10-708: Probabilistic Graphical Models, Spring 2015

7 : Exact Inference

Lecturer: Eric P. Xing *Scribes: Vrushali Fangal, Xiao Liu, and Vipul Singh*

1 Question: Is there a BN that is a perfect map for a given MN?

No. The diamond MN. Accroding to the figure below, in the diamond MN, the conditional independencies include $A \perp C|\{B, D\}$ and $B \perp D|\{A, C\}$. However, no BN can include the same set of independencies. For example, the middle panel represents conditional independencies $A \perp C|\{B, D\}$ and $B \perp D|A$ but $B \perp D|C$ does not hold. And the right panel represents conditional independencies $A \perp C|\{B, D\}$ and $B \perp D$ but $B \perp D|\{A, C\}$ does not hold.

Summary

- Investigated the relationship between BNs and MNs

They represent different families of independence assumptions

- Not mentioned: Chain networks $\rightarrow$ superset of both BNs and MNs
- Why we care about this:

  BN and MN offer different semantics for designer to capture or expression (conditional) independences among variables

  Under certain condition BN can be represented as an MN and vice versa: Trees, Trangulated graphs

  In the future, for certain operation (i.e., inference), we will be using a single representation as the data structure for which an algorithm can operate on.

  This makes algorithm design, and analysis of the algorithms simpler

2 Probabilistic Inference and Learning

A probabilistic graphical model (PGM) M represents a unique probability distribution P over a set of random variables. Given such a representation, the following are some of the important tasks we can accomplish.

1. Querying: Computing the likelihood of certain variables, optionally conditioned on another set of variables. This is generally called *Probabilistic Inference*.
2. Estimation: When the model itself is unknown, estimating a plausible model M from data D. This process is called *Learning*.

Our current focus is on inference. Though learning is usually seen as a task different from inference, it is not always the case. Learning typically involves estimating parameters of the model M so that it explains or "fits" the data D. In Bayesian learning, we particularly seek $p(M|D)$, which is done by assigning prior distributions $p(M)$ over the parameters, and then estimating the parameters given the data $p(M|D)$, which

is essentially inference. Sometimes learning has to be done from incomplete data. In such cases, we run inference to estimate hidden variables representing missing data.

There are three important kinds of queries given a distribution P_M represented by a graphical model M: likelihood, posteriori belief and most probable assignment. We describe each of these in detail.

Likelihood This is the value of the marginal probability of a subset of variables E in the distribution. Likelihood is also calculated to get the conditional probability of a different subset of variables conditioned on E. In that case, E is called the evidence. Let us define a joint probability distribution P over variables $X_{1\ldots n}$. Let $E = X_{k+1}, \ldots, X_n$. To calculate $P(E)$, we do the following.

$$P(E) = \sum_{X_1} \ldots \sum_{X_k} P(X_1 \ldots X_n)$$

Posteriori Belief This is the conditional probability distribution of some query nodes conditioned on an evidence, $P(X|e)$.

$$P(X|e) = \frac{P(X,e)}{P(e)} = \frac{P(X,e)}{\sum_x P(X=x,e)}$$

In case there is a subset of X that we do not care about, that is if $X = Y, Z$ and we want to calculate $P(Y|e)$,

$$P(Y|e) = \sum_z P(Y, Z=z|e)$$

This process of summing out z is called marginalization. A generalization of the same idea is shown in Section 3 as variable elimination.

Posteriori belief has multiple applications in statistical analysis and learning. Given a causal trail, calculating the conditional probability of effects given causes is called prediction. In this case, the query node is a descendant of the evidence. Diagnosis is calculating the conditional probability of causes given effects, and this is useful in finding the probability of a disease given the symptoms. In this case, the query node is an ancestor of the evidence node in the trail. While learning under partial observation, posteriori belief of the unobserved variables given the observed ones is calculated.

Most Probable Assignment In this query, we are interested in finding only one set of values for the query variables that maximize the given conditional probability instead of finding the entire distribution.

$$MPA(Y|e) = argmax_{y \in Y} P(Y|e)$$

Calculating the posteriori belief is not a necessary pre-requisite for finding MPA as we can usually find some clever tricks to directly calculate the latter. MPA has direct applications in tasks like classification and explanation.

2.1 Complexity of Inference

Computing $P(X = x|e)$ in a GM is an NP-hard problem, which means that there is not generalized efficient algorithm for inference given an arbitrary PGM, query and evidence nodes. However, for some families of models, there are polynomial time algorithms for inference. Another solution to deal with the hardness of the problem is finding approximate solutions. The following is a list of some of the exact and approximate algorithms on graphical models.

Exact Inference

- Elimination algorithm
- Message passing algorithm
- Junction tree algorithms

Approximate Inference

- Stochastic simulation
- Markov chain Monte Carlo methods
- Variational algorithms

The current focus is on exact algorithms, and in particular we deal with the Elimination algorithm in detail. A key observation is that given query and evidence nodes in a PGM, calculating the posteriori belief does not involve some of the variables in the joint distribution. Hence the idea is to avoid the marginalization involving a naive summation over an exponential number of terms.

3 Marginalization and Elimination

3.1 Variable Elimination on a Chain

Let us consider a simple example of directed chain of 5 random variables A, B, C, D, and E, as in Figure 1. For simplicity suppose that each variable can take n number of values. A straightforward probabilistic description would require a full joint probability table containing n^5 entries. As we can see the size of this table is exponential in the number of variables. Now if we want to evaluate the probability that variable E takes on value e, i.e. $P(E = e)$ under this model description, we have to compute:

$$P(e) = \sum_{a,b,c,d} P(a, b, c, d, e)$$

However, this operation requires marginalizing/integrating/summing out the other 4 variables, which involves $O(n^4)$ addition operations, i.e. exponential number of operations in the number of variables.

Figure 1: A simple directed probabilistic model with a chain structure

We can be smarter by exploiting the structure of the chain, i.e. factorize the joint probability according to the chain and rewrite as:

$$\begin{aligned} P(e) &= \sum_{a,b,c,d} P(a, b, c, d) \\ &= \sum_{a,b,c,d} P(a)P(b|a)P(c|b)P(d|c)P(e|d) \end{aligned}$$

Now in the factorized joint distribution, not all terms have a dependence on all variables. So if we want to sum over A, then $P(c|b), P(d|c)$, and $P(e|d)$ are not affected by the value of a. So we can pull those three terms that don't depend on a out of this summation. Then we have to perform summation over the only terms that are affected, i.e. $P(a)$ and $P(b|a)$.

$$\begin{aligned} P(e) &= \sum_{a,b,c,d} P(a)P(b|a)P(c|b)P(d|c)P(e|d) \\ &= \sum_{b,c,d} P(c|b)P(d|c)P(e|d) \sum_{a} P(a)P(b|a) \\ &= \sum_{b,c,d} P(c|b)P(d|c)P(e|d)P(b) \end{aligned}$$

The result of the summation would marginal distribution of B which is a function that depends only on the value of B and no other variables. Next, we can repeat the same process using the summation over another variable, such as B. First we pull out all the terms in the joint distribution that don't depend on B, and then we compute the value of the sum:

$$\begin{aligned} P(e) &= \sum_{b,c,d} P(c|b)P(d|c)P(e|d)P(b) \\ &= \sum_{c,d} P(d|c)P(e|d) \sum_{b} P(c|b)P(b) \\ &= \sum_{c,d} P(d|c)P(e|d)P(c) \end{aligned}$$

We can continue this process by performing the same steps for the remaining summations over C and D:

$$\begin{aligned} P(e) &= \sum_{c,d} P(d|c)P(e|d)P(c) \\ &= \sum_{d} P(e|d) \sum_{c} P(d|c)P(c) \\ &= \sum_{d} P(e|d)P(d) \end{aligned}$$

Using this procedure, we can eliminate one variable at a time. This reduces the amount of computation by performing the summations one at a time consisting of $O(n^2)$ operations (due to the number of values taken by the variable (node) being marginalized those taken by its child node). Then total elimination steps would be $O(kn^2)$ where k is the number of variables.

