Introduction to information theory and coding

Louis WEHENKEL

Set of slides No 3

- Probabilistic *deductive* inference in graphical models
  - Important classes of DAGs
  - Hidden Markov chains and propagation algorithms
  - Comments on generalized versions of these algorithms
- Automatic learning of graphical models
  - Classes of learning problems for BNs
  - Top down induction of decision trees

Probabilistic *deductive* **inference in Bayesian networks**

Approaches

- Brute force approach (table lookup plus marginalization)
  $\Rightarrow$ intractable for large networks...

- Network reduction (transfiguration)
  To determine impact of observation $\mathcal{X}_i = X_i^j$ on distribution of $\mathcal{X}$ remove all other variables from graph
  Removal of a variable creates new links among its neighbors

  $\Rightarrow$ analogy with circuit theory (e.g. optimal ordering of node elimination)

- Monte-Carlo simulation techniques

- Local propagation of information (using the network structure)
  $\Rightarrow$ generalized forward-backward procedure

  Works only for singly-connected graphs (polytrees)
  $\Rightarrow$ need to transform general DAG into polytree

Bayesian network structures (DAGs)

- **Unconnected graphs**
  Decompose overall problem into independent subproblems

- **Markov chains**
  Every variable has at most one father and one son.

- **Hidden Markov chain**
  Like a Markov chain where each node has one additional son

- **Trees**
  All nodes except one (called root) has exactly one father.
  NB. Markov (hidden or not) chains are a subclass of trees

- **Polytrees**
  DAG such that the corresponding undirected graph has no cycles (i.e. is singly connected)

  This is the most general type of DAG on which local propagation works

  NB. Trees are also polytrees.

Building of Bayesian belief networks

Algorithm

Choose variables, and choose their ordering $\mathcal{X}_i$.

For $i = 1, 2, \ldots$

- create node $\mathcal{X}_i$;
- select a minimal subset of $\{\mathcal{X}_1, \ldots, \mathcal{X}_{i-1}\}$ to become $\mathcal{P}(\mathcal{X}_i)$.
- Create arcs from $\mathcal{P}(\mathcal{X}_i) \rightarrow \mathcal{X}_i$.
- Create table $P(\mathcal{X}_i|\mathcal{P}(\mathcal{X}_i))$.

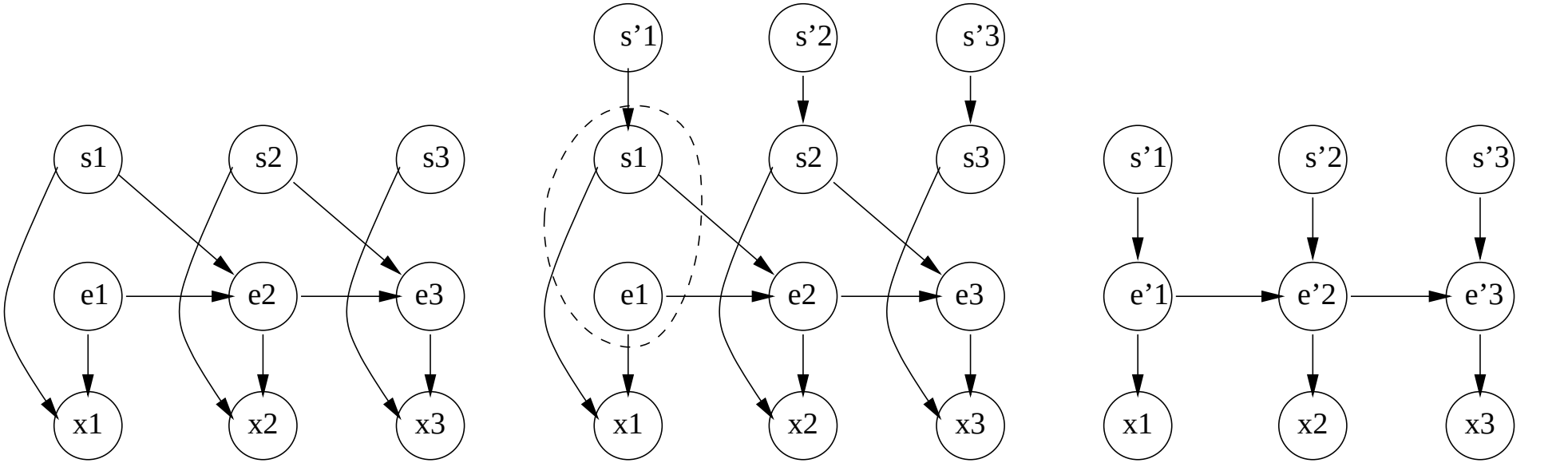
Comments

Complexity of resulting structure is highly dependent on node ordering.

In practice, use physical insight to choose appropriate order.

Duplication and merging of nodes in a DAG

Can be used to transform graph into polytree :

