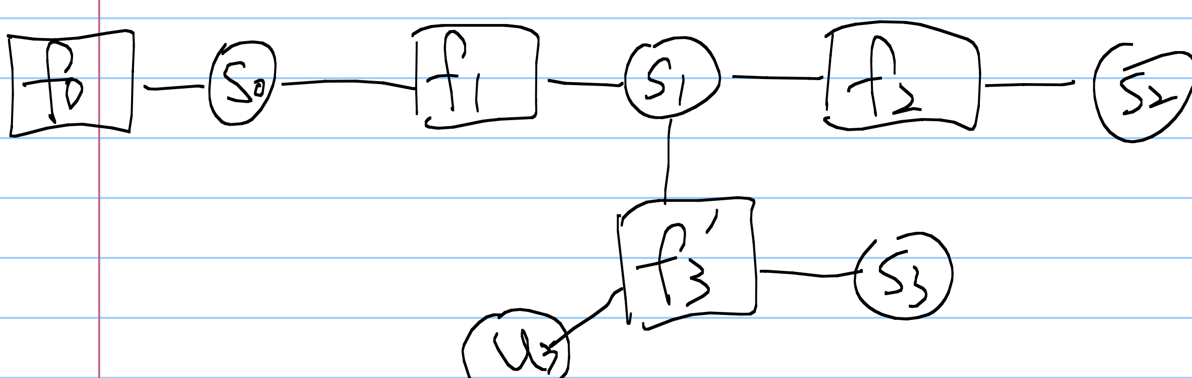
Lecture 13

Example 4: The factor graph representation is not unique.

Q: Continue from the same tree-based convolutional codes. Find out the ML value of U_3. (Basically, we would like to apply the BCJR to this tree-code.)

Ans: Note that the problem of the previous FG representation is that the target bit U_3 is hidden in the function $f_3(S_1, S_3)$ of the trellis structure

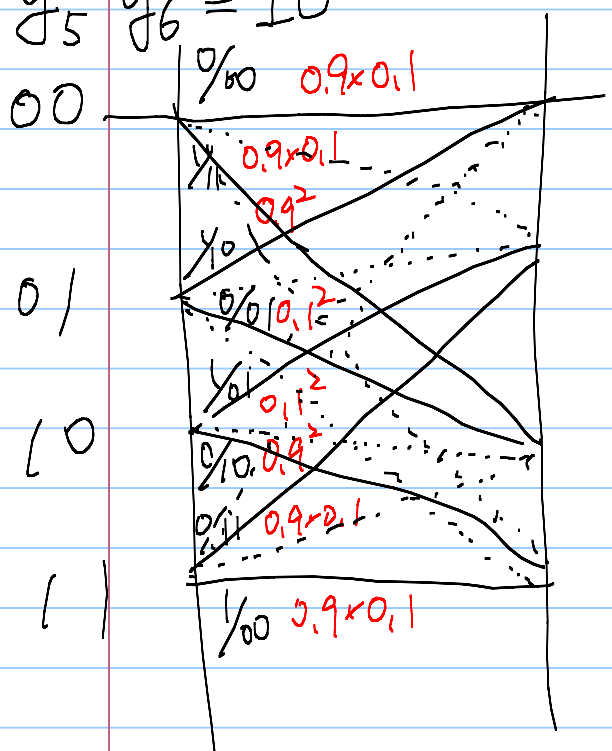
Instead of using the previous factor-graph representation, we can construct a new FG that facilitates the derivation. by singling out the ~~effect~~ of U_3



