MAC6916 PROBABILISTIC GRAPHICAL MODELS
LECTURE 7: THE SUM-PRODUCT ALGORITHM

DENIS D. MAUÁ

1. Introduction

As we have seen, variable elimination is sound but innefficient in networks with high treewidth. Kwisthout et al. [1] showed that if the sum-product problem can be solved in subexponential time in the treewidth, then the propositional logic satisfiability problem (SAT) can also solved in subexponential time in the number of propositions. The latter is *believed* to not be true, implying that the former is probably also not true. Thus, exact inference algorithms with polynonial-time behavior in high treewidth models are very unlikely to exist. This motivates the study of approximate algorithms. We have already seen sampling algorithms which provide approximate solutions (in polynomial time). In this lecture, we will look at another approximate algorithm, called the sum-product algorithm, which is a syntatic variant of the famous *loopy belief propagation* algorithm for approximate inference in Bayesian networks.

loopy belief propagation

To recap, given potentials $\phi_1, \dots, \phi_m$ over a set of variables $\mathcal{V}$ and a set $\mathcal{X} \subseteq \mathcal{V}$, we wish to compute

$$\sum_{\mathcal{V}-\mathcal{X}} \prod_j \phi_j \, . \qquad \text{[sum-product problem]}$$

We will assume that the set $\mathcal{X}$ is either a single variable or the empty set (which is the case when we compute belief updating or the partition function).

Loopy belief propagation in Bayesian networks was first proposed by Pearl as a direct generalization of its belief propagation algorithm that performs exact inference in tree-shaped Bayesian networks. Pearl warned that the algorithm was not guaranteed to converge, and did not recommend its use. A different guise of the algorithm was independently developed in the statistical mechanics community and used for image processing tasks due to its relative efficiency and accuracy (compared to sampling) in models with thousands of variables. Later, it was realized that turbo coder, one of the most efficient error-correcting algorithms used in communication theory, could be interpreted as the sum-product algorithm in a particular Markov network [2]. This revived the theoretical interest in investigating the properties of the sum-product algorithm. Yedidia et al. [3, 4] showed that the algorithm can be cast as an optimization problem, where the objective is to find the tree-shaped network that minimizes the KL-divergence with respect to the model distribution constrained so that the node and edge marginals in each model agree. This led to a deeper understanding of the convergence of the algorithm, to a

Date: April 5, 2016.

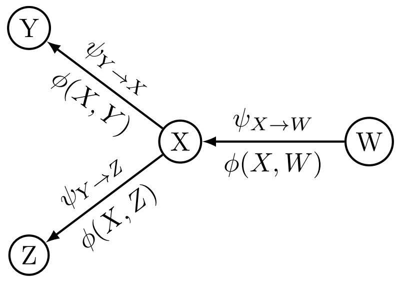


FIGURE 1. Illustration of variable elimination in trees.

connection between the algorithm and Bethe free energy minimization, an old and well-studied problem in statistical mechanics, and to the development of convergent variants such as the tree-reweighted algorithm.

2. Variable Elimination in Markov Trees

Variable elimination takes a very peculiar form in tree-shaped Markov networks. Recall that the distribution of a factored Markov tree has the form

$$p(\mathcal{V}) = \frac{1}{Z} \prod_{X-Y\in\mathcal{E}} \phi(X,Y)\,,$$

rooted tree

where $\mathcal{E}$ is the set of edges of the tree. We can transform any tree into a *rooted tree*, which is a directed tree where the arcs are pointed away from the root (the node without incoming arcs). As usual, we refer to the neighbors of a node in a tree as the parent (the one that is closer to the root) and children. If $T_R = (\mathcal{V}, \mathcal{E})$ is tree rooted at $R \in \mathcal{V}$, we define the *subtree* rooted at S as the subgraph $T_S = (\{S\} \cup \text{de}(S), \mathcal{E}_S)$ induced by S and its descendants T_R.

subtree

Proposition 1. *Let $X_n, \ldots, X_1$ be a topological ordering of the nodes in the tree rooted at X_n. Then $X_1, \ldots, X_n$ is a perfect elimination ordering.*

