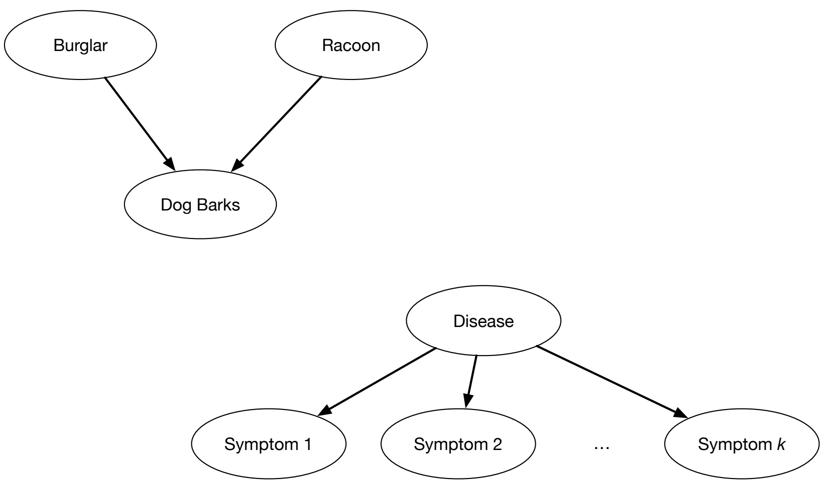
Peter Szolovits

ai6034.mit.edu

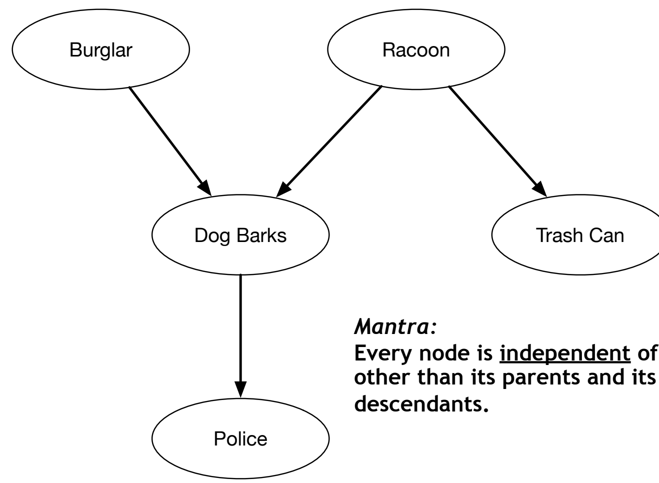
November 13, 2019

Graphical Probabilistic Models

Graphical Models Show Dependencies

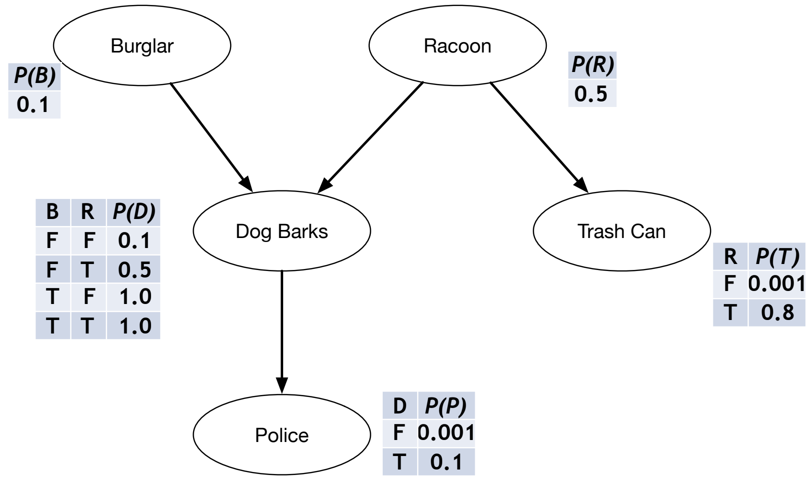


Consider More Complex Barking Story



Mantra:
Every node is <u>independent</u> of nodes other than its parents and its descendants.