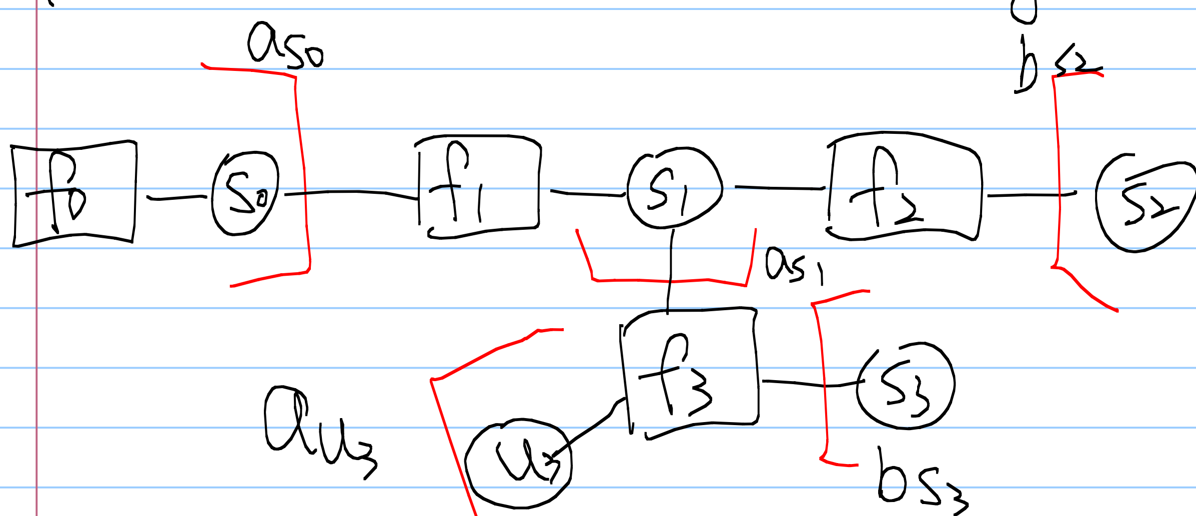
The $f_0(\cdot)$, $f_1(\cdot)$, $f_2(\cdot)$ remain the same

$$f_3'(S_1, S_3, U_3) = \begin{cases} 0.9\times 0.1 & 00\ 00\ 0 \\ 0.9\times 0.1 & 00\ 10\ 1 \\ 0.9^2 & 01\ 00\ 0 \\ 0.1^2 & 01\ 10\ 1 \\ 0.1^2 & 10\ 01\ 0 \\ \vdots & \\ 0 & \text{all other cases} \end{cases} \qquad (S_1\ S_3\ U_3)$$

$y_5\, y_6 = 10$



The BCJR decoder is then straightforward.



Make the decision of U_3 here

The update rules are

$$a_{s_0} = f_0(s_0)$$

$$b_{s_2} = 1 \quad \text{for any } s_2 \text{ value}$$

$$a_{s_1} = \sum_{s_0, s_2} a_{s_0} f_1(s_0, s_1) f_2(s_1, s_2) b_{s_2}$$

$$b_{s_3} = 1 \quad \text{for any } s_3.$$

$$a_{u_3} = \sum_{s_1 s_3} a_{s_1} f_3(s_1, s_3, u) b_{s_3}$$

Note that u_3 takes value in $\{0, 1\}$.

We thus have two values

$a_{u_3=0}$ and $a_{u_3=1}$

the decision of u_3 is $\arg\max_x a_{u_3=x}$

A formal discussion of the factor graph

Def: Each factor graph contains

① n variables $x_1, \cdots, x_n$, representa by a circle

② k factors $f_1 \cdots f_k$, represented by a rectangle

③ and the j-th function f_j takes

$$x_i : i \in \underbrace{\partial j}_{\text{a set}} \subseteq \{1, \cdots, n\} \text{ as input}$$

& output a non-negative value

The j-th rectangle is connected to ∂j circles

④ The objective function is

$$F(x_1, \cdots, x_n) = \prod_{j=1}^{k} f_j(x_i : i \in \partial j)$$

Theorem: If a factor graph is cycle-free, then an efficient Viterbi/BCJR-like algorithm exists when we are interested in finding