Proof. By the time a variable X_i is eliminated, all its children have been eliminated, so that its single neighbor is its parent, making X_i simplicial. □

This allows us to re-define variable elimination in trees as follows. Let $X_n, \ldots, X_1$ be a topological ordering of the nodes. Compute for $i = 1$ to $i = n$

$$\psi_i(\text{pa}(X_i)) = \Big[\phi(X_i, \text{pa}(X_i)) \times \psi_{i_1}(X_i) \times \cdots \times \psi_{i_k}(X_i)\Big] - X_i\,,$$

where $\text{ch}(X_i) = \{X_{i_1}, \ldots, X_{i_k}\}$. Since $\text{pa}(X_n) = \emptyset$, it follows that $\psi_n = Z$ (as all variables are eliminated). The marginal distribution of X_n can be obtained as $\psi_{i_1} \times \cdots \times \psi_{i_k}/Z_n$, where Z_n is a normalization constant. The choice of X_n as the root of the tree is arbitrary, and the algorithm computes the partition function for any choice of root. This allows us to re-formulate the algorithm as follows:

(1) Initialize $\psi^{(0)}_{i\to j}(X_j) = 1$, and $t \leftarrow 1$
(2) Repeat until convergence:
    (a) For $i \leftarrow 1$ to $i \leftarrow n$ compute

$$\psi^{(t)}_{i\to j}(X_j) = \Big[\phi(X_i, X_j) \times \psi^{(t-1)}_{i_1\to i}(X_i) \times \cdots \times \psi^{(t-1)}_{i_r\to i}(X_i)\Big] - X_i$$

for each $X_j \in \text{ch}(X_i)$, where $\text{ch}(X_i) = \{X_{i_1}, \ldots, X_{i_r}\} \cup \{X_j\}$.

(b) Update $t \leftarrow t+1$

We call the potential $\psi_i^{(t)}(X_j)$ the t-th message from X_i to X_j, and the previous algorithm *synchronous message-passing*. Convergence is verified by computing $\max_{i,j} |\psi_{i\to j}^{(t)} - \psi_{i\to j}^{(t-1)}|$.

message-passing

Before giving a formal proof of convergence, let us analyze a few iterations of the algorithm. After the first iteration of Step (2), every leaf has sent a message to its single neighbor, as it would have in variable elimination with a consistent reverse topological ordering. This message remains the same in every subsequent iteration (so that leaf nodes converge after one iteration). Now consider a node X_i where all but one of its neighbors are leaves (call the non-leaf neighbor X_j). That is, there is at most one path from X_i to a leaf with length greater than one. Then, after 2 iterations, this node has sent the same message to its non-leaf neighbor as it would have had we been performing variable elimination with a consistent reverse topological ordering. Moreover, since the messages X_i receives from its leaf neighbors remain unaltered in future iterations, also the message from X_i to X_j remains unaltered for $t \geq 2$ (so that X_i converges after 2 iterations). In general, message-passing can be seen as a parallel simulation of variable elimination with many perfect elimination orderings and with memory (message) sharing. This enables a more efficient computation of all the marginals of the network (compared to multiple runs of variable elimination).

The formal analysis of convergence uses the height of rooted trees: The *height of a rooted tree* is the length of the longest path from the root to a leaf. The height of a tree containing a single node is zero. Note that the height of a subtree is always smaller than that of the tree.

tree height

Lemma 1. *Root the tree and let X_i be an arbitrary node and $X_j = pa(X_i)$. Then $\psi_{i\to k}^{(t)} = \psi_{i\to k}^{(h+1)}$ for every $t > h$, where h is the height of the subtree rooted at X_i.*

Proof. We use an inductive argument in the height of the tree. Let X_i be a leaf (hence $h = 0$). After one iteration this node has sent a message to its single neighbor (its parent), as it would have in variable elimination with any consistent reverse topological ordering. This message remains the same in every subsequent iteration. Now consider a node X_i with height h, and assume that the result holds for every subtree rooted at one of its children. Then every message $\psi_{j\to i}^{(t)}$ sent by a child of X_i are unaltered for $t > h-1$. But this implies that the message from X_i to its parent X_j is the same for every $t > h$. $\square$

