Learning the structure of Bayesian networks

Jüri Reimand
Bayesian Networks seminar
08042009
Search procedures
Search procedures

• Find the highest-scoring network structure
  • among all structures?! super-exponential search space
Search procedures

• Find the highest-scoring network structure
  • among all structures?! super-exponential search space

• Heuristics to explore the search space
  • small iterative changes in a given structure (DAG)
Search procedures

- Find the highest-scoring network structure
  - among all structures?! super-exponential search space
- Heuristics to explore the search space
  - small iterative changes in a given structure (DAG)
- Use search operators
Search procedures

• Find the highest-scoring network structure
  • among all structures?! super-exponential search space

• Heuristics to explore the search space
  • small iterative changes in a given structure (DAG)

• Use search operators
Search procedures

- Find the highest-scoring network structure
  - among all structures?! super-exponential search space
- Heuristics to explore the search space
  - small iterative changes in a given structure (DAG)
- Use search operators
  - Add an edge
  - Delete an edge
  - Reverse an edge
decomposability

$$\text{score}(\mathcal{D}, S) = \sum_{i=1}^{n} \text{score}(X_i, \text{pa}(X_i), \mathcal{D})$$
decomposability

\[ \text{score}(\mathcal{D}, S) = \sum_{i=1}^{n} \text{score}(X_i, \text{pa}(X_i), \mathcal{D}) \]

- Global score is a sum of local scores
- Search operators cause local changes
decomposability

\[ \text{score}(\mathcal{D}, S') = \sum_{i=1}^{n} \text{score}(X_i, \text{pa}(X_i), \mathcal{D}) \]

- Global score is a sum of local scores
- Search operators cause local changes
decomposability

\[
score(\mathcal{D}, S') = \sum_{i=1}^{n} score(X_i, pa(X_i), \mathcal{D})
\]

- Global score is a sum of local scores
- Search operators cause local changes
decomposability

\[ \text{score}(D, S') = \sum_{i=1}^{n} \text{score}(X_i, \text{pa}(X_i), D) \]

- Global score is a sum of local scores
- Search operators cause local changes
decomposability

\[ \text{score}(\mathcal{D}, S) = \sum_{i=1}^{n} \text{score}(X_i, \text{pa}(X_i), \mathcal{D}) \]

- Global score is a sum of local scores
- Search operators cause local changes
decomposability

$$\text{score}(D, S) = \sum_{i=1}^{n} \text{score}(X_i, \text{pa}(X_i), D)$$

- Global score is a sum of local scores
- Search operators cause local changes

- Alter global score through local changes
decomposability

\[ \text{score}(D, S) = \sum_{i=1}^{n} \text{score}(X_i, \text{pa}(X_i), D) \]

- Global score is a sum of local scores
- Search operators cause local changes

- Alter global score through local changes

\[ \Delta(X_i \rightarrow X_j) = \text{score}(X_j, \text{pa}(X_j) \cup \{X_i\}, D) - \text{score}(X_j, \text{pa}(X_j), D) \]
Greedy search

1. Let $S$ be an initial structure.
2. Repeat
   a) Calculate $\Delta(A)$ for each legal arc operation $A$
      - Let $\Delta^* = \max_A \Delta(A)$ and $A^* = \arg \max_A \Delta(A)$.
   b) If $\Delta^* > 0$, then
      - Set $S = op(S, A^*)$.
3. Until $\Delta^* \leq 0$. 
Greedy search

1. Let $S$ be an initial structure.
2. Repeat
   a) Calculate $\Delta(A)$ for each legal arc operation $A$
      - Let $\Delta^* = \max_A \Delta(A)$ and $A^* = \arg \max_A \Delta(A)$.
   b) If $\Delta^* > 0$, then
      - Set $S = \text{op}(S, A^*)$.
3. Until $\Delta^* \leq 0$.  

We need to preserve acyclic graph structure.
Greedy search

1. Let $S$ be an initial structure.
2. Repeat
   a) Calculate $\Delta(A)$ for each legal arc operation $A$
      - Let $\Delta^* = \max_A \Delta(A)$ and $A^* = \arg \max_A \Delta(A)$.
   b) If $\Delta^* > 0$, then
      - Set $S = \text{op}(S, A^*)$.
3. Until $\Delta^* \leq 0$.

