Bayesian Networks

CS 760@UW-Madison

Goals for Part 1

you should understand the following concepts

- the Bayesian network representation
- inference by enumeration
- the parameter learning task for Bayes nets
- the structure learning task for Bayes nets
- maximum likelihood estimation
- Laplace estimates
- *m*-estimates

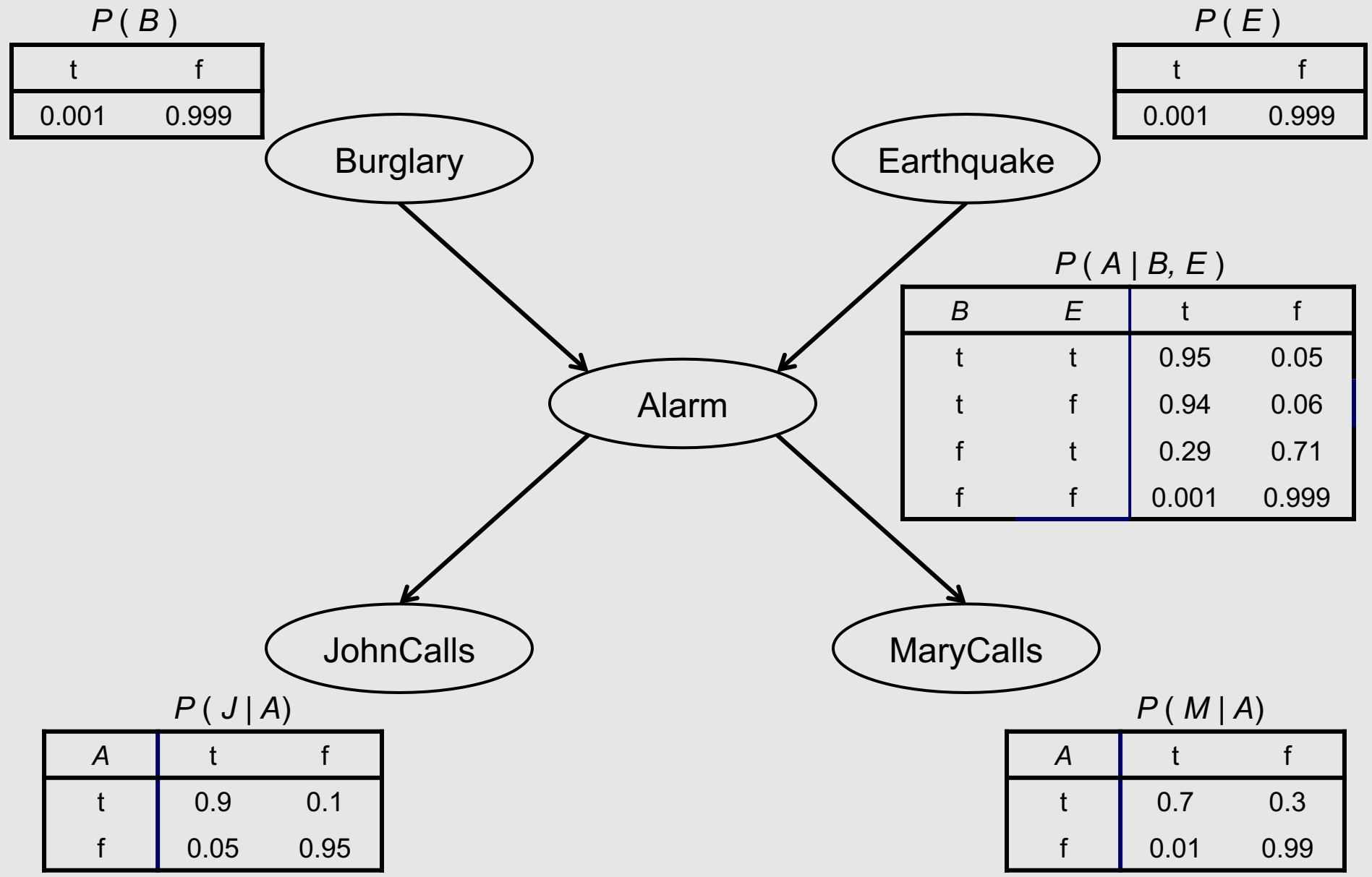
Bayesian network example

- Consider the following 5 binary random variables:

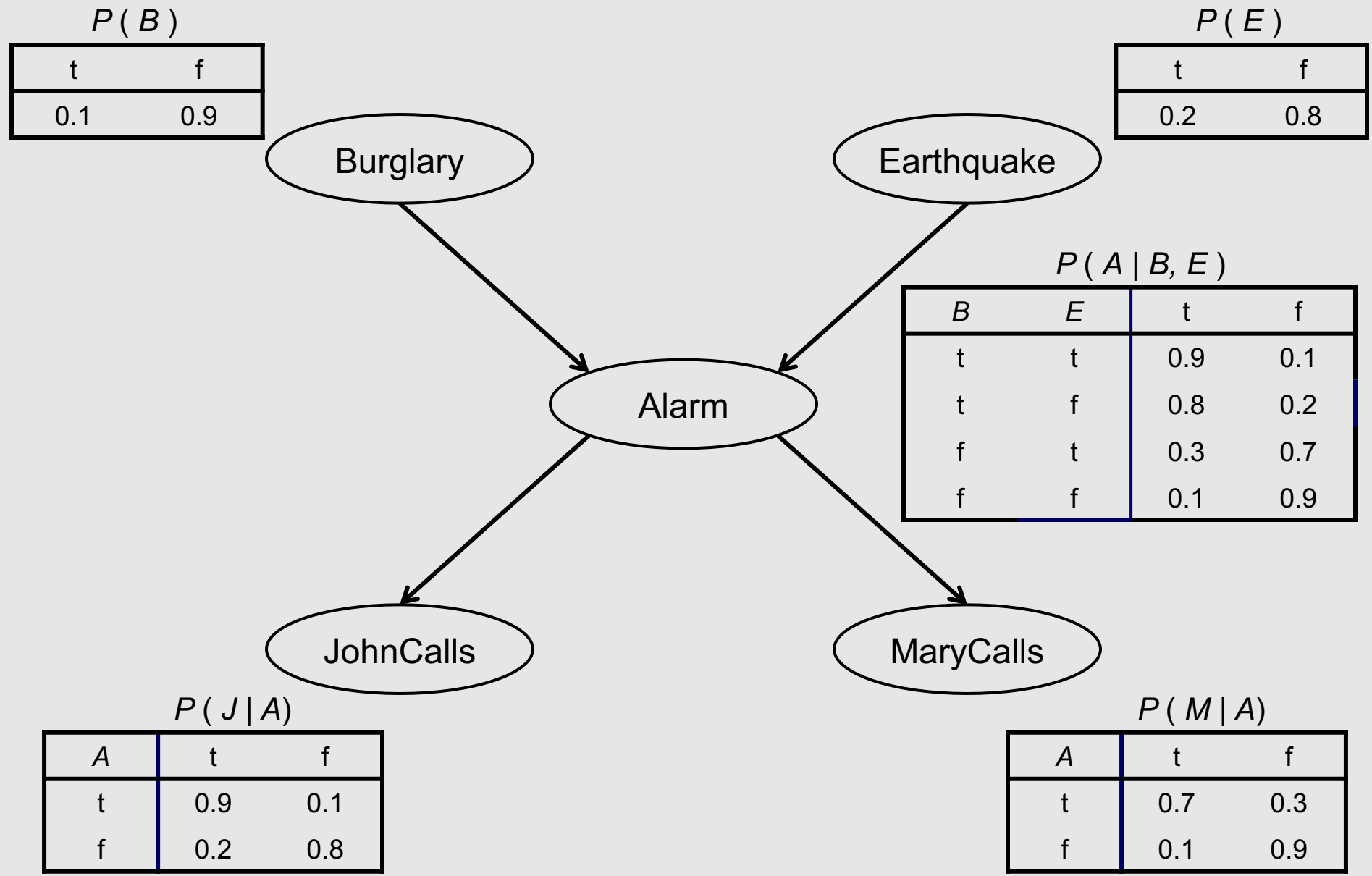
  B = a burglary occurs at your house
  E = an earthquake occurs at your house
  A = the alarm goes off
  J = John calls to report the alarm
  M = Mary calls to report the alarm

- Suppose we want to answer queries like what is $P(B \mid M, J)$?

Bayesian network example



Bayesian network example



Bayesian networks

- a BN consists of a Directed Acyclic Graph (DAG) and a set of conditional probability distributions

- in the DAG
  - each node denotes random a variable
  - each edge from X to Y represents that X *directly influences* Y
  - formally: each variable X is independent of its non-descendants given its parents

- each node X has a *conditional probability distribution* (CPD) representing $P(X \mid Parents(X))$

Bayesian networks

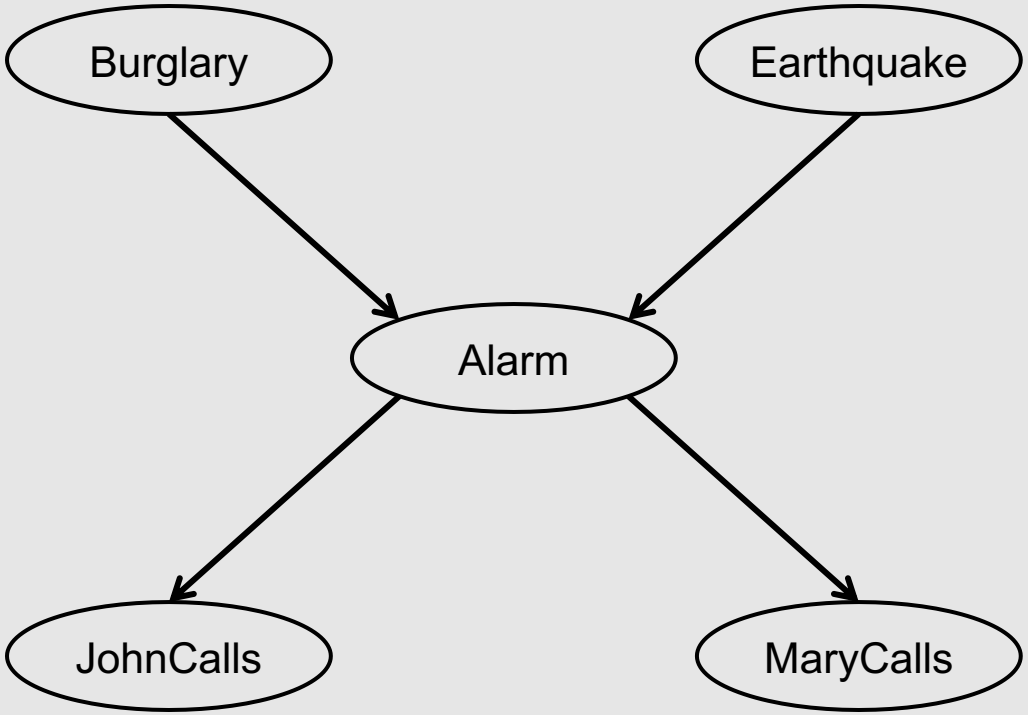
- using the chain rule, a joint probability distribution can be expressed as