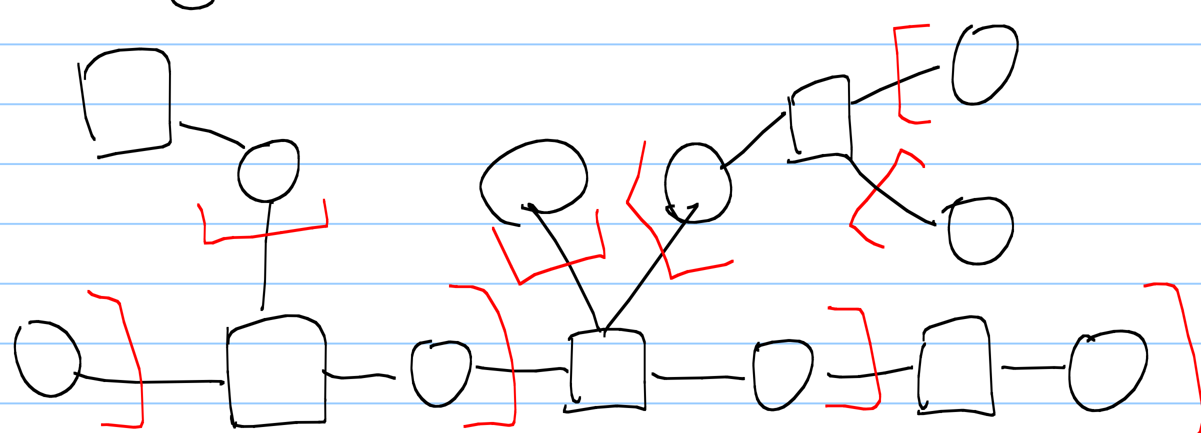
$$\underset{\vec{x}}{\operatorname{argmax}} \prod_{j=1}^{k} f_j(x_i : i \in \partial j)$$

or

$$\underset{x_i}{\operatorname{argmax}} \sum_{\substack{\text{all } \vec{x} \text{ with} \\ \text{fixed } x_i}} \prod_{j=1}^{k} f_j(x_i : i \in \partial j)$$

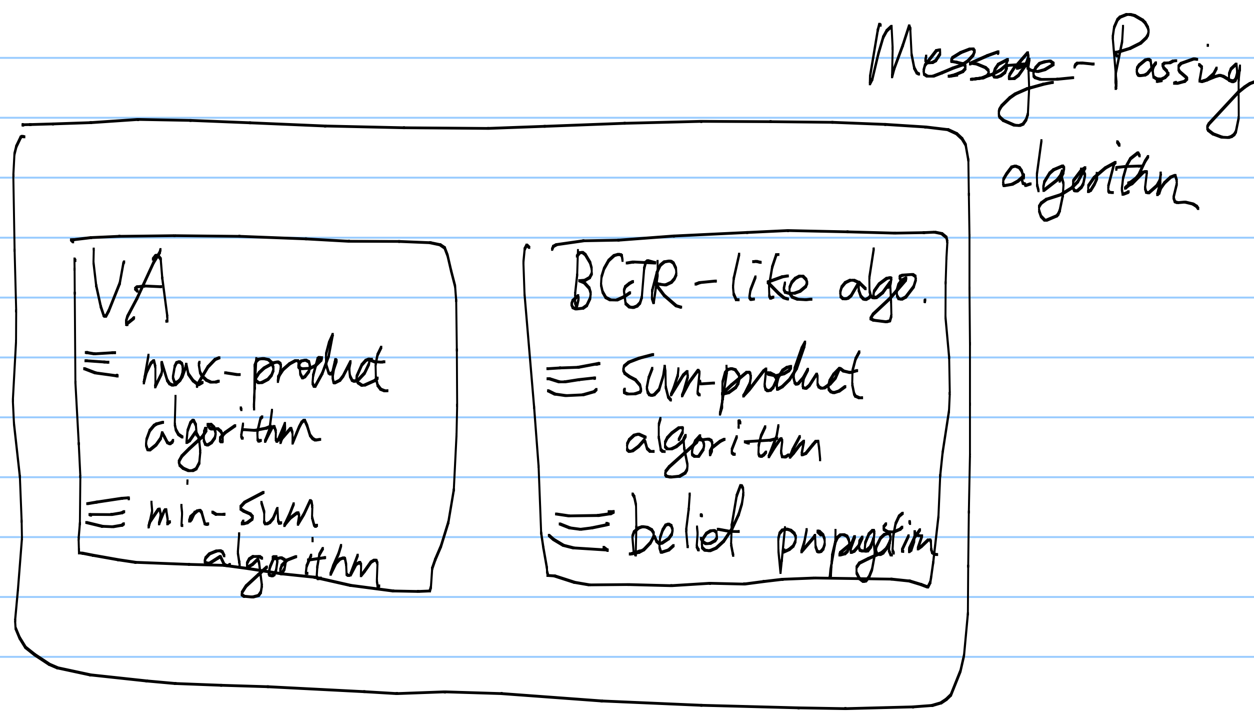
* The corresponding algorithm needs to exchange state-summarizing metrics $a_{x_i}(x)$ which is a function for different x values. (x is a possible value of X_i)
* The computation of a_{x_i} starts from the leaves back to the root based on any arbitrary orientation of the tree.

