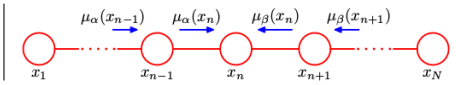
READING 5: BISHOP 8.3-8.4

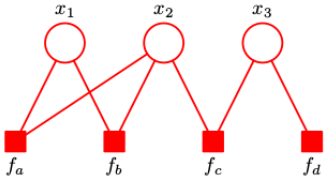
Apoorv Vikram Singh

1. In Markov network, distribution is defined over maximal cliques $p(\boldsymbol{x}) = (1/Z)\prod \psi_c(\boldsymbol{x}_c)$, ($\psi_c \geq 0$, Z= partition function). To convert a graph from directed to undirected, form a clique with elements of conditional distribution: *Moralization*.
2. **Chain Inference**: has a nice interpretation in language of message passing (see fig). Using marginalization, we can express $p(x_n) = (1/Z)\mu_\alpha(x_n)\mu_\beta(x_n)$.

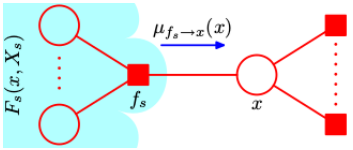


3. To evaluate marginal of each variables, propagate $\mu_\beta(x_{N-1})$ to x_1, and $\mu_\alpha(x_2)$ to x_N. Store the computations in between and use the formula in point 2. Suppose $x_n = \hat{x}_n$ is observed, then $p(\boldsymbol{x}) = (1/Z)\mathbb{1}(x_n = \hat{x}_n)\prod \psi_c(\boldsymbol{x}_c)$.
4. Trees are nice to work with because moralization adds no new links.
5. **Factor graphs**: to generalize both directed and undirected graphs, as can be seen below. There can be multiple factor graphs for same underlying model, and every graph (un/directed, with cycles) can be made a tree, using factor graph

$$p(\boldsymbol{x}) = \prod_s f_s(\boldsymbol{x}_s), \quad \text{e.g.,} \quad p(\boldsymbol{x}) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3).$$



6. **Sum-product algo**: evaluating local marginal over subset of nodes. The marginal $p(x)$ is the product of all the incoming messages arriving at node x: $p(x) = \prod_{s\in \text{ne}(x)} \mu_{f_s\to x}(x)$.
7. Evaluating $\mu_{f_s\to x}(x)$: take product of the incoming messages along all incoming links to factor node, multiply by the associated factor, and marginalize over variables associated with the incoming messages. Note: if f or x is leaf then $\mu_{x\to f}(x) = 1$, $\mu_{f\to x}(x) = f(x)$.



8. **Algorithm description**: (for evaluating $p(x)$) Let x be root of factor graph, and initiate message passing from leaves to root recursively (node can only send message forward if received from all neighbours). Once the root node has received messages from all of its neighbours, the marginal can be evaluated. Like in chain inference, if we want to evaluate marginal of every node, just send back message, root-to-leaves (see fig), store these values and use the formula in point 6.

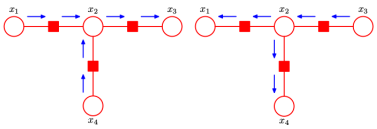


Fig: leaf-root & root-leaf

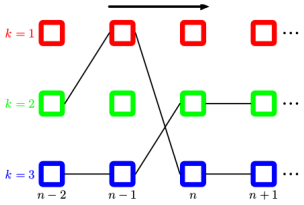
9. **Max-sum algo**: to find a setting of the variables that has the largest probability and the value of that probability. Similar settings, we work with $\max_{\boldsymbol{x}} \log p(\boldsymbol{x})$, to prevent underflow in computations. We define the messages that the algorithm passes. Messages sent by the leaf nodes: $\mu_{x\to f}(x) = 0$, $\mu_{f\to x}(x) = \log f(x)$.

$$\mu_{f\to x}(x) = \max_{x_1,\ldots,x_M} \left[\log f(x, x_1, \ldots, x_M) + \sum_{m\in \text{ne}(f_s)\setminus x} \mu_{m_m\to f}(x_m)\right], \quad \mu_{x\to f}(x) = \sum_{l\in \text{ne}(x)\setminus f} \mu_{f_l\to x}(x).$$

Suppose we are interested in finding $x^{\max}$ for some x. Like before, make it a root node and pass the message from leaf nodes. We get that $x^{\max} = \arg\max_x(\sum_{s\in \text{ne}(x)} \mu_{f_s\to x}(x))$. If we are interested in finding joint maximum over all variable, traversing back like is not direct, due to multiple possible max configuration.

10. **Back-tracking**: Suppose we have chain-graph like in point 2. Then $x_N^{\max} = \arg\max_x(\mu_{f_{N-1},N\to x_N}(x_N))$. We need to determine the states of the previous variables that correspond to the same maximizing configuration: keep track of which values of the variables gave rise to the maximum state of each variable, i.e., store $\phi(x_n) = \mu_{f_{n-1,n}\to x_n}(x_n)$.

For each state of a given variable, there is a unique state $\phi(x_n)$ of the previous variable that maximizes the probability: indicated by lines connecting nodes in fig. Once we know most probable value of x_N, follow link back to x_{N-1},..., to x_1. This corresponds to propagating a message back down the chain using $x_{n-1}^{\max} = \phi(x_n^{\max})$. Easily generalizable to trees. Important application: *Viterbi* algorithm.



11. The message passing framework can be generalized to arbitrary graph topologies, giving an exact inference procedure known as the *junction tree algorithm*. Loopy belief propagating works: apply sum-product algo (even if no guarantee), works well in practice.