Probabilistic Graphical Models

Prof. Joschka Boedecker and Hao Zhu

Department of Computer Science
University of Freiburg

universität freiburg

4. Bayesian networks

- Representation

- Inference

- Parameter learning

- Structure learning

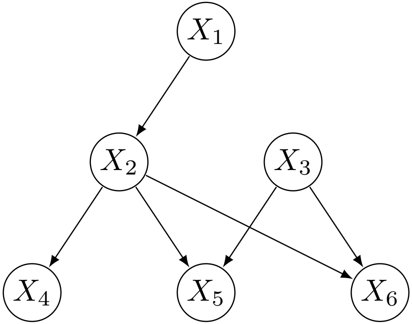
Outline

- Representation
- Inference
- Parameter learning
- Structure learning

Bayesian network

- **structure**: directed acyclic graph (DAG), each node corresponds to one variable
- **parameters**: conditional probability table, contains the probability of each instance of the variable given its parents

$$\mathbf{P}(X \mid \mathbf{pa}(X))$$



Structure

given a probability distribution $\mathbf{P}$ of $X = (X_1, \ldots, X_n)$, and its graphical representation G

mappings: correspondence between the conditional independence in $\mathbf{P}$ and in G

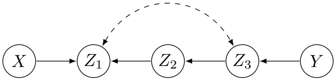
- types of mappings
  - D-map: all the conditional independence relations in $\mathbf{P}$ are satisfied in G
  - I-map: all the conditional independence relations in G are true in $\mathbf{P}$
  - P-map: or perfect map, it is a D-map and an I-map
- graph G and probability distribution $\mathbf{P}$ are compatible if G is an I-map of $\mathbf{P}$
- minimal I-map: all the conditional independence relations implied by G are true in P, and if any arc is deleted in G this condition is lost

Structure

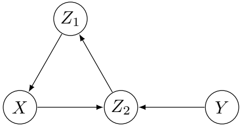
d-separation

given a graph G and sets of nodes X, Y, and Z

- a path p is d-separated (blocked) by a set of nodes Z if and only if
  1. p contains a chain $i \to m \to j$ or a fork $i \leftarrow m \to j \implies m \in Z$
  2. p contains a collider $i \to m \leftarrow j \implies m \notin Z$ & no descendant of m is in Z
- $(X \perp\!\!\!\perp Y \mid Z) \implies Z$ blocks every path from X to Y
- examples:



  - X and Y are d-separated given Z_2
  - X and Y are d-connected given Z_1



  - X and Y cannot be d-separated by any set of nodes

Structure

Markov assumption: any node X is conditionally independent of all nodes in graph G that are not descendants of X given $\mathbf{pa}(X)$

- $\mathbf{pa}(X)$: contour of X

Markov blanket

$$(X \perp\!\!\!\perp G_{-X} \mid \mathbf{mb}(X)) \Longleftrightarrow \mathbf{P}(X \mid G_{-X}) = \mathbf{P}(X \mid \mathbf{mb}(X))$$

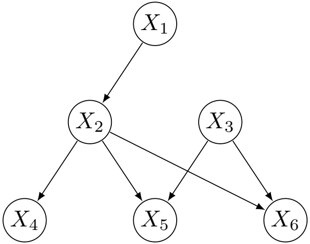
- Markov blanket of X consists of
  - the parents of X
  - the children of X
  - the other parents of the children of X

Structure

example: Bayesian network representation of probability distribution

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

$$\left\{ \begin{array}{l} \mathbf{pa}(X_1) = \emptyset, \\ \mathbf{pa}(X_2) = \{X_1\}, \\ \mathbf{pa}(X_3) = \emptyset, \\ \mathbf{pa}(X_4) = \{X_2\}, \\ \mathbf{pa}(X_5) = \{X_2, X_3\}, \\ \mathbf{pa}(X_6) = \{X_2, X_3\} \end{array} \right\}$$



$$\mathbf{P}(X_1, \ldots, X_6) = \mathbf{P}(X_1)\mathbf{P}(X_2 \mid X_1)\mathbf{P}(X_3)\mathbf{P}(X_4 \mid X_2)\mathbf{P}(X_5 \mid X_2, X_3)\mathbf{P}(X_6 \mid X_2, X_3)$$

Parameters

$$\mathbf{P}(X_i \mid \mathbf{pa}(X_i)), \quad i = 1, 2, \ldots$$

canonical models

- mostly for binary variables
- examples: noisy-OR, noisy-AND, noisy-max, noise-min

Parameters

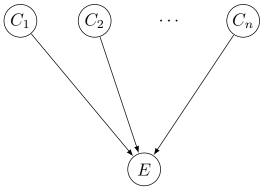
noisy-OR

- given effect E and possible causes $C_1, \dots, C_n$
- assumptions:
  - independence of exceptions (not necessarily true for $E = \text{true}$):

$$\mathbf{P}(E = \text{false} \mid C_1, \dots, C_n) = \prod_{i=1}^{n} \mathbf{P}(E = \text{false} \mid C_i)$$

  - responsibility:

$$\mathbf{P}(E = \text{false} \mid C_i = \text{false}) = 1, \quad i = 1, \dots, n$$



- representation: let $q_i = \mathbf{P}(E = \text{false} \mid C_i = \text{true})$, if k out of n causes are true

$$\mathbf{P}(E = \text{false} \mid C_1, \dots, C_n) = \prod_{i=1}^{k} q_i \quad \text{and} \quad \mathbf{P}(E = \text{true} \mid C_1, \dots, C_n) = 1 - \prod_{i=1}^{k} q_i$$