Ex:

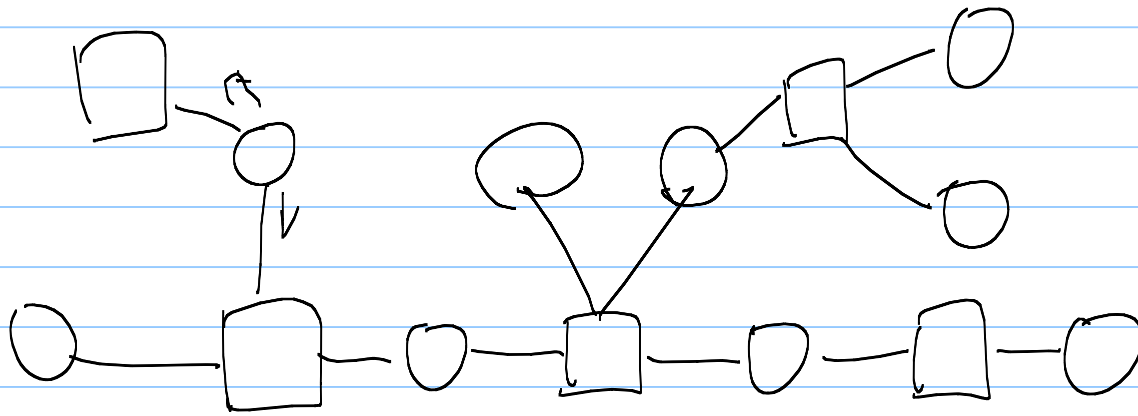


* The "state-summarizing function" $a_{x_i}(\cdot)$ is generally considered as a "message" that is passed from nodes to nodes.

* Since the Viterbi / BCJR-like algorithms pass <u>the function $a_{x_i}(\cdot)$</u> as messages, they are special instances of the message-passing algorithm

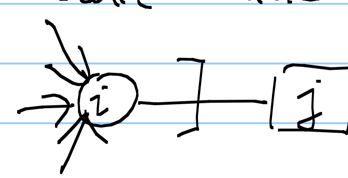


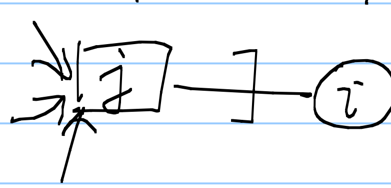
* A parallel implementation of the Viterbi-like, max product algorithm.