$$P(X_1,...,X_n) = P(X_1)\prod_{i=2}^{n} P(X_i \mid X_1,...,X_{i-1})$$

- a BN provides a compact representation of a joint probability distribution

$$P(X_1,...,X_n) = P(X_1)\prod_{i=2}^{n} P(X_i \mid Parents(X_i))$$

Bayesian networks



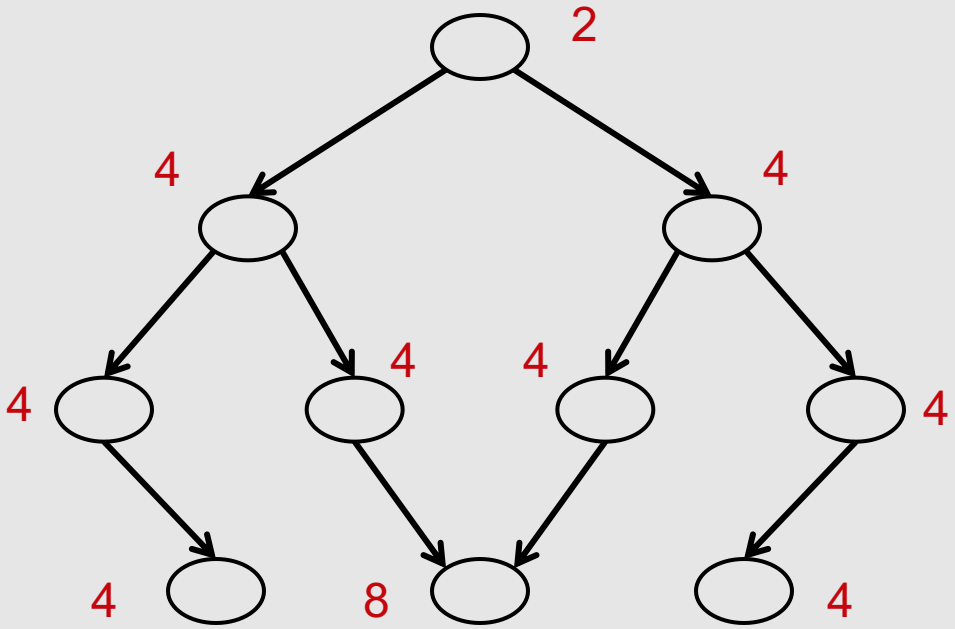
$$
\begin{aligned}
&P(B,E,A,J,M) \\
&= P(B) \\
&\times P(E) \\
&\times P(A \mid B,E) \\
&\times P(J \mid A) \\
&\times P(M \mid A)
\end{aligned}
$$

- a standard representation of the joint distribution for the Alarm example has $2^5 = 32$ parameters
- the BN representation of this distribution has 20 parameters

Bayesian networks

- consider a case with 10 binary random variables

- How many parameters does a BN with the following graph structure have?



= 42

- How many parameters does the standard table representation of the joint distribution have?

= 1024

Advantages of Bayesian network representation

- Captures independence and conditional independence where they exist
- Encodes the relevant portion of the full joint among variables where dependencies exist
- Uses a graphical representation which lends insight into the complexity of inference

The inference task in Bayesian networks

Given: values for some variables in the network (*evidence*), and a set of *query* variables

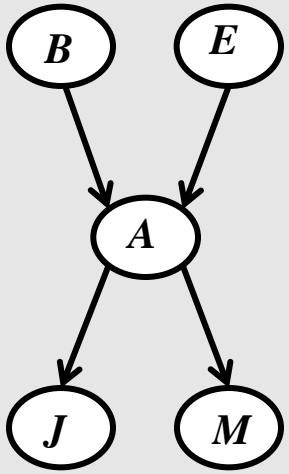
Do: compute the posterior distribution over the query variables

- variables that are neither evidence variables nor query variables are *hidden* variables
- the BN representation is flexible enough that any set can be the evidence variables and any set can be the query variables

Inference by enumeration

- let a denote A=true, and $\neg a$ denote A=false
- suppose we're given the query: $P(b \mid j, m)$

  "probability the house is being burglarized given that John and Mary both called"
- from the graph structure we can first compute:



$$P(b, j, m) = \sum_{e,\neg e} \sum_{a,\neg a} P(b)P(E)P(A \mid b, E)P(j \mid A)P(m \mid A)$$

sum over possible values for E and A variables (e, $\neg e$, a, $\neg a$)

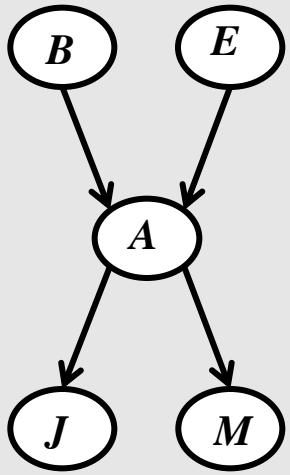
Inference by enumeration

$$P(b,j,m) = \sum_{e,\neg e}\sum_{a,\neg a} P(b)P(E)P(A\mid b,E)P(j\mid A)P(m\mid A)$$

$$= P(b)\sum_{e,\neg e}\sum_{a,\neg a} P(E)P(A\mid b,E)P(j\mid A)P(m\mid A)$$

P(B)
0.001

P(E)
0.001



B	E	P(A)
t	t	0.95
t	f	0.94
f	t	0.29
f	f	0.001

A	P(J)
t	0.9
f	0.05

A	P(M)
t	0.7
f	0.01

$B \quad E \quad A \quad J \quad M$

$$= 0.001 \times (0.001 \times 0.95 \times 0.9 \times 0.7 + \quad e, a$$

$$0.001 \times 0.05 \times 0.05 \times 0.01 + \quad e, \neg a$$

$$0.999 \times 0.94 \times 0.9 \times 0.7 + \quad \neg e, a$$

$$0.999 \times 0.06 \times 0.05 \times 0.01) \quad \neg e, \neg a$$

Inference by enumeration

- now do equivalent calculation for $P(\neg b,\ j, m)$
- and determine $P(b \mid j, m)$

$$P(b \mid j,m) = \frac{P(b, j,m)}{P(j,m)} = \frac{P(b, j,m)}{P(b, j,m) + P(\neg b, j,m)}$$

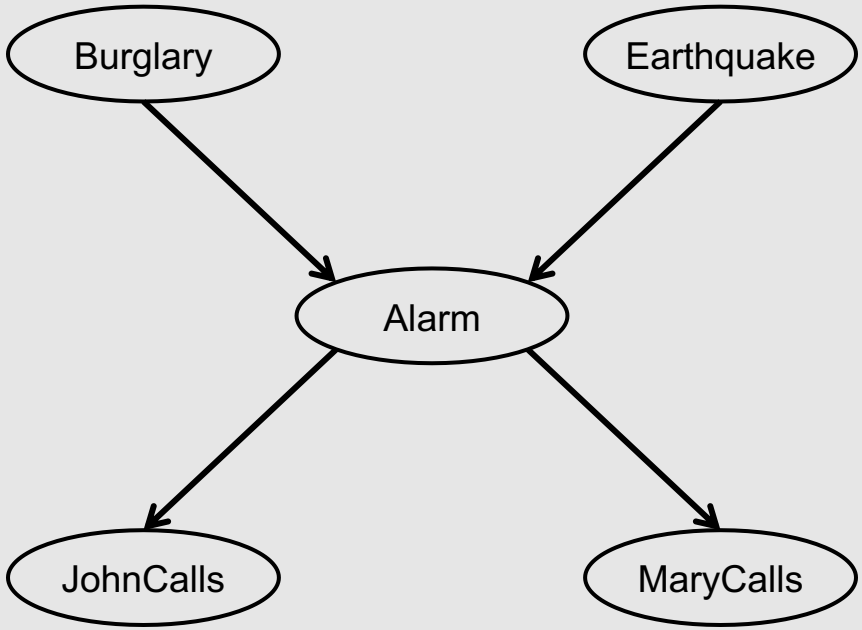
Comments on BN inference

- *inference by enumeration* is an *exact* method (i.e. it computes the exact answer to a given query)
- it requires summing over a joint distribution whose size is exponential in the number of variables
- in many cases we can do exact inference efficiently in large networks
  - key insight: save computation by pushing sums inward
- in general, the Bayes net inference problem is NP-hard
- there are also methods for approximate inference – these get an answer which is "close"
- in general, the approximate inference problem is NP-hard also, but approximate methods work well for many real-world problems

The parameter learning task

- Given: a set of training instances, the graph structure of a BN

B	E	A	J	M
f	f	f	t	f
f	t	f	f	f
f	f	t	f	t
		…		



- Do: infer the parameters of the CPDs

The structure learning task

- Given: a set of training instances

B	E	A	J	M
f	f	f	t	f
f	t	f	f	f
f	f	t	f	t
		…		

- Do: infer the graph structure (and perhaps the parameters of the CPDs too)

Parameter learning and MLE

- *maximum likelihood estimation* (MLE)
  - given a model structure (e.g. a Bayes net graph) G and a set of data D
  - set the model parameters θ to maximize $P(D \mid G, \theta)$

- i.e. make the data D look as likely as possible under the model $P(D \mid G, \theta)$

Maximum likelihood estimation

consider trying to estimate the parameter θ (probability of heads) of a biased coin from a sequence of flips

