

C.M.Bishop: Pattern Recognition and Machine Learning, Springer, 2006.

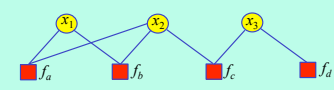
## Approximate Energy Minimisation by Belief Propagation

COMPSCI 773 S1 T  
VISION GUIDED CONTROL  
A/P Georgy Gimel'farb

## Graph Models: Joint P.D.

- **Joint probability distribution** (p.d.) over a set of variables: a product of factors (functions depending each on a subset of variables  $x_i$ )  

$$\mathbf{x} = (x_1, x_2, \dots, x_n) \Rightarrow p(\mathbf{x}) = \prod_i f_i(\mathbf{x}_i)$$
- **Factor graph**: a node for every variable  $x_i$  and an additional node for each factor  $x_i$  in the joint p.d.
  - Factor graph for a p.d.  $p(\mathbf{x}) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$

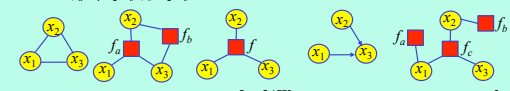


CompSci 773 1

## Factor Graphs

- Factor graphs convey more detailed information about the underlying factorisation of a p.d. than models with only the variable nodes
- Factor graphs are **bipartite**: two distinct kinds of nodes; all links go between nodes of opposite type
  - The same undirected or directed graph has many different factor graphs

$$p(\mathbf{x}) = f(x_1, x_2, x_3) = \frac{f_a(x_1) f_b(x_2) f_c(x_3, x_2, x_1)}{f_d(x_1, x_2, x_3) f_e(x_2, x_3)}$$

$$p(\mathbf{x}) = f(x_1, x_2, x_3) = p_1(x_1) p_2(x_2) p(x_3 | x_1, x_2)$$


CompSci 773 2

## The Sum-product Algorithm

- Evaluating local marginal probability distributions over a single node or a subset of nodes
  - Joint probability distribution:  $p(\mathbf{x}) = p(x_1, \dots, x_n) = \prod_i f_i(\mathbf{x}_i)$
  - Marginal distribution of a single variable:  

$$p(x_i) = \sum_{x_1, \dots, x_n} p(\mathbf{x}) = \sum_{x_1} \dots \sum_{x_{i-1}} \sum_{x_{i+1}} \dots \sum_{x_n} p(x_1, \dots, x_i, \dots, x_n)$$
    - **Summing** - for the assumed discrete variables
    - **Integration** - for continuous variables (e.g. linear-Gaussian models)
- **Belief propagation** for exact inference on **digraphs without loops** is a special case of the **sum-product** algorithm

CompSci 773 3

## The Sum-product Algorithm

- Original graph - an undirected or directed tree, or a polytree
  - The corresponding factor graph has a tree structure
- To compute the marginal  $p(x_i) = \sum_{x_1, \dots, x_n} p(\mathbf{x})$ :
  - Substitute for  $p(\mathbf{x})$  using the factor graph expression  $p(\mathbf{x}) = \prod_i f_i(\mathbf{x}_i)$
  - Then interchange summations and products for obtaining a computationally efficient algorithm:

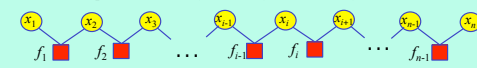
$$p(x_i) = \sum_{x_1, \dots, x_n} \prod_i f_i(\mathbf{x}_i) \Rightarrow \prod_{j \in \text{Nei}(i)} \sum_{x_j} f_j(x_j, \mathbf{x}_{\setminus j})$$

CompSci 773 4

## Computing Marginals

- Joint distribution of  $n$  variables  $p(x_1, x_2, \dots, x_n)$ ;  $x_i$  in  $[0, 1, \dots, X-1]$
- Marginal distribution of a variable  $x_i$ : generally,  $X^{n-1}$  operations!
 