Obj: $\underset{\vec{x}}{\operatorname{argmax}} \prod_{j=1}^{k} f_j(x_i : i \in \partial j)$

① Assign a processor for each node

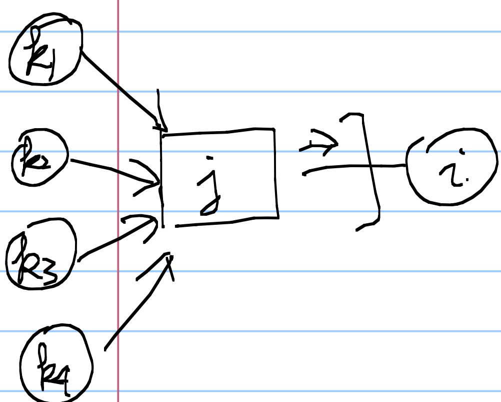
② Each circle sends an $\alpha_{i,j}(x_i)$ message which is a function of the state value x_i summarizing the effect for 

③ Each rectangle sends an $\beta_{j,i}(x_i)$ message which is a function of the x_i, summarizing the effect for 

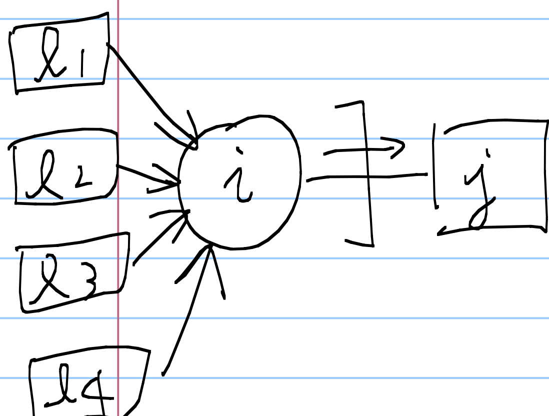
④ The update rules (VA-like algorithm)

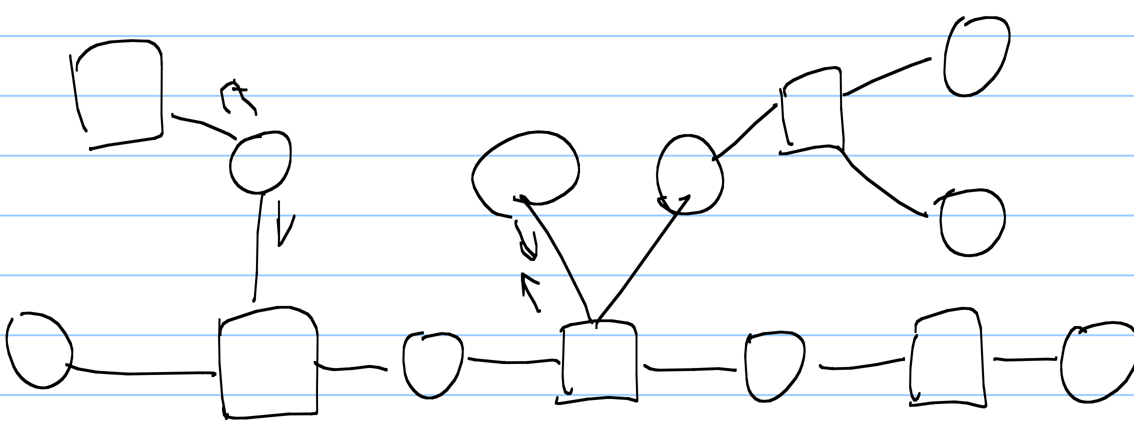
Iterative update

$$\beta_{j,i}(x_i) = \max_{\text{all } \{x_k \,:\, k \in \partial j \setminus i\}} f_j(\underbrace{x_k : k \in \partial j}_{\text{including the fixed } x_i}) \cdot \prod_{k \in \partial j \setminus i} \alpha_{k,j}(x_k)$$