Parameters

example: noisy-OR

- $q_1 = q_2 = q_3 = 0.1$

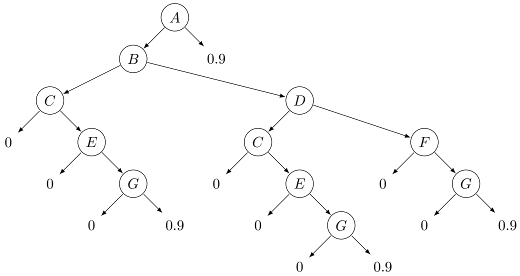
C_1	0	0	0	0	1	1	1	1
C_2	0	0	1	1	0	0	1	1
C_3	0	1	0	1	0	1	0	1
$\mathbf{P}(E=0)$	1	0.1	0.1	0.01	0.1	0.01	0.01	0.001
$\mathbf{P}(E=1)$	0	0.9	0.9	0.99	0.9	0.99	0.99	0.999

Parameters

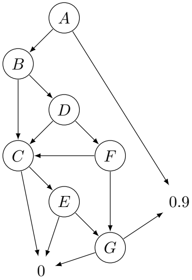
graphical representations

- idea: within each conditional probability table, the same probability values tend to be repeated several times

- structure: decision tree, decision diagram



decision tree



decision diagram

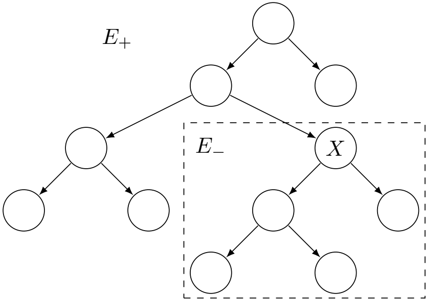
Outline

- Representation
- Inference
- Parameter learning
- Structure learning

Belief propagation

- exact inference for singly connected graphs (trees, polytrees)
- convergence is not guaranteed on general Bayesian networks

$$\mathbf{P}(x_i \mid E) = \frac{\mathbf{P}(E \mid x_i)\mathbf{P}(x_i)}{\mathbf{P}(E)}$$



- node X divide the network into two independent subtrees:
  - E_-: evidence of the rooted tree in X
  - E_+: all other evidence

Belief propagation

$$\begin{aligned}
\mathbf{P}(x_i \mid E) &= \frac{\mathbf{P}(E \mid x_i)\mathbf{P}(x_i)}{\mathbf{P}(E)} \\
&= \frac{\mathbf{P}(E_-, E_+ \mid x_i)\mathbf{P}(x_i)}{\mathbf{P}(E)} \\
&= \frac{\mathbf{P}(E_- \mid x_i)\mathbf{P}(E_+ \mid x_i)\mathbf{P}(x_i)}{\mathbf{P}(E)} \\
&= \frac{\mathbf{P}(E_- \mid x_i)\mathbf{P}(x_i \mid E_+)\mathbf{P}(E_+)\cancel{\mathbf{P}(x_i)}}{\mathbf{P}(E)\cancel{\mathbf{P}(x_i)}} \\
&= \frac{1}{Z}\mathbf{P}(x_i \mid E_+)\mathbf{P}(E_- \mid x_i) \\
&= \frac{1}{Z}\mu(x_i)\lambda(x_i)
\end{aligned}$$

- $\frac{1}{Z} = \frac{\mathbf{P}(E_+)}{\mathbf{P}(E)}$: normalization constant
- auxiliary variables:
  - μ message

    $$\mu(x_i) = \mathbf{P}(x_i \mid E_+)$$

  - λ message

    $$\lambda(x_i) = \mathbf{P}(E_- \mid x_i)$$

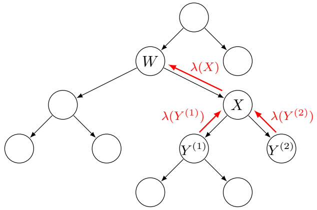
Belief propagation

bottom-up propagation

$$\lambda(x_i) = \mathbf{P}(E_- \mid x_i) = \prod_k \mathbf{P}(E_-^{(k)} \mid x_i)$$

- $E_-^{(k)}$: evidence coming from the tree rooted in the kth child $Y^{(k)}$ of X

$$\begin{aligned}\mathbf{P}(E_-^{(k)} \mid x_i) &= \sum_{y^{(k)} \in Y^{(k)}} \mathbf{P}(E_-^{(k)} \mid x_i, y^{(k)}) \mathbf{P}(y^{(k)} \mid x_i) \\ &= \sum_{y^{(k)} \in Y^{(k)}} \mathbf{P}(E_-^{(k)} \mid y^{(k)}) \mathbf{P}(y^{(k)} \mid x_i) \\ &= \sum_{y^{(k)} \in Y^{(k)}} \lambda(y^{(k)}) \mathbf{P}(y^{(k)} \mid x_i)\end{aligned}$$



Belief propagation

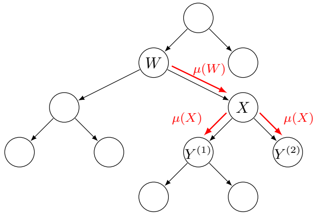
top-down propagation

$$\mu(x_i) = \mathbf{P}(x_i \mid E_+) = \sum_{w \in W} \mathbf{P}(x_i \mid E_+, w)\mathbf{P}(w \mid E_+) = \sum_{w \in W} \mathbf{P}(x_i \mid w)\mathbf{P}(w \mid E_+)$$

- $W = \mathbf{pa}(X)$: the parent node of X

$$\mathbf{P}(w \mid E_+) = \frac{1}{Z}\mu(w) \prod_{E_{W-}^{(k)} \neq E_-} \mathbf{P}(E_{W-}^{(k)} \mid w)$$

$$= \frac{1}{Z}\mu(w) \prod_{X^{(k)} \neq X} \sum_{x^{(k)} \in X^{(k)}} \lambda(x^{(k)})\mathbf{P}(x^{(k)} \mid w)$$