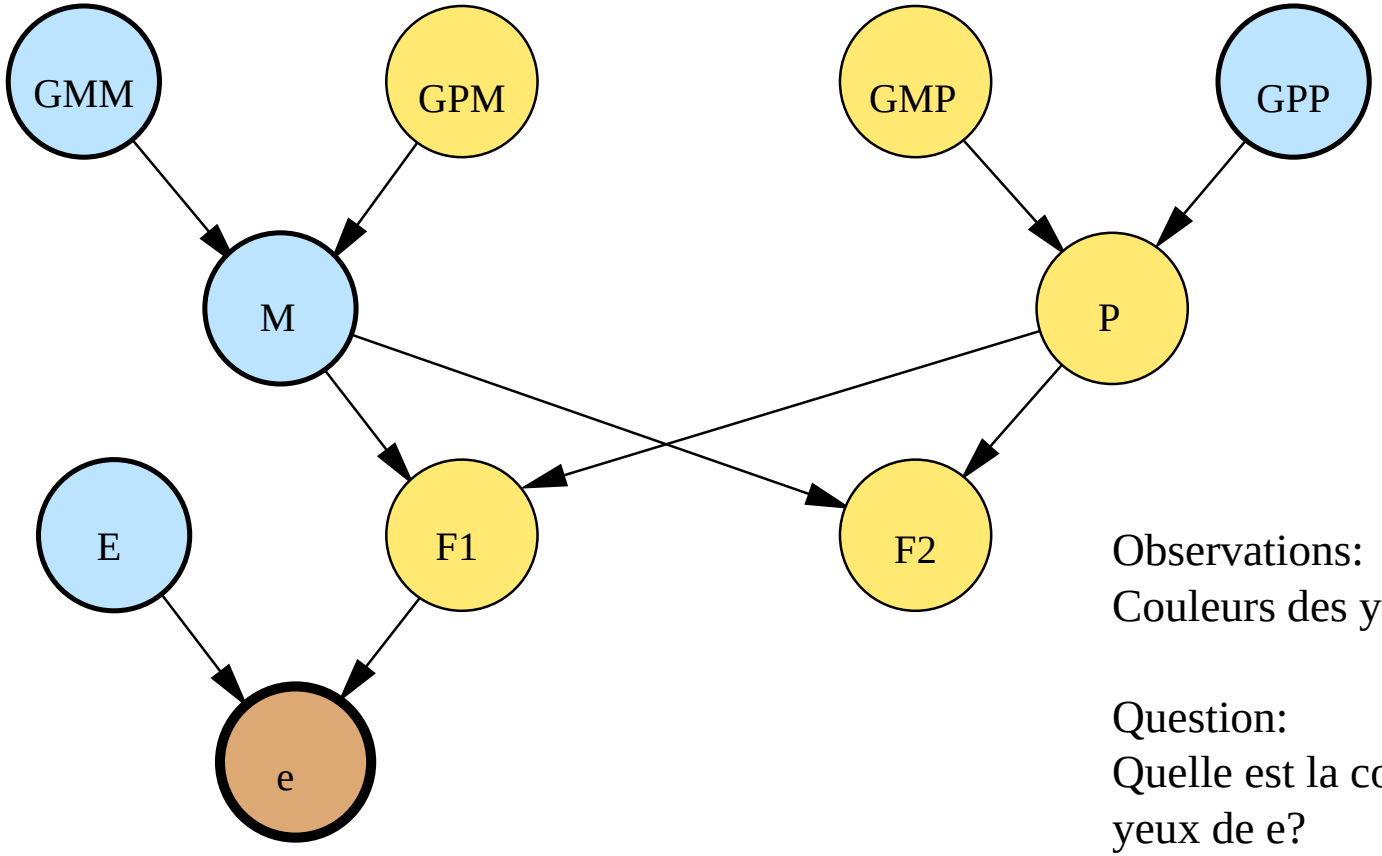


(a) General DAG (b) duplication of inputs (c) Polytree obtained after merging

General algorithm : based on *junction trees*...

Inference

For example :

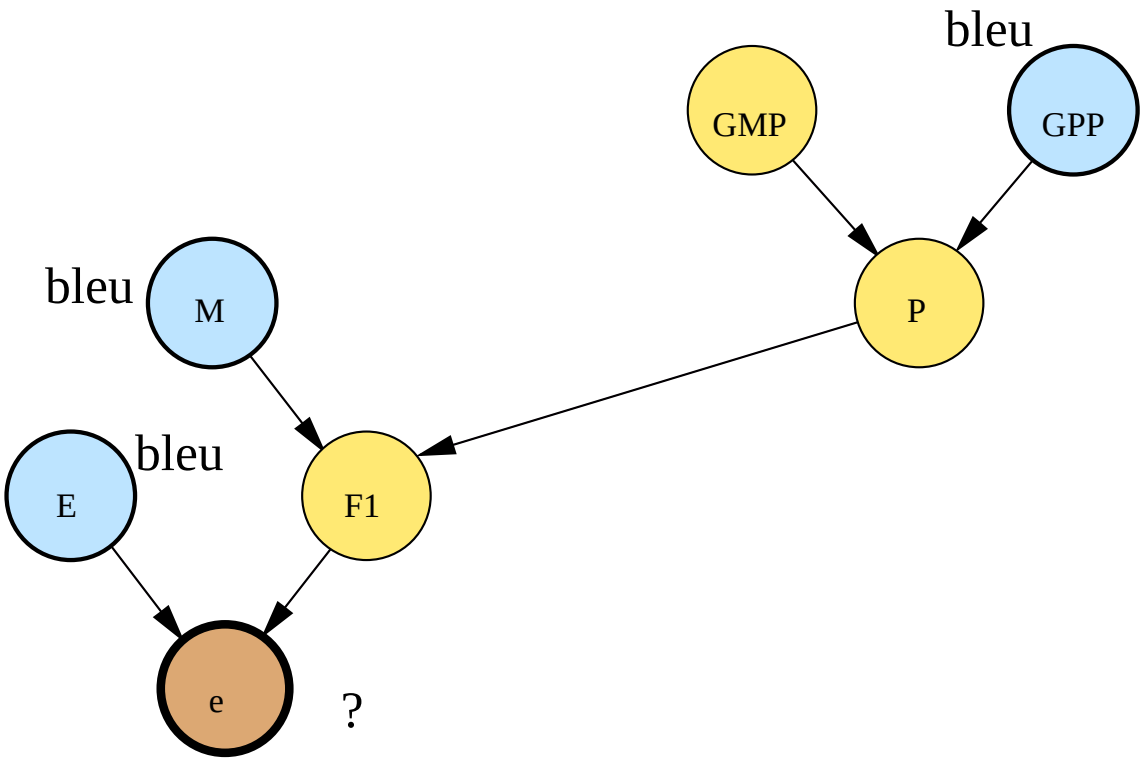


Observations:
Couleurs des yeux de GPP, GMM, M et E

Question:
Quelle est la couleur la plus probable des yeux de e?

Typical inference problem.
Given some observations O (the values for a subset of variables) determine the possible values of all other variables and their probabilities $P(\mathcal{X}_i|O)$.

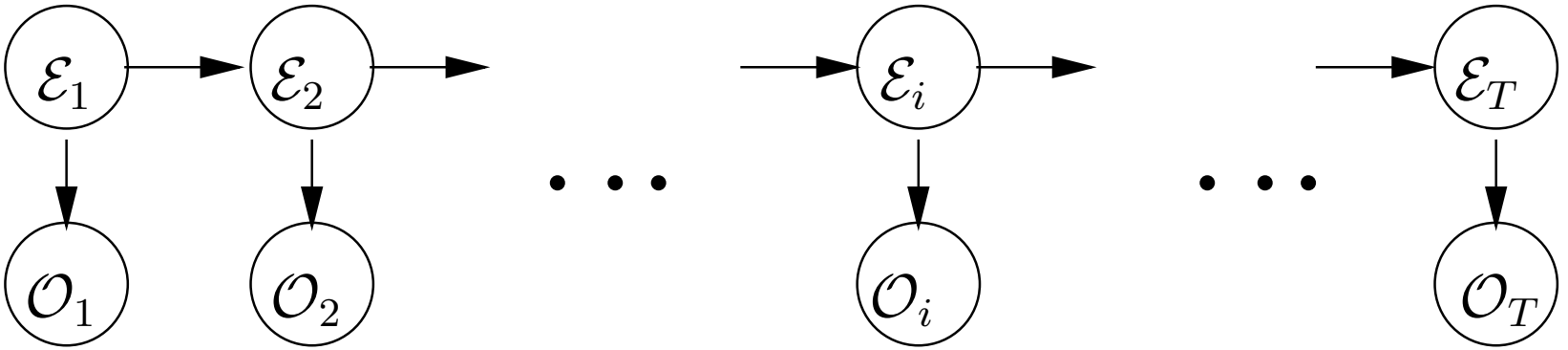
Remark.
Not all variables are used in all inference problems.
D-separation may be exploited to remove nodes from network.
$\Rightarrow$ often possible to simplify network structure before solving inference problem.