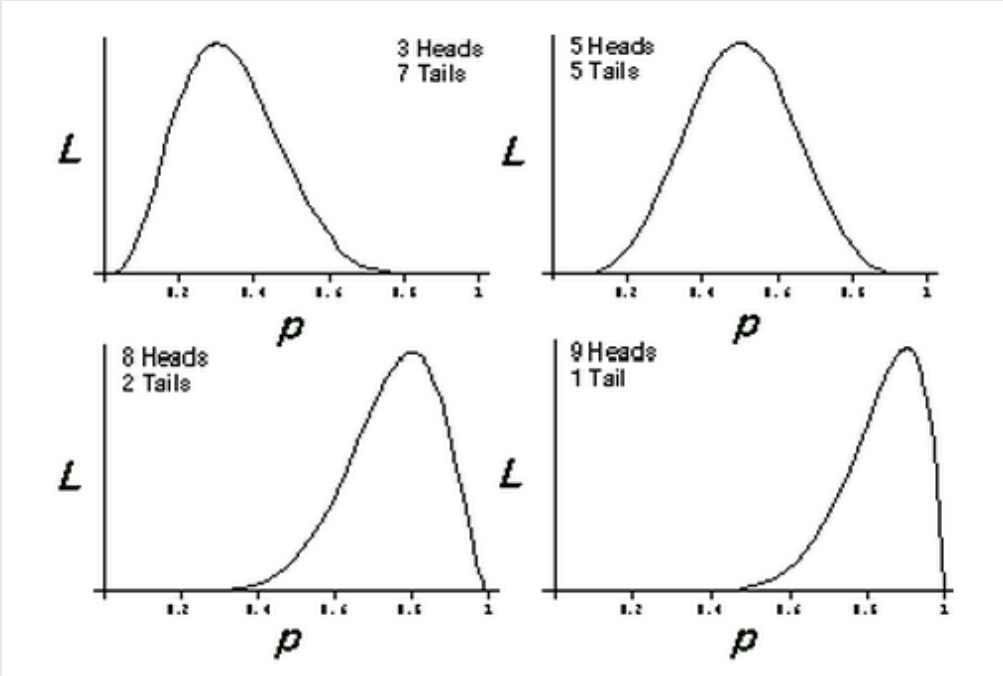
$$\boldsymbol{x} = \{1,1,1,0,1,0,0,1,0,1\}$$

the likelihood function for θ is given by:

$$L(\theta : x_1,\ldots,x_n) = \theta^{x_1}(1-\theta)^{1-x_1}\cdots\theta^{x_n}(1-\theta)^{1-x_n}$$

$$= \theta^{\sum x_i}(1-\theta)^{n-\sum x_i}$$

for h heads in n flips
the MLE is h/n



MLE in a Bayes net

$$L(\theta : D, G) = P(D \mid G, \theta) = \prod_{d \in D} P(x_1^{(d)}, x_2^{(d)}, \ldots, x_n^{(d)})$$

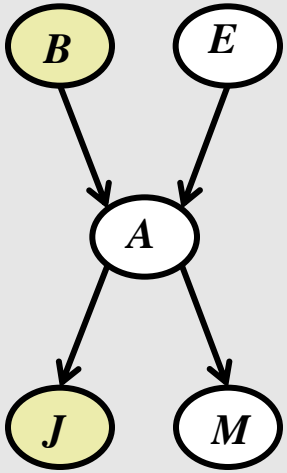
$$= \prod_{d \in D} \prod_i P(x_i^{(d)} \mid Parents(x_i^{(d)}))$$

$$= \prod_i \left(\prod_{d \in D} P(x_i^{(d)} \mid Parents(x_i^{(d)})) \right)$$

independent parameter learning problem for each CPD

Maximum likelihood estimation

now consider estimating the CPD parameters for B and J in the alarm network given the following data set



B	E	A	J	M
f	f	f	t	f
f	t	f	f	f
f	f	f	t	t
t	f	f	f	t
f	f	t	t	f
f	f	t	f	t
f	f	t	t	t
f	f	t	t	t

$$P(b) = \frac{1}{8} = 0.125$$

$$P(\neg b) = \frac{7}{8} = 0.875$$

$$P(j \mid a) = \frac{3}{4} = 0.75$$

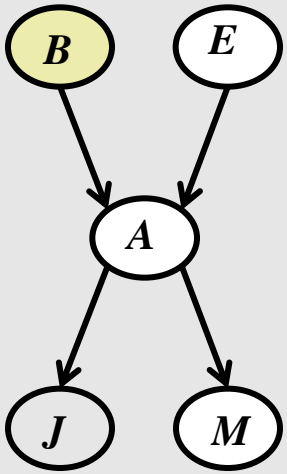
$$P(\neg j \mid a) = \frac{1}{4} = 0.25$$

$$P(j \mid \neg a) = \frac{2}{4} = 0.5$$

$$P(\neg j \mid \neg a) = \frac{2}{4} = 0.5$$

Maximum likelihood estimation

suppose instead, our data set was this…



B	E	A	J	M
f	f	f	t	f
f	t	f	f	f
f	f	f	t	t
f	f	f	f	t
f	f	t	t	f
f	f	t	f	t
f	f	t	t	t
f	f	t	t	t

$$P(b) = \frac{0}{8} = 0$$

$$P(\neg b) = \frac{8}{8} = 1$$

do we really want to set this to 0?

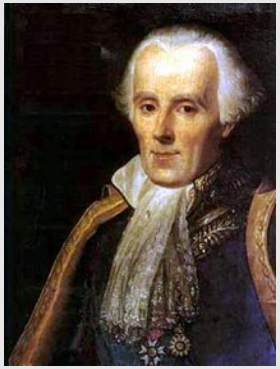
Maximum a posteriori (MAP) estimation

- instead of estimating parameters strictly from the data, we could start with some prior belief for each

- for example, we could use *Laplace estimates*

$$P(X = x) = \frac{n_x + 1}{\sum_{v \in \text{Values}(X)} (n_v + 1)}$$

pseudocounts



- where n_v represents the number of occurrences of value v

Maximum a posteriori (MAP) estimation

a more general form: *m-estimates*

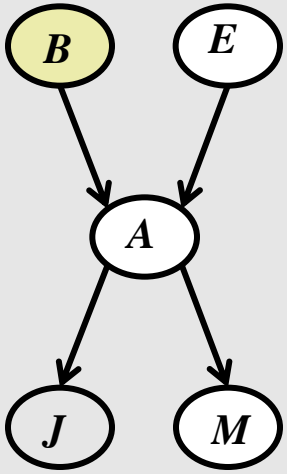
$$P(X = x) = \frac{n_x + p_x m}{\left(\sum_{v \in \text{Values}(X)} n_v \right) + m}$$

prior probability of value x

number of "virtual" instances

M-estimates example

now let's estimate parameters for B using $m=4$ and $p_b=0.25$



B	E	A	J	M
f	f	f	t	f
f	t	f	f	f
f	f	f	t	t
f	f	f	f	t
f	f	t	t	f
f	f	t	f	t
f	f	t	t	t
f	f	t	t	t

$$P(b) = \frac{0 + 0.25 \times 4}{8 + 4} = \frac{1}{12} = 0.08 \qquad P(\neg b) = \frac{8 + 0.75 \times 4}{8 + 4} = \frac{11}{12} = 0.92$$

Goals for Part 2

you should understand the following concepts

- missing data in machine learning
  - hidden variables
  - missing at random
  - missing systematically
- the EM approach to imputing missing values in Bayes net parameter learning

- the Chow-Liu algorithm for structure search

Missing data

- Commonly in machine learning tasks, some feature values are missing
- some variables may not be observable (i.e. *hidden*) even for training instances
- values for some variables may be *missing at random*: what caused the data to be missing does not depend on the missing data itself
  - e.g. someone accidentally skips a question on an questionnaire
  - e.g. a sensor fails to record a value due to a power blip
- values for some variables may be *missing systematically*: the probability of value being missing depends on the value
  - e.g. a medical test result is missing because a doctor was fairly sure of a diagnosis given earlier test results
  - e.g. the graded exams that go missing on the way home from school are those with poor scores

Missing data

- hidden variables; values *missing at random*
  - these are the cases we'll focus on
  - one solution: try impute the values

- values *missing systematically*
  - may be sensible to represent "*missing*" as an explicit feature value

Imputing missing data with EM

Given:

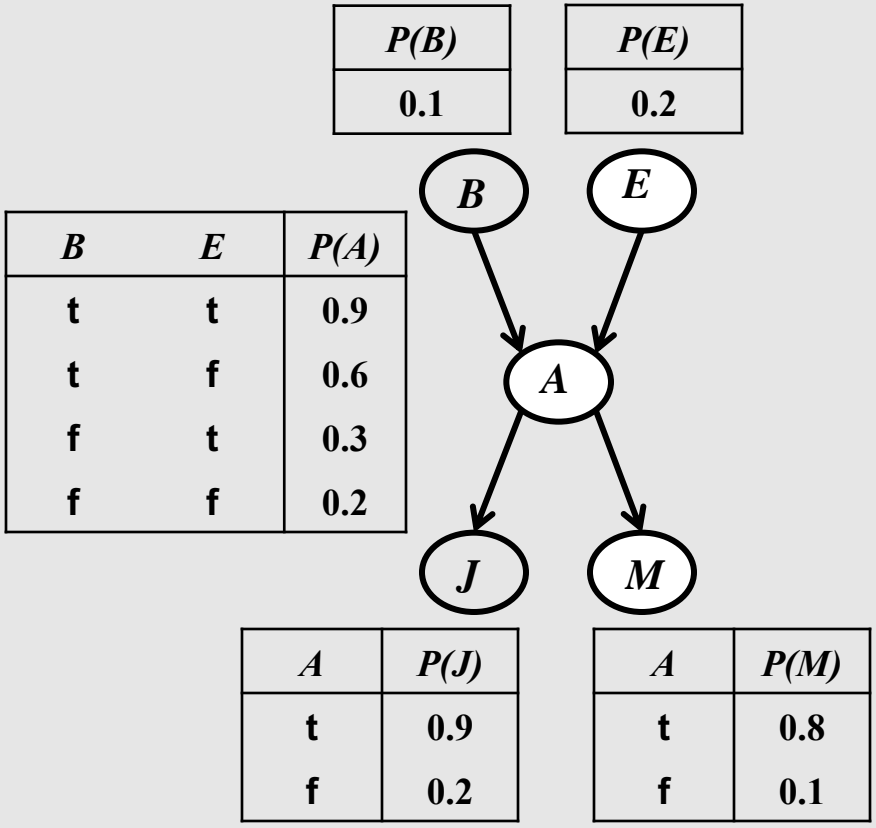
- data set with some missing values
- model structure, initial model parameters

Repeat until convergence

- *Expectation* (E) step: using current model, compute expectation over missing values
- *Maximization* (M) step: update model parameters with those that maximize probability of the data (MLE or MAP)

Example: EM for parameter learning

suppose we're given the following initial BN and training set



B	*E*	*A*	*J*	*M*
f	f	?	f	f
f	f	?	t	f
t	f	?	t	t
f	f	?	f	t
f	t	?	t	f
f	f	?	f	t
t	t	?	t	t
f	f	?	f	f
f	f	?	t	f
f	f	?	f	t