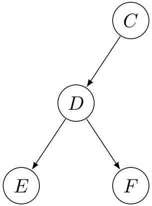


- $X^{(k)}$: the kth child of W
- $E_{W-}^{(k)}$: the evidence coming from the tree rooted in $X^{(k)}$

Belief propagation

example

1. bottom-up propagation:
   - initial conditions: $\lambda(E) = (1,0)$, $\lambda(F) = (1,1)$



$$\lambda(D) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}^T \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \end{bmatrix}^T \begin{bmatrix} 0.7 & 0.4 \\ 0.3 & 0.6 \end{bmatrix}$$

$$= \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix}$$

$$\lambda(C) = \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix}^T \begin{bmatrix} 0.9 & 0.7 \\ 0.1 & 0.3 \end{bmatrix} = \begin{bmatrix} 0.86 \\ 0.78 \end{bmatrix}$$

c_1	c_2
0.8	0.2

	c_1	c_2
d_1	0.9	0.7
d_2	0.1	0.3

	d_1	d_2
e_1	0.9	0.5
e_2	0.1	0.5

	d_1	d_2
f_1	0.7	0.4
f_2	0.3	0.6

- evidence: $E = e_1$

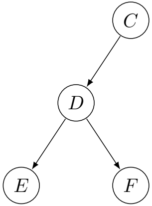
Belief propagation

2. top-down propagation
   - initial condition: $\mu(C) = (0.8, 0.2)$

$$\mu(D) = \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix}^T \begin{bmatrix} 0.9 & 0.1 \\ 0.7 & 0.3 \end{bmatrix} = \begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix}$$

$$\mu(F) = \left(\begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \end{bmatrix}^T \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix} \right)^T \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}$$

$$= \begin{bmatrix} 0.57 \\ 0.27 \end{bmatrix}$$



c_1	c_2
0.8	0.2

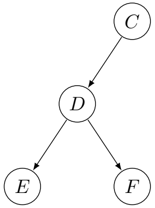
	c_1	c_2
d_1	0.9	0.7
d_2	0.1	0.3

	d_1	d_2
e_1	0.9	0.5
e_2	0.1	0.5

	d_1	d_2
f_1	0.7	0.4
f_2	0.3	0.6

- evidence: $E = e_1$

Belief propagation



3. obtain the posterior probabilities

$$\mathbf{P}(C) = \frac{1}{Z}\begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix} \odot \begin{bmatrix} 0.86 \\ 0.78 \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} 0.69 \\ 0.16 \end{bmatrix} = \begin{bmatrix} 0.81 \\ 0.19 \end{bmatrix}$$

$$\mathbf{P}(D) = \frac{1}{Z}\begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix} \odot \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} 0.77 \\ 0.07 \end{bmatrix} = \begin{bmatrix} 0.92 \\ 0.08 \end{bmatrix}$$

$$\mathbf{P}(F) = \frac{1}{Z}\begin{bmatrix} 0.57 \\ 0.27 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} 0.57 \\ 0.27 \end{bmatrix} = \begin{bmatrix} 0.68 \\ 0.32 \end{bmatrix}$$

c_1	c_2
0.8	0.2

	c_1	c_2
d_1	0.9	0.7
d_2	0.1	0.3

	d_1	d_2
e_1	0.9	0.5
e_2	0.1	0.5

	d_1	d_2
f_1	0.7	0.4
f_2	0.3	0.6

- evidence: $E = e_1$

Variable elimination

given a Bayesian network representing the joint probability distribution

$$X = \{X_1, \dots, X_n\} = \{X_H \cup X_E \cup X_R\}$$

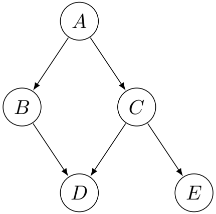
- X_H: variables to calculate the posterior probability
- X_E: evidence variables
- X_R: remaining variables

$$\mathbf{P}(X_H \mid X_E) = \frac{\mathbf{P}(X_H, X_E)}{\mathbf{P}(X_E)}$$

$$\mathbf{P}(X_H, X_E) = \sum_{X_R} \mathbf{P}(X) \quad \text{and} \quad \mathbf{P}(X_E) = \sum_{X_H} \mathbf{P}(X_H, X_E)$$

Variable elimination

example

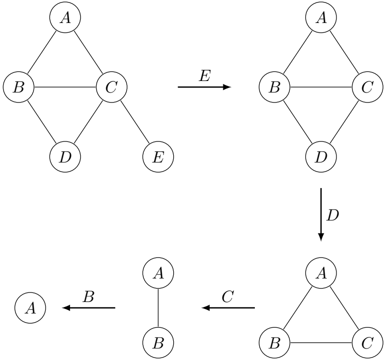


$$\begin{aligned}
\mathbf{P}(A, D) &= \sum_B \sum_C \sum_E \mathbf{P}(A, B, C, D, E) \\
&= \sum_B \sum_C \sum_E \mathbf{P}(A)\mathbf{P}(B \mid A)\mathbf{P}(C \mid A)\mathbf{P}(D \mid B, C)\mathbf{P}(E \mid C) \\
&= \mathbf{P}(A) \sum_B \left[\mathbf{P}(B \mid A) \sum_C \left[\mathbf{P}(C \mid A)\mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C)\right]\right]
\end{aligned}$$

Variable elimination

interaction graph

- heuristic for selecting a good elimination order
- obtaining interaction graphs through elimination:
  - eliminate the direction of the arcs from the original Bayesian network, and add additional arcs between each pair of non-connected variables having common children
  - each time a variable X_j is eliminated, the interaction graph is modified by adding an arc between each pair of neighbors of X_j that are not connected, and deleting variable X_j from the graph