3.2 Variable Elimination on a HMM

A more realistic example would be to consider a Hidden Markov Model (HMM), as shown in Figure 2. Suppose we are interested in finding out the state the HMM is in when it emitted x_i, then we would like to evaluate $P(y_i|x_1, ..., x_T)$. As in the chain, we begin by writing out the full joint distribution and marginalizing out all of the variables that we don't care about:

$$\begin{aligned} P(y_i|x_1, ..., x_T) &\propto \sum_{y_1,...,y_{i-1},y_{i+1},...,y_T} p(x_1, x_2, ..., x_T, y_1, y_2, ..., y_T) \\ &= \sum_{y_1,...,y_{i-1},y_{i+1},...,y_T} p(y_1)p(x_1|y_1)p(y_2|y_1)p(x_2|y_2)...p(y_T|y_{T-1})p(x_T|y_T) \end{aligned}$$

Once again, we can reduce computational cost by exploiting the structure in the HMM or its graphical model. The procedure would be to move in any order through the variables being eliminated (all Y's except Y_i), select only the terms in the factored joint distribution that depend on the current elimination variable Y_j, and calculate the summation over values y_j to generate a new factor that doesn't depend on Y_j. For example, suppose we start by eliminating Y_1. Then the first summation we compute will be:

$$\sum_{y_1} P(y_1)P(x_1|y_1)P(y_2|y_1) = \phi_1(x_1, y_2)$$

Next we eliminate Y_2 by performing the following summation:

$$\sum_{y_2} \phi_1(x_1, y_2)P(x_2|y_2)P(y_3|y_2) = \phi_2(x_1, x_2, y_3)$$

Then we eliminate each of $Y_3, ..., Y_{i-1}, Y_{i+1}, ..., Y_T$ in turn. We can calculate the conditional probability of y_i after all the variables have been eliminated. This process is exactly the Forward-Backward algorithm for HMMs. To make the parallel clearer, let us define the α and β probabilities.

$$\alpha(y_i) = \phi_i(x_1, x_2, ..., x_i, y_i)$$

$$\beta(y_i) = \phi_{i+1}(x_{i+1}, ..., x_T, y_i)$$

Now,

$$p(y_i|x_1, ..., x_T) \propto P(x_i|y_i)\alpha(y_i)\beta(y_i)$$

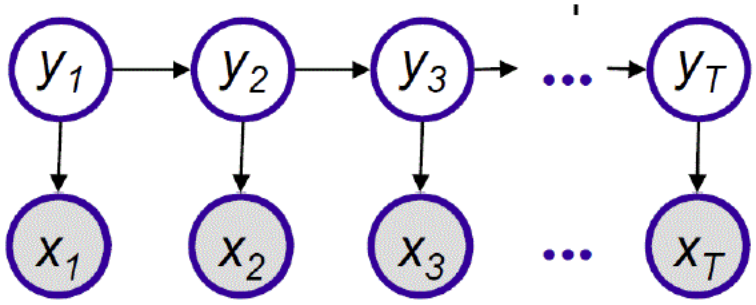


Figure 2: Hidden Markov Model

We next need to normalize to obtain a proper probability:

$$P(y_i|x_1, ..., x_T) = \frac{P(x_i|y_i)\alpha(y_i)\beta(y_i)}{\sum_{y_i} P(x_i|y_i)\alpha(y_i)\beta(y_i)}$$

Just as we saw in the chain example, the core operation is the marginalization of each variable, which takes $O(n^2)$ time. Since we perform T such steps, the total cost is again $O(Tn^2)$.

3.3 Variable Elimination on Undirected Chains

Now consider a simple example of undirected chain of 5 random variables A, B, C, D, and E, as in Figure 3. Now if we want to evaluate the probability that variable E takes on value e, i.e. $P(E = e)$ under this model description, we have to compute:

$$P(e) = \sum_{a,b,c,d} \frac{1}{Z}\phi(a, b)\phi(b, c)\phi(c, d)\phi(d, e)$$

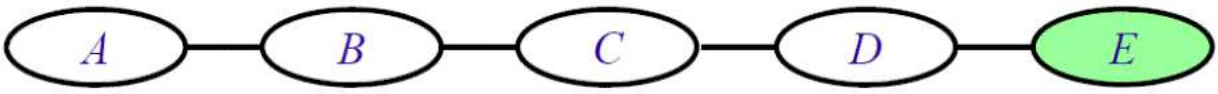


Figure 3: A simple undirected probabilistic model with a chain structure

Like before we can see that not all terms have a dependence on all variables. So if we want to sum over A, then $\phi(b,c), \phi(c,d)$, and $\phi(d,e)$ are not affected by the value of a. So we can pull those three terms that don't depend on a out of this summation. Then we have to perform summation over the only $\phi(a,b)$. We keep repeating this procedure.

$$\begin{aligned}
P(e) &\propto \sum_{a,b,c,d} \phi(a,b)\phi(b,c)\phi(c,d)\phi(d,e) \\
&= \sum_{b,c,d} \phi(b,c)\phi(c,d)\phi(d,e) \sum_{a} \phi(a,b) \\
&= \sum_{b,c,d} \phi(b,c)\phi(c,d)\phi(d,e) m_a(b) \\
&= \sum_{c,d} \phi(c,d)\phi(d,e) \sum_{b} \phi(b,c) m_a(b) \\
&= \sum_{c,d} \phi(c,d)\phi(d,e) m_b(c) \\
&= \sum_{d} \phi(d,e) \sum_{c} \phi(c,d) m_b(c) \\
&= \sum_{d} \phi(d,e) m_c(d) \\
&= \sum_{d} \phi(d,e) m_c(d) \\
&= m_d(e)
\end{aligned}$$

Finally we normalize to obtain a proper probability:

$$P(e) = \frac{m_d(e)}{\sum_e m_d(e)}$$

Next we present a formal general algorithm to solve such problems.

3.4 The General Variable Elimination Algorithm

Suppose $\mathcal{X}$ be set of all random variables involved in the graphical model. Let $\mathfrak{F}$ denote the set of factors and then for each factor $\phi \in \mathfrak{F}$ denote by $\mathsf{scope}[\phi] \subseteq \mathcal{X}$ the set of all variables involved in ϕ. Suppose $\mathcal{Y} \subset \mathcal{X}$ be the set of query variables, then $Z = \mathcal{X} - \mathcal{Y}$ would be the set of variables to be eliminated. Further we can have a set of observed variables, termed as evidence denoted by $\mathcal{E}$. We deal with evidence, i.e. observed variables by defining evidence potential:

$$\delta(\mathcal{E}_i, \bar{e}_i) = \begin{cases} 1 & \text{if } \mathcal{E}_i = \bar{e}_i \\ 0 & \text{if } \mathcal{E}_i \neq \bar{e}_i \end{cases}$$

and in turn the total evidence potential:

$$\delta(\mathcal{E}, \bar{e}) = \prod_{i \in \mathcal{I}} \delta(\mathcal{E}_i, \bar{e}_i)$$

This trick of evidence potential allows us to treat marginalization and conditioning as formally equivalent as the evidence potentials basically converts evaluations into sums.