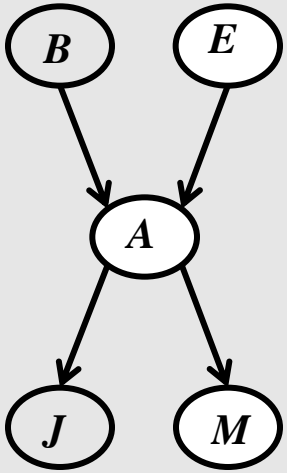
Example: E-step

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

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

P(B)
0.1

P(E)
0.2



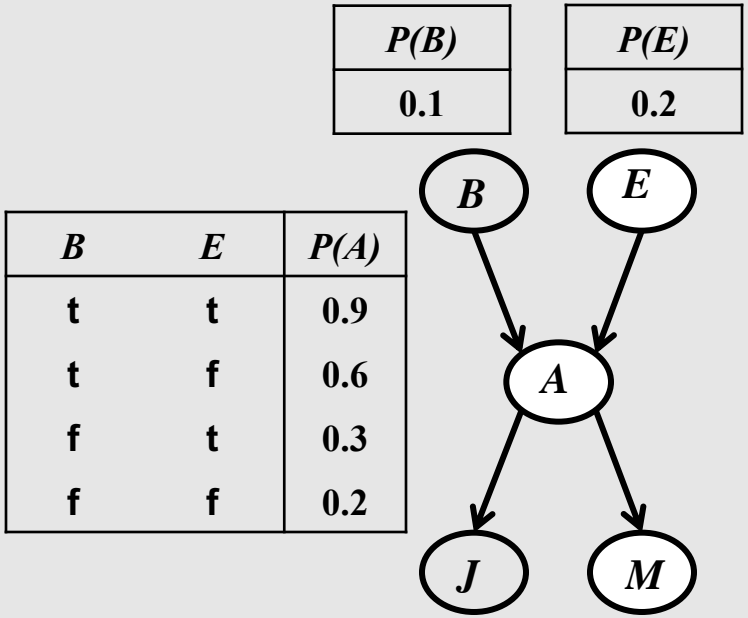
B	E	P(A)
t	t	0.9
t	f	0.6
f	t	0.3
f	f	0.2

A	P(J)
t	0.9
f	0.2

A	P(M)
t	0.8
f	0.1

B	E	A	J	M
f	f	t: 0.0069 f: 0.9931	f	f
f	f	t:0.2 f:0.8	t	f
t	f	t:0.98 f: 0.02	t	t
f	f	t: 0.2 f: 0.8	f	t
f	t	t: 0.3 f: 0.7	t	f
f	f	t:0.2 f: 0.8	f	t
t	t	t: 0.997 f: 0.003	t	t
f	f	t: 0.0069 f: 0.9931	f	f
f	f	t:0.2 f: 0.8	t	f
f	f	t: 0.2 f: 0.8	f	t

Example: E-step



B	E	P(A)
t	t	0.9
t	f	0.6
f	t	0.3
f	f	0.2

A	P(J)
t	0.9
f	0.2

A	P(M)
t	0.8
f	0.1

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

$$= \frac{P(\neg b, \neg e, a, \neg j, \neg m)}{P(\neg b, \neg e, a, \neg j, \neg m) + P(\neg b, \neg e, \neg a, \neg j, \neg m)}$$

$$= \frac{0.9 \times 0.8 \times 0.2 \times 0.1 \times 0.2}{0.9 \times 0.8 \times 0.2 \times 0.1 \times 0.2 + 0.9 \times 0.8 \times 0.8 \times 0.8 \times 0.9}$$

$$= \frac{0.00288}{0.4176} = 0.0069$$

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

$$= \frac{P(\neg b, \neg e, a, j, \neg m)}{P(\neg b, \neg e, a, j, \neg m) + P(\neg b, \neg e, \neg a, j, \neg m)}$$

$$= \frac{0.9 \times 0.8 \times 0.2 \times 0.9 \times 0.2}{0.9 \times 0.8 \times 0.2 \times 0.9 \times 0.2 + 0.9 \times 0.8 \times 0.8 \times 0.2 \times 0.9}$$

$$= \frac{0.02592}{0.1296} = 0.2$$

Example: M-step

re-estimate probabilities using expected counts

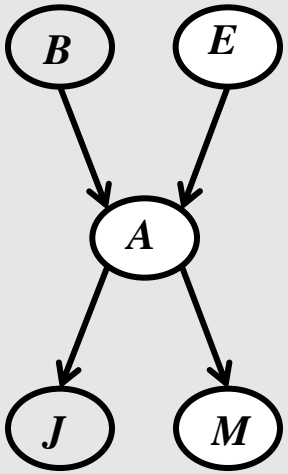
$$P(a \mid b,e) = \frac{E\#(a \wedge b \wedge e)}{E\#(b \wedge e)}$$

$$P(a \mid b,e) = \frac{0.997}{1}$$

$$P(a \mid b,\neg e) = \frac{0.98}{1}$$

$$P(a \mid \neg b,e) = \frac{0.3}{1}$$

$$P(a \mid \neg b,\neg e) = \frac{0.0069 + 0.2 + 0.2 + 0.2 + 0.0069 + 0.2 + 0.2}{7}$$



B	*E*	*P(A)*
t	t	0.997
t	f	0.98
f	t	0.3
f	f	0.145

re-estimate probabilities for $P(J \mid A)$ and $P(M \mid A)$ in same way

B	*E*	*A*	*J*	*M*
f	f	t: 0.0069 f: 0.9931	f	f
f	f	t:0.2 f:0.8	t	f
t	f	t:0.98 f: 0.02	t	t
f	f	t: 0.2 f: 0.8	f	t
f	t	t: 0.3 f: 0.7	t	f
f	f	t:0.2 f: 0.8	f	t
t	t	t: 0.997 f: 0.003	t	t
f	f	t: 0.0069 f: 0.9931	f	f
f	f	t:0.2 f: 0.8	t	f
f	f	t: 0.2 f: 0.8	f	t

Example: M-step

re-estimate probabilities using expected counts

$$P(j \mid a) = \frac{E\#(a \wedge j)}{E\#(a)}$$

$$P(j \mid a) = \frac{0.2 + 0.98 + 0.3 + 0.997 + 0.2}{0.0069 + 0.2 + 0.98 + 0.2 + 0.3 + 0.2 + 0.997 + 0.0069 + 0.2 + 0.2}$$

$$P(j \mid \neg a) = \frac{0.8 + 0.02 + 0.7 + 0.003 + 0.8}{0.9931 + 0.8 + 0.02 + 0.8 + 0.7 + 0.8 + 0.003 + 0.9931 + 0.8 + 0.8}$$

B	E	A	J	M
f	f	t: 0.0069 f: 0.9931	f	f
f	f	t:0.2 f:0.8	t	f
t	f	t:0.98 f: 0.02	t	t
f	f	t: 0.2 f: 0.8	f	t
f	t	t: 0.3 f: 0.7	t	f
f	f	t:0.2 f: 0.8	f	t
t	t	t: 0.997 f: 0.003	t	t
f	f	t: 0.0069 f: 0.9931	f	f
f	f	t:0.2 f: 0.8	t	f
f	f	t: 0.2 f: 0.8	f	t

Convergence of EM

- E and M steps are iterated until probabilities converge
- will converge to a maximum in the data likelihood (MLE or MAP)
- the maximum may be a local optimum, however
- the optimum found depends on starting conditions (initial estimated probability parameters)

Learning structure + parameters

- number of structures is superexponential in the number of variables
- finding optimal structure is NP-complete problem
- two common options:
  - search very restricted space of possible structures (e.g. networks with tree DAGs)
  - use heuristic search (e.g. sparse candidate)

The Chow-Liu algorithm

- learns a BN with a tree structure that maximizes the likelihood of the training data
- algorithm
  1. compute weight $I(X_i, X_j)$ of each possible edge (X_i, X_j)
  2. find maximum weight spanning tree (MST)
  3. assign edge directions in MST

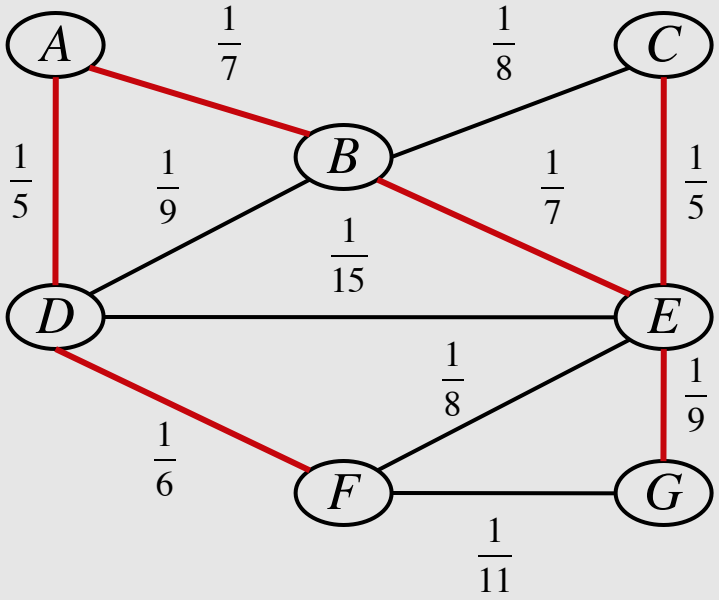
The Chow-Liu algorithm

1. use mutual information to calculate edge weights

$$I(X,Y) = \sum_{x \in \text{values}(X)} \sum_{y \in \text{values}(Y)} P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)}$$

The Chow-Liu algorithm

2. find maximum weight spanning tree: a maximal-weight tree that connects all vertices in a graph



The Chow-Liu algo always have a complete graph, but here we use a non-complete graph as the example for clarity.