Variable elimination

- **min-degree elimination**: eliminate the variable with the smallest number of neighbors in the current interaction graph
- **min-fill elimination**: eliminate the variable that leads to adding the minimum number of edges to the interaction graph

Conditioning

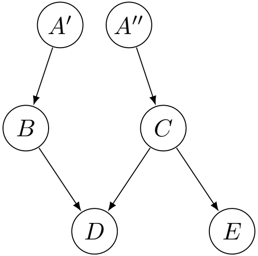
idea: instantiated variables block the propagation of the evidence in a Bayesian network

- cut the graph at instantiated variables, transform a multi-connected graph into a polytree
- apply the belief propagation algorithm

- if the variables to instantiate are unknown, set them to each of their possible values, and then do propagation for each value
- the posterior probabilities for unknown variables are then a weighted combination of the probabilities from each propagation

Conditioning

suppose instantiating variable A transforms the multi-connected Bayesian network to a polytree



$$\mathbf{P}(X \mid E) = \sum_{a \in A} \mathbf{P}(X \mid E, a)\mathbf{P}(a \mid E)$$

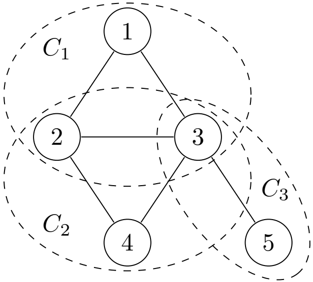
- $\mathbf{P}(X \mid E, a)$: posterior probability of X obtained from belief propagation for each $a \in A$
- $\mathbf{P}(a \mid E)$: combination weight

$$\mathbf{P}(a \mid E) = \frac{1}{Z}\mathbf{P}(a)\mathbf{P}(E \mid a)$$

- $\frac{1}{Z}$: normalization constant
- $\mathbf{P}(a)$: obtained from belief propagation without evidence
- $\mathbf{P}(E \mid a)$: probability of evidence variables obtained from propagation with $A = a$

Graph theory background for junction tree algorithm

- G is a **complete graph** if there is an edge between each pair of nodes
- **complete set** of G is a set that induces a complete subgraph of G
- **clique** C is a subset of graph G that is a maximal complete set (there is no other complete set in G that contains C)



Graph theory background for junction tree algorithm

ordering

given graph $G = (V, E)$ with n nodes

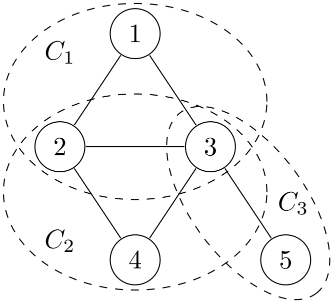
- ordering α assigns a label to each node

$$\alpha = (V_1, \ldots, V_n)$$

- V_i is before V_j according to the ordering if $i < j$
- an ordering α of G is a **perfect ordering** if

$$\mathbf{adj}(V_i) \cap \{V_1, \ldots, V_{i-1}\}$$

  is a complete subgraph of G for all V_i



Graph theory background for junction tree algorithm

running intersection property

given graph $G = (V, E)$ with m cliques

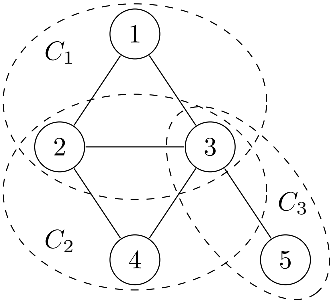
- ordering β assigns a label to each clique

$$\beta = (C_1, \ldots, C_m)$$

- an ordering β has the running intersection property if for every C_i with $i > 1$, there exists some C_j with $j < i$ such that

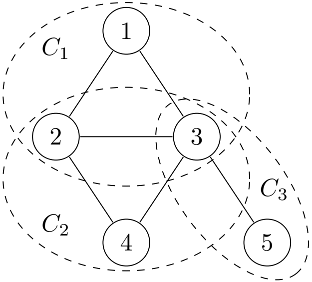
$$C_i \cap \{C_1, \ldots, C_{i-1}\} \subseteq C_j$$

  - C_j: the parent of C_i



Graph theory background for junction tree algorithm

- **chord** is an edge that connects two of the nodes in a circuit but is not part of the circuit



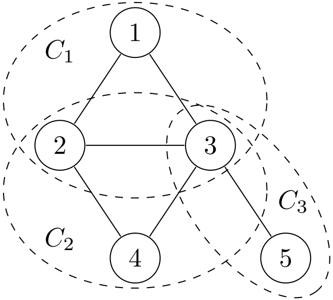
  - the circuit $1 \to 2 \to 4 \to 3 \to 1$ has a chord that connects nodes 2 and 3

- G is **triangulated** if every simple circuit of length greater than three in G has a chord

Graph theory background for junction tree algorithm

maximum cardinality search

- finding a perfect ordering of triangulated graphs



---

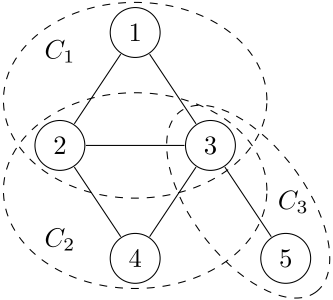
given triangulated graph $G = (V, E)$ with n nodes.
1. Assign index 1 to any node from V.
repeat
2. Assign the next index to one non-indexed node with the highest number of adjacent indexed nodes.
until all nodes are numbered.