Probabilities



P(B)
0.1

P(R)
0.5

B	R	P(D)
F	F	0.1
F	T	0.5
T	F	1.0
T	T	1.0

R	P(T)
F	0.001
T	0.8

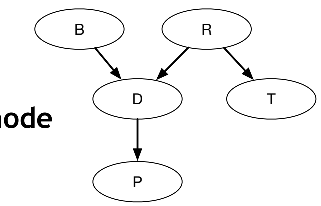
D	P(P)
F	0.001
T	0.1

Bayes Network Allows Reconstruction of Probability Tables

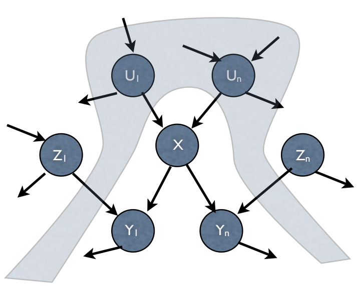
- $P(p,d,b,t,r) = P(p\,|\,d,b,t,r)P(d\,|\,b,t,r)P(b\,|\,t,r)P(t\,|\,r)P(r)$ by the chain rule
- But by conditional independence in the graph,

$$P(p,d,b,t,r) = P(p\,|\,d,\cancel{b},\cancel{t},\cancel{r})P(d\,|\,b,\cancel{t},r)P(b\,|\,\cancel{t},\cancel{r})P(t\,|\,r)P(r)$$
$$= P(p\,|\,d)P(d\,|\,b,r)P(b)P(t\,|\,r)P(r)$$

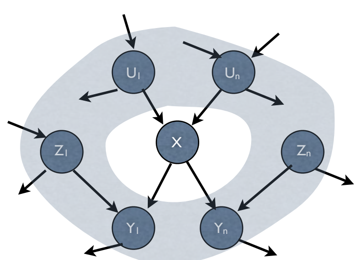
- $P(x_1,\ldots,x_n) = \prod_{i=n}^{1} P(x_i\,|\,\mathrm{Par}(x_i))$
where Par gives the parents of a node



Topological Interpretations



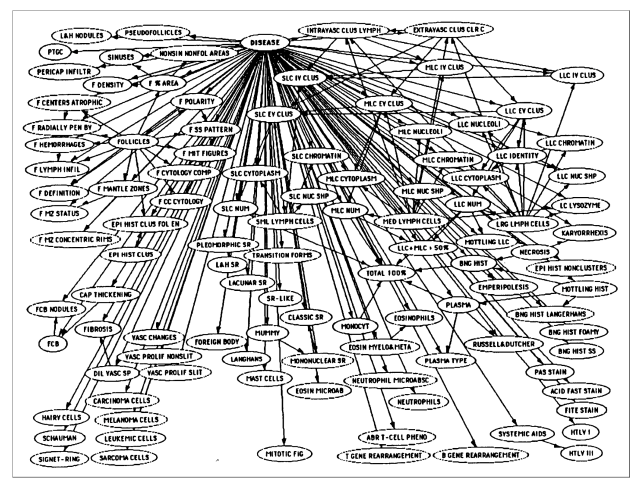
A node, *X*, is *conditionally independent* of its non-descendants, Z_i, given its parents, U_i.



A node, *X*, is *conditionally independent* of all other nodes in the network given its *Markov blanket*: its parents, U_i, children, Y_i, and children's parents, Z_i.

A Very Large Bayes Net

David Heckerman, *Pathfinder/Intellipath*, around 1990



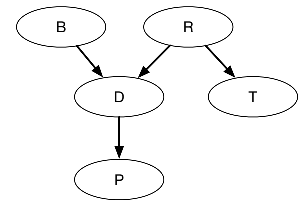
How (not) to do Inference

- So, we can reconstruct the probability of any particular scenario
- But, normally we want to know the probabilities of some nodes *given* that we have observed some others
  - E.g., what is the probability of a burglar given that the police were called and the trash can was *not* knocked over?
- By abuse of notation, we write a variable x to represent whatever its value is, and x^+, x^- if its value is known to be T or F (binary case)
  - $P(b^+ | p^+, t^-) = \dfrac{P(b^+, p^+, t^-)}{P(b^+, t^-)}$

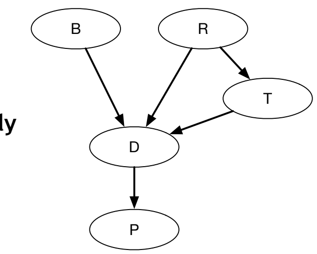
$$\frac{\sum_{d,r} P(p^+, d, b^+, t^-, r)}{\sum_{b,d,r} P(p^+, d, b, t^-, r)}$$

- Downside: exponential number of terms in the "don't care" variables

For Poly-Trees, simple propagation



- Suppose we observe B
  - Reduce c.p. table of its children (D) to the B=T or B=F cases
  - Propagate
- Suppose we observe P
  - Use Bayes' Rule to update D
  - Propagate
- Suppose we observe D
  - Do both of the above
- Because everything is singly connected, one pass updates all probabilities
- Much more complex if the network is multiply connected! Propagation doesn't work.