Observations:
Couleurs des yeux de GPP, M et E

Question:
Quelle est la couleur la plus probable des yeux de e?

Hidden Markov chains



- set of indices $i = 1, 2, \ldots, T$ (e.g. to represent time)
- successive states of the chain $\mathcal{E}_i$: form a markov chain
- the observations $\mathcal{O}_i$: $P(\mathcal{O}_i|\mathcal{E}_i, \mathcal{W}) = P(\mathcal{O}_i|\mathcal{E}_i)$ where $\mathcal{W}$ denotes any subset of states and observations.

$\Rightarrow$ Model instance characterized by $P(\mathcal{E}_1)$, $P(\mathcal{E}_{i+1}|\mathcal{E}_i)$, and $P(\mathcal{O}_i|\mathcal{E}_i)$,

Time invariance : $P(\mathcal{E}_{i+1}|\mathcal{E}_i)$ and $P(\mathcal{O}_i|\mathcal{E}_i)$ are independent of time (i.e. of i).

$\Rightarrow$ fundamental assumption in coding theory, and here it makes life easier...

Notation : $\pi_i \stackrel{\triangle}{=} P(\mathcal{E}_1 = i), \Pi_{i,j} \stackrel{\triangle}{=} P(\mathcal{E}_{k+1} = j|\mathcal{E}_k = i), \Sigma_{i,j} \stackrel{\triangle}{=} P(\mathcal{O}_k = j|\mathcal{E}_k = i)$.

Examples of applications of Hidden Markov chains

A highly structured generalization of Markov chains, when states are not observed directly $\Rightarrow$ large class of stochastic processes can be modelled in this way.

- *Artificial intelligence :* handwritten text and spoken text recognition.
- *Information and coding theories :* memoryless channel fed with a Markov chain, channel coding and decoding processes
- *Bioinformatics :* models for complex molecules (proteins, DNA, . . .)
- *Computer Science :* stochastic automata

Further generalizations

Markov random fields and hidden markov fields (e.g. image restoration).

Inference with hidden Markov chains : problem No 1

Let us consider a time invariant HMC characterized by

$$\pi_i \stackrel{\triangle}{=} P(\mathcal{E}_1 = i), \quad \Pi_{i,j} \stackrel{\triangle}{=} P(\mathcal{E}_{k+1} = j | \mathcal{E}_k = i), \text{ et } \quad \Sigma_{i,j} \stackrel{\triangle}{=} P(\mathcal{O}_k = j | \mathcal{E}_k = i).$$

Possible values of states $1, 2, \ldots, N$, possible values of observations $1, 2, \ldots, M$.

Problem formulation

Given a sequence of observations $O^T = O_1, O_2, \ldots, O_T$ (a realization of the r.v. $\mathcal{O}^T = \mathcal{O}_1, \ldots, \mathcal{O}_T$), and the model specification π, Π, Σ :

Determine (compute) the probability $P(O^T)$

Applications

Pattern recognition : given a set of candidate HMCs and an observation sequence, which one among the candidate models is the most likely *explanation*.

Inference with hidden Markov chains : problem No 2

Problem formulation

Given an observation sequence O^T and the model specification π, Π, Σ :

Determine the most likely state sequence

Applications

Decoding : looking at the observations as noisy versions of the states, we try to recover the original state sequence.

Variants

Determine the single most likely state sequence

Determine (or approximate) the conditional probability distribution $P(\mathcal{E}^T|O^T)$.

Algorithms

BCJR and Viterbi are two efficient algorithms

Forward algorithm : efficient solution to problem No 1

NB : see notes for full details and comments.

Idea : find recursive algorithm to compute $P(O^T, \mathcal{E}_T)$ and marginalize over $\mathcal{E}_T$

Let us denote by $\alpha_t(i) = P(O_1 O_2 \cdots O_t, \mathcal{E}_t = i)$ (for each $t : N$ numbers in $[0; 1]$)

Then we have :

- Initialization : $\alpha_1(i) = P(O_1, \mathcal{E}_1 = i) = P(\mathcal{E}_1 = i) P(O_1 | \mathcal{E}_1 = i) = \pi_i \Sigma_{i,O_1}$.
- Induction : $\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) \Pi_{i,j}\right] \Sigma_{j,O_{t+1}}$ (Proof follows on next slide).
- Termination : $P(O^T) = \sum_{i=1} \alpha_T(i)$

Comment

Number of computations linear in $T \times N^2$

Trivial method (marginalization w.r.t. $\mathcal{E}^T$) : $2T \times N^T$

Derivation of the induction step :