$$p(x_i) = \sum_{x_1=0}^{X-1} \dots \sum_{x_{i-1}=0}^{X-1} \sum_{x_{i+1}=0}^{X-1} \dots \sum_{x_n=0}^{X-1} p(x_1, \dots, x_n)$$
  - A very simple particular case:  

$$p(x_1, \dots, x_n) = f_1(x_1, x_2) f_2(x_2, x_3) \dots f_{n-1}(x_{n-1}, x_n)$$



CompSci 773 5

# Minimum Cut / Maximum Flow

## Computing a Marginal...

•  $p(x_i)$  for  $p(x_1, \dots, x_n) = f_1(x_1, x_2) f_2(x_2, x_3) \dots f_{n-1}(x_{n-1}, x_n)$  :  
only  $(N-1)X^2$  operations by interchanging the sums and the products:

$$p(x_i) = \sum_{x_{i-1}=0}^{X-1} \underbrace{f_{i-1}(x_{i-1}, x_i)}_{\psi_{i-1}(x_i)} \dots \sum_{x_{i+1}=0}^{X-1} \underbrace{f_i(x_i, x_{i+1})}_{\psi_i(x_i)} \dots \sum_{x_{n-1}=0}^{X-1} \underbrace{f_{n-1}(x_{n-1}, x_n)}_{\psi_{n-1}(x_i)}$$

$X_i \Rightarrow X_j$        $X_j \Leftarrow X_i$

CompSci 773 6

## The Sum-product Algorithm

$p(\mathbf{x}) = \prod_{i \in \text{Nei}(x)} F_i(x, X_i)$   
 $p(x) = \sum_{x_{i \in \text{Nei}(x)}} p(\mathbf{x})$

- Partitioning the factors in the joint distribution  $p(\mathbf{x})$  into groups due to the tree structure of the graph:
  - One group is associated with each of the factor nodes that is a neighbour of the variable node  $x$
  - $\text{Nei}(x)$  - the set of factor nodes  $f_i$  that are neighbours of  $x$
  - $X_i$  - the set of all variables in a subtree connected to the variable node  $x$  via the factor node  $f_i$
  - $F_i(x, X_i)$  - the product of all the factors in the group associated with the factor node  $f_i$

A fragment of a factor graph: the evaluation of the marginal  $p(x)$

Messages from  $f_i$  to  $x$ :  $\mu_{f_i \rightarrow x}(x) = \sum_{X_i} F_i(x, X_i)$

CompSci 773 7

## Computing a Marginal...

Particular case:  $p(x) = \prod_{i \in \text{Nei}(x), i \neq \{f_{i-1}, f_i\}} F_i(x, X_i) = \prod_{k=1}^{i-1} \underbrace{f_k(x_k, x_{k+1})}_{F_{k-1}(x, X_{k-1})} \prod_{k=i+1}^{n-1} \underbrace{f_k(x_k, x_{k+1})}_{F_k(x, X_k)}$

$$p(x_i) = \prod_{i \in \text{Nei}(x_i), X_j} F_j(x_i, X_j) = \mu_{f_{i-1} \rightarrow x_i}(x_i) \mu_{f_i \rightarrow x_i}(x_i)$$

$X_{f_{i-1}} = \{x_1, \dots, x_{i-1}\}$        $\text{Nei}(x_i) = \{f_{i-1}, f_i\}$        $X_{f_i} = \{x_{i+1}, \dots, x_n\}$

$\mu_{f_{i-1} \rightarrow x_i}(x_i) = \sum_{x_1, \dots, x_{i-1}} F_{i-1}(x_i, \{x_1, \dots, x_{i-1}\})$        $\mu_{f_i \rightarrow x_i}(x_i) = \sum_{x_{i+1}, \dots, x_n} F_i(x_i, \{x_{i+1}, \dots, x_n\})$

CompSci 773 8

## The Sum-product Algorithm

- The goal marginal is the product of all the incoming messages:
 
$$p(x) = \sum_{\mathbf{x}} p(\mathbf{x}) = \sum_{\mathbf{x}} \prod_{i \in \text{Nei}(x)} F_i(x, X_i) = \prod_{i \in \text{Nei}(x)} \left[ \sum_{X_i} F_i(x, X_i) \right] = \prod_{i \in \text{Nei}(x)} \mu_{f_i \rightarrow x}(x)$$
- Each factor  $F_i(x, X_i)$  is described by a factor (sub)-graph and so can itself be factorised:  $F_i(x, X_i) = f_i(x, x_1, \dots, x_M) G_1(x_1, X_{G_1}) \dots G_M(x_M, X_{G_M})$ 
  - $x_1, \dots, x_M$  are the variables associated with factor  $f_i$  in addition to  $x$