The high level idea is to express any query in the following form:

$$P(\mathcal{Y}|\mathcal{E}) \propto \sum_{\mathcal{Z}} \prod_{\phi \in \mathfrak{F}} \phi(\mathsf{scope}[\phi])\delta(\mathcal{E}, \bar{e})$$

Throughout the algorithm we maintain an active set of potential functions. The active list is initialized to hold all factors and the evidence potentials. Then iteratively we find all factors involving the the next variable in the elimination ordering. These factors are removed from the active list and sum of their product with respect to current variable in the elimination order is carried out. In simple words we move all irrelevant terms outside of the innermost sum and compute the innermost sum. Thus obtaining a new intermediate factor, which we re-insert into the active list. We stop when all variables in $\mathcal{Z}$ are eliminated in the ordering $\mathcal{I}$. Finally we wrap-up with normalization as:

$$P(\mathcal{Y}|\mathcal{E}) = \frac{\phi(\mathcal{Y}, \mathcal{E})}{\sum_{\mathcal{E}} \phi(\mathcal{Y}, \mathcal{E})}$$

3.5 Pseudocode

If G is the graphical model, the pseudocode for the algorithm can be given as

Eliminate$(G, \mathcal{E}, \mathcal{Z}, \mathcal{Y})$:
Initialize$(G, \mathcal{Y})$
Evidence$(\mathcal{E})$
Sum-Product-Variable-Elimination$(\mathcal{F}, \mathcal{Z}, \prec)$
Normalization$(\mathcal{F})$

Let us see each of the sub-procedures in detail:

1. **Intialize**$(\mathbf{G}, \mathcal{Z})$:
   (a) Choose an elimination ordering with the query variable at the end of the ordering. Also, if $\mathcal{Z}_1, \mathcal{Z}_2, ...\mathcal{Z}_k$ is the ordering then $\mathcal{Z}_i \prec \mathcal{Z}_j$ iff $i < j$.
   (b) Initialize the active list, which is essentially a stack $\mathcal{F}$ with the full set of factors, $\phi_1, \phi_2, ...$
2. **Evidence**$(\mathcal{E})$:
   (a) Grow the stack $\mathcal{F}$ further by loading all evidences
3. **Sum-Product-Variable-Elimination**$(\mathcal{F}, \mathcal{Z}, \prec)$:
   (a) For all variables in the ordering, $\mathcal{Z}_1, \mathcal{Z}_2, ...\mathcal{Z}_k k$ call subroutine:

$$\mathcal{F} \leftarrow \mathit{Sum\text{-}Product\text{-}Eliminate\text{-}Var}(\mathcal{F}, Z_i); i = 1, ..., k$$

   Each iteration creates a new stack and overwrites the existing one. The subroutine sums over the product of all factors of the variable to be eliminated and returns a new stack with the variable eliminated.

(b) After all iterations are complete, we get a stack of final potentials. We take product of all of these to get our query out.

$$\phi^* \leftarrow \prod_{\phi \in \mathcal{F}} \phi$$

4. **Normalization($\mathcal{F}$)**:

   (a) Finally, we normalize the potential thus obtained ϕ^*.

$$P(X|E) \leftarrow \phi^*(X) / \sum_x \phi^*(X)$$

5. **Sum-Product-Eliminate-Var** $(\mathcal{F}, Z)$:

   (a) Define $\mathcal{F}'$ as scope of Z, i.e., all terms in the stack $\mathcal{F}$ having Z_i as an argument.

$$\mathcal{F}' \leftarrow \{\phi \in \mathcal{F} : Z \in Scope[\phi]\}$$

   (b) Define $\mathcal{F}$" as the remaining stack.

$$\mathcal{F}" \leftarrow \mathcal{F} - \mathcal{F}'$$

   (c) Take product of all factors in F'. This is the product step of the Sum-Product operation.

$$\psi \leftarrow \prod_{\phi \in \mathcal{F}'} \phi$$

   (d) Take sum on these product factors over Z, so that Z is eliminated. A new factor τ is generated.

$$\tau \leftarrow \sum_Z \psi$$

   (e) Put τ on top of stack and return.

$$return \quad \mathcal{F}" \cup \tau$$

3.6 Variable Elimination on more complex network

Consider the a little more complicated directed graphical model as shown in figure 3. The complete set of random variables is $\mathcal{X} = A, B, C, D, E, F, G, H$. Suppose we need to evaluate the query $P(A|H = h)$, then $\mathcal{Y} = A$ and we need to eliminate $\mathcal{Z} = B, C, D, E, F, G, H$. The evidence set is simply $\mathcal{E} = H$ with the evidence potential being $\delta(\mathcal{E}, \bar{e}) = \delta(H, h)$.
So the initially with all factors we have

$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$

and also push in the evidence potential $\delta(H, h)$. Suppose we choose an elimination order $\mathcal{I} = H, G, F, E, D, C, B$ (more on how to choose a good elimination order will follow). Then we follow the general variable elimination algorithm as follows:

1. **Conditioning H**: As discussed earlier this step is isomorphic to a marginalization step wherein we fix the evidence node on its observed value:

$$m_h(e, f) = \sum_h p(h|e, f)\delta(h, \tilde{h})$$

   Leaving the factors as

$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e, f)$$

2. **Eliminate G**: We compute
$$m_g(e) = \sum_g P(g|e) = 1$$
to yield
$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)m_h(e,f)$$

3. **Eliminate F**: We compute
$$m_f(e,a) = \sum_f p(f|a)m_h(e,f)$$
to yield
$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)m_f(e,a)$$

4. **Eliminate E**: We compute
$$m_e(a,c,d) = \sum_e P(e|c,d)m_f(e,a)$$
to yield
$$P(a)P(b)P(c|b)P(d|a)m_e(a,c,d)$$

5. **Eliminate D**: We compute
$$m_d(a,c) = \sum_d P(d|a)m_e(a,c,d)$$
to yield
$$P(a)P(b)P(c|b)m_d(a,c)$$

6. **Eliminate C**: We compute
$$m_c(a,b) = \sum_c P(c|b)m_d(a,c)$$
to yield
$$P(a)P(b)m_c(a,b)$$

7. **Eliminate B**: We compute
$$m_b(a) = \sum_b P(b)m_c(a,b)$$
to yield
$$P(a)m_b(a)$$

8. **Normalization**: We wrap-up with normalization as
$$P(a|h=\tilde{h}) = \frac{P(a)m_b(a)}{\sum_a P(a)m_b(a)}$$

3.7 Complexity of Variable Elimination

For each variable x to be eliminated, the Sum-Product operation can be broken down into two parts:

1. **Product**:
$$m'_x(x,y_1,..,y_k) = \prod_{i=1}^{k} m_i(x,y_{c_i})$$

2. **Sum**:

$$m_x(y_1, .., y_k) = \sum_x m'_x(x, y_1, ..., y_k)$$

Now, Total number of multiplications required = $k \times |Val(X)| \times \prod_i |Val(Y_{C_i})|$,
where Y_{c_i} is the i^{th} configuration of the clique c defined on Y

Total number of additions = $|Val(X)| \times \prod_i |Val(Y_{C_i})|$

Following important points are made about the Sum-Product operation:

1. Both the above steps are polynomial, not exponential to the number of states in every random variable.
2. $Val(Y_{C_i})$ is the local exponential term, dependent on the size of the clique used to define intermediate term. However, outside the clique, this becomes multiplicative.
3. This is the major benefit of the Sum-Product operation. It reduced a globally exponential operation to one which is locally exponential but globally polynomial. The polynomial depends on the number of cliques used to define the model and the complexity depends on the local cliques produced.