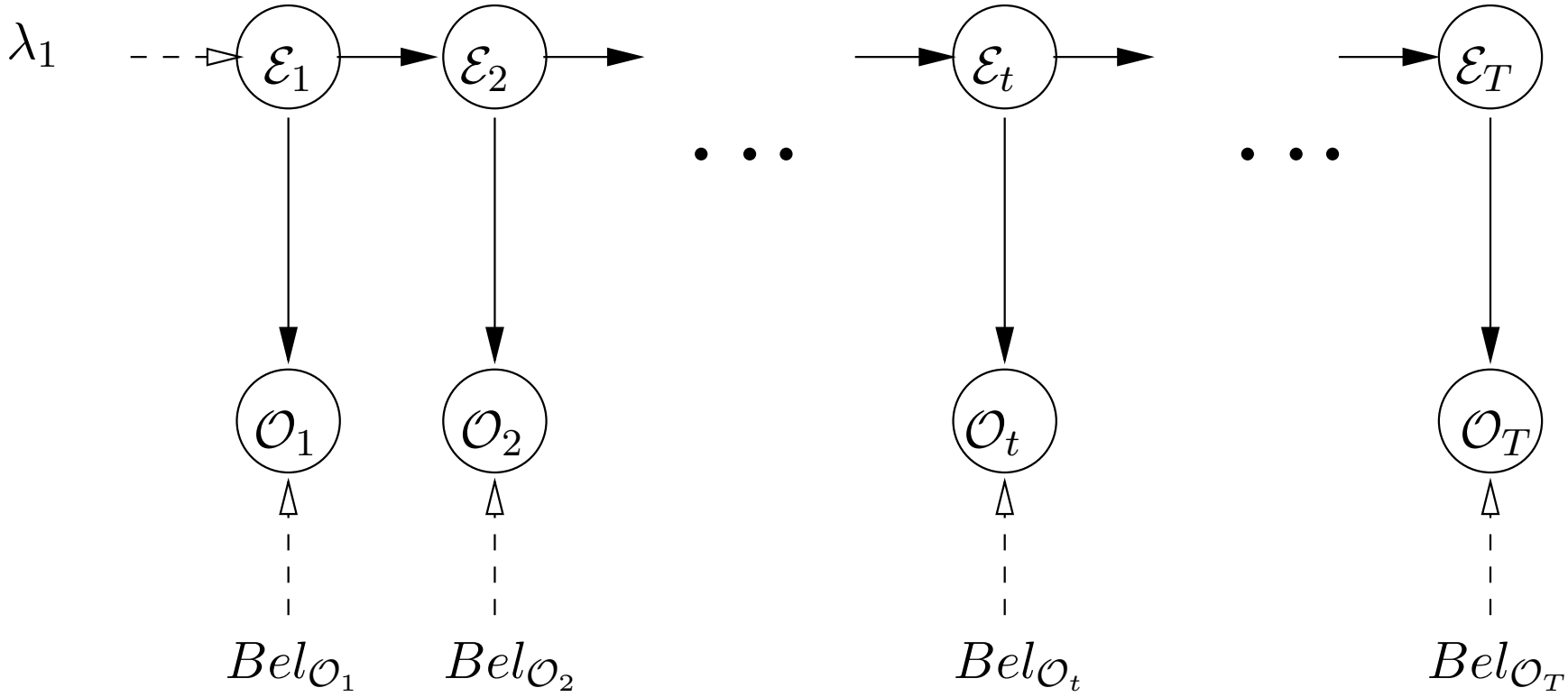
Exploiting conditional independence one time...

$$\begin{aligned}
\alpha_{t+1}(j) &= P(O_1 O_2 \cdots O_t, \mathcal{E}_{t+1} = j, O_{t+1}) \\
&= P(O_1 O_2 \cdots O_t, \mathcal{E}_{t+1} = j) P(O_{t+1} | O_1 O_2 \cdots O_t, \mathcal{E}_{t+1} = j) \\
&= P(O_1 O_2 \cdots O_t, \mathcal{E}_{t+1} = j) P(O_{t+1} | \mathcal{E}_{t+1} = j) \\
&= P(O_1 O_2 \cdots O_t, \mathcal{E}_{t+1} = j) \Sigma_{j, O_{t+1}},
\end{aligned}$$

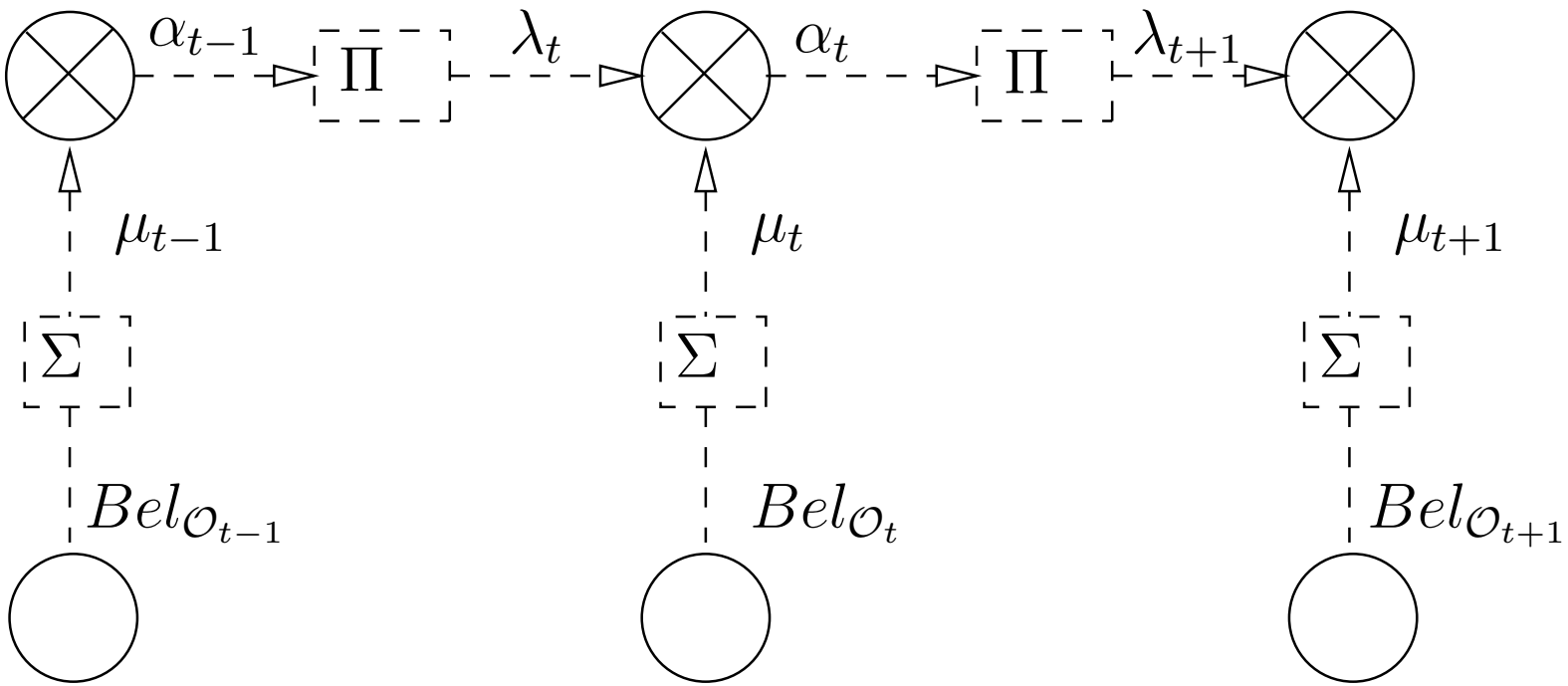
and a second time

$$\begin{aligned}
P(O^t, \mathcal{E}_{t+1} = j) &= \sum_{i=1}^{N} P(O_1 \cdots O_t, \mathcal{E}_t = i, \mathcal{E}_{t+1} = j) \\
&= \sum_{i=1}^{N} P(O_1 \cdots O_t, \mathcal{E}_t = i) P(\mathcal{E}_{t+1} = j | O_1 \cdots O_t, \mathcal{E}_t = i) \\
&= \sum_{i=1}^{N} P(O_1 \cdots O_t, \mathcal{E}_t = i) P(\mathcal{E}_{t+1} = j | \mathcal{E}_t = i) \\
&= \sum_{i=1}^{N} \alpha_t(i) \Pi_{i,j},
\end{aligned}$$

Initialization



Local propagation



Forward-backward algorithm (BCJR) : efficient computation of $P(\mathcal{E}_t|O^T)$ ($\forall t$).