Messages from factor nodes to variable nodes:  $\mu_{f_i \rightarrow x}(x)$

Messages from variable nodes to factor nodes:  $\mu_{x_m \rightarrow f_i}(x_m) \equiv \sum_{X_m} G_m(x_m, X_m)$

CompSci 773 9

## The Sum-product Algorithm

- Computing the messages from factor nodes to variable nodes
  - $\text{Nei}(f_i)$  - the set of variable nodes being neighbours of the factor node  $f_i$
  - $\text{Nei}(f_i), x$  - the same set but without node  $x$ :

$$\mu_{f_i \rightarrow x}(x) = \sum_{x_i} F_i(x, X_i) = \sum_{x_i} \sum_{x_M} f_i(x, x_1, \dots, x_M) \prod_{m \in \text{Nei}(f_i), m \neq x} \left[ \sum_{X_m} G_m(x_m, X_m) \right]$$

$$= \sum_{x_M} \dots \sum_{x_1} f_i(x, x_1, \dots, x_M) \prod_{m \in \text{Nei}(f_i), m \neq x} \mu_{x_m \rightarrow f_i}(x_m)$$

Evaluating the message sent by a factor node to a variable node along their connecting link: (i) Take the product of the incoming messages along all other links coming to the factor node, (ii) Multiply by the factor associated with that node, and (iii) Marginalise over all the variables associated with the incoming messages

CompSci 773 10

## The Sum-product Algorithm

- A factor node can send a message to a variable node once it has received incoming messages from all other neighbouring variable nodes
  - Evaluating the messages from variable nodes to factor nodes - again by the (sub)-graph factorization
    - The term  $G_m(x_m, X_m)$  associated with node  $x_m$  is given by a product of terms  $F_l(x_m, X_{ml})$  each associated with one of the factor nodes  $f_l$  that is linked to node  $x_m$  (excluding node  $f_i$ ):

$$G_m(x_m, X_m) = \prod_{l \in \text{Nei}(x_m), l \neq f_i} F_l(x_m, X_{ml})$$

- The product over all neighbours of node  $x_m$  except for node  $f_i$
- Each of the factors  $F_l(x_m, X_{ml})$ : a subtree of the original graph of the same kind as the joint  $p(\mathbf{x})$

CompSci 773 11

# Minimum Cut / Maximum Flow

$\mu_{f \rightarrow x}(x) = \sum_{x'} F_f(x, x')$

$\mu_{x \rightarrow f}(x_m) = \sum_{x_m'} G_m(x_m, x_m')$

$G_m(x_m, X_m) = \prod_{i \in \text{Neigh}(f)} F_i(x_m, X_m)$

## The Sum-product Algorithm

- Evaluating the message sent by a variable node to an adjacent factor node along the connecting link:
  - The product of the incoming messages along all of the other links:
 
$$\mu_{x \rightarrow f}(x_m) = \prod_{i \in \text{Neigh}(x) \setminus \{f\}} \sum_{x_m'} F_i(x_m, X_m) = \prod_{i \in \text{Neigh}(x) \setminus \{f\}} \mu_{f_i \rightarrow x_m}(x_m)$$
  - Any variable node having only two neighbours simply passes messages through unchanged
  - A variable node can send a message to a factor node once it has received incoming messages from all other neighbouring factor nodes
- The marginal for variable node  $x$  is given by the product of incoming messages along all of the links arriving at that node
  - Each of these messages can be computed recursively in terms of other messages by viewing  $x$  as the root of the tree and starting at the leaf nodes

CompSci 773
12

$p(x) = \prod_{i \in \text{Neigh}(x)} \mu_{f_i \rightarrow x}(x)$

$p(\mathbf{x}) = f_r(\mathbf{x}) \prod_{i \in \text{Neigh}(f_r)} \mu_{x_i \rightarrow f_r}(x_i)$