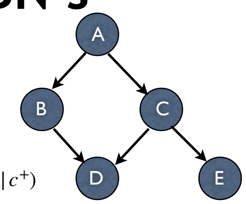


Rules and Probabilities

- Many have wanted to put a probability on assertions and on rules, and compute with likelihoods
- E.g., Mycin's *certainty factor* framework
  – A (p=.3) & B (p=.7) ==(p=.8)==> C (p=?)
- Problems:
  – How to combine uncertainties of preconditions and of rule
  – How to combine evidence from multiple rules
- Theorem: There is NO such algebra that works when rules are considered independently.
- Need BN for a consistent model of probabilistic inference

Exact Solution of BN's

(non-poly-trees)



- $P(a, b, c, d, e) = P(a)P(b | a)P(c | a)P(d | b, c)P(e | c)$
- What is the probability of a specific state, say A=T, B=F, C=T, D=T, E=F?
  $P(a^+, b^-, c^+, d^+, d^-) = P(a^+)P(b^- | a^+)P(c^+ | a^+)P(d^+ | b^-, a^+)P(e^- | c^+)$
- What is the probability that E=t given B=t?
  $P(e^+ | b^+) = P(e^+, b^+)/P(b^+)$
- Consider the term $P(e^+, b^+)$

$$P(e^+, b^+) = \sum_{a,c,d} P(a, b^+, c, d, e^+)$$

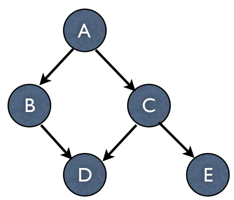
$$= \sum_{a,c,d} P(a)P(b^+ | a)P(c | a)P(D | b^+, c)P(e^+ | c)$$

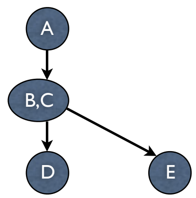
$$= \sum_{c} P(e^+ | c)\left(\sum_{a} P(a)P(c | a)P(b^+ | a)\right)\left(\sum_{d} P(d, b^+ | c)\right)$$

- 12 instead of 32 multiplications (even in this small example)

Alas, optimal
factoring is NP-hard

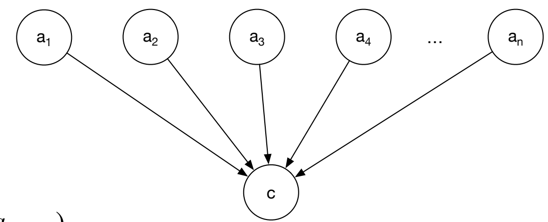
Other Exact Methods





- *Join-tree*: Merge variables into (small!) sets of variables to make graph into a poly-tree. Most commonly-used; aka *Clustering*, *Junction-tree*, *Potential*)
- *Cutset-conditioning*: Instantiate a (small!) set of variables, then solve each residual problem, and add solutions weighted by probabilities of the instantiated variables having those values
- ...
- All these methods are essentially equivalent; with some time-space tradeoffs.

We don't want to model high-arity dependence



- $p(c \mid a_1, a_2, \ldots)$
  - too many probabilities needed in conditional probability table
- Can we simplify?
  - Noisy or
  - noisy and
  - noisy max/min
  - ?

Simplifying Conditional Probability Tables via Noisy-OR

- Do we know any structure in the way that $\mathrm{Par}(x)$ "cause" x?
- If each destroyer can sink the ship with probability $P(s \mid d_i)$, what is the probability that the ship will sink if it's attacked by both?
  $1 - P(s \mid d_1, d_2) = (1 - P(s \mid d_1))(1 - P(s \mid d_2))(1 - l)$
- For $|\mathrm{Par}(x)| = n$, this requires $O(n)$parameters, not $O(k^n)$