Remark : $P(\mathcal{E}_t|O^T)$ allows one to guess also a likely value of $E_t \Rightarrow$ BCJR algorithm can be used to minimize BER in channel decoding

Idea : backward recursive algorithm to compute $\beta_t(i) = P(O_{t+1}, \ldots, O_T|\mathcal{E}_t = i)$.

Indeed, we have (step (a) by d-separation of $\mathcal{O}_1, \ldots, \mathcal{O}_t$ and $\mathcal{O}_{t+1}, \ldots, \mathcal{O}_T$ by $\mathcal{E}_t$)

$$\begin{aligned} P(O^T, \mathcal{E}_t = i) &= P(O_1, \ldots, O_t, \mathcal{E}_t = i, O_{t+1}, \ldots O_T) \\ &= P(O_1, \ldots, O_t, \mathcal{E}_t = i) P(O_{t+1}, \ldots O_T|O_1, \ldots, O_t, \mathcal{E}_t = i) \\ &\stackrel{a}{=} P(O_1, \ldots, O_t, \mathcal{E}_t = i) P(O_{t+1}, \ldots O_T|\mathcal{E}_t = i) \\ &= \alpha_t(i)\beta_t(i), \end{aligned}$$

Hence $P(O^T) = \sum_{i=1}^N \alpha_t(i)\beta_t(i)$ and $P(\mathcal{E}_t = i|O^T) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^N \alpha_t(i)\beta_t(i)}$.

Notice that

$$P(O^T) = \sum_{i=1}^N \alpha_t(i)\beta_t(i)$$

provides also a forward-backward formula for the computation of $P(O^T)$.

Backward recursion for $\beta_t(i)$

1. Initialization : $\beta_T(i) = 1, \quad 1 \leq i \leq N$
2. Induction : $\beta_t(j) = \sum_{i=1}^{N} \Pi_{j,i} \Sigma_{i,O_{t+1}} \beta_{t+1}(i)$.

   Proof : try to find it as an exercise

Viterbi algorithm : another (efficient) solution to problem No 2

Suppose that we want to find the state sequence maximizing $P(E^T|O^T)$.

This is the single sequence which best explains the observation.

Solution : forward-backward algorithm based on dynamic programming principle.

See notes : we will explain this more in detail in the context of channel decoding.

Remark :

Viterbi algorithm minimizes probability of choosing the wrong code word.

Probability propagation in polytrees

NB: generalization published in the late 1980's (Pearl, Spiegelhalter et al)