## The Sum-product Algorithm

- If a leaf node is a variable node, then  $\mu_{x \rightarrow f}(x) = 1$
- If the leaf node is a factor node, then  $\mu_{f \rightarrow x}(x) = f(x)$
- General algorithm:
  - Arbitrarily pick any (variable or factor) node and designate it as the root
  - Propagate messages from the leaves to the root until the root node will have received messages from all of its neighbours
  - Once this message propagation is complete, then propagate messages from the root to all of its neighbours and further along all of the links outwards from the root all the way to the leaves
    - Now, a message will pass in both directions across every link in the graph, and every node will receive a message from all of its neighbours
    - The marginal distribution is readily calculated for every variable in the graph because every variable node has received messages from all its neighbours

CompSci 773
13

## The Sum-product Algorithm

- After one message has passed in each direction across each link, the marginals are evaluated as  $p(x) = \prod_{i \in \text{Neigh}(x)} \mu_{f_i \rightarrow x}(x)$
- Normalisation of the distribution
  - If the factor graph is derived from a directed graph, then the joint distribution is already correctly normalised
    - So the computed marginals will be similarly normalised
  - For an undirected graph, in general there is an unknown factor  $1/Z$ 
    - The sum-product algorithm is to be run first with an unnormalised version of the joint distribution in order to find the unnormalised marginals
    - The factor  $1/Z$  is then easily obtained by normalising any one of these marginals (a computationally efficient approach because the normalisation is done over a single variable rather than over the entire set of variables)

CompSci 773
14

$p(x) = \prod_{i \in \text{Neigh}(x)} \mu_{f_i \rightarrow x}(x)$

$p(\mathbf{x}) = f_r(\mathbf{x}) \prod_{i \in \text{Neigh}(f_r)} \mu_{x_i \rightarrow f_r}(x_i)$

## The Sum-product Algorithm: An Example

- Unnormalised joint distribution:
 
$$\tilde{p}(\mathbf{x}) = f_r(x_1, x_2) f_1(x_2, x_3) f_2(x_2, x_4)$$
- Starting with the leaf nodes:
 
$$\mu_{x_1 \rightarrow f_1}(x_1) = 1; \quad \mu_{f_1 \rightarrow x_2}(x_2) = \sum_{x_1} f_1(x_1, x_2)$$

$$\mu_{x_3 \rightarrow f_2}(x_3) = 1; \quad \mu_{f_2 \rightarrow x_2}(x_2) = \sum_{x_3} f_2(x_2, x_3)$$

$$\mu_{x_4 \rightarrow f_3}(x_4) = 1; \quad \mu_{f_3 \rightarrow x_2}(x_2) = \sum_{x_4} f_3(x_2, x_4)$$
- Propagating messages from the root node out to the leaf nodes:
 
$$\mu_{x_2 \rightarrow f_1}(x_2) = 1; \quad \mu_{f_1 \rightarrow x_1}(x_1) = \sum_{x_2} f_1(x_1, x_2)$$

$$\mu_{x_2 \rightarrow f_2}(x_2) = \mu_{x_3 \rightarrow f_2}(x_2) \mu_{f_2 \rightarrow x_2}(x_2);$$

$$\mu_{x_2 \rightarrow f_3}(x_2) = \sum_{x_4} f_3(x_2, x_4) \mu_{f_3 \rightarrow x_2}(x_2)$$

$$\mu_{x_3 \rightarrow f_1}(x_3) = \mu_{x_2 \rightarrow f_1}(x_2) \mu_{f_1 \rightarrow x_3}(x_3);$$

$$\mu_{x_4 \rightarrow f_2}(x_4) = \sum_{x_2} f_2(x_2, x_4) \mu_{x_2 \rightarrow f_2}(x_2)$$

$$\mu_{x_1 \rightarrow f_2}(x_1) = \sum_{x_2} f_2(x_2, x_1) \mu_{x_2 \rightarrow f_2}(x_2)$$