Graph theory background for junction tree algorithm

given a perfect ordering, the following process orders the cliques of G that has the running intersection property:

---

given perfectly ordered, triangulated graph $G = (V, E)$ with m cliques.
1. Assign index m to the clique that has the node with the highest index.
repeat
2. Assign index $m-1$ to one non-indexed clique that includes the next highest indexed node.
until all cliques are numbered.



Junction tree algorithm

idea: transform Bayesian network to singly connected graph via clustering of nodes

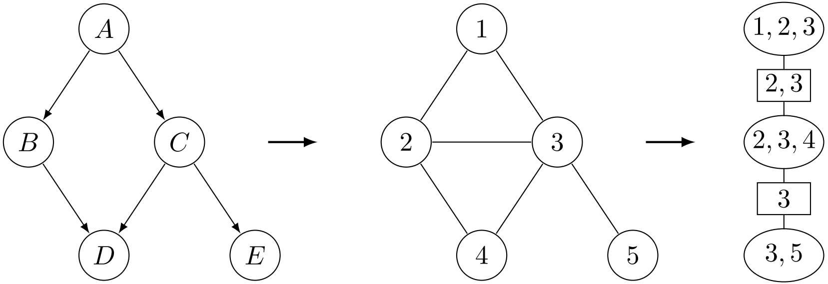
process: transformation + belief propagation

transformation

1. Eliminate the directionality of the arcs.
2. Moralize the graph by adding an arc between pairs of nodes with common children, and add additional arcs if necessary to make the graph triangulated.
3. Order the nodes in the graph with maximum cardinality search.
4. Obtain and order the cliques of the graph such that the order satisfies the running intersection property.
5. Build a junction tree according to the clique ordering.

Junction tree algorithm

example



Junction tree algorithm

preprocessing

1. Determine the set of variables for each clique C_i.
2. Determine the set of variables that are shared with the previous (parent) clique S_i.
3. Determine the set of other variables R_i that are in C_i but not in S_i.
4. Calculate the potential of each clique as $\psi(C_i) = \prod_{X \in R_i} \mathbf{P}(X \mid \mathbf{pa}(X))$.

Junction tree algorithm

bottom-up propagation

---

1. Start from the leaf clique, calculate the λ message to send to the parent clique: $\lambda(C_i) = \sum_{R_i} \psi(C_i)$.
2. Update the potential of each clique with the λ messages from its children: $\psi'(C_j) = \lambda(C_i)\psi(C_j)$.
3. Repeat the previous two steps until reaching the root clique, and obtain $\mathbf{P}(C_{\text{root}}) = \psi'(C_{\text{root}})$.

Junction tree algorithm

top-down propagation

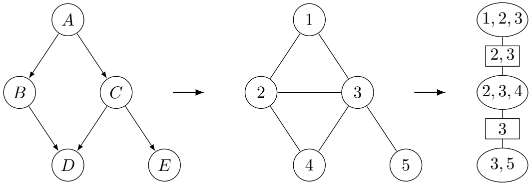
---

1. Start from the root clique, calculate the μ message to send to each child node C_i by its parent C_j: $\mu(C_i) = \sum_{C_j - S_i} \mathbf{P}(C_j)$.
2. Update the potential of each clique when receiving the μ message from its parent and obtain: $\mathbf{P}(C_i) = \psi'(C_i)\frac{\mu(C_i)}{\lambda(C_i)}$.
3. Repeat the previous two steps until reaching the leaf nodes in the junction tree.

- after belief propagation, each clique has the joint marginal probability of the variables that conform it
- the marginal posterior probabilities of each variable can be obtained from the clique via marginalization

Junction tree algorithm

example



1. preprocessing:

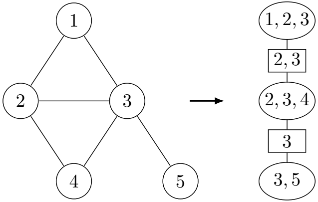
$$
\begin{array}{lll}
C_1 = \{A, B, C\} & C_2 = \{B, C, D\} & C_3 = \{C, E\} \\
S_1 = \emptyset & S_2 = \{B, C\} & S_3 = \{C\} \\
R_1 = \{A, B, C\} & R_2 = \{D\} & R_3 = \{E\} \\
\psi(C_1) = \mathbf{P}(A)\mathbf{P}(B \mid A)\mathbf{P}(C \mid A) & \psi(C_2) = \mathbf{P}(D \mid B, C) & \psi(C_3) = \mathbf{P}(E \mid C)
\end{array}
$$

Junction tree algorithm

2. bottom-up propagation:

$$\lambda(C_3) = \sum_E \psi(C_3) = \sum_E \mathbf{P}(E \mid C)$$

$$\psi'(C_2) = \psi(C_2)\lambda(C_3) = \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C)$$

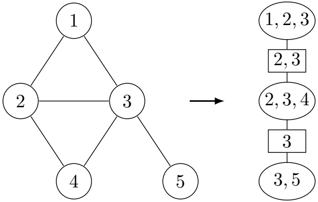


$$\lambda(C_2) = \sum_D \psi'(C_2) = \sum_D \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C)$$

$$\begin{aligned}\psi'(C_1) &= \psi(C_1)\lambda(C_2) \\ &= \mathbf{P}(A)\mathbf{P}(B \mid A)\mathbf{P}(C \mid A) \sum_D \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C)\end{aligned}$$

Junction tree algorithm

$$
\begin{aligned}
\mathbf{P}(C_1) &= \psi'(C_1) \\
&= \mathbf{P}(A)\mathbf{P}(B \mid A)\mathbf{P}(C \mid A) \sum_{D} \mathbf{P}(D \mid B, C) \sum_{E} \mathbf{P}(E \mid C) \\
&= \sum_{D,E} \mathbf{P}(A)\mathbf{P}(B \mid A)\mathbf{P}(C \mid A)\mathbf{P}(D \mid B, C)\mathbf{P}(E \mid C) \\
&= \sum_{D,E} \mathbf{P}(A, B, C, D, E) \\
&= \mathbf{P}(A, B, C)
\end{aligned}
$$

