Learning Bayesian Networks (part 2)

Mark Craven and David Page
Computer Sciences 760
Spring 2018

www.biostat.wisc.edu/~craven/cs760/

Some of the slides in these lectures have been adapted/borrowed from materials developed by Tom Dietterich, Pedro Domingos, Tom Mitchell, David Page, and Jude Shavlik

Goals for the lecture

you should understand the following concepts

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

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

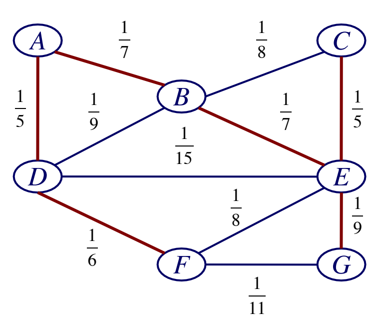
The Chow-Liu algorithm

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



Prim's algorithm for finding an MST

given: graph with vertices V and edges E

V_{new} ← { v } where v is an arbitrary vertex from V
E_{new} ← { }
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

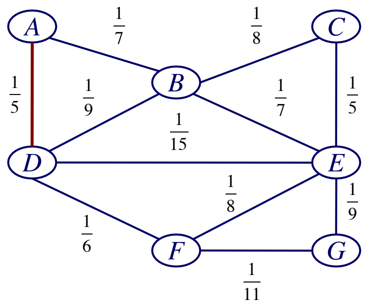
Kruskal's algorithm for finding an MST

given: graph with vertices V and edges E

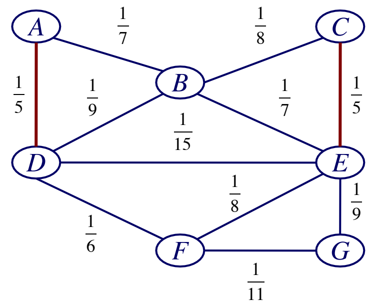
E_{new} ← { }
for each (u, v) in E ordered by weight (from high to low)
{
    remove (u, v) from E
    if adding (u, v) to E_{new} does not create a cycle
        add (u, v) to E_{new}
}
return V and E_{new} which represent an MST

Finding MST in Chow-Liu

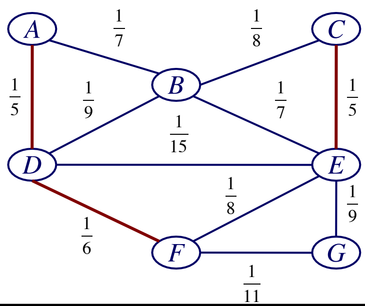
i.



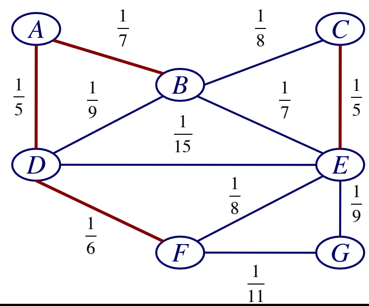
ii.



iii.

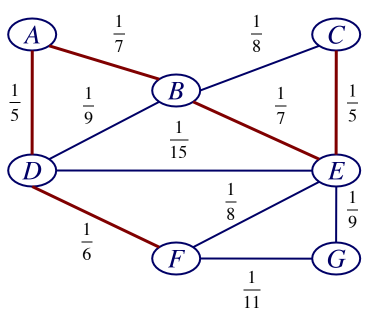


iv.

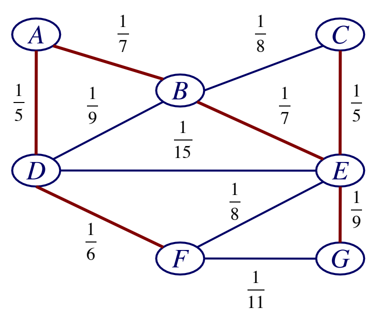


Finding MST in Chow-Liu

v.