4 Graph Elimination

4.1 Graph Elimination on Undirected and Directed Graphs

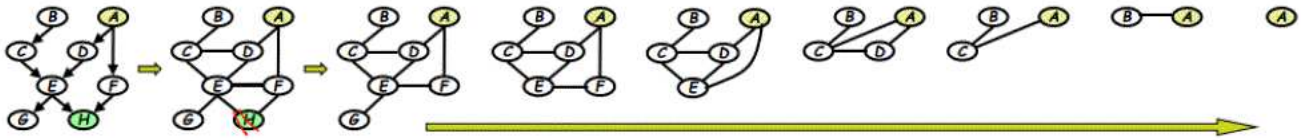


Figure 4: A Graph Elimination procedure

A graph elimination procedure for undirected graphs is as follows:

1. Begin with the graph and an elimination ordering.
2. For each variable in the ordering, connect all neighbors of the variable.
3. Eliminate the variable.

For a directed graph, first the graph is moralized. The remaining procedure stays the same as in undirected graphs. At each iteration, a sequence of subgraphs is obtained that includes the variable to be eliminated, remaining variables coupled with the variable to be eliminated and edges resultant from elimination. After each iteration, we record the elimination cliques of the graph, i.e., clique formed by the neighbors of the variable being eliminated and the variable itself. These cliques are equivalent to the sets of variables on which summations operate in probabilistic inference using Variable Elimination.
Graph Elimination presents two interesting intuitions:

1. The elimination ordering determines the intermediate terms being produced. Different subgraphs would be produced for different elimination orderings. The structure of the graph can give us intuition about how to structure our elimination ordering.

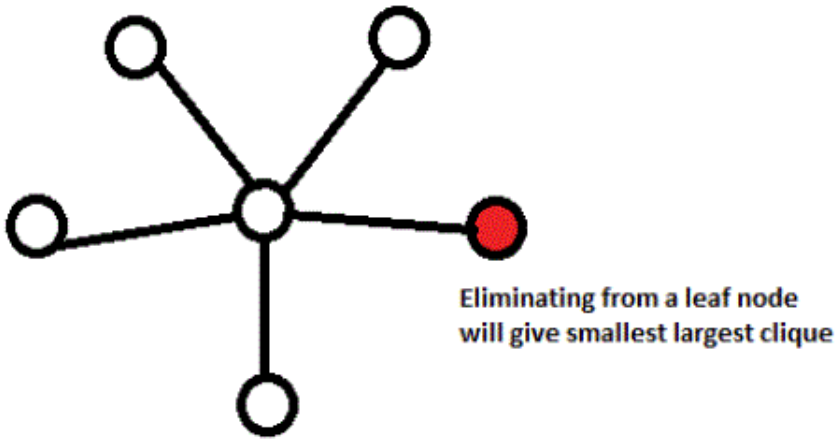


Figure 5: A Star Graph

2. Intermediate terms can be viewed as different messages being passed from clique being eliminated to the next clique to be eliminated.

4.2 Complexity of Graph Elimination

The complexity for a Graph Elimination on a graph is proportional to the size of the largest intermediate message (intermediate term). The size of the largest intermediate clique would be the bottleneck of the algorithm. This is called the *tree-width* of the graph. Essentially, the problem of finding the largest intermediate clique would be NP-hard. However, we can use intuition from the structure of the graph to find a good ordering that will result in smallest largest intermediate clique.

As an example, for a star graph, it is best to eliminate all the leaf nodes first. However, if we start from the middle node, it will result in one large clique consisting of all leaf nodes.

Another example is of tree graphs like the one shown in Figure 6. Starting from the leaf nodes is the best elimination ordering.

Let us now consider the Ising Model as shown in Figure 7. The complexity or treewidth of this model is equal to the number of nodes in a single dimension of the grid. Such models can be used to represent images. However, in such a case memory requirements are too big. For a binary 255×255 image, we need 2^{255} bits. Hence, graph elimination is not practical in this case.

4.3 Message Passing

A major limitation of the graph elimination algorithm is that the entire process needs to be carried for elimination of a each query variable. An important observation is that most of the intermediate sums or messages computed for calculating one marginal can be reused for other marginals. This leads us to the *Message Passing* algorithm.

The limitation of the Variable Elimination procedure is that it answers only one query, e.g., on one node. We would need to do a complete elimination for every such query. Variable elimination involves message passing along a clique tree, as can be seen in Figure 8.

Messages are re-used along the way. For example,

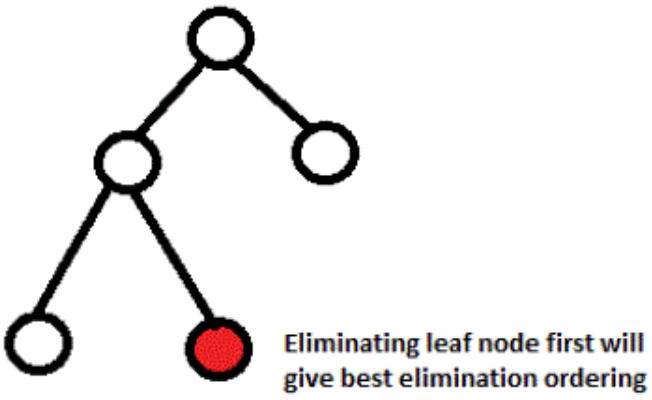


Figure 6: A Tree Graph

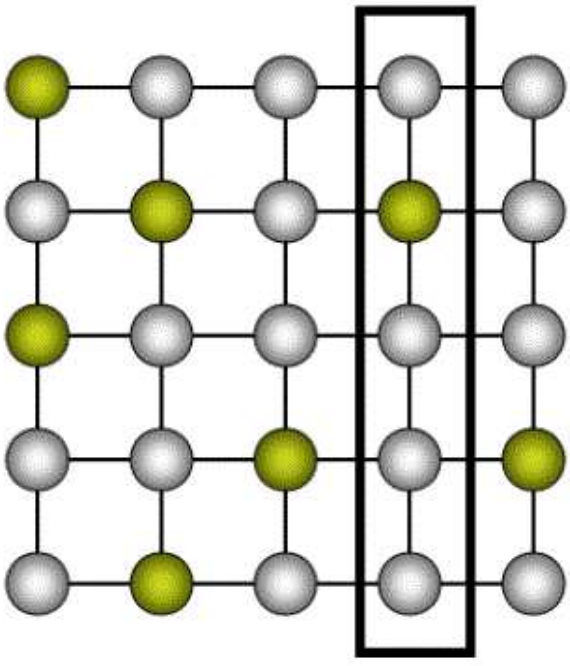


Figure 7: Graph Elimination on an Ising Model

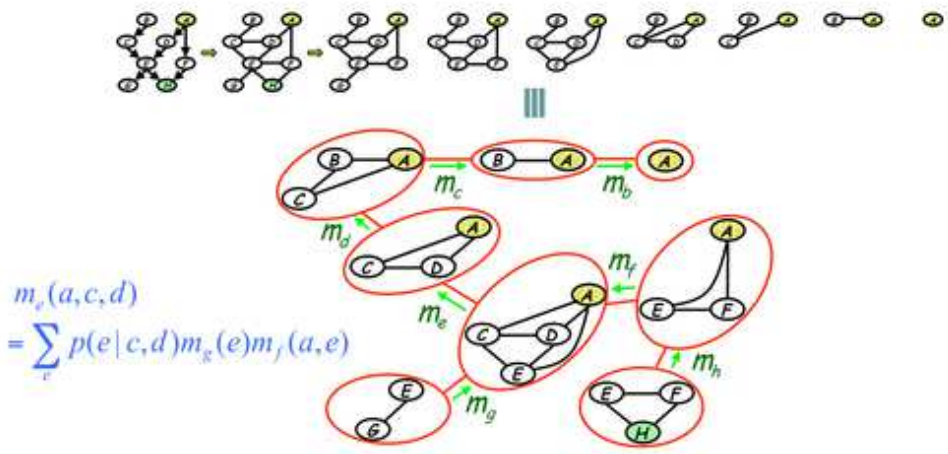


Figure 8: Message passing along the clique tree for the example discussed in lecture

$$m_e(a,c,d) = \sum_e p(e|c,d)m_g(e)m_f(a,e)$$

Now, suppose we had a query on variable G. Then, as explained in Figure 9, we would re-use messages m_f and m_h from Figure 8, but the other messages would have to be recomputed.