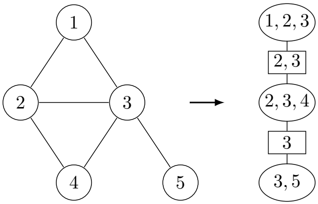


Junction tree algorithm

3. top-down propagation

$$\mu(C_2) = \sum_{C_1 - S_2} \mathbf{P}(C_1) = \sum_A \mathbf{P}(A, B, C)$$

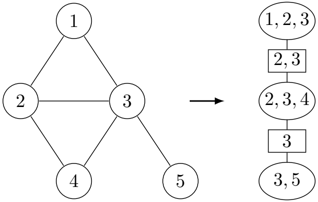
$$\begin{aligned}
\mathbf{P}(C_2) &= \psi'(C_2)\frac{\mu(C_2)}{\lambda(C_2)} \\
&= \frac{\mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C) \sum_A \mathbf{P}(A, B, C)}{\sum_D \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C)} \\
&= \mathbf{P}(D \mid B, C)\mathbf{P}(B, C) \\
&= \mathbf{P}(B, C, D)
\end{aligned}$$



Junction tree algorithm

$$\mu(C_3) = \sum_{C_2 - S_3} \mathbf{P}(C_2) = \sum_{B,D} \mathbf{P}(B, C, D)$$

$$\begin{aligned}
\mathbf{P}(C_3) &= \psi'(C_3)\frac{\mu(C_3)}{\lambda(C_3)} \\
&= \frac{\mathbf{P}(E \mid C) \sum_{B,D} \mathbf{P}(B, C, D)}{\sum_E \mathbf{P}(E \mid C)} \\
&= \mathbf{P}(E \mid C)\mathbf{P}(C) \\
&= \mathbf{P}(C, E)
\end{aligned}$$



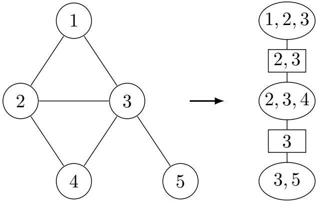
Junction tree algorithm

4. marginalization

$$\mathbf{P}(A) = \sum_{B,C} \mathbf{P}(C_1)$$
$$\mathbf{P}(B) = \sum_{A,C} \mathbf{P}(C_1)$$
$$\mathbf{P}(C) = \sum_{A,B} \mathbf{P}(C_1)$$

$$\mathbf{P}(D) = \sum_{B,C} \mathbf{P}(C_2)$$

$$\mathbf{P}(E) = \sum_{C} \mathbf{P}(C_3)$$



Sampling based methods

idea

- simulate the Bayesian network several times
- posterior probabilities of unknown variables are approximated by the frequency of each value in the sample space
- estimation accuracy depends on the number of samples
- computational cost is not affected by the complexity of the network

Sampling based methods

logic sampling

1. Generate sample values for all root nodes of the Bayesian network according to their prior probabilities $\mathbf{P}(X)$.
2. Generate samples for the children of the sampled nodes, according to their conditional probabilities $\mathbf{P}(X \mid \mathbf{pa}(X))$.
3. Repeat the second step until all leaf nodes are reached.

$$\mathbf{P}(X = x_k) = \frac{1}{n} \sum_{i=1}^{n} I_{x_k}(x_i)$$

- $I_{x_k}(x_i) = 1$ if $x_k = x_i$, and 0 otherwise
- if there is evidence, all samples that are not consistent with the evidence are discarded

Sampling based methods

likelihood weighting

- generate weights for all samples instead of discarding the non-consistent ones

given non-instantiated nodes H and evidence E, calculate weight for each sample i:

$$w_i = \mathbf{P}(E \mid H_i)$$

then the posterior probability of possible values of each variable is estimated as a weighted average over all n samples:

$$\mathbf{P}(X = x_k) = \frac{\sum_{i=1}^{n} w_i I_{x_k}(x_i)}{\sum_{i=1}^{n} w_i}$$

Outline

- Representation
- Inference
- Parameter learning
- Structure learning

Parameter learning

objective

- **given** the network structure
- **estimating** the conditional probability tables from data

example: estimate the conditional probability table for variable C with two parents A and B given the observed n samples

$$\mathbf{P}(C = c_k \mid A = a_i, B = b_j) = \frac{\sum_{i'=1}^{n} I_{a_i,b_j,c_k}(a_{i'}, b_{i'}, c_{i'})}{\sum_{i'=1}^{n} I_{a_i,b_j}(a_{i'}, b_{i'})}$$

- I is the indicator function
- useful only if the dataset is ‘good’

Smoothing

objective: dealing with non-observed events, which leads to zero probability value

idea: estimate the posterior distribution of the parameters given some priors

uniform prior (additive smoothing)

given m-valued discrete variable X, and a dataset with n samples

$$\mathbf{P}(x_i) = \frac{\alpha + \sum_{i'=1}^{n} I_{x_i}(x_{i'})}{\alpha m + n}, \quad i = 1, \ldots, m$$

- with no observed sample:

$$\mathbf{P}(x_i) = \frac{1}{m}, \quad i = 1, \ldots, m$$

- parameter estimation converges to the true data distribution with $n \to \infty$:

$$\lim_{n\to\infty} \mathbf{P}(x_i) = \lim_{n\to\infty} \frac{\alpha + \sum_{i'=1}^{n} I_{x_i}(x_{i'})}{\alpha m + n} = \frac{\sum_{i'=1}^{n} I_{x_i}(x_{i'})}{n}, \quad i = 1, \ldots, m$$