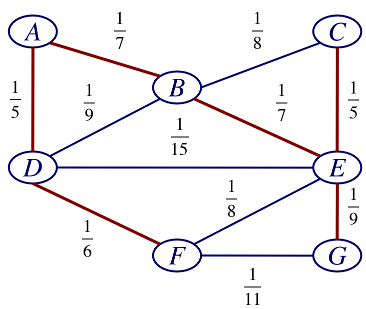


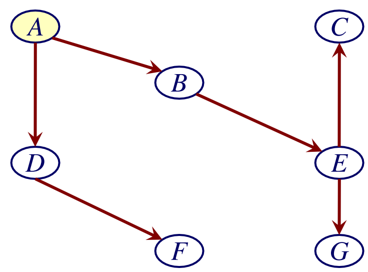
vi.



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

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

$$= |D| \sum_i \left(I(X_i, Parents(X_i)) - H(X_i) \right)$$

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

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

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

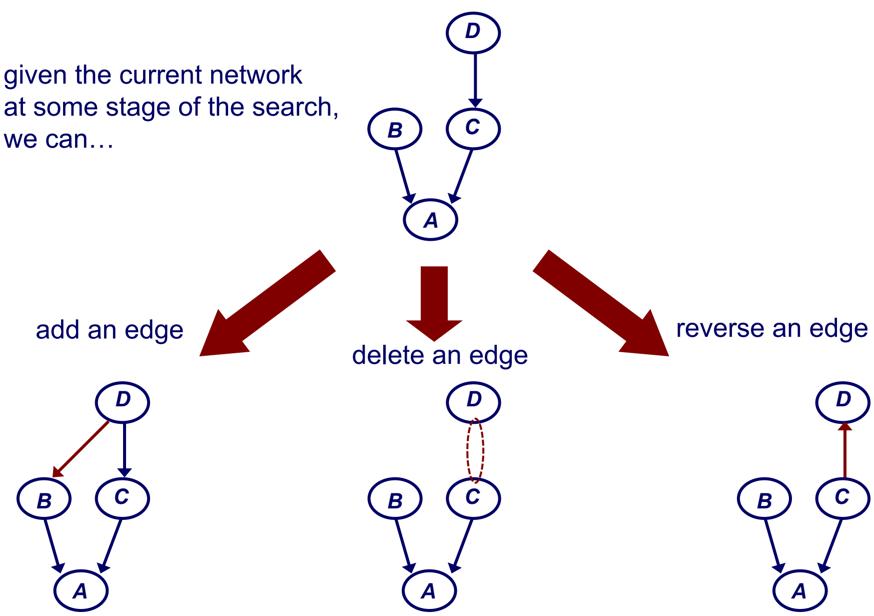
$$\arg\max_{G,\,\theta_G} \; \log P(D \mid G, \theta_G) - \underbrace{f(m)\left|\theta_G\right|}_{\text{complexity penalty}}$$

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…



Bayesian network search: *hill-climbing*

given: data set D, initial network B_0

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

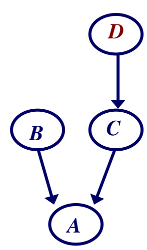
- to identify candidate parents in the <u>first</u> iteration, can compute the *mutual information* between pairs of variables

$$I(X,Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$$

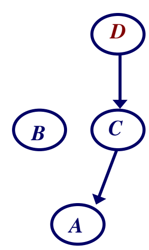
The *restrict* step in Sparse Candidate

- Suppose:

true distribution

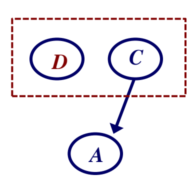


current network



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

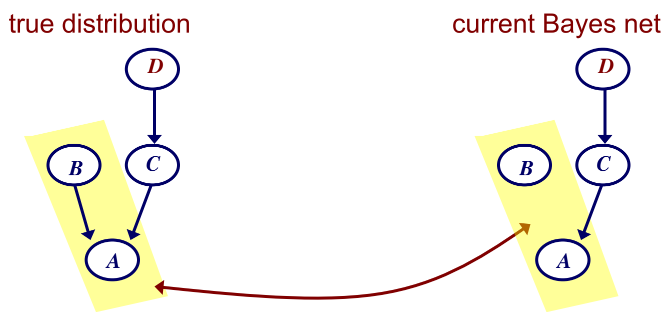
$$P(X,Y)$$

$$P(X)P(Y)$$ (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 $P(X, Y)$

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



$$D_{KL}(P(A,B)) \| P_{net}(A,B))$$

- 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