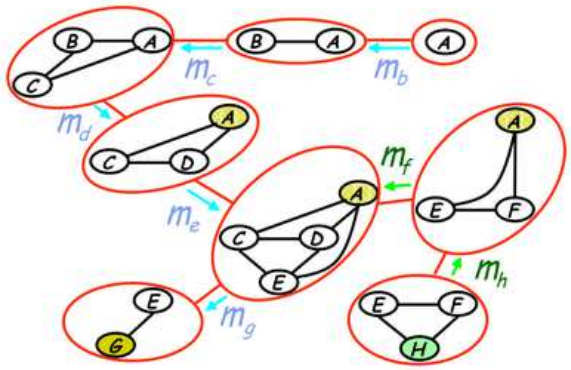


Figure 9: Message passing along the clique tree for a different query

5 Tree Graphical Models

The different types of trees are:

- Undirected trees have a unique path between any pair of nodes.
- In directed trees, all nodes except the root have exactly one parent.
- In a poly-tree, nodes can have multiple parents.

These are illustrated in Figure 10

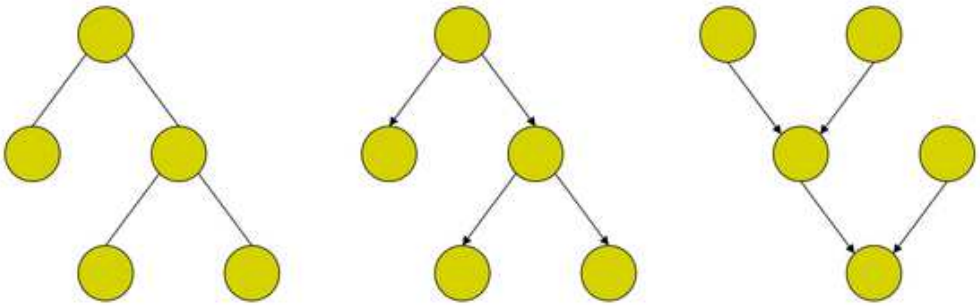


Figure 10: L-R: an undirected tree, a directed tree and a poly tree

5.1 Equivalence of directed and undirected trees

- Any undirected tree can be converted to a directed tree by choosing a root node and directing all edges away from it.
- A directed tree and the corresponding undirected tree make the same conditional independence assertions.
- Parameterizations are essentially the same.

$$\text{Undirected tree: } p(x) = \tfrac{1}{Z}(\textstyle\prod_{i\in V}\psi(x_i)\prod_{(i,j)\in E}\psi(x_i,x_j))$$
$$\text{Directed tree: } p(x) = p(x_r)\textstyle\prod_{(i,j)\in E}p(x_j|x_i)$$
$$\text{Equivalence: } \psi(x_r) = p(x_r);\ \psi(x_i,x_j) = p(x_j|x_i);\ Z = 1;\ \psi(x_i) = 1$$

5.2 Elimination on a tree

Consider the example tree shown in Figure 11. Let $m_{ji}(x_i)$ denote the factor resulting from eliminating variables from below up to i, which is a function of x_i:

$$m_{ji}(x_i) = \textstyle\sum_{x_j}(\psi(x_j)\psi(x_i,x_j)\prod_{k\in N(j)\setminus i}m_{kj}(x_j))$$

This is of the same form as a message sent from j to i during variable elimination. $m_{ij}(x_i)$ represnets a *belief* of x_i from x_j. In general,

$$p(x_f) \propto \psi(x_f)\textstyle\prod_{e\in N(f)}m_{ef}(x_f)$$

Elimination on trees is equivalent to message passing along tree branches.
The steps in ELIMINATION algorithm are:

- Choose an ordering Z in which query node f is the final node.
- Place all potentials on an active list.
- Eliminate node i by removing all potentials containing i, take sum/product over x_i.
- Place the resultant factor back on the list.

For a TREE graph, the steps are:

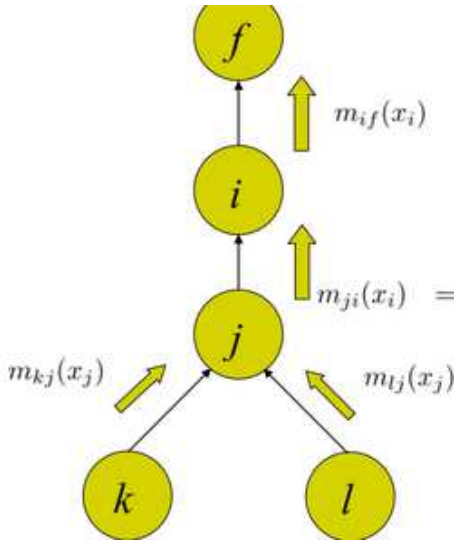


Figure 11: Elimination on a tree graph

- Choose query node f as the root of the tree.
- View tree as a directed tree with edges pointing towards leaves from f.
- Elimination ordering is based on depth-first traversal.
- Elimination of each node can be considered as message-passing (or Belief Propagation) directly along tree branches, rather than on some transformed graphs.

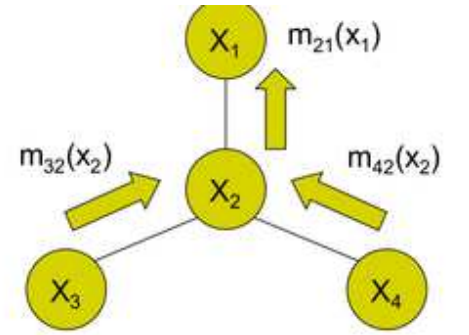
Thus, we can use the tree itself as a data-structure to do general inference.

5.3 The message passing protocol

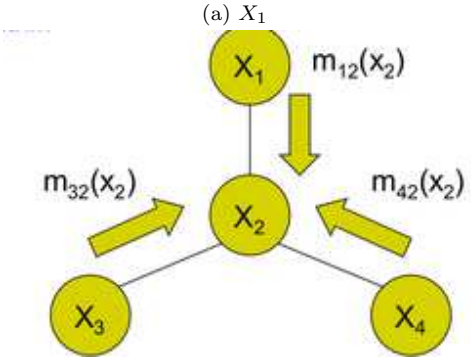
A node can send a message to its neighbors when (and only when) it has received messages from all its other neighbors.

To compute node marginals, the naive approach is to consider each node as the root and execute the message passing algorithm. Keeping the protocol in mind, Figure 12 show the messages that would need to be computed for finding the marginal distribution of three nodes in the tree.