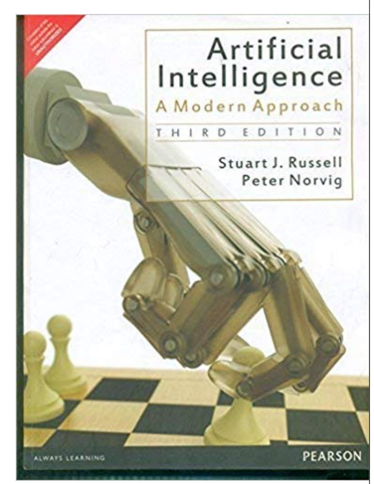
d_1

s

d_2

Sampling Methods to Evaluate Bayes Networks

Following Russel & Norvig



Approximate Inference in BN's

- Direct Sampling
- Rejection Sampling
- Likelihood Weighting
- Markov chain Monte Carlo
  - Gibbs and other similar sampling methods

Direct Sampling

```
function Prior-Sample(bn) returns an event sampled from bn
  inputs: bn, a Bayes net specifying the joint distribution P(X1, ... Xn)
 x := an event with n elements
 for i = 1 to n do
   xi := a random sample from P(Xi|Par(Xi))
 return x
```

$$\lim_{n \to \infty} \frac{N_{PS}(x_1, \ldots, x_n)}{N} = P(x_1, \ldots, x_n) \qquad P(x_1, \ldots, x_m) \approx \frac{N_{PS}(x_1, \ldots, x_m)}{N}$$

- From a large number of samples, we can estimate all joint probabilities
  - The probability of an event is the fraction of all complete events generated by PS that match the partially specified event
    - hence we can compute all conditionals, etc.

Rejection Sampling

```
function Rejection-Sample(X, e, bn, N) returns an estimate of P(X|e)
  inputs: bn, a Bayes net
          X, the query variable
          e, evidence specified as an event
          N, the number of samples to be generated
   local: K, a vector of counts over values of X, initially 0

 for j = 1 to N do
  y := PriorSample(bn)
  if y is consistent with e then
    K[v] := K[v]+1 where v is the value of X in y
 return Normalize(K[X])
```

- Uses PriorSample to estimate the proportion of times each value of X appears in samples that are consistent with e
- But, most samples may be irrelevant to a specific query, so this is quite inefficient

Likelihood Weighting

- In trying to compute P(X|e), where e is the *evidence* (variables with known, observed values),
  - Sample only the variables other than those in e
  - Weight each sample by how well it predicts e

$$\begin{aligned} S_{WS}(\boldsymbol{z}, \boldsymbol{e}) w(\boldsymbol{z}, \boldsymbol{e}) &= \prod_{i=1}^{l} P(z_i|\mathrm{Par}(Z_i)) \prod_{i=1}^{m} P(e_i|\mathrm{Par}(E_i)) \\ &= P(\boldsymbol{z}, \boldsymbol{e}) \end{aligned}$$

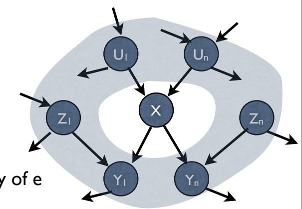
Likelihood Weighting

$$
\begin{aligned}
S_{WS}(\boldsymbol{z},\boldsymbol{e})w(\boldsymbol{z},\boldsymbol{e}) &= \prod_{i=1}^{l} P(z_i|\mathrm{Par}(Z_i)) \prod_{i=1}^{m} P(e_i|\mathrm{Par}(E_i)) \\
&= P(\boldsymbol{z},\boldsymbol{e})
\end{aligned}
$$

```
function Likelihood-Weighting(X, e, bn, N) returns an estimate of P(X|e)
   inputs: bn, a Bayes net
           X, the query variable
           e, evidence specified as an event
           N, the number of samples to be generated
   local: W, a vector of weighted counts over values of X, initially 0
 for j = 1 to N do
   y,w := WeightedSample(bn)
   if y is consistent with e then
     W[v] := W[v]+w where v is the value of X in y
 return Normalize(W[X])

function Weighted-Sample(bn,e) returns an event and a weight
 x := an event with n elements; w := 1
 for i = 1 to n do
   if Xi has a value xi in e
     then w := w * P(Xi = xi | Par(Xi))
     else xi := a random sample from P(Xi | Par(Xi))
 return x,w
```