$$\alpha_{i,j}(x_i) = \prod_{l \in \partial i \setminus j} \beta_{l,i}(x_i)$$



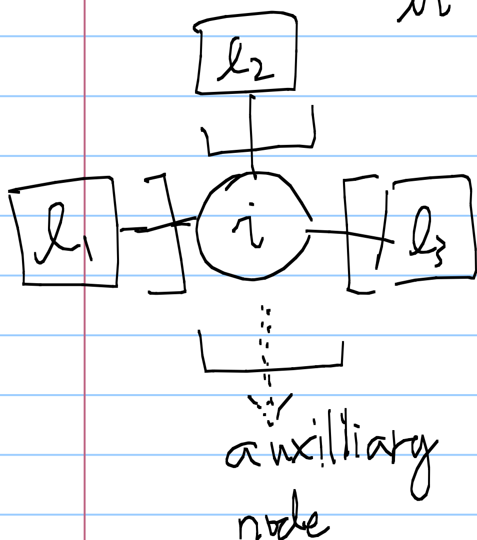


Initialization:

$$\alpha_{ij}(x_i) = 1$$

* Decoding stage:

$$\max_{x_i} \alpha_i(x_i) = \max_{x_i} \prod_{l \in \partial i} \beta_{l,i}(x_i)$$



Once the optimal x_i is decided, we backward trace the max decisions & obtain the maximizing $\bar{x}$

* The order of activating each circle & rectangle to perform message updates can be arbitrary.
* A popular choice is Update all $\alpha_{i,j}$ → Update all $\beta_{j,i}$

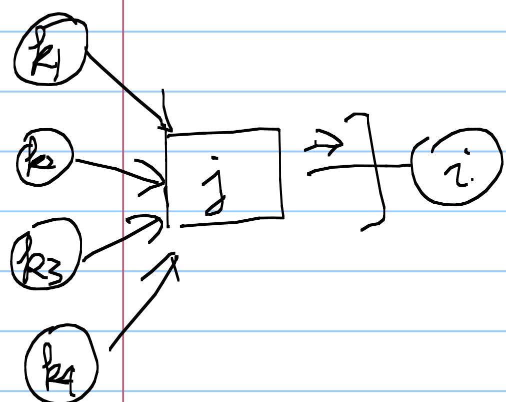
Parallel BCJR algorithm.

Obj: $$\arg\max_{x_i} \sum_{\bar{x}:\text{ with the given } x_i} \prod_{j=1}^{k} f_j(x_i : i \in \partial j)$$