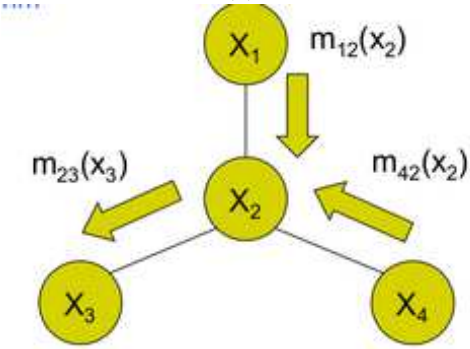
The complexity of this naive approach to computing all the node marginals is NC where N is the number of nodes in the tree and C is the complexity of a complete message passing. We now look at an alternative dynamic programming approach which consists of just 2 passes and hence, a complexity of $2C$ for computing all the node marginals. In this two-pass algorithm, we go up the tree (from leaves to root) and then down. For example, in Figure 13, we compute and send the messages in the following order: $m_{32}, m_{42}, m_{21}, m_{12}, m_{23}, m_{24}$.



(a) X_1



(b) X_2



(c) X_3

Figure 12: Messages that need to be computed for finding the marginal distribution of X_1, X_2 and X_3

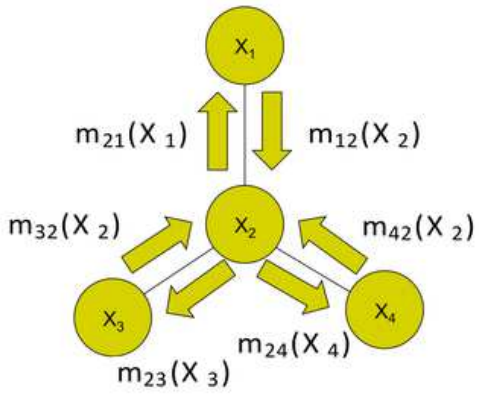


Figure 13: A two-pass algorithm for computing all node marginals on the same tree as before

6. Factor Graphs and Factor Trees

6.1 Factor Graph

Factor graphs aim at capturing factorizations. A factor graphs is a bipartite graph that represents the factorization for a joint probability of a directed or an undirected graph. Consider the Bayesian network given in Figure 14. For each marginal and conditional probability, we can define corresponding factors as $P(X_1) \equiv f_a(X_1)$ and add these factors in the original graph. These factors, corresponding to their respective nodes are connected with the neighbours of the corresponding nodes in the original graph. We can convert a directed Bayesian network to a Markov Random fields using factor graphs.

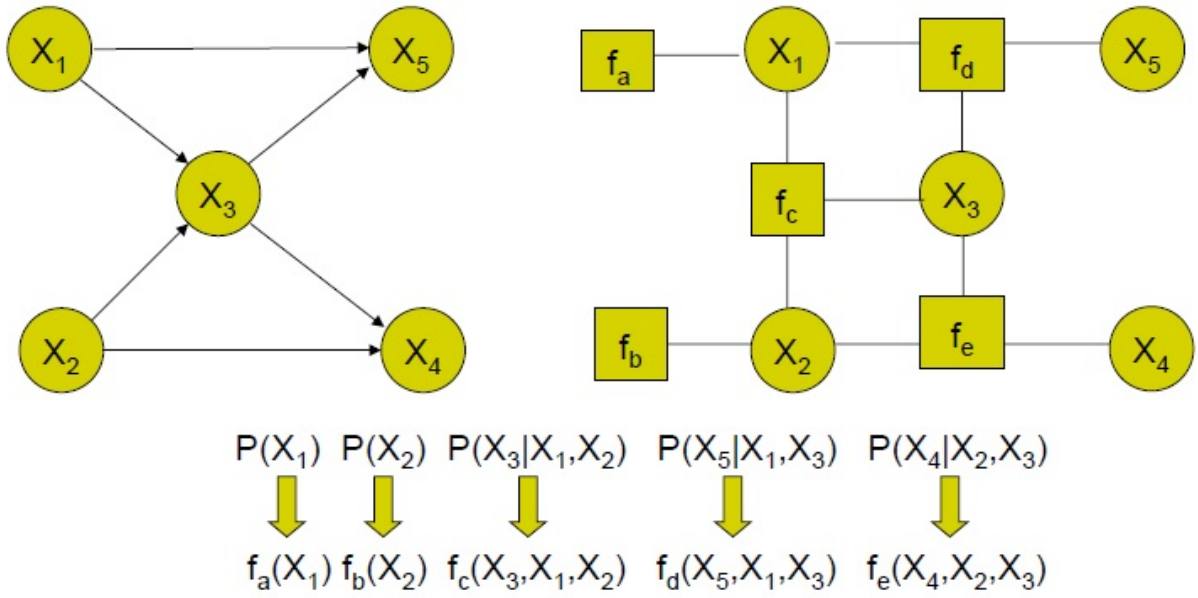


Figure 14: Bayesian network (left) to Factor Graph (right)

For an undirected graph, the corresponding factor graph is not unique. There can be multiple factor graphs representing the same Markov Random Field. Consider the example given in Figure 15, where the same undirected graph is represented with different factor graphs.The first factor graph contains three factors (f_a, f_b, f_c) while the second factor graph contains only one factor (f_a) placed centrally at the graph. Thus factor graphs can be used to represent a Bayesian network or a Markov Random Field in a tree-like structure, so that we can apply belief propagation algorithm to the factor graph.

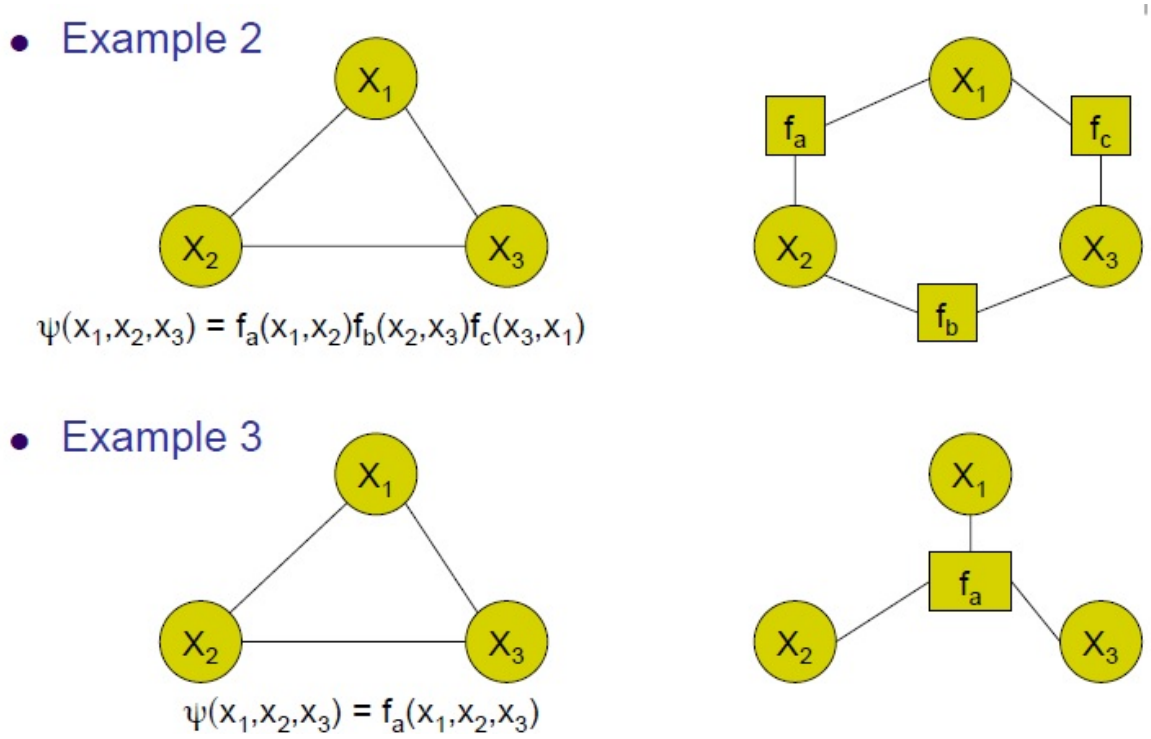


Figure 15: Different factor graphs for an undirected graph

6.2 Factor Trees

A factor graph is a factor tree if the undirected graph obtained by ignoring the distinction between variable nodes and factor nodes is an undirected tree. Please refer to Figure 15.