We thus have that:

Theorem 1. *Synchronous message-passing converges in tree-shaped graphs after ρ iterations, where ρ is the diameter of the tree (the size of the longest path). Moreover, $Z = \prod_{X_j \in ne(X_i)} \psi_{j\to i}^{(\rho)}(X_i) - X_i$ and $p(X_i) \propto \prod_{X_j \in ne(X_i)} \psi_{j\to i}^{(\rho)}(X_i)$ for any node X_i.*

Proof sketch. Root the node at a node X_i such that the height of the rooted tree is maximized. This height equals the diameter of the tree, and by the previous lemma, the messages converge after $t > \rho$ iterations. Fix a reverse topological ordering $X_1, \ldots, X_n$. Then the messages $\psi_{X_i \to \mathrm{pa}(X_i)}^{(i)}$ are the same potentials generated during variable elimination with that elimination ordering. $\square$

2.1. **Asynchronous Message-Passing.** Message-passing can also be performed asynchronously. Moreover, we can also normalize the messages (this increases numerical accuracy):

(1) Initialize $\psi_{i \to j}(X_j) = 1/r_j$
(2) Repeat until convergence:
   (a) For $i \leftarrow 1$ to $i \leftarrow n$ compute

$$\psi_{i \to j}(X_j) \propto \Big[\phi(X_i, X_j) \times \psi_{i_1 \to i}(X_i) \times \cdots \times \psi_{i_r \to i}(X_i)\Big] - X_i$$

   for each $X_j \in \text{ch}(X_i)$, where $\text{ch}(X_i) = \{X_{i_1}, \ldots, X_{i_r}\} \cup \{X_j\}$.

The constant $r_j = |\text{dom}(X_j)|$ denotes the cardinality of variable X_j. The messages are normalized by dividing $\psi_{i \to j}/(\psi_{i \to j} - X_j)$. It can be shown that the asynchronous version also terminates after a finite number of steps and produces estimates of the marginal probabilities and the partition function. Moreover, since the asynchronous version updates messages more often than the synchronous version, the former converges faster. Note that the normalization can also be performed to the synchronous version.

3. Sum-Product in Factor Graphs

The (synchronous or asynchronous) message-passing algorithm can be immediately applied on loopy pairwise Markov networks. However, in loopy networks, the algorithm might not converge and a bound on the number of iterations must be imposed. Even when the algorithm does converge, there is no guarantee on the quality of the estimates induced from the messages. In fact, the algorithm can be understood as implicitly assuming a larger set of independences, which results in either under or overestimation of probability marginals. The accuracy of the algorithm need not to increase with time (in fact, it is common that accuracy oscillates as the number of iterations grows). To further extend the algorithm to arbitrary (non-pairwise) Markov networks, we need to introduce an auxiliary representation.

factor graph

Definition 1. The *factor graph* of a set of factors $\mathcal{F} = \{\phi_1, \ldots, \phi_m\}$ is the bipartite undirected graph $G = (\mathcal{V} \cup \mathcal{F}, \mathcal{E})$ where $\mathcal{V} = \cup_j \text{sc}(\phi_j)$ is the set of variables and there is an edge $X - \phi_j$ in $\mathcal{E}$ iff $X \in \text{sc}(\phi_j)$. The graph induces a probability distribution through

$$p(\mathcal{V}) = \frac{1}{Z} \prod_{j=1}^{m} \phi_m \,.$$

Figure 2 depicts an example of a simple factor graph. Note that a factor graph is a more detailed description of the distribution than a Markov network, since the latter makes no (graphical) distinction between the parametrization of the model (i.e., whether the potentials are over maximal or non-maximal cliques). Thus, while a pairwise Markov network over a complete graph and a Markov network with single factor have the same graphical representation, they have very different factor graph representations.