Smoothing

Beta prior

for random variable $X \sim \mathrm{Beta}(\alpha, \beta)$:

$$\mathbf{E}_{\mathrm{Beta}(\alpha,\beta)}[X] = \mathbf{P}(X = 1 \mid \alpha, \beta) = \frac{\alpha}{\alpha + \beta}$$

given binary variable X, and a dataset with n samples

$$\mathbf{P}(X = 1) = \frac{\alpha + \sum_{i'=1}^{n} I_1(x_{i'})}{\alpha + \beta + n}$$

- $\mathbf{P}(X = 0) = 1 - \mathbf{P}(X = 1)$
- (α, β): shape parameters
  - $\frac{\alpha}{\alpha+\beta}$: expert's prior for $X = 1$
  - $\alpha + \beta$: confidence about the prior

Smoothing

example

- prior: $\mathbf{E}_{\mathrm{Beta}(\alpha,\beta)}[X] = 0.7$
- dataset: 40 positive cases among 100 samples

parameter estimation for different confidences:

- low confidence ($\alpha + \beta = 10$): $\mathbf{P}(X = 1) = \frac{7+40}{10+100} = 0.43$
- medium confidence ($\alpha + \beta = 100$): $\mathbf{P}(X = 1) = \frac{70+40}{100+100} = 0.55$
- high confidence ($\alpha + \beta = 1000$): $\mathbf{P}(X = 1) = \frac{700+40}{1000+100} = 0.67$

Smoothing

Dirichlet prior: extending the Beta prior to m-valued random variables

for m-dimensional random vector $X \sim \mathrm{Dir}(\alpha)$:

$$\mathbf{E}_{\mathrm{Dir}(\alpha)}[X_i] = \mathbf{P}(x_i \mid \alpha) = \frac{\alpha_i}{\alpha^T \mathbf{1}}, \quad i = 1, \ldots, m$$

given m-valued variable X, and a dataset with n samples

$$\mathbf{P}(x_i) = \frac{\alpha_i + \sum_{i'=1}^{n} I_{x_i}(x_{i'})}{\alpha^T \mathbf{1} + n}, \quad i = 1, \ldots, m$$

- $\alpha \in \mathbf{R}^m$: shape parameters
  - $\frac{\alpha_i}{\alpha^T \mathbf{1}}$: expert's prior for $X = x_i$
  - $\alpha^T \mathbf{1}$: confidence about the prior

Missing data

missing values for one or more variables in some samples:

- remove all the samples with missing values
  - acceptable only if there is sufficient data
- substitute the missing value by the most common value of that variable
  - may bias the model since the information from the other variables is not taken into account
- estimate the missing value based on the other variables in the corresponding sample:

---

1. Learn Bayesian network network parameters based on the samples with complete observations.
2. For each sample with missing values:
   2.1 Instantiate all the known variables in the sample.
   2.2 Through probabilistic inference obtain the posterior probabilities of the missing variables.
   2.3 Assign to each unknown variable the value with highest posterior probability, or sample one value according to the posterior probability.
   2.4 Add this completed sample to the database.
3. Re-estimate the model parameters based on the completed dataset.

Missing data

hidden nodes: a variable or set of variables in the model cannot be observed

- expectation-maximization (EM) algorithm

---

1. Initializing the missing parameters with random values.
2. E-step: the missing data values are estimated based on the current parameters.
3. M-step: the parameters are updated based on the estimated data.
4. Repeat the last two steps until convergence.

Discretization

unsupervised discretization

- **equal width**:
  - dividing the range of a variable into k equal bins
  - each bin has a size of $\frac{\sup(X)-\inf(X)}{k}$
- **equal data**:
  - dividing $(\sup(X), \inf(X))$ into k intervals with each having the same number of data points
  - the intervals not necessarily have the same width

supervised discretization

- variables are discretized to optimize this task
- determine the optimal partition of $(\inf(X), \sup(X))$ w.r.t. some score function (accuracy, likelihood, etc.)
- solve a combinatorial optimization problem using **hill-climbing**, **simulated annealing**, **genetic algorithms**, etc.

Outline

- Representation
- Inference
- Parameter learning
- Structure learning

Tree learning

- dependencies between random variables can be represented with a tree-structure
- procedure:
  - establishing undirected edges between variables (tree skeleton learning)
  - determining the direction of the edges

Tree learning

skeleton learning: Chow-Liu procedure (CLP)

given a set of random variables $X = \{X_1, \ldots, X_n\}$

$$D_{\mathrm{KL}}(\mathbf{P}, \widetilde{\mathbf{P}}) = \sum_{x \in X} \mathbf{P}(x) \log \left(\frac{\mathbf{P}(x)}{\widetilde{\mathbf{P}}(x)} \right)$$

- approximation error of the joint distribution of X by a tree-structure
- D_{KL}: **KL-divergence** measure
- $\mathbf{P}(x)$: true distribution
- $\widetilde{\mathbf{P}}(x)$: distribution obtained from some tree including variables X

- evaluating the KL-divergence for all possible trees is very expensive

Tree learning

- **mutual information** between any pair of variables $X_i, X_j \in X$:

$$I(X_i, X_j) = \sum_{x_i \in X_i, x_j \in X_j} \mathbf{P}(x_i, x_j) \log \left(\frac{\mathbf{P}(x_i, x_j)}{\mathbf{P}(x_i)\mathbf{P}(x_j)} \right)$$

- given tree $G = (X, E)$, the sum of the mutual information of the edges:

$$W(X) = \sum_{(X_i, X_j) \in E} I(X_i, X_j) = \sum_{i=1}^{n-1} I(X_i, \mathbf{pa}(X_i))$$

  - minimizing $D_{\mathrm{KL}}(\mathbf{P}, \widetilde{\mathbf{P}})$ is equivalent to maximizing $W(X)$ over E

Tree learning

Chow-Liu procedure (CLP)

1. Obtain the mutual information $I(X_i, X_j)$ for all pairs of variables $X_i \in X$, $X_j \in X$.
2. Order the mutual information values in descending order.
3. Select the pair (X_i, X_j) with maximum $I(X_i, X_j)$ and connect the two variables with an edge, this constitutes the initial tree.
4. Add the pair with the next highest mutual information to the tree if they do not make a cycle, otherwise skip it and continue with the following pair.
5. Repeat the previous step until all the variables are in the tree.