Message passing on a factor tree: We can use message passing algorithm to calculate the marginal probability of a variable. There are two types of messaging in a factor tree as follows:

- ν : From variables to factors
- μ : From factors to variables

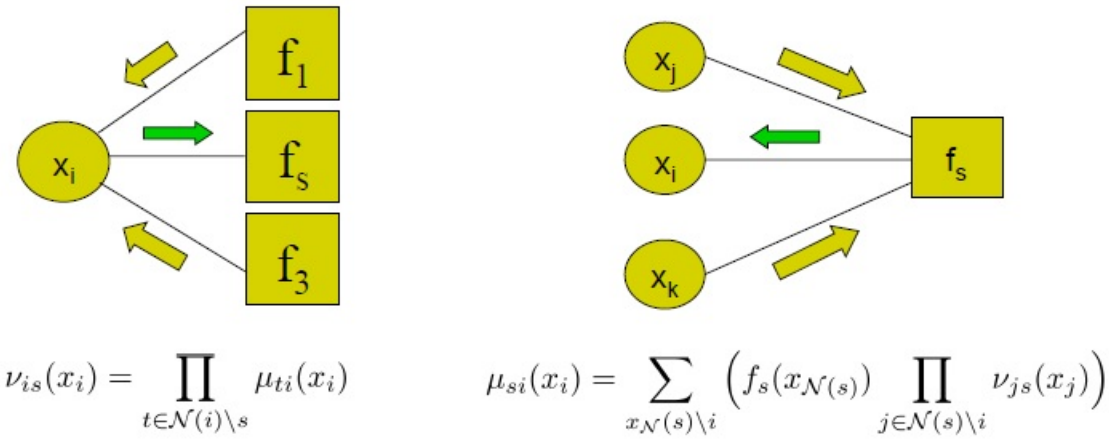


Figure 16: Message passing on a factor tree

In a factor tree, a node can send messages to neighbouring nodes only when it receives messages from other nodes. As you can see from Figure.16, the messages that variables send to factors is a product of all the messages that the variable receives from other factors. The messages that factors send to variables is a summation over product of all the messages it receives from other variables multiplied by the weight of the factor. The factor tree methodology can be used to calculate the marginal probabilities of the variables by converting a graph into a factor tree. The marginal probability is proportional to the product of all the messages it receives and it is proportional to the product of the message it sends to a factor and it receives from a factor.

We cannot use message passing on a non-tree graph because the correctness of belief propagation cannot be guaranteed. The factor tree technique is useful for converting tree-like graphs and poly-trees to factor trees which can be used to derive marginal probabilities using message passing algorithm.