Prim's algorithm for finding an MST

given: graph with vertices V and edges E

$V_{new} \leftarrow \{ v \}$ where v is an arbitrary vertex from V

$E_{new} \leftarrow \{ \}$

repeat until $V_{new} = V$

{

  choose an edge (u, v) in E with max weight where u is in V_{new} and v is not

  add v to V_{new} and (u, v) to E_{new}

}

return V_{new} and E_{new} which represent an MST

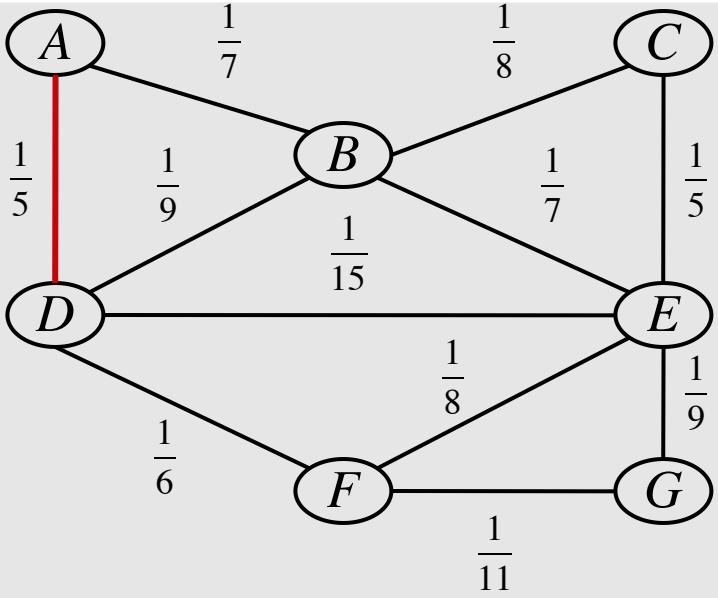
Kruskal's algorithm for finding an MST

given: graph with vertices V and edges E

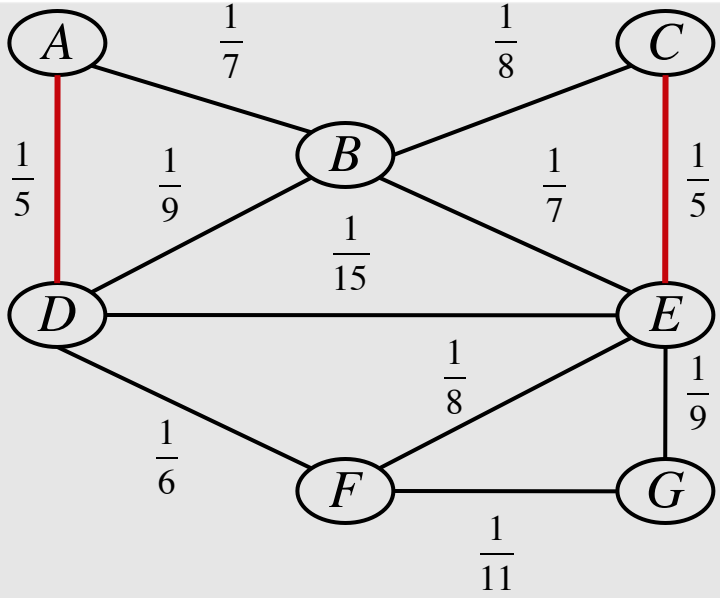
$E_{new} \leftarrow \{\ \}$
for each (u, v) in E ordered by weight (from high to low)
{
  remove (u, v) from E
  if adding (u, v) to E_{new} does not create a cycle
    add (u, v) to E_{new}
}
return V and E_{new} which represent an MST

Finding MST in Chow-Liu

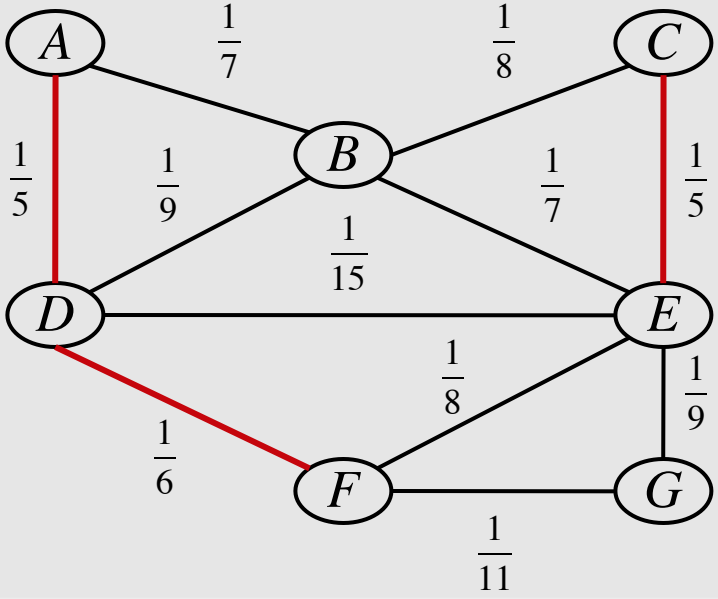
i.



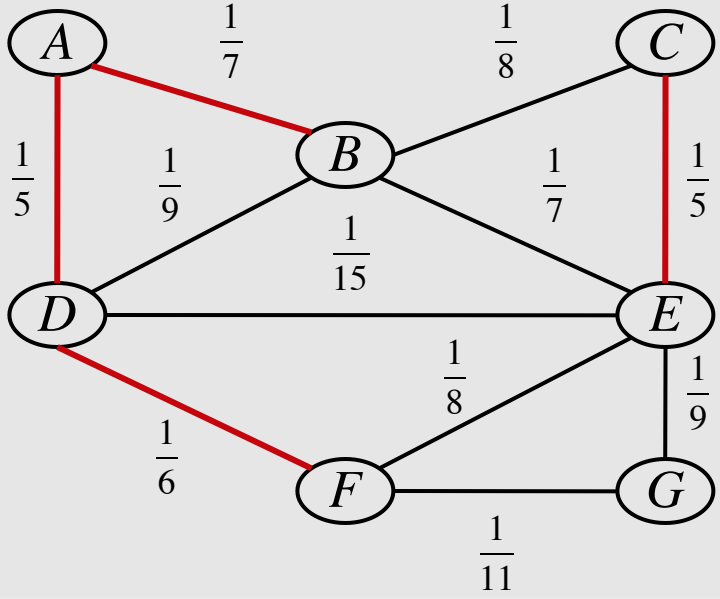
ii.



iii.

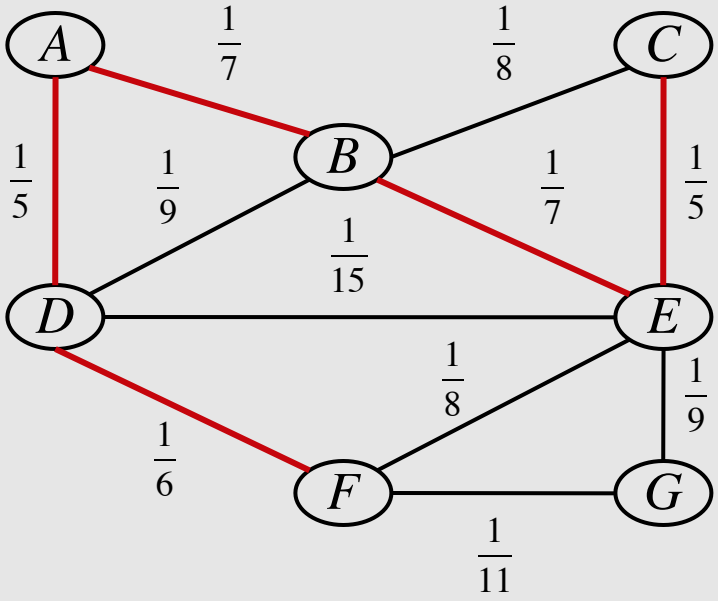


iv.

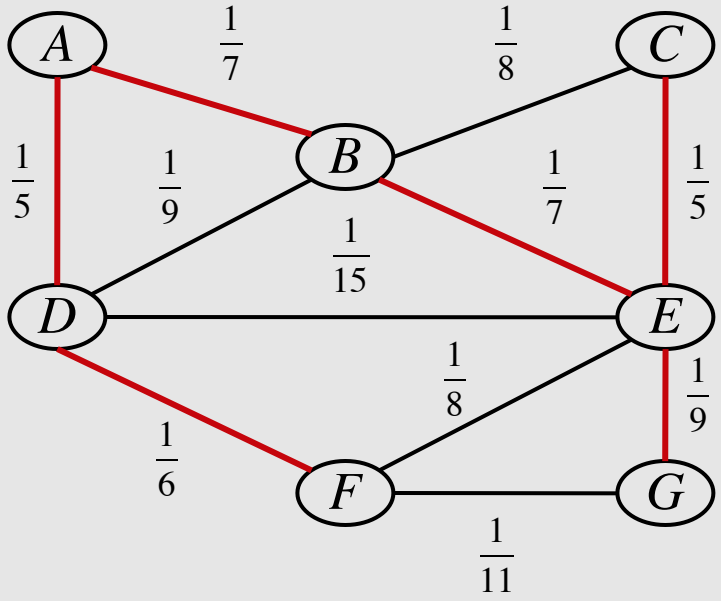


Finding MST in Chow-Liu

v.

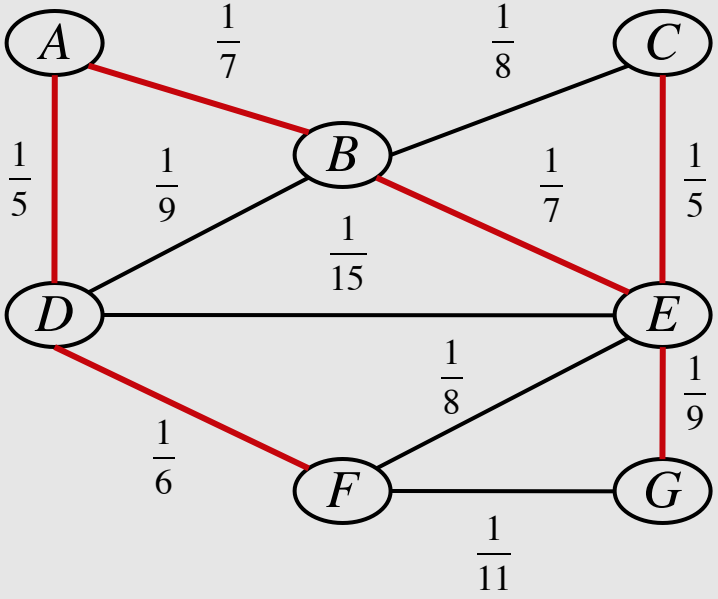


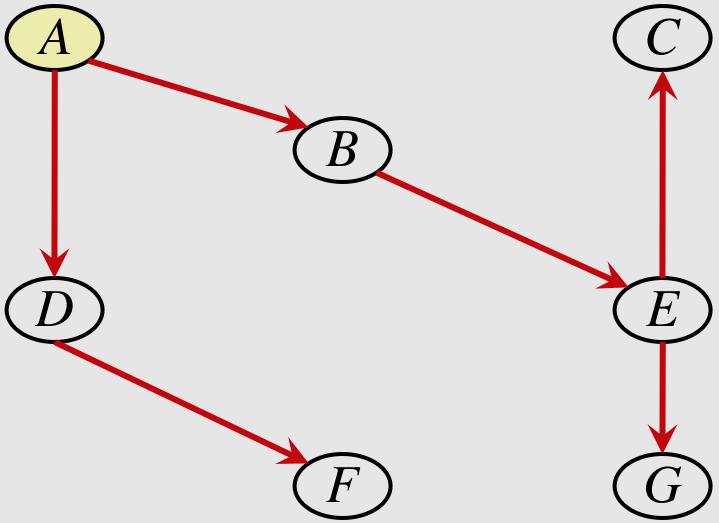
vi.