Markov chain Monte Carlo




```
function MCMC(X, e, bn, N) returns an estimate of P(X|e)
   local: K[X], a vector of counts over values of X, initially 0
          Z, the non-evidence variables in bn (includes X)
          x, the current state of the network, initially a copy of e
 initialize x with random values for the vars in Z
 for j = 1 to N do
   for each Zi in Z do
      sample the value of Zi in x from P(Zi|mb(Zi)), given the values of mb(Zi) in x
      K[v] := K[v]+1 where v is the value of X in x
 return Normalize(K[X])
```

- **Wander incrementally from the last state sampled, instead of re-generating a completely new sample**
- **For every unobserved variable, choose a new value according to its probability given the values of vars in it Markov blanket (remember, it's independent of all other vars)**
- **After each change, tally the sample for its value of X; this will only change sometimes**
- **Problem: "narrow passages"**

Learning Probabilistic Models: A Simple Example

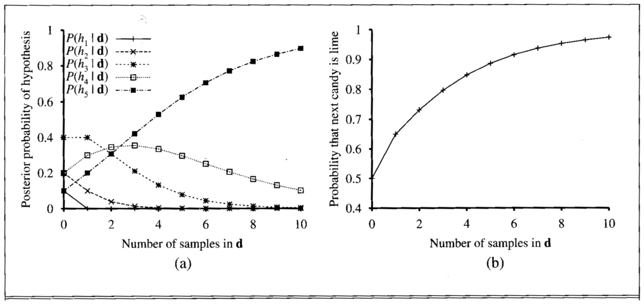
- Surprise Candy Corp. makes two flavors of candy: *cherry* and *lime*
- Both flavors come in the same opaque wrapper
- Candy is sold in large bags, which have one of the following distributions of flavors, but are visually indistinguishable:
  - h_1: 100% cherry
  - h_2: 75% cherry, 25% lime
  - h_3: 50% cherry, 50% lime
  - h_4: 25% cherry, 75% lime
  - h_5: 100% lime
- Relative prevalence of these types of bags is (.1, .2, .4, .2, .1)
- As we eat our way through a bag of candy, predict the flavor of the next piece; actually a probability distribution.

Bayesian Learning about a Fixed Simple Set of Hypotheses

- **Calculate the probability of each hypothesis given the data**
  $P(h_i\,|\,\mathbf{d}) = \alpha P(\mathbf{d}\,|\,h_i)P(h_i)$
- **To predict the probability distribution over an unknown quantity, X,**
  $P(X\,|\,\mathbf{d}) = \sum_i P(X\,|\,\mathbf{d},h_i)P(h_i\,|\,\mathbf{d}) = \sum_i P(X\,|\,h_i)P(h_i\,|\,\mathbf{d})$
- **If the observations $\mathbf{d}$ are independent, then**
  $P(\mathbf{d}\,|\,h_i) = \prod_j P(d_j\,|\,h_i)$
- **E.g., suppose the first 10 candies we taste are all lime**
  $P(\mathbf{d}\,|\,h_3) = 0.5^{10} \approx 0.001$

Learning Hypotheses and Predicting from Them

- (a) probabilities of ***h_i*** after ***k lime*** candies; (b) prob. of next ***lime***



- MAP prediction: predict just from most probable hypothesis
  - After 3 limes, ***h_5*** is most probable, hence we predict ***lime***
  - Even though, by (b), it's only 80% probable

Observations

- Bayesian approach asks for prior probabilities on ***hypotheses***!
  - Natural way to encode bias against complex hypotheses: make their prior probability very low
- Choosing h_{MAP} to maximize $P(h_i | \mathbf{d}) = \alpha P(\mathbf{d} | h_i) P(h_i)$
  - is equivalent to minimizing $-\log P(\mathbf{d} | h_i) - \log P(h_i)$
  - but from our earlier discussion of entropy as a measure of information, these two terms are
    - # of bits needed to describe the data given hypothesis
    - # bits needed to specify the hypothesis
  - Thus, MAP learning chooses the hypothesis that maximizes ***compression*** of the data; ***Minimum Description Length*** principle
- Assuming uniform priors on hypotheses makes MAP yield h_{ML}, the ***maximum likelihood hypothesis***, which maximizes $P(h_i | \mathbf{d}) = \alpha P(\mathbf{d} | h_i)$

How to Build a Bayes Network

- Human expertise
  - E.g., like building the Acute Renal Failure program, Pathfinder, Alarm, …
- Human expertise to determine structure, data to determine parameters
  - Point parameter estimation
  - Smoothing
  - Useful distributions:
    - Common: Beta (binomial), Dirichlet (multinomial)
    - Any of the "exponential family", e.g., normal, Poisson, Gamma, etc.
- Automated methods to discover structure and parameters
  - "Best" model is the one that predicts the highest probability for the data actually observed.