Tree learning

direction learning

- based on independence tests on variable triplets
- given three variables X, Y, and Z, there are three possibilities for their dependency:
  - sequential: $X \to Y \to Z$
  - divergent: $X \leftarrow Y \to Z$
  - convergent: $X \to Y \leftarrow Z$
- $(X \perp\!\!\!\perp Z \mid Y) \implies X \to Y \to Z$ or $X \leftarrow Y \to Z$ (indistinguishable)
- $(X \not\perp\!\!\!\perp Z \mid Y) \implies X \to Y \leftarrow Z$ (used for edge direction assignment)

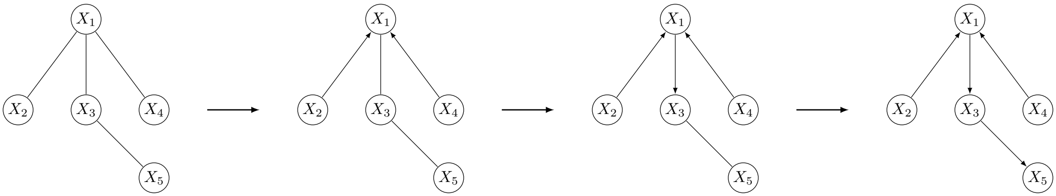
Tree learning

1. Iterate over the network until a convergent variable triplet is found. We will call the variable to which the arcs converge a multi-parent node.
2. Starting with a multi-parent node, determine the directions of other arcs using independence tests for variable triplets. Continue this procedure until it is no longer possible.
3. Repeat the first two steps until no other directions can be determined.

- no guarantee that the direction for all the arcs in the tree can be obtained
- external semantics can be used to infer the directions of the left undirected edges

Tree learning

example



- test for $\{X_1, X_2, X_4\}$: $(X_2 \not\perp\!\!\!\perp X_4 \mid X_1) \implies X_2 \to X_1 \leftarrow X_4$
- test for $\{X_1, X_2, X_3\}$ and $\{X_1, X_3, X_4\}$:
  - $(X_2 \perp\!\!\!\perp X_3 \mid X_1)$, $(X_3 \perp\!\!\!\perp X_4 \mid X_1) \implies X_1 \to X_3$
  - otherwise, $X_1 \leftarrow X_3$
- test for $\{X_1, X_3, X_5\}$, the same as above

Score-based methods

idea

- structure learning as combinatorial optimization w.r.t. some score function S
- generally NP-hard
- heuristic methods: **hill-climbing**, **simulated annealing**, **genetic algorithms**

likelihood score

given observed dataset $\mathcal{D}$, graph G and its parameters θ_G:

$$S_{\mathrm{LL}}(G) = l_{\mathcal{D}}(\theta_G) = \log \mathbf{P}(\mathcal{D} \mid \theta_G, G)$$

- $l_{\mathcal{D}}(\theta_G)$: log-likelihood of dataset $\mathcal{D}$ parameterized by θ_G
- find network structure by **maximum likelihood estimation (MLE)**
  - may result in overfitting

Score-based methods

obtain the posterior probability of the structure given the data with the Bayes rule

$$\mathbf{P}(G \mid \mathcal{D}) = \frac{\mathbf{P}(\mathcal{D} \mid G)\mathbf{P}(G)}{\mathbf{P}(\mathcal{D})}$$

- $\mathbf{P}(\mathcal{D})$: normalization factor

Bayesian score

$$S_{\mathrm{B}}(G) = \log \mathbf{P}(\mathcal{D} \mid G) + \log \mathbf{P}(G)$$

- $\mathbf{P}(G)$: prior over network structures

Score-based methods

- $\mathbf{P}(\mathcal{D} \mid G)$: **marginal likelihood** of the data

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

  - $\mathbf{P}(\mathcal{D} \mid \theta_G, G)$: likelihood of the data given the network G and its parameter θ_G
  - $\mathbf{P}(\theta_G \mid G)$: prior distribution over different parameter values for the network G

- find network structure by marginal likelihood maximization
  - measuring the expected likelihood, averaged over different possible choices of θ_G, instead of the maximum (most optimistic) likelihood
  - more conservative in the estimation of the goodness of the mode, avoid overfitting

Score-based methods

example: **Bayesian information criterion (BIC)**

- Dirichlet parameter prior for all parameters in the network
- number of samples $n \to \infty$

$$
\begin{aligned}
S_{\mathrm{BIC}}(G) &= l_{\mathcal{D}}(\theta_G) - \frac{k}{2} \log n \\
&= \log \mathbf{P}(\mathcal{D} \mid \theta_G, G) - \frac{k}{2} \log n
\end{aligned}
$$

- k: number of parameters in the network
- n: number of samples in the dataset
- trade off fit to data with model complexity

PC algorithm

idea

- first recovers the skeleton of the network, then determines the direction of the edges
- both steps are based on independence tests

---

1. Establish a fully connected undirected graph between all variables.
2. For each pair of variables, determines their conditional independence given some subset of the other variables. Eliminate the edge between this pair of variables if the independence measure is below some threshold value.
3. Determine the direction of the network skeleton based on independence tests for variable triplets.