Returning directed graph in Chow-Liu

3. pick a node for the root, and assign edge directions





The Chow-Liu algorithm

- How do we know that Chow-Liu will find a tree that maximizes the data likelihood?
- Two key questions:
  - Why can we represent data likelihood as sum of $I(X;Y)$ over edges?
  - Why can we pick any direction for edges in the tree?

Why Chow-Liu maximizes likelihood (for a tree)

data likelihood given directed edges

$$\log_2 P(D \mid G, \theta_G) = \sum_{d \in D} \sum_i \log_2 P(x_i^{(d)} \mid Parents(X_i))$$

$$\mathrm{E}\left[\log_2 P(D \mid G, \theta_G)\right] = \mid D \mid \sum_i (I(X_i, Parents(X_i)) - H(X_i))$$

we're interested in finding the graph G that maximizes this

$$\arg\max_G \log_2 P(D \mid G, \theta_G) = \arg\max_G \sum_i I(X_i, Parents(X_i))$$

if we assume a tree, each node has at most one parent

$$\arg\max_G \log_2 P(D \mid G, \theta_G) = \arg\max_G \sum_{(X_i, X_j) \in \text{edges}} I(X_i, X_j)$$

edge directions don't matter for likelihood, because MI is symmetric

$$I(X_i, X_j) = I(X_j, X_i)$$

Goals for Part 3

you should understand the following concepts

- structure learning as search
- Kullback-Leibler divergence
- the Sparse Candidate algorithm

- the Tree Augmented Network (TAN) algorithm

Heuristic search for structure learning

- each state in the search space represents a DAG Bayes net structure
- to instantiate a search approach, we need to specify
  - scoring function
  - state transition operators
  - search algorithm

Scoring function decomposability

- when the appropriate priors are used, and all instances in D are complete, the scoring function can be decomposed as follows

$$\text{score}(G, D) = \sum_i \text{score}(X_i, Parents(X_i) : D)$$

- thus we can
  - score a network by summing terms over the nodes in the network
  - efficiently score changes in a *local* search procedure

Scoring functions for structure learning

- Can we find a good structure just by trying to maximize the likelihood of the data?

$$\arg\max_{G,\theta_G} \log P(D \mid G, \theta_G)$$

- If we have a strong restriction on the the structures allowed (e.g. a tree), then maybe.

- Otherwise, no! Adding an edge will never decrease likelihood. Overfitting likely.

Scoring functions for structure learning

- there are many different scoring functions for BN structure search
- one general approach

$$\arg\max_{G,\theta_G} \log P(D \mid G, \theta_G) - f(m) \mid \theta_G \mid$$

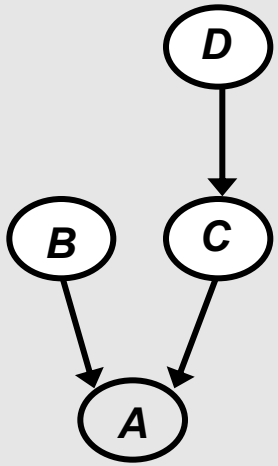
complexity penalty

Akaike Information Criterion (AIC): $f(m) = 1$

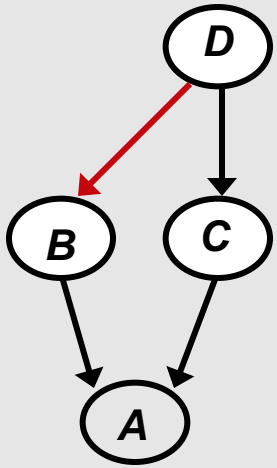
Bayesian Information Criterion (BIC): $f(m) = \frac{1}{2}\log(m)$

Structure search operators

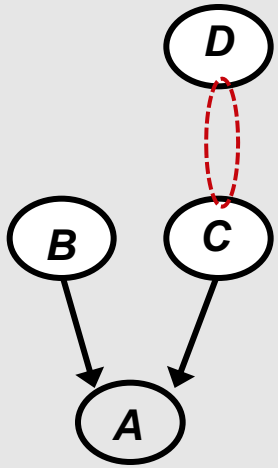
given the current network
at some stage of the search,
we can…



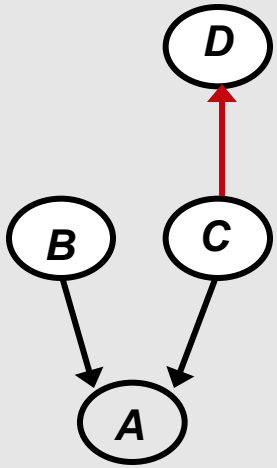
add an edge



delete an edge



reverse an edge



Bayesian network search: *hill-climbing*

given: data set D, initial network B_0

$i = 0$
$\mathrm{B}_{best} \leftarrow B_0$
while stopping criteria not met
{
  for each possible operator application a
  {
    $B_{new} \leftarrow \text{apply}(a, B_i)$
    if score(B_{new}) > score(B_{best})
      $\mathrm{B}_{best} \leftarrow B_{new}$
  }
  $++i$
  $B_i \leftarrow B_{best}$
}
return B_i

Bayesian network search: the *Sparse Candidate* algorithm [Friedman et al., *UAI* 1999]

given: data set D, initial network B_0, parameter k

$i = 0$

repeat

{

$++i$

// restrict step

select for each variable X_j a set C_j^i of candidate parents ($|C_j^i| \leq k$)

// maximize step

find network B_i maximizing score among networks where $\forall X_j$, Parents(X_j) $\subseteq C_j^i$

} until convergence

return B_i

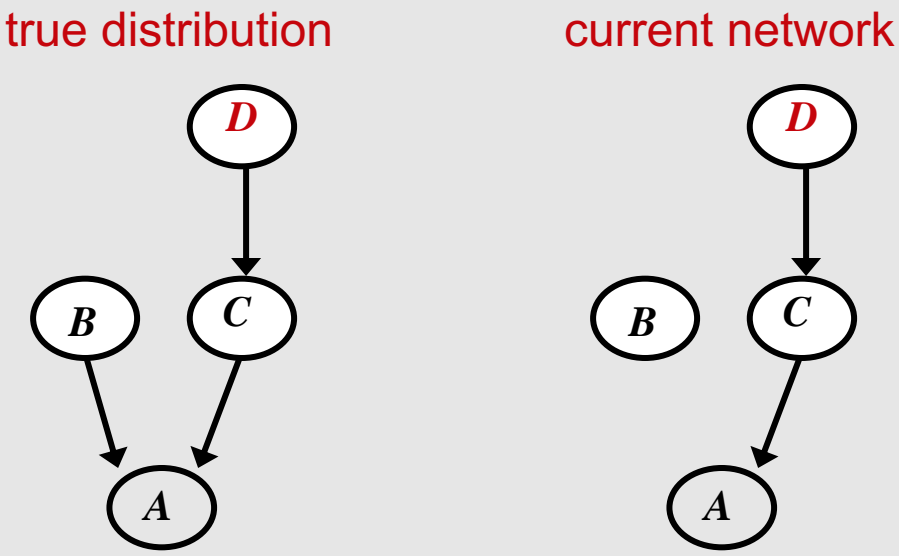
The *restrict* step in Sparse Candidate

- to identify candidate parents in the first iteration, can compute the *mutual information* between pairs of variables

$$I(X,Y) = \sum_{x \in \text{values}(X)} \sum_{y \in \text{values}(Y)} P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)}$$

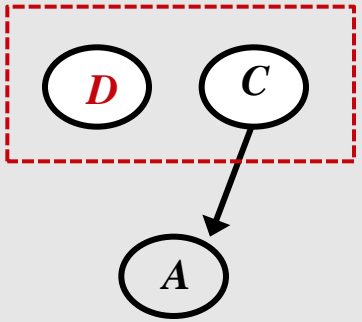
The *restrict* step in Sparse Candidate

- Suppose:



we're selecting two candidate parents for A, and $I(A, C) > I(A, D) > I(A, B)$

- with mutual information, the candidate parents for A would be C and D



- how could we get B as a candidate parent?

The *restrict* step in Sparse Candidate

- *Kullback-Leibler* (KL) *divergence* provides a distance measure between two distributions, P and Q

$$D_{KL}(P(X) \| Q(X)) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

- mutual information can be thought of as the KL divergence between the distributions