Learning Structure

- In general, we are trying to determine not only parameters for a known structure but in fact which structure is best
  - or the probability of each structure, so we can average over them to make a prediction

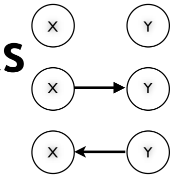
Structure Learning

- Recall that a Bayes Network is fully specified by
  - a DAG *G* that gives the (in)dependencies among variables
  - the collection of parameters θ that define the conditional probability tables for each of the $P(x_i \mid \text{Par}(x_i))$
- Then $P(G \mid D) = \dfrac{P(D \mid G)P(G)}{P(D)} \propto P(D \mid G)P(G)$
- We define the ***Bayesian score*** as $\log P(D \mid G) + \log(P(G))$

$$P(D \mid G) = \int_{\theta_G} P(D \mid \theta_G, G)P(\theta_g \mid G)P(G)d\theta_G$$

- But
  - First term: usual marginal likelihood calculation
  - Second term: parameter priors
  - Third term: "penalty" for complexity of graph
- Define a search problem over all possible graphs & parameters

Searching for Models



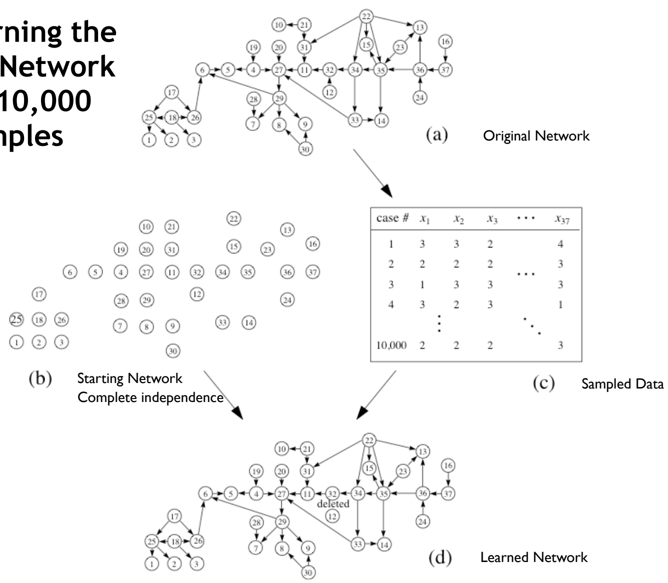
1	2.000000e+00
2	3.200000e+01
3	3.072000e+03
4	1.572864e+06
5	4.026532e+09
6	4.947802e+13
7	2.837268e+18
8	7.437727e+23
9	8.773900e+29
10	4.600050e+36
11	1.061171e+44
12	1.068209e+52
13	4.659610e+60
14	8.755632e+69
15	7.050966e+79

- How many possible DAGs are there for *n* variables?
  - $< 3^{n^2}$ = all possible directed graphs on *n* vars
  - Not all are DAGs
- To get a closer estimate, imagine that we order the variables so that the parents of each var come before it in the ordering. Then
  - there are $n\,!$possible ordering, and
  - the *i*-th var can have any of the previous vars as a parent

$$n!\prod_{i=1}^{n} 2^{i-1} = n! \cdot 2^{\sum_{i=1}^{n}(i-1)} = O(n! \cdot 2^{n^2})$$

- If we can choose a particular ordering, say based on prior knowledge, then we need consider "merely" $O(2^{n^2})$ models
- If we restrict |Par(X)| to no more than *k*, consider $\leq \sum_{i=1}^{n} \binom{n}{k}$ models
  - this is actually practical
- Search actions: add, delete, reverse an arc
- Hill-climb on P(D|G) or on P(G|D)
- All "usual" tricks in search: simulated annealing, random restart, ...

Re-Learning the ALARM Network from 10,000 Samples



Caution about Hidden Variables (Confounders)

- Suppose you are given a dataset containing data on patients' smoking, diet, exercise, chest pain, fatigue, and shortness of breath
- You would probably learn a model like the one below left
- If you can hypothesize a "hidden" variable (not in the data set), e.g., *heart disease*, the learned network might be much simpler, such as the one below right
- But, there are potentially infinitely many such variables