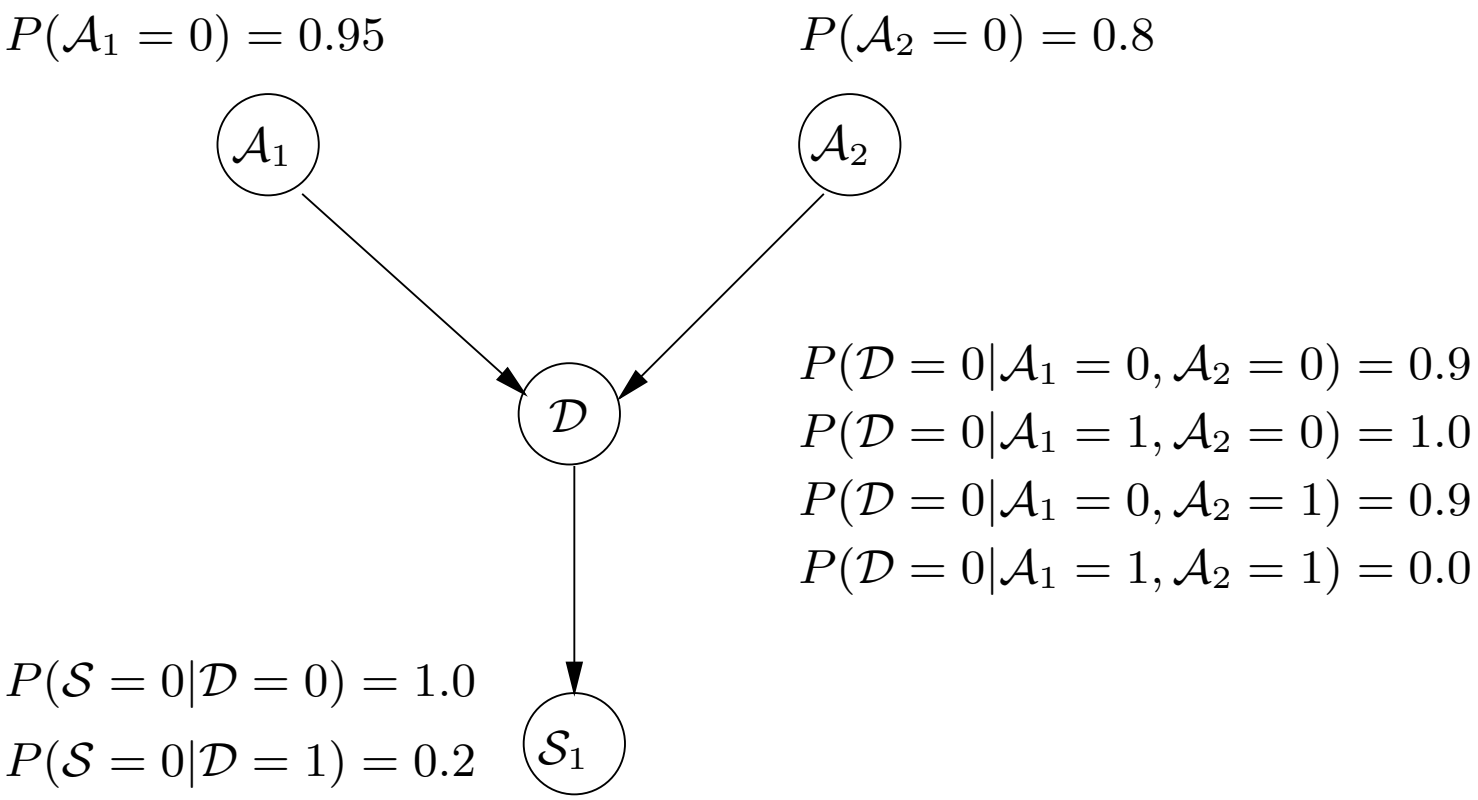
Main steps

- Convert polytree to *factor graph* (which ignores edge directions)
- Arrange factor graph as a horizontal tree with an arbitrary node chosen as root (on the right extreme)
- Starting with the left most levels : forward passing of messages towards the root
- Each node in the factor graph stores received forward messages
- Backward pass : once message has reached root it is sent back towards the leaves.
- Each node in the factor graph stores received backward messages
- Finalization : each node $\mathcal{X}_i$ combines received messages to compute $P(\mathcal{X}_i|OBS)$

Observed variables : send $\delta_{i,j}$ messages indicating belief of the observed value.

Unobserved leaf nodes : send uniform message $1(i)$

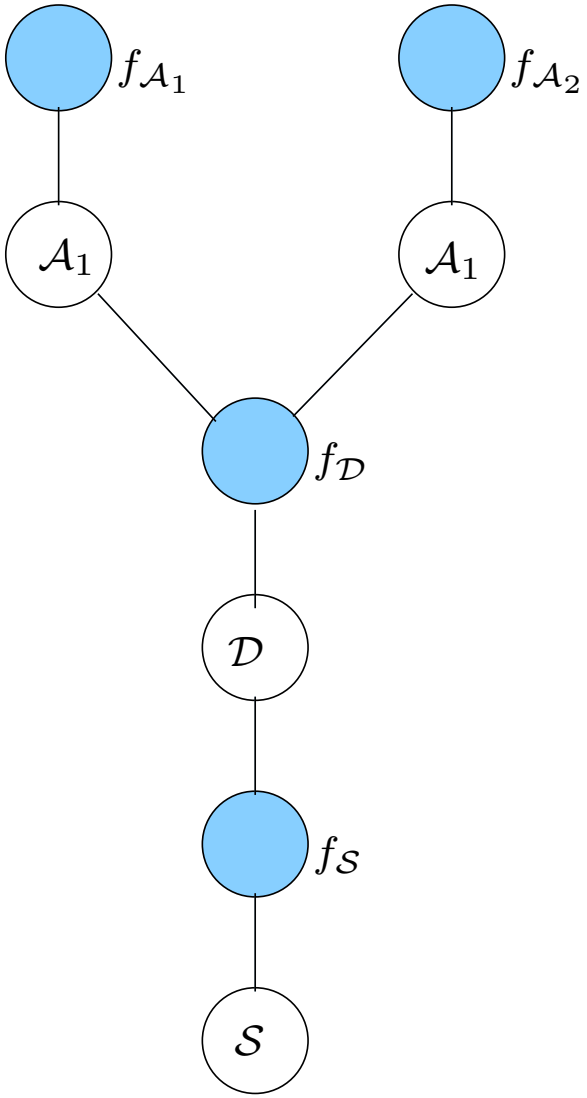
Example : medical diagnosis



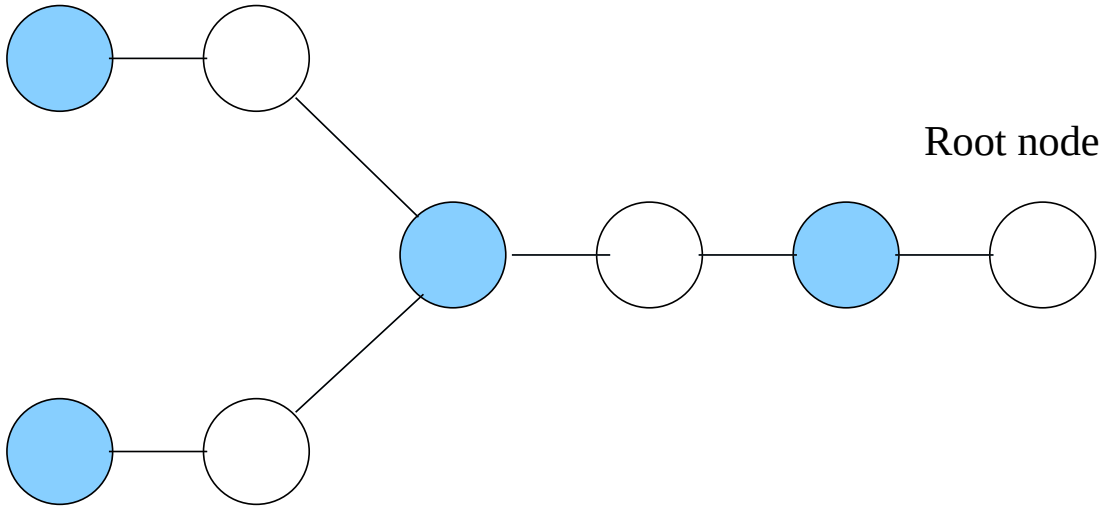
Let's compute $P(\mathcal{D}|\mathcal{S} = 0)$