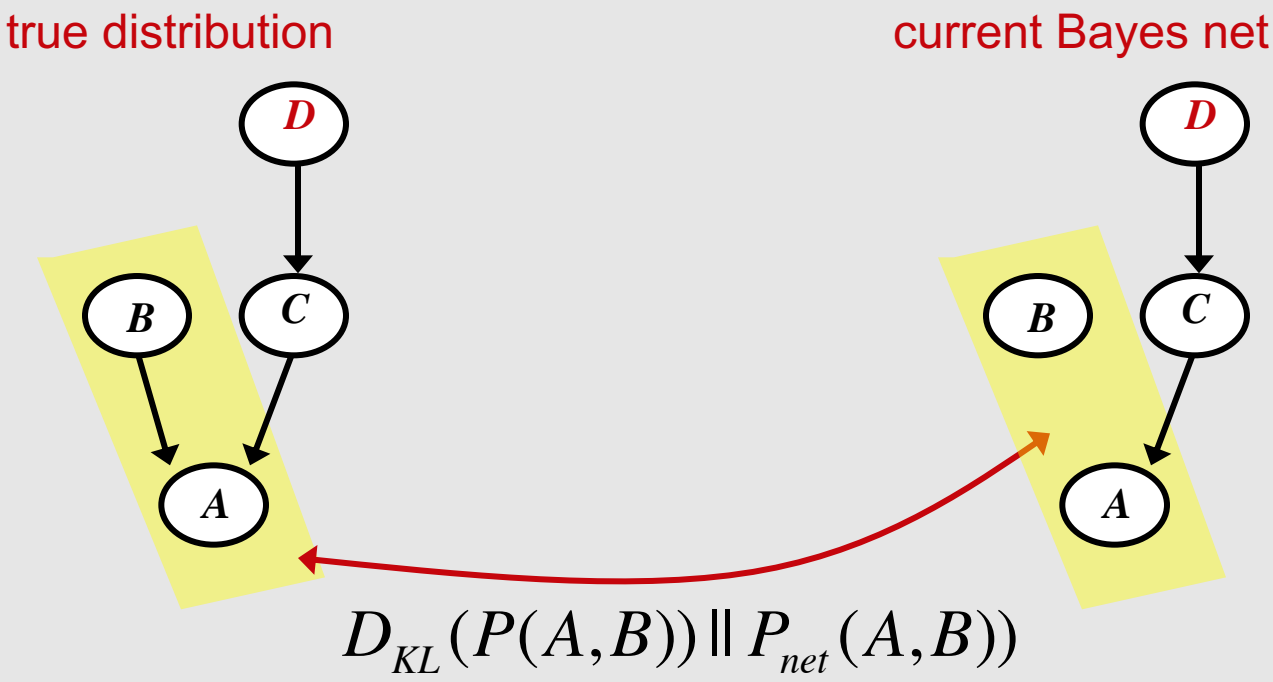
$$P(X,Y)$$

$$P(X)P(Y)$$ (assumes X and Y are independent)

The *restrict* step in Sparse Candidate

- we can use KL to assess the discrepancy between the network's $P_{net}(X, Y)$ and the empirical $P(X, Y)$

$$M(X,Y) = D_{KL}(P(X,Y)) \| P_{net}(X,Y))$$



- can estimate $P_{net}(X, Y)$ by sampling from the network (i.e. using it to generate instances)

The *restrict* step in Sparse Candidate

given: data set D, current network B_i, parameter k

for each variable X_j

{

calculate M(X_j , X_l) for all $X_j \neq X_l$ such that $X_l \notin$ Parents(X_j)

choose highest ranking $X_1 \ldots X_{k-s}$ where $s=$ | Parents(X_j) |

// include current parents in candidate set to ensure monotonic

// improvement in scoring function

C_j^i =Parents(X_j) $\cup$ $X_1 \ldots X_{k-s}$

}

return { C_j^i } for all X_j

The *maximize* step in Sparse Candidate

- hill-climbing search with *add-edge*, *delete-edge*, *reverse-edge* operators
- test to ensure that cycles aren't introduced into the graph

Efficiency of Sparse Candidate

n = number of variables

	possible parent sets for each node	changes scored on first iteration of search	changes scored on subsequent iterations
ordinary greedy search	$O(2^n)$	$O(n^2)$	$O(n)$
greedy search w/at most k parents	$O\left(\binom{n}{k}\right)$	$O(n^2)$	$O(n)$
Sparse Candidate	$O(2^k)$	$O(kn)$	$O(k)$

after we apply an operator, the scores will change only for edges from the parents of the node with the new impinging edge

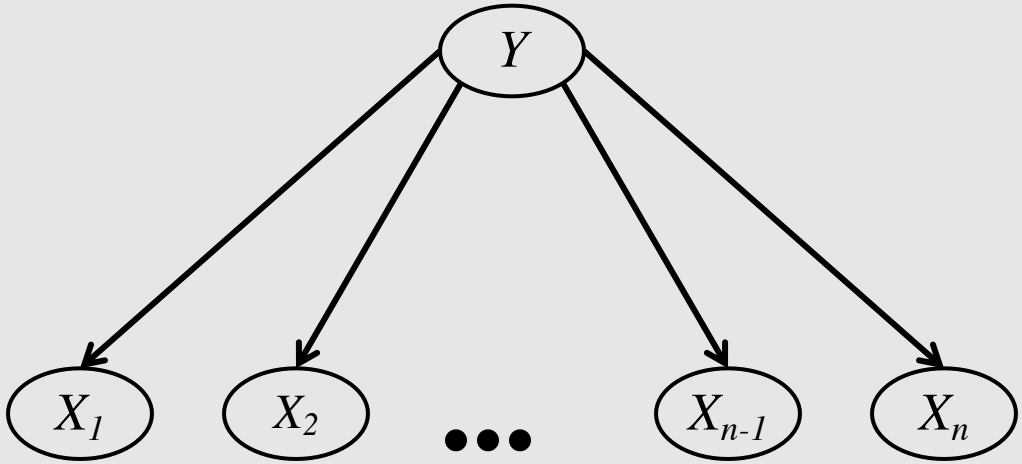
Bayes nets for classification

- the learning methods for BNs we've discussed so far can be thought of as being unsupervised
  - the learned models are not constructed to predict the value of a special class variable
  - instead, they can predict values for arbitrarily selected query variables

- now let's consider BN learning for a standard supervised task (learn a model to predict Y given $X_1 \ldots X_n$)

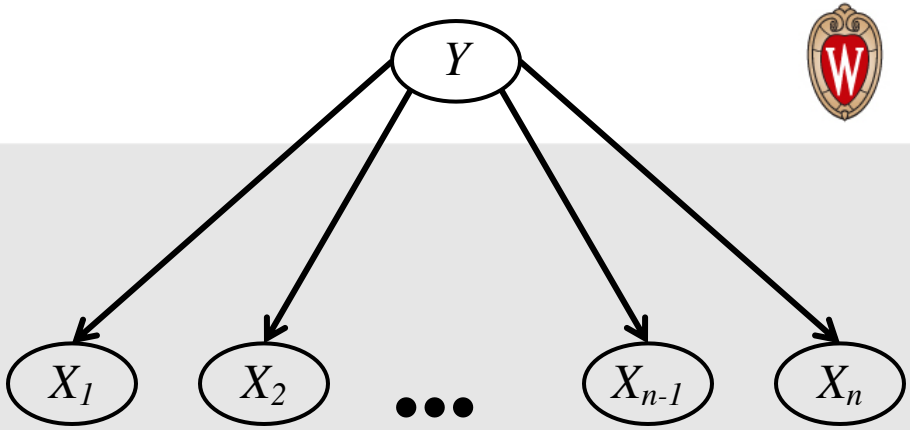
Naïve Bayes

- one very simple BN approach for supervised tasks is *naïve Bayes*
- in naïve Bayes, we assume that all features X_i are conditionally independent given the class Y



$$P(X_1,\ldots,X_n,Y) = P(Y)\prod_{i=1}^{n} P(X_i \mid Y)$$

Naïve Bayes



Learning

- estimate $P(Y = y)$ for each value of the class variable Y
- *estimate* $P(X_i = x \mid Y = y)$ *for each* X_i

Classification: use Bayes' Rule

$$P(Y = y \mid \boldsymbol{x}) = \frac{P(y)P(\boldsymbol{x} \mid y)}{\sum_{y'} P(y')P(\boldsymbol{x} \mid y')} = \frac{P(y)\prod_{i=1}^{n} P(x_i \mid y)}{\sum_{y'}\left(P(y')\prod_{i=1}^{n} P(x_i \mid y')\right)}$$

Naïve Bayes vs. BNs learned with an unsupervised structure search

test-set error on 25 classification data sets from the UC-Irvine Repository

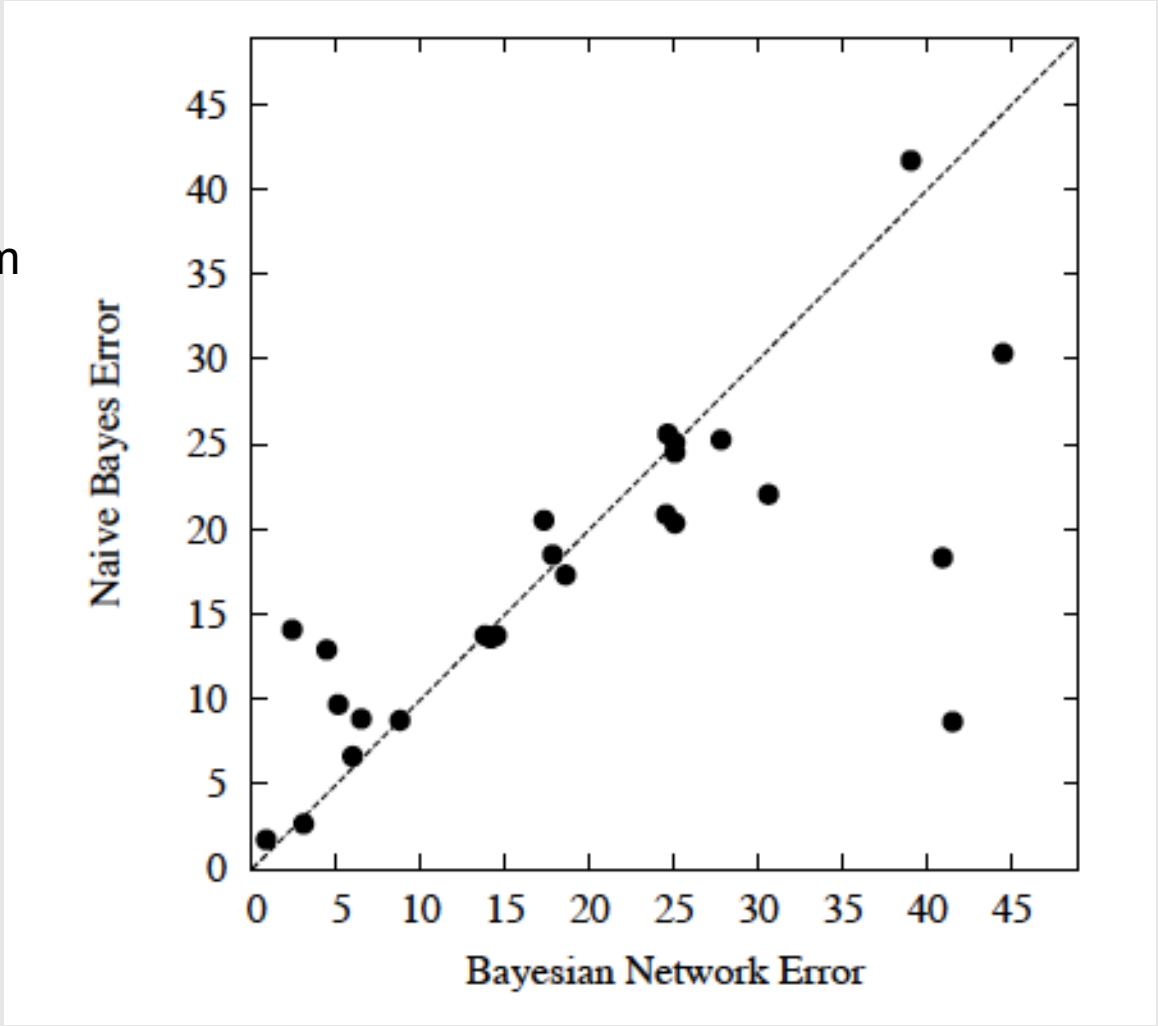


Figure from Friedman et al., *Machine Learning* 1997

The Tree Augmented Network (TAN) algorithm

[Friedman et al., *Machine Learning* 1997]