• Global optimal structure is not guaranteed!
  • may get stuck in local maxima
  • multiple random restarts, choose best-scoring structure
Prior information 1

- Use causal rules to group nodes
  - constrain structure search space
  - disallow edges that violate causal hierarchy
Prior information I

- Use causal rules to group nodes
  - constrain structure search space
  - disallow edges that violate causal hierarchy
Prior information I

• Use causal rules to group nodes
  • constrain structure search space
  • disallow edges that violate causal hierarchy
Prior information II
Prior information II

• Introduce a partial node ordering $\leq$
  
  • only allow $x_i \rightarrow x_j$ if $x_i \leq x_j$
Prior information II

• Introduce a partial node ordering $\leq$
  • only allow $x_i \rightarrow x_j$ if $x_i \leq x_j$

• Special case: full linear ordering [K2]
  • $x_1 \leq \ldots \leq x_i \leq x_{i+1} \leq \ldots \leq x_n$
  • $x_i$ can have up to $i-1$ parents and $2^{i-1}$ parent sets
  • subtract the ‘super’ from superexponential structure space
Prior information II

- Introduce a partial node ordering $\leq$
  - only allow $x_i \rightarrow x_j$ if $x_i \leq x_j$

- Special case: full linear ordering [K2]
  - $x_1 \leq \ldots \leq x_i \leq x_{i+1} \leq \ldots \leq x_n$
  - $x_i$ can have up to $i-1$ parents and $2^{i-1}$ parent sets
  - subtract the ‘super’ from superexponential structure space

$$\prod_{i=1}^{n} 2^{n-1} = 2^{\sum_{i=1}^{n-1} i} = 2^{n(n-1)/2}$$
Equivalence class search

• data alone cannot discriminate structures with equivalent d-separation properties
  • therefore, many of the network structures are score-equivalent

• greedy equivalence search
  • explore equivalence classes, rather than all DAGs
  • apply complex search operators, defined as dependence statements
Equivalence class search
Equivalence class search
Equivalence class search
Equivalence class search
Equivalence class search

- Start from ♠:
  find a local maximum, while moving upwards (e.g. ♦)

- Start from ♦:
  find a local maximum, while moving downwards (e.g. ✓)
Equivalence class search

- start from ♠:
  find a local maximum, while moving upwards (e.g. ♣)

- start from ♣:
  find a local maximum, while moving downwards (e.g. ✔)

If DB is “large enough”, resulting equivalence class is guaranteed to include the generating Bayesian network.
Equivalence class search

- start from ♠: find a local maximum, while moving upwards (e.g. ♠)
- start from ♣: find a local maximum, while moving downwards (e.g. ✓)

If DB is “large enough”, resulting equivalence class is guaranteed to include the generating Bayesian network

Unfortunately, equivalence classes are still superexponential to node count
Chow-Liu trees I

- Max one parent per node
  - constrain complexity
  - efficient max.likelihood computation
Chow-Liu trees I

• Max one parent per node
  • constrain complexity
  • efficient max.likelihood computation

1. Calculate the mutual information MI($X_i, X_j$) for each pair ($X_i, X_j$).
2. Consider the complete MI-weighted graph: the complete undirected graph over \{X, ..., X_n\}, where the links ($X_i, X_j$) have the weight MI($X_i, X_j$).
3. Build a maximal-weight spanning tree for the complete MI-weighted graph.
4. Direct the resulting tree by choosing any variable as a root and setting the directions of the links to be outward from it.
5. Learn the parameters.
Chow-Liu trees I

- Max one parent per node
  - constrain complexity
  - efficient max.likelihood computation

1. Calculate the **mutual information** $\text{MI}(X_i, X_j)$ for each pair $(X_i, X_j)$.
2. Consider the **complete MI-weighted graph**: the complete undirected graph over $\{X, \ldots, X_n\}$, where the links $(X_i, X_j)$ have the weight $\text{MI}(X_i, X_j)$.
3. Build a **maximal-weight spanning tree** for the complete MI-weighted graph.
4. Direct the resulting tree by choosing any variable as a root and setting the directions of the links to be outward from it.
5. Learn the parameters.
Chow-Liu trees II

• Mutual information
Chow-Liu trees II

- Mutual information

\[ MI(X, Y) = \sum_{X,Y} P(X, Y) \log_2 \left( \frac{P(X, Y)}{P(X)P(Y)} \right) \]
Chow-Liu trees II

- Mutual information

\[ \text{MI}(X, Y) = \sum_{X,Y} P(X, Y) \log_2 \left( \frac{P(X, Y)}{P(X)P(Y)} \right) \]

- Max-weight spanning tree
  - covers all nodes
  - includes no cycles
  - iteratively includes heaviest edges
Chow-Liu trees II

- Mutual information

\[ \text{MI}(X, Y) = \sum_{X,Y} P(X,Y) \log_2 \left( \frac{P(X,Y)}{P(X)P(Y)} \right) \]