Even though the factor graph is defined as an undirected graph, it is convenient to represent it as a bi-directed graph (i.e., each edge is represented by two symmetric arcs). Let $\phi_1, \ldots, \phi_r$ denote the neighbors of a variable X in the graph, and $X_1, \ldots, X_s$ be the neighbors of a factor ϕ. Message-passing in sum-product graphs is performed by applying the following *sum-product rules* to each (directed) arc:

sum-product rules

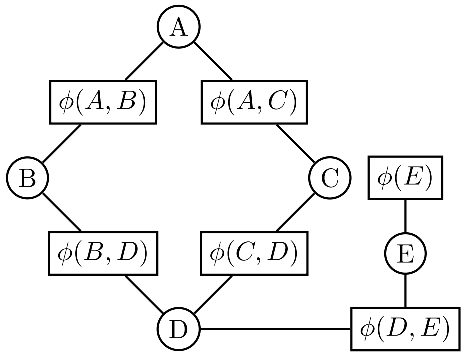
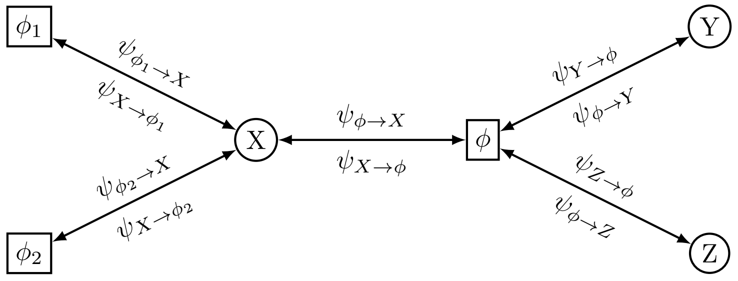


FIGURE 2. Factor graph of the potentials $\phi(A,B)$, $\phi(A,C)$, $\phi(B,D)$, $\phi(C,D)$, $\phi(D,E)$, $\phi(E)$.



$$\psi_{X\to\phi}(X) = \psi_{\phi_1\to X}(X) \times \psi_{\phi_2\to X}(X)$$
$$\psi_{\phi\to X}(X) = \big(\phi(X,Y,Z) \times \psi_{Y\to\phi}(Y) \times \psi_{Z\to\phi}(Z)\big) - \{Y,Z\}$$

FIGURE 3. Representation of the sum-product rules in factor graphs.

$$\begin{aligned}
\psi_{X\to\phi_j}(X) &= \psi_{\phi_1\to X} \times \cdots \times \psi_{\phi_{j-1}\to X} \times \psi_{\phi_{j+1}\to X} \times \cdots \times \psi_{\phi_r\to X} && \text{[variable to factor]}\\
\psi_{\to\phi}(X_j) &= \psi_{X_1\to\phi} \times \cdots \psi_{X_{j-1}\to\phi} \times \psi_{X_{j+1}\to\phi} \times \cdots \times \psi_{X_s\to\phi} &&\\
\psi_{\phi\to X_j}(X_j) &= (\phi \times \psi_{\to\phi}) - \{X_1,\ldots,X_{j-1},X_{j+1},\ldots,X_s\} && \text{[factor to variable]}
\end{aligned}$$

The sum-product rules messages are graphically represented in Figure 3.

The messages are initialized with ones and often updated asynchronously (although synchronous update is possible). The algorithm terminates when messages converge or after a maximum number of iterations is reached.

3.1. Marginals.

The node and clique marginal distributions can be obtained from the respective incoming messages at any iteration by

$$\begin{aligned}
p(X) &\propto \psi_{\phi_1\to X} \times \cdots \times \psi_{\phi_r\to X}\,, && \text{[node marginal]}\\
p(\mathcal{C}) &\propto \phi(\mathcal{C}) \times \psi_{X_1\to\phi} \times \cdots \times \psi_{X_s\to\phi}\,, && \text{[clique marginal]}
\end{aligned}$$

where $\phi_1,\ldots,\phi_r$ are the neighbors of X in the factor graph and $X_1,\ldots,X_s$ are the neighbors of $\phi(\mathcal{C})$. Note that these estimated need no to converge and can oscillate arbitrarily from iteration to iteration.