- learns a tree structure to augment the edges of a naïve Bayes network

- algorithm
  1. compute weight $I(X_i, X_j \mid Y)$ for each possible edge (X_i, X_j) between features
  2. find maximum weight spanning tree (MST) for graph over $X_1 \ldots X_n$
  3. assign edge directions in MST
  4. construct a TAN model by adding node for Y and an edge from Y to each X_i

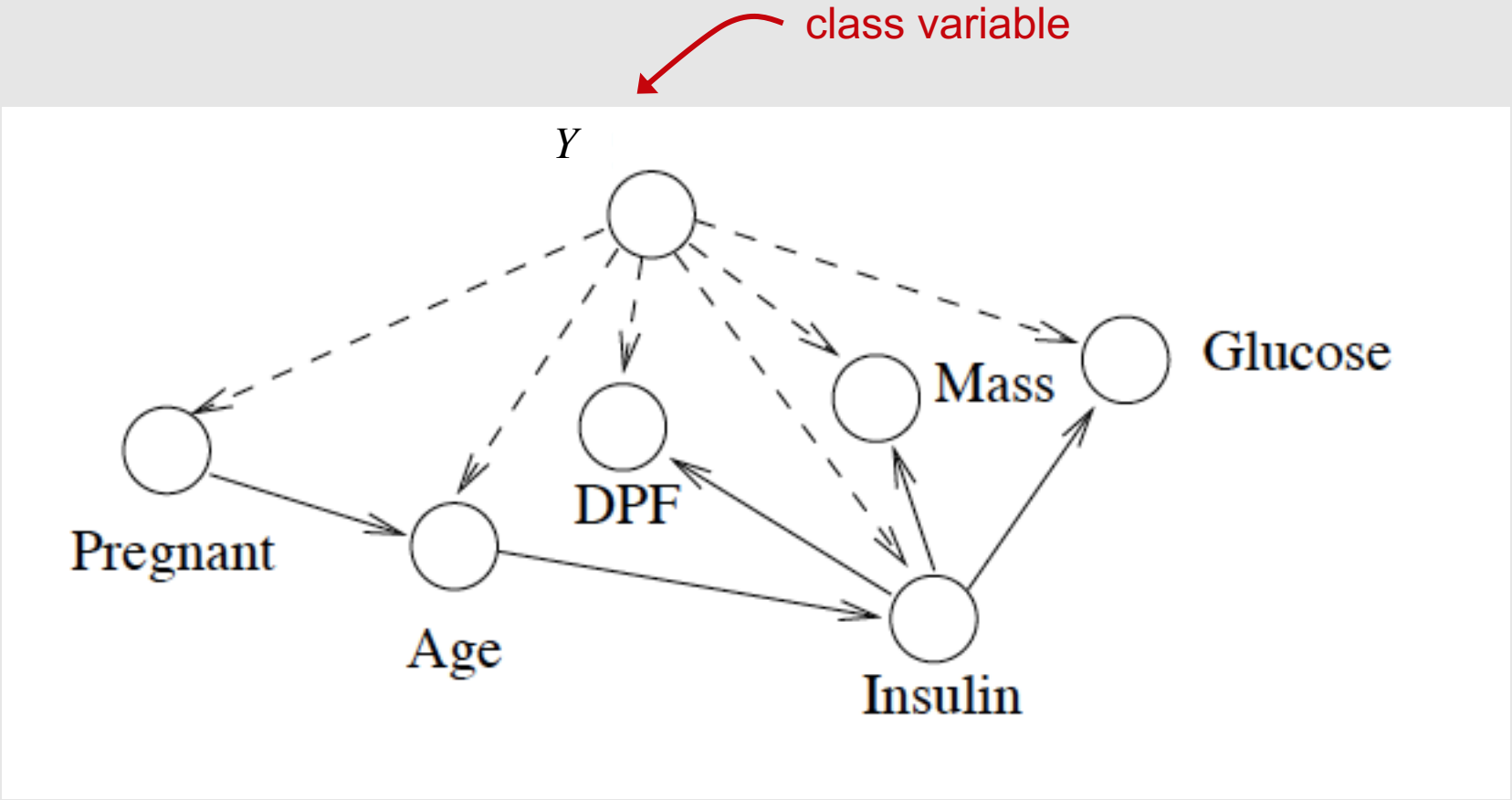
Conditional mutual information in TAN

conditional mutual information is used to calculate edge weights

$$I(X_i, X_j \mid Y) = \sum_{x_i \in \text{values}(X_i)} \sum_{x_j \in \text{values}(X_j)} \sum_{y \in \text{values}(Y)} P(x_i, x_j, y) \log_2 \frac{P(x_i, x_j \mid y)}{P(x_i \mid y) P(x_j \mid y)}$$

"how much information X_i provides about X_j when the value of Y is known"

Example TAN network



naïve Bayes edges

edges determined by MST

TAN vs. Chow-Liu

- TAN is focused on learning a Bayes net specifically for classification problems
- the MST includes only the feature variables (the class variable is used only for calculating edge weights)
- conditional mutual information is used instead of mutual information in determining edge weights in the undirected graph
- the directed graph determined from the MST is added to the $Y \rightarrow X_i$ edges that are in a naïve Bayes network

TAN vs. Naïve Bayes

test-set error on 25 data sets from the UC-Irvine Repository

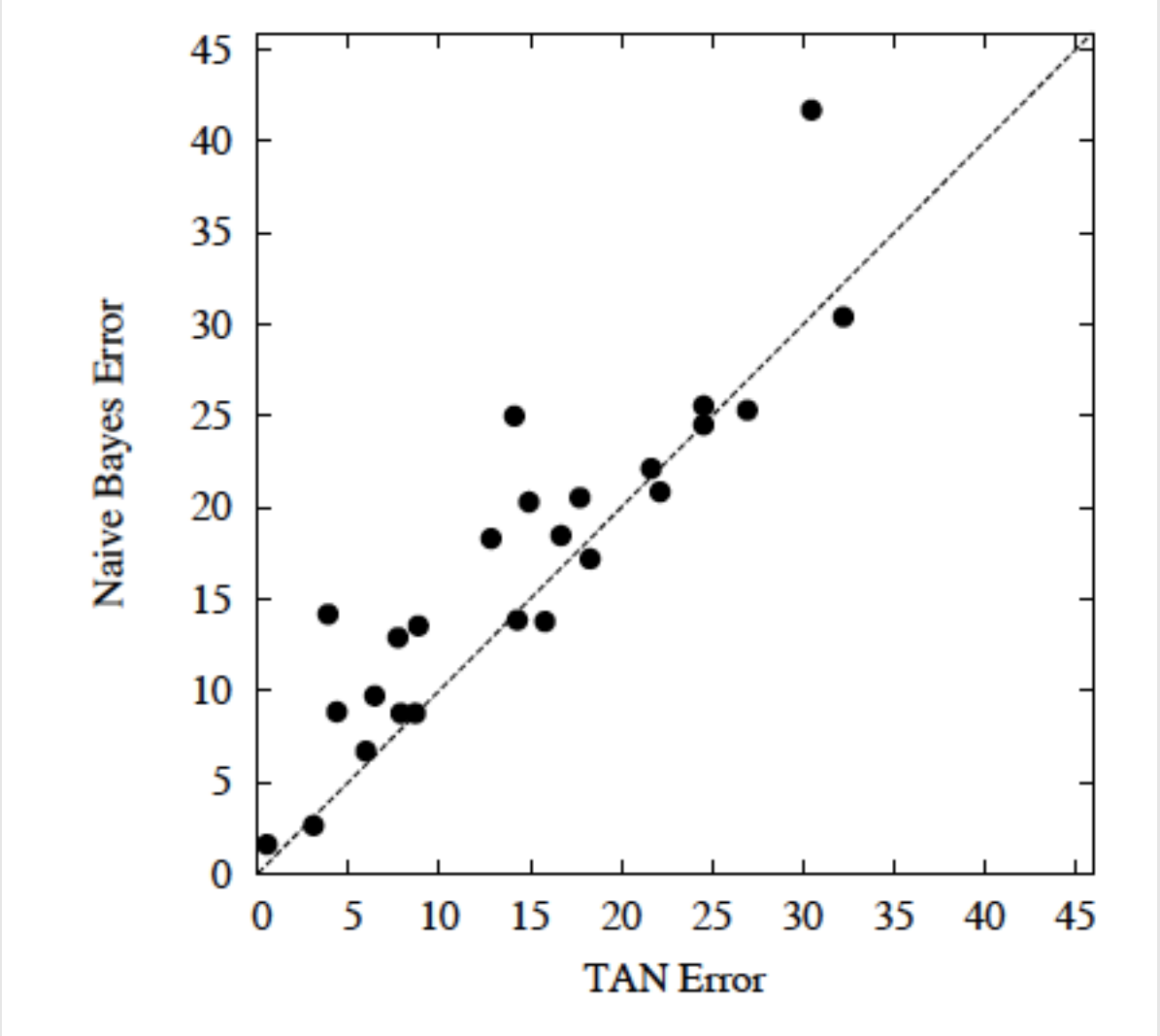


Figure from Friedman et al., *Machine Learning* 1997

Comments on Bayesian networks

- the BN representation has many advantages
  - easy to encode domain knowledge (direct dependencies, causality)
  - can represent uncertainty
  - principled methods for dealing with missing values
  - can answer arbitrary queries (in theory; in practice may be intractable)
- for supervised tasks, it may be advantageous to use a learning approach (e.g. TAN) that focuses on the dependencies that are most important
- although very simplistic, naïve Bayes often learns highly accurate models
- BNs are one instance of a more general class of *probabilistic graphical models*

THANK YOU

Some of the slides in these lectures have been adapted/borrowed from materials developed by Yingyu Liang, Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, and Pedro Domingos.