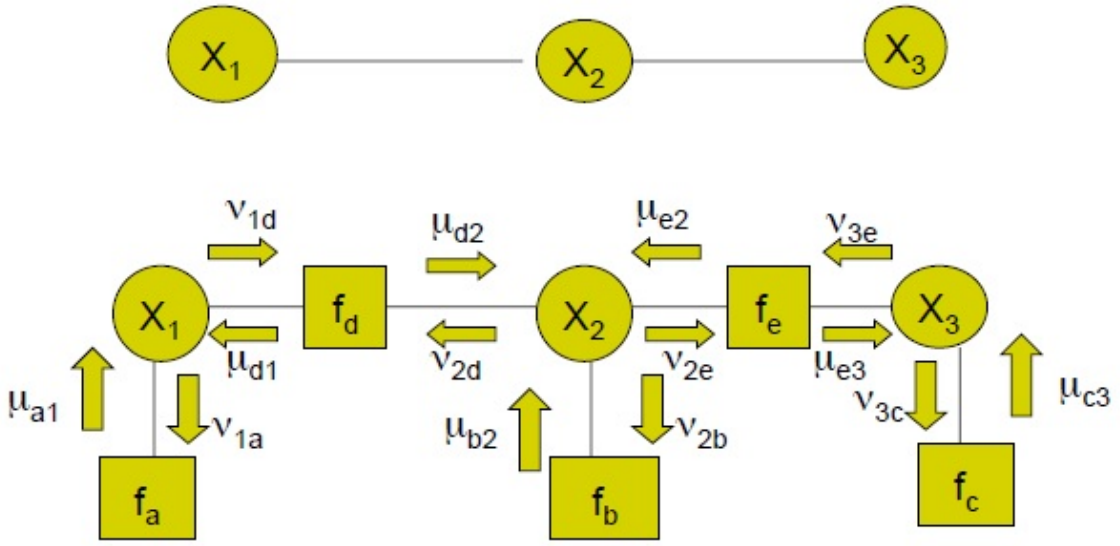


Figure 17. BP on a factor tree

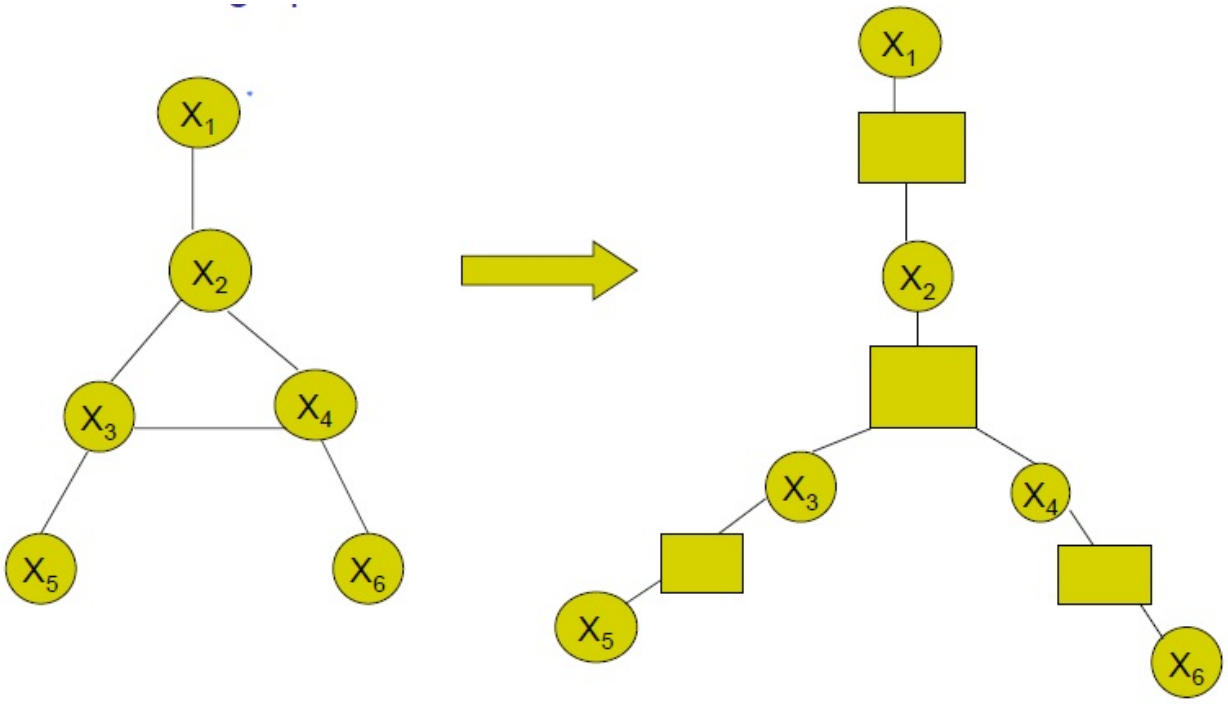


Figure 18. Tree-like graphs to factor trees

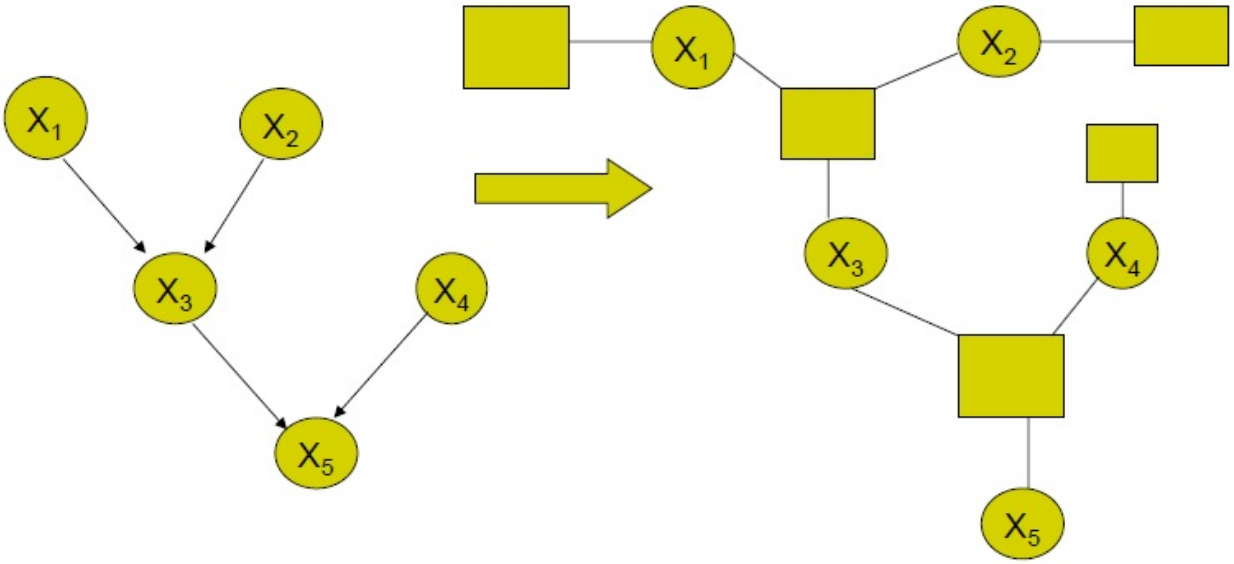


Figure 19. Poly-trees to factor trees

7. Max-Product algorithm :

Max- Product algorithm is used for computing maximum a posteriori probability of the nodes in a graph. By making little modifications to the message passing algorithm, we can find most likely configuration of all variables in a graphical model.

Following modifications are to be made:

- Instead of summing over all possible values of a variable like in the Sum-Product algorithm, find the variable that maximizes the message it sends.
- Instead of running the algorithm from leaf node, run it from the root node.

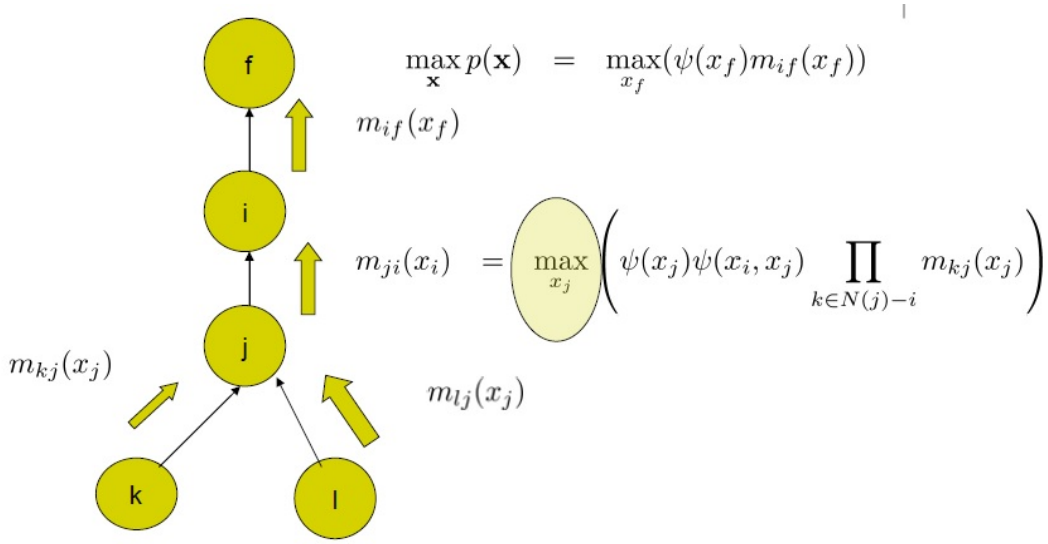


Figure 20. Step 1 of Max-product algorithm

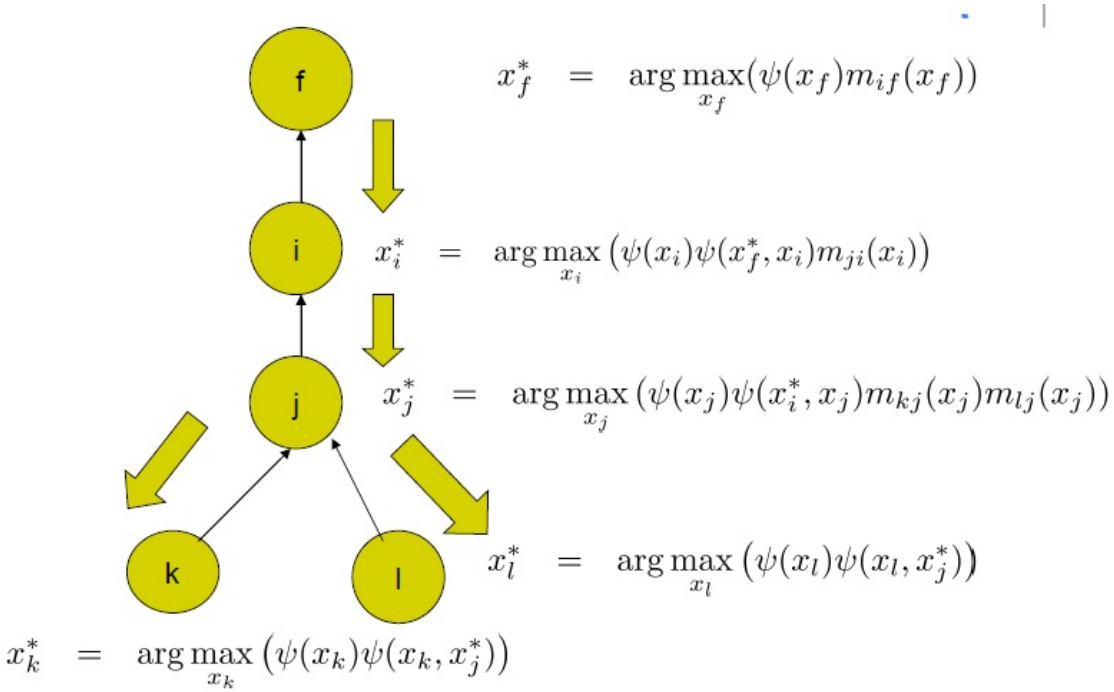


Figure 21.Step 2 of Max-Product Algorithm