3.2. Complexity.

Sending a message from a potential ϕ to a variable X requires combining $(d-1)$ potentials and eliminating $d-1$ variables, where $d = |\text{ne}(\phi)| =$

$|\mathrm{sc}(\phi)|$. Thus, the cost of the computation is $O(d \cdot c^d)$, where c is the maximum variable cardinality. Sending a message from X to ϕ requires combining $d' - 1$ potentials with scope $\{X\}$ taking time $O(d' \cdot c)$, where d' is the number of neighbors of X in the associated domain graph of the model. Hence, the time is dominated by the complexity of sending messages from potential nodes. At each iteration, two messages are sent on each edge. Thus, if there are n variables and m potentials (factors), an iteration takes time $O((m + n) \cdot c^d)$, where d is the size of the largest scope of a potential in the input. The overall running time is $O((m+n) \cdot c^2 \cdot T)$, where T is the maximum number of iterations. Hence, the algorithm takes polynomial time in the size of the input (the number of variables, potentials and variable cardinalities), independently of the treewidth of the domain graph.

3.3. **Convergence.** It is widely *believed* that the sum-product algorithm is more accurate when graph has few short loops (long loops are less problematic); more accurate when $\phi(X_i, X_j) = \phi(X_i)\phi(X_j) + \phi'(X_i, X_j)$ with $|\phi'(X_i, X_j)| < \epsilon$, small ϵ. The convergence of the algorithm can be investigated as an non-convex optimization problem where one tries to find a Markov tree approximation to any given model. It can be shown that the fixed points of the sum-product algorithm correspond to stationary points of the optimization [9].

4. Reading

Read the article [5].

5. Exercises

No exercises this week.

6. Assignment

- Implement approximate inference algorithms by sampling and message passing (logical sampling, likelihood weighting, Gibbs sampling, synchronous and asynchronous sum-product).
- Evaluate each algorithm's performance on the provided set of Bayesian networks (generate evidence test cases). Discuss convergence and accuracy (Use variable elimination to compute the correct values).
- Write report describing implementation and discussing empirical analysis.

Deadline: April, 25.

References

[1] J.H.P. Kwisthout, A.H.L. Bodlaender and L.C. van der Gaag. The Necessity of Bounded Treewidth for Efficient Inference in Bayesian Networks. In *Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pp. 237-242, 2010.

[2] R. McEliece, D. MacKay and J. Cheng. Turbo decoding as an instance of Pearl's 'belief propagation' algorithm. IEEE Journal on Selected Areas in Comm., volume 16, issue 2, pp. 140–152, 1998.

[3] J.S. Yedidia, W.T. Freeman, and Y. Weiss. Generalized belief propagation. In Advances in Neuroprocessing Systems 13, pp. 689–695, 2001.

[4] J.S. Yedidia, W.T. Freeman and Y. Weiss. Constructing Free-Energy Approximations and Generalized Belief Propagation Algorithms. IEEE Transactions on Information Theory, volume 51, issue 7, pp. 2282–2312, 2005.

[5] F.R. Kschischang, B.J. Frey and H.-A. Loeliger. Factor graphs and the sum-product algorithm IEEE Transactions on Information Theory, volume 47, issue 2, pp. 498–519, 2001.

[6] A. Darwiche. Modeling and reasoning with Bayesian networks. Cambridge University Press, 2009.

[7] D. Koller and N. Friedman. Probabilistic Graphical Models MIT Press, 2009.

[8] P. Shenoy and G. Shafer. Axioms for probability and belief-function propagation. In *Proceedings of the 4th Conference on Uncertainty in Artificial Intelligence*, pp. 169–198, 1988.

[9] T. Heskes Stable Fixed Points of Loopy Belief Propagation Are Minima of the Bethe Free Energy. In *Advances in Neural Information Processing Systems 15*, pp. 359–366, 2002.