Factor graph : undirected graph representing factorization of $P(\mathcal{G})$

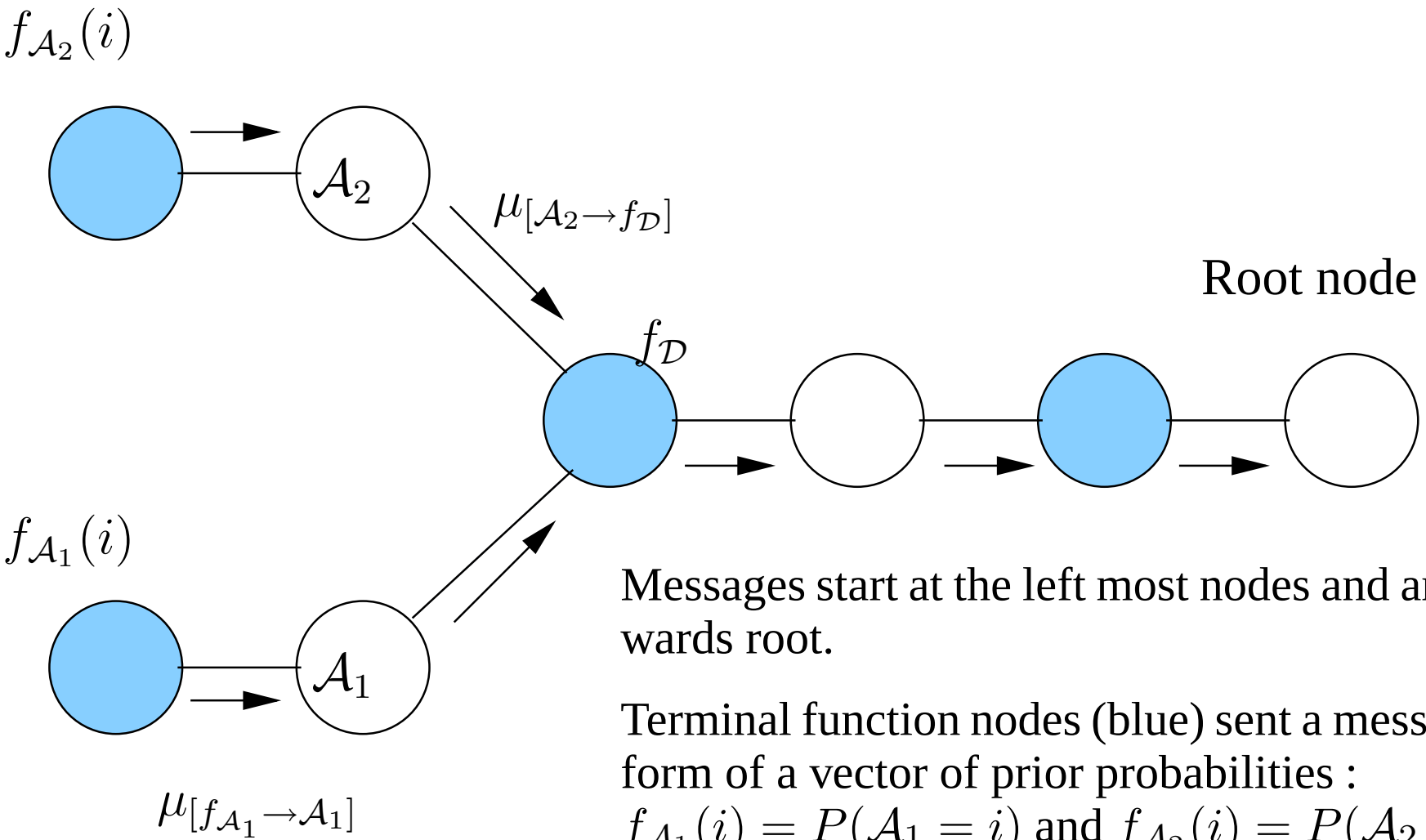
1 *variable* node for each variable + 1 *function* node for each probability table



Notation : we denote by $f_{\mathcal{X}}$ the function node associated to variable $\mathcal{X}$ and by $f_{\mathcal{X}}(\mathcal{X}, \mathcal{P}(\mathcal{X}))$ its table $P(\mathcal{X}|\mathcal{P}(\mathcal{X}))$



Forward pass



Messages start at the left most nodes and are sent towards root.

Terminal function nodes (blue) sent a message in the form of a vector of prior probabilities :
$f_{\mathcal{A}_1}(i) = P(\mathcal{A}_1 = i)$ and $f_{\mathcal{A}_2}(i) = P(\mathcal{A}_2 = i)$

Other nodes store incoming messages, then combine and propagate towards right.

Combination rules :

Need to distinguish among function nodes and variables nodes.

A. Variable nodes $\mathcal{X}$: propagate $\mu_{[\mathcal{X}\to f]}(i)$ towards function node f on the right

1. If variable is observed $\mathcal{X} = j$: $\mu_{[\mathcal{X}\to f]}(i) = \delta_{i,j}$

2. If node is terminal in factor graph and variable is not observed : $\mu_{[\mathcal{X}\to f]}(i) = 1$

3. Otherwise : multiply messages received from left (say function nodes f_1 and f_2) :

$$\mu_{[\mathcal{X}\to f]}(i) = \mu_{[f_1\to\mathcal{X}]}(i)\mu_{[f_2\to\mathcal{X}]}(i)$$

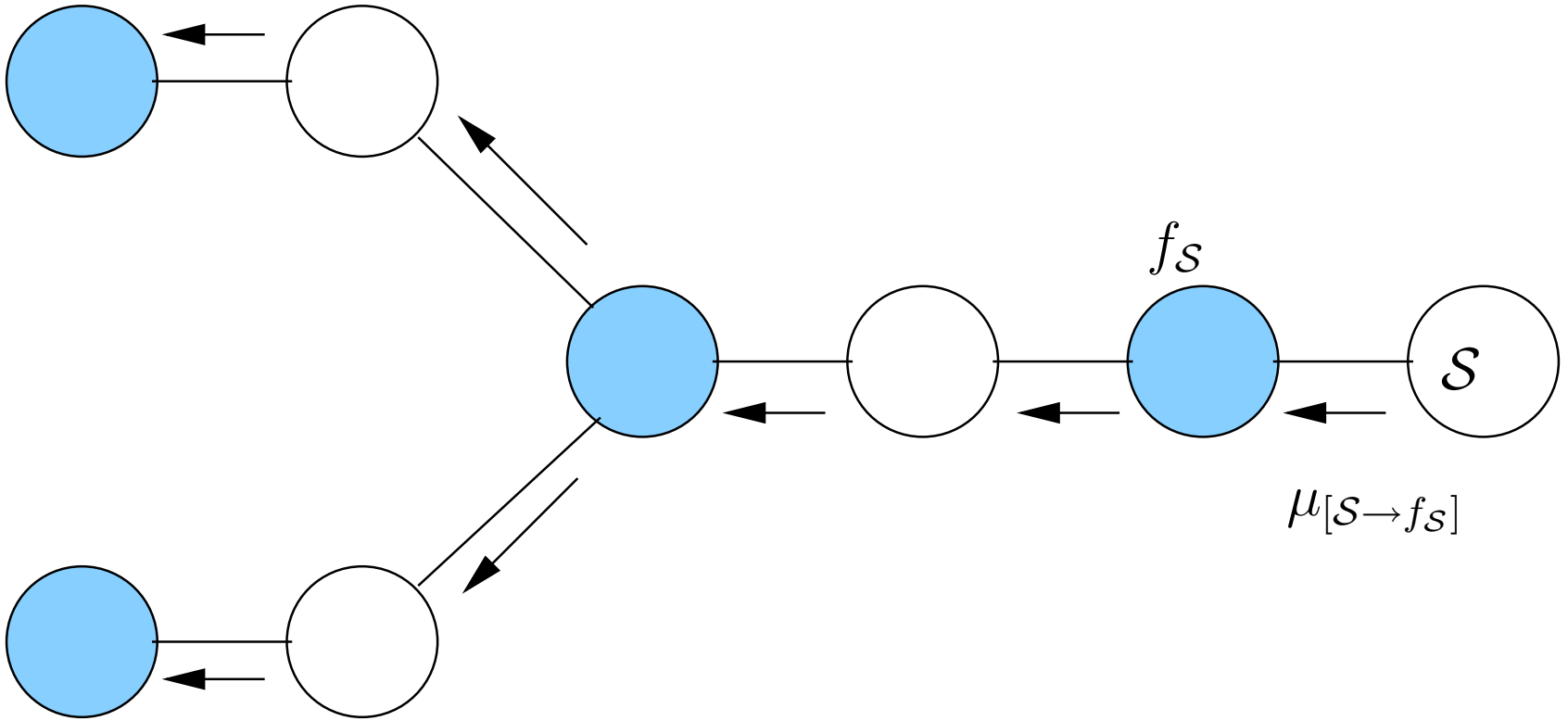
B. Function nodes f : propagate $\mu_{f\to\mathcal{X}}(i)$ towards variable node $\mathcal{X}$ on the right.