CompSci 773
15

## The Max-sum Algorithm

- An application of *dynamic programming* in the graphical models
- Find the set of values that jointly have the max probability:
 
$$\mathbf{x}^{\max} = \arg \max_{\mathbf{x}} p(\mathbf{x}) \Leftrightarrow p(\mathbf{x}^{\max}) = \max_{\mathbf{x}} p(\mathbf{x})$$
  - An example for a chain of nodes:
 
$$p(\mathbf{x}) = \frac{1}{Z} \psi_{1,2}(x_1, x_2) \cdot \dots \cdot \psi_{N-1,N}(x_{N-1}, x_N)$$

$$\max_{\mathbf{x}} p(\mathbf{x}) = \frac{1}{Z} \max_{x_1} \left[ \max_{x_2} \psi_{1,2}(x_1, x_2) \left[ \dots \max_{x_N} \psi_{N-1,N}(x_{N-1}, x_N) \right] \right]$$

$$\ln(\max_{\mathbf{x}} p(\mathbf{x})) = \max_{\mathbf{x}} \ln p(\mathbf{x}) = \max_{\mathbf{x}} \left[ \ln \psi_{1,2}(x_1, x_2) + \dots + \ln \psi_{N-1,N}(x_{N-1}, x_N) \right]$$

$$\max_{\mathbf{x}} \ln p(\mathbf{x}) = -\ln Z + \max_{x_1} \left[ \max_{x_2} \ln \psi_{1,2}(x_1, x_2) + \left[ \dots + \max_{x_N} \ln \psi_{N-1,N}(x_{N-1}, x_N) \right] \right]$$

CompSci 773
16

## The Max-sum Algorithm

- Similarly to the sum-product algorithm, the max-sum algorithm is readily written down in terms of message passing by replacing 'sum' with 'max' and products with sums of logarithms:
 
$$\mu_{f \rightarrow x}(x) = \max_{x_1, \dots, x_M} \left[ \ln f(x, x_1, \dots, x_M) + \sum_{m \in \text{neigh}(f) \setminus \{x\}} \mu_{x_m \rightarrow f}(x_m) \right]$$

$$\mu_{x \rightarrow f}(x) = \sum_{i \in \text{neigh}(f)} \mu_{f_i \rightarrow x}(x)$$
- The initial messages sent by the leaf nodes:
 
$$\begin{cases} \mu_{f \rightarrow x}(x) = 0 \\ \mu_{x \rightarrow f}(x) = \ln f(x) \end{cases}$$

CompSci 773
17

# Minimum Cut / Maximum Flow

## The Max-sum Algorithm

- The maximum probability at the root node:
 
$$p^{\max} = \max_x \left[ \sum_{i \in \text{leaf}(x)} \mu_{f_i \rightarrow x}(x) \right]; \quad x^{\max} = \arg \max_x \left[ \sum_{i \in \text{leaf}(x)} \mu_{f_i \rightarrow x}(x) \right]$$
  - The maximum of the joint probability is found by propagating messages from leaves to an arbitrary chosen root node
  - The result is the same irrespective of which node is chosen as the root
- But the propagation of the messages back from the root to the leaves (with maximisation instead of summing) fails to return  $x^{\max}$ 
  - Individual variable values found by maximising the product of messages at each node may belong to different maximizing configurations  $x^{\max}$
  - An overall configuration may no longer correspond to a maximum

CompSci 773 18

## The Max-sum Algorithm

Global maximum path

- To resolve the problem: a different kind of message passing from the root node to the leaves
  - Example:** a chain of  $N$  variables  $x_1, \dots, x_N$  each having  $K$  states

CompSci 773 19

## The Max-sum Algorithm

- Suppose node  $x_N$  is taken to be the root node

**Phase 1:** Propagate messages from the leaf node  $x_1$  to the root node:

$$\mu_{x_n \rightarrow f_{n,n}}(x_n) = \mu_{f_{n,n} \rightarrow x_n}(x_n); \quad \mu_{x_n \rightarrow f_{1,2}}(x_1) = 0$$

$$\mu_{f_{n-1,n} \rightarrow x_n}(x_n) = \max_{x_{n-1}} \left[ \ln f_{n-1,n}(x_{n-1}, x_n) + \mu_{x_{n-1} \rightarrow f_{n-1,n}}(x_{n-1}) \right]$$

$$\phi(x_n) = \arg \max_{x_{n-1}} \left[ \ln f_{n-1,n}(x_{n-1}, x_n) + \mu_{x_{n-1} \rightarrow f_{n-1,n}}(x_{n-1}) \right]$$

- There could be several values  $x_{n-1}$  giving above the same maximum

**Phase 2: Backtracking** - propagate a message back down the chain:

$$x_N^{\max} = \arg \max_{x_N} \left[ \mu_{f_{N,N} \rightarrow x_N}(x_N) \right]; \quad x_{n-1}^{\max} = \phi(x_n^{\max})$$