- Max-weight spanning tree
  - covers all nodes
  - includes no cycles
  - iteratively includes heaviest edges
Prior distribution over structures
Prior distribution over structures

\[ P(S | \mathcal{D}) = \frac{P(\mathcal{D}, S)}{P(\mathcal{D})} = \frac{P(S)P(\mathcal{D}|S)}{P(\mathcal{D})} = \mu P(S)P(\mathcal{D}|S). \]
Prior distribution over structures

\[ P(S \mid D) = \frac{P(D, S)}{P(D)} = \frac{P(S)P(D \mid S)}{P(D)} = \mu P(S)P(D \mid S); \]
Prior distribution over structures

\[ P(S | D) = \frac{P(D, S)}{P(D)} = \frac{P(S)P(D | S)}{P(D)} = \mu P(S)P(D | S). \]

data, given the structure e.g. \textit{likelihood}

prior knowledge about the \textit{structure}
Prior distribution over structures

\[ P(S | \mathcal{D}) = \frac{P(\mathcal{D}, S)}{P(\mathcal{D})} = \frac{P(S)P(\mathcal{D}|S)}{P(\mathcal{D})} = \mu P(S)P(\mathcal{D}|S). \]

\[ P(S) = c \cdot \prod_{i=1}^{n} \rho(X_i, \text{pa}(X_i)), \]

data, given the structure e.g. likelihood

prior knowledge about the structure
Prior distribution over structures

\[ P(S \mid \mathcal{D}) = \frac{P(\mathcal{D}, S)}{P(\mathcal{D})} = \frac{P(S)P(\mathcal{D} \mid S)}{P(\mathcal{D})} = \mu P(S)P(\mathcal{D} \mid S). \]

\[ P(S) = c \cdot \prod_{i=1}^{n} \rho(X_i, \text{pa}(X_i)), \]

decomposable

sum/product over parent-child families

data, given the structure e.g. likelihood

prior knowledge about the structure
Prior distribution over structures

\[
P(S|D) = \frac{P(D, S)}{P(D)} = \frac{P(S)P(D|S)}{P(D)} = \mu P(S)P(D|S).
\]

\[
P(S) = c \cdot \prod_{i=1}^{n} \rho(X_i, \text{pa}(X_i)), \quad \rho(X_i, \text{pa}(X_i)) = 1
\]

A decomposable sum/product over parent-child families

data, given the structure e.g. \textit{likelihood}

prior knowledge about the \textbf{structure}
Prior distribution over structures

\[ P(S | D) = \frac{P(D, S)}{P(D)} = \frac{P(S)P(D|S)}{P(D)} = \mu P(S)P(D|S). \]

\[ P(S) = c \cdot \prod_{i=1}^{n} \rho(X_i, \text{pa}(X_i)), \quad \rho(X_i, \text{pa}(X_i)) = 1 \]

A decomposable sum/product over parent-child families that makes all families equally likely...
Prior distribution over structures

\[ P(S | D) = \frac{P(D, S)}{P(D)} = \frac{P(S)P(D|S)}{P(D)} = \mu P(S)P(D|S). \]

\[ P(S) = c \cdot \prod_{i=1}^{n} \rho(X_i, \text{pa}(X_i)), \quad \rho(X_i, \text{pa}(X_i)) = 1 \]

A decomposable sum/product over parent-child families that makes all families equally likely...

\[ \rho(X_i, \text{pa}(X_i)) = \kappa \sum_{i=1}^{n} \delta_i \]

\[ \delta_i = |(\text{pa}(X_i)_S \cup \text{pa}(X_i)_{BP}) \setminus (\text{pa}(X_i)_S \cap \text{pa}(X_i)_{BP})| \]
Prior distribution over structures

\[ P(S \mid D) = \frac{P(D, S)}{P(D)} = \frac{P(S)P(D \mid S)}{P(D)} = \mu P(S)P(D \mid S). \]

\[ P(S) = c \cdot \prod_{i=1}^{n} \rho(X_i, \text{pa}(X_i)), \quad \rho(X_i, \text{pa}(X_i)) = 1 \]

a decomposable sum/product over parent-child families

that makes all families equally likely...

.. or assigns exponentially less preference to families further away from a prior structure B

\[ \rho(X_i, \text{pa}(X_i)) = \kappa \sum_{i=1}^{n} \delta_i \]

\[ \delta_i = |(\text{pa}(X_i)_S \cup \text{pa}(X_i)_{BP}) \setminus (\text{pa}(X_i)_S \cap \text{pa}(X_i)_{BP})| \]