All incoming variables are marginalized out, for example for node $f_{\mathcal{D}}$:

$$\mu_{[f_{\mathcal{D}}\to\mathcal{D}]}(i) = \sum_{j\in\mathcal{A}_1}\sum_{k\in\mathcal{A}_2} f_{\mathcal{D}}(i,j,k)\mu_{[\mathcal{A}_1\to f_{\mathcal{D}}]}(j)\mu_{[\mathcal{A}_2\to f_{\mathcal{D}}]}(k)$$

where $f_{\mathcal{D}}(i,j,k) = P(\mathcal{D} = i|\mathcal{A}_1 = j, \mathcal{A}_2 = k)$

Backward pass



Same propagation rules are used as for the forward pass (replace right by left)

Again, each variable node stores its incoming messages (coming from the right)

Note that, here the root node intializes the process : since the value $S = 0$ is observed it will send the message $\mu_{[\mathcal{S}\to f_{\mathcal{S}}]} = \delta_{i,0}$ towards node $f_{\mathcal{S}}$.

Termination (local)

After forward and backward passes, each variable node $\mathcal{X}$ has received a message in the form of a table $\mu_{f\to\mathcal{X}}(i)$ from each of its adjacent function nodes (left and right).

Using this information, the variable node $\mathcal{X}$ can compute its local probability distribution

$$P(\mathcal{X}|O)$$

where O denotes all the observations (except the observed variable nodes).

E.g. we have

$$P(\mathcal{D}=i|O) = \beta\mu_{[f_{\mathcal{D}}\to\mathcal{D}]}(i)\mu_{[f_{\mathcal{S}}\to\mathcal{D}]}(i)$$

where β is a normalization constant, i.e.

$$\beta = \sum_i \mu_{[f_{\mathcal{D}}\to\mathcal{D}]}(i)\mu_{[f_{\mathcal{S}}\to\mathcal{D}]}(i).$$

Remark.

Algorithm can be made more elegant (??) and more general by adding extra nodes to the graph for the observations : these nodes are called constraint nodes.

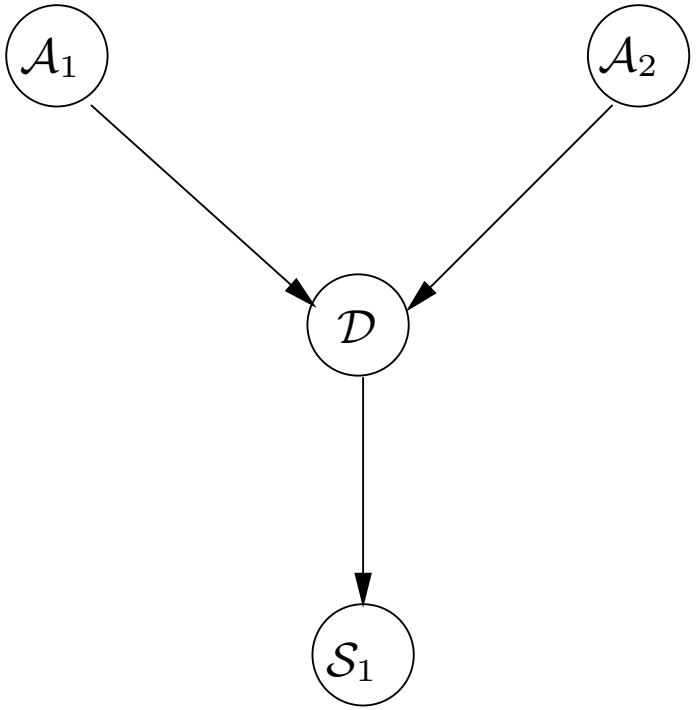
Proof not given here...

Automatic learning of graphical models (a very quick introduction)

A. Bayesian networks

Suppose structure is given and data is completeley observed

Example :



Base de données DB

No	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{D}$	$\mathcal{S}_1$
1	F	F	F	F
2	F	F	F	F
3	F	F	F	F
4	F	T	T	T
5	F	F	F	F
6	T	F	F	F
7	F	T	F	F
⋮	⋮	⋮	⋮	⋮
N-1	F	F	T	F
N	F	F	F	F

Learning problem : estimate probability tables from database.

Basic idea : consider relevant subsets of data base and estimate probabilities by relative frequencies.

E.g. To estimate $P(\mathcal{S}_1|\mathcal{D} = T)$ consider all lines of table suchthat $\mathcal{D} = T$ then count number of lines suchthat also $\mathcal{S}_1 = T$ (and $\mathcal{S}_1 = F$)...

See course notes, for further details and arguments showing that this procedure estimates the model by maximum likelihood.

What if structure is unknown ?

Try out different structures (but combinatorial explosion)

What if some attribute values are missing ? (missing values or hidden variables)

Remove corresponding objects from tables... Estimate missing values from data...

How to take into account constraints among tables ?

E.g. what if we have a time invariance assumption in HMC

B. Automatic learning of decision trees

Same basic idea : estimate conditional probabilities from data base by relative frequency counts.

Tree is built top-down.

$\Rightarrow$ leads to recursive partitioning algorithm (very fast)

NB : needs complexity tradeoff to avoid overfitting

See demonstration : dtapplet