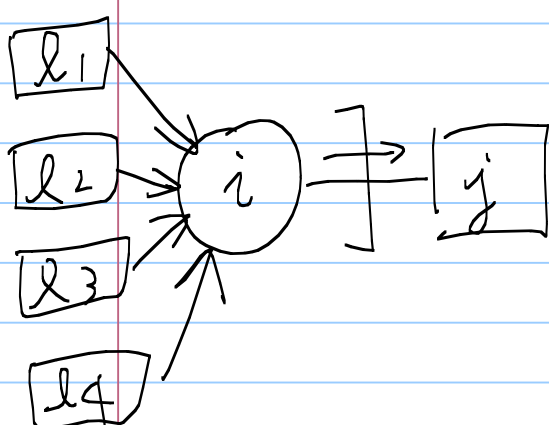
* The update rules

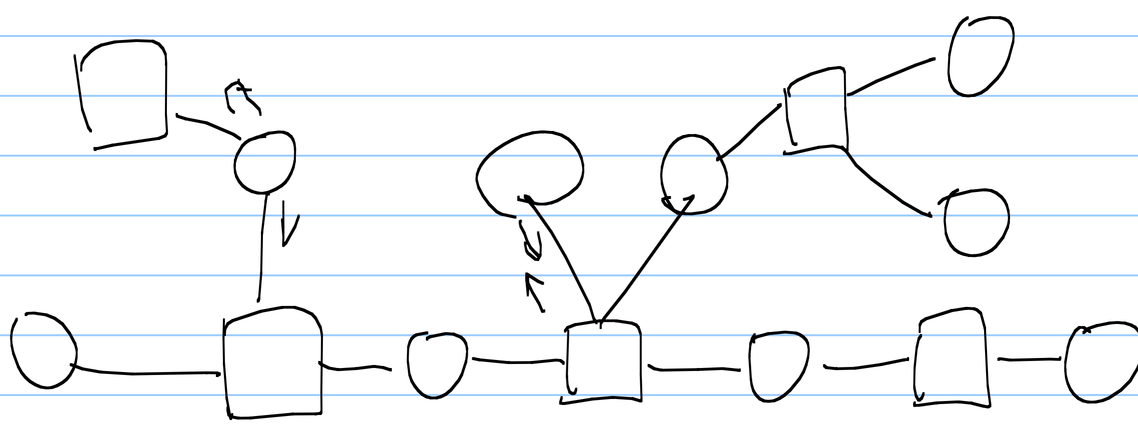
Iterative update

$$\beta_{j,i}(x_i) = \sum_{\text{all } \{x_k : k \in \partial j \setminus i\}} f_j\underbrace{(x_k : k \in \partial j)}_{\text{including the fixed } x_i} \cdot \prod_{k \in \partial j \setminus i} \alpha_{k,j}(x_k)$$



$$\alpha_{i,j}(x_i) = \prod_{l \in \partial i \setminus j} \beta_{l,i}(x_i)$$



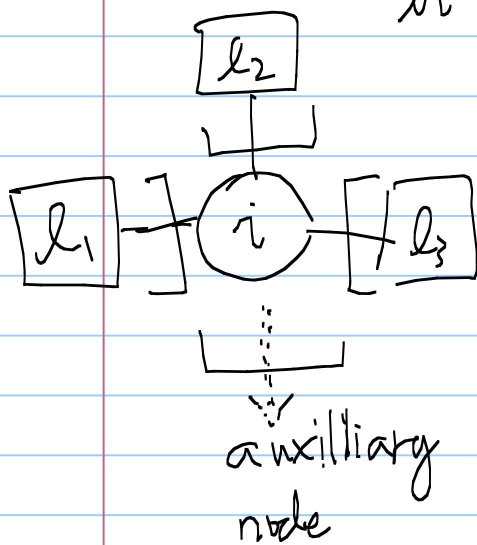


Initialization:

$$\alpha_{ij}(x_i) = 1$$

Decoding stage:

$$\max_{x_i} \alpha_i(x_i) = \max_{x_i} \prod_{l \in \partial i} \beta_{l,i}(x_i)$$



Theorem: The above parallel algorithms converge for acyclic FGs. Namely, after a finite number of parallel update rounds, the message functions $\alpha_{i,j}(\cdot)$ & $\beta_{j,i}(\cdot)$ do not change anymore.

Intuition: Since the FG is acyclic, the messages successfully sumarize the effects on the other side of the FG by accumulating non-overlaped info.

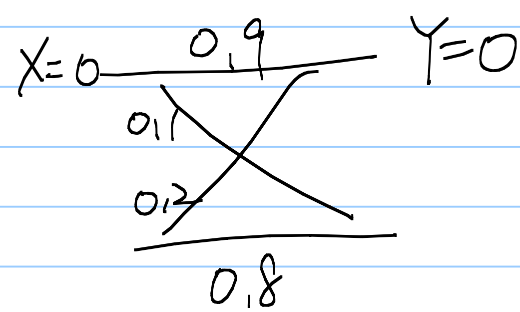
Let us look at two more examples of the FG.

Example 5: linear code: $H = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$

$\vec{y}_{obs} = (1, 1, 1, 0)$

Each codeword is equally likely
(Or when considering the ML instead of the MAP decoder)

Channel



Q: Use the FG to find the bit-based ML decoder

A: $\underset{x \in \{0,1\}}{\operatorname{argmax}} \sum_{\substack{\vec{x}|_i = x \\ H\vec{x}=0}} \prod_{i=1}^{4} P_{Y_i|X_i}(y_i|x_i)$

$= \underset{x \in \{0,1\}}{\operatorname{argmax}} \sum_{\vec{x}|_i = x} f_1(x_1, x_2, x_3)\, f_2(x_2, x_3) \prod_{i=1}^{4} P_{Y_i|X_i}(y_i|x_i)$