CompSci 773 20

## The Max-sum Algorithm

- To have a global maximum of a joint probability distribution:
  - Keep track of the maximising states during the forward pass ( $\phi(x_n)$ )
  - Use back-tracking to find a consistent solution
- Extension to a general tree-structured factor graph:
  - If a message is sent from a factor node  $f$  to a variable node  $x$ , a maximisation is performed over all other variable nodes  $x_1, \dots, x_M$  that are neighbours of that factor node
  - When this maximisation is performed, we keep a record of which values of the variables  $x_1, \dots, x_M$  gave rise to the maximum
  - Then in the backtracking step, having found  $x^{\max}$ , we use the stored values to assign consistent maximising states  $x_1^{\max}, \dots, x_M^{\max}$
- Important application: the *Viterbi* algorithm (to find a HMM model)

CompSci 773 21

## MRF Model for Stereo

J. Sun et al., Stereo Matching Using Belief Propagation, IEEE T. PAMI, vol.25 (7), 787-800, 2003

Posterior model:  $P(\mathbf{X} | \mathbf{Y}) \propto \prod_j \psi_j(X_j, Y_j) \prod_{i \in \text{Neib}(j)} \varphi_i(X_j, X_i)$

CompSci 773 22

## Max-sum Algorithm for Stereo


J. Sun et al., Stereo Matching Using Belief Propagation, IEEE T. PAMI, vol.25 (7), 787-800, 2003

- Log-likelihood:  $\log P(\mathbf{X} | \mathbf{Y}) \propto \sum_j \left( \log \psi_j(X_j, Y_j) + \sum_{i \in \text{Neib}(j)} \log \varphi_i(X_j, X_i) \right)$ 
  - Simplification:  $m_j(X_j, X_i) = m_j(X_i)$ ;  $m_j(X_j, Y_j) = m_j(X_j)$
- Max-sum BP:
  - Initialise all messages  $m_{ij}(X_j) = 0$  and  $m_j(X_j) = \log \psi_j(X_j, Y_j)$
  - Update messages  $m_{ij}(X_j)$  iteratively for  $j = 1, \dots, J$ 

$$m_{ij}^{[r+1]}(X_j) \leftarrow \max_{x_j} \left\{ \log \varphi_{ij}(X_j, X_i) + m_j^{[r]}(X_j) + \sum_{k \in \text{Neib}(j) \setminus i} m_{kj}^{[r]}(X_j) \right\}$$
  - Compute beliefs:
 
$$b_j(X_j) \leftarrow m_j^{[r]}(X_j) + \sum_{k \in \text{Neib}(j)} m_{kj}^{[r]}(X_j); \quad \mathbf{X}_j^{\text{MAP}} = \arg \max_{\mathbf{X}} b_j(\mathbf{X})$$

CompSci 773 23



# Minimum Cut / Maximum Flow



## Exact Inference in General Graphs

- The MAP solution  $y^* = \arg \max_y p(y|x)$  can be found by special random sampling from  $p(y|x)$  called *simulated annealing*
  - But its convergence is too slow in most practical cases
- The *sum-product* and *max-sum* algorithms provide efficient and exact solutions to inference problems in tree-structured graphs
- For many practical applications, graphs have loops
  - The message passing framework can be generalised to arbitrary graphs topologies to give an exact inference procedure
  - But in the case of discrete variables, its computational complexity grows exponentially with the maximum number of interdependent variables
  - Thus for many problems of practical interest, it is not feasible to use exact inference, so that effective approximation methods have to be exploited

CompSci 773 24



## Approximate Inference

- **Loopy belief propagation (LBP)**
  - Is possible because the message passing rules for the sum-product algorithm are purely local
  - But because now the graph has cycles, information can flow many times around the graph
    - For some models, the algorithm converges, whereas for others it will not
  - **Message passing schedule:** when each node should send a message across link from the node after receiving messages from all other links
    - Transmit only *pending* messages: after a node receives a message on one of its links (convergence to the exact marginal if no more pending messages)
- **Graph-cut-based algorithms**
  - Exact inference only for binary vector signals  $x$  and decisions  $y$

CompSci 773 25

