Graphical Models Reading Group
Week 1

Nicholas Ruozzi

1 Inference and Graphical Models

Historically, the study of graphical models was developed to perform statistical inference. Let $p(x_1, ..., x_n)$ be a probability distribution. Because of their importance in a wide variety of applications, two inference problems are typically studied in this context: the maximum a posteriori (MAP) estimation problem and the computation of marginal probabilities.

A marginal of a probability distribution p is the function obtained by fixing the value of one of the random variables and summing out over the remaining variables. For example, the marginal function, $p_i(x_i)$, corresponding to the i^{th} variable is given by:

$$p_i(x_i) = \sum_{x' \text{ st. } x'_i = x_i} p(x'_1, ..., x'_n) \tag{1}$$

Computing the marginals of a given probability distribution is, in general, an expensive operation. However, when the probability distribution has some additional structure, we may be able to compute the marginals much faster.

Example 1.1 (Efficient marginalization).
Suppose $p(x_1, x_2, x_3)$ is a probability distribution over three random variables that take values in the set $\{1, ..., k\}$. Further, suppose that p can be written as a product of functions as follows:

$$p(x_1, x_2, x_3) = q_{12}(x_1, x_2) q_{13}(x_1, x_3) \tag{2}$$

Now, consider computing the marginal of p for the variable x_1:

$$p_1(x_1) = \sum_{x_2} \sum_{x_3} p(x_1, x_2, x_3) \tag{3}$$

As x_2 and x_3 can each take one of k different values, this summation contains k^2 distinct terms. However, if we exploit the observation that p can be written as a product, we can rewrite the summation as

$$p_1(x_1) = \sum_{x_2} \sum_{x_3} q_{12}(x_1, x_2) q_{13}(x_1, x_3) \tag{4}$$

$$= \sum_{x_2} q_{12}(x_1, x_2) \Big[\sum_{x_3} q_{13}(x_1, x_3) \Big] \tag{5}$$

which only requires summing $2k$ distinct terms.

As an alternative to computing the marginals, we may also be interested in computing the most likely configuration of our joint probability distribution. This problem is typically referred to as maximum a posteriori estimation, MAP estimation for short, and is a solution to the following optimization problem:

$$x^* \in \arg\max_x p(x_1, ..., x_n) \tag{6}$$

Again, notice that, like summation, computing the maximum can be computationally expensive, but if the probability distribution can be written as a product of smaller functions, then we may be able to compute the maximum more efficiently. A similar idea can be applied over any semiring [1] (e.g. sum and product, max and product, min and sum, etc.).

Graphical models describe the relationship between the pieces of the decomposition, called potentials, of a particular probability distribution. When the graphical model has a certain structure, then we can efficiently compute the marginals and the MAP estimate.

2 Graphical Models

Let $f : \prod_i \mathcal{X}_i \to \mathbb{R} \cup \{\infty, -\infty\}$, where each $\mathcal{X}_i$ is an arbitrary set (e.g. $\mathbb{R}$, $\{0, 1\}$, $\mathbb{Z}$, etc.). We will in general allow f to take the value ∞ over its domain. However, as we will see later, some results will only apply when f is a real-valued function (i.e. f does not take the value ∞ or $-\infty$ for any element in its domain).

Often, we will be interested in finding an element $(x_1, ..., x_n) \in \prod_i \mathcal{X}_i$ that maximizes/minimizes f, and as such, we will assume that there is such an element. For an arbitrary function, computing this maximum/minimum may be computationally expensive, especially if n is large. A typical scientific application may involve hundreds of thousands of variables and potential functions and storing the entire problem on one computer may be difficult, if not impossible. In other applications, such as sensor networks, processing power and storage are limited. Because local message passing algorithms are decentralized and distributed, they can operate on scales at which typical algorithms would be impractical.

2.1 Dynamic Programming

As a motivation for studying the general problem, we will begin by examining the connection between message passing algorithms and more standard notions in computer science.

A common problem solving strategy in computer science is to break apart large problems into smaller subproblems, solve the subproblems, and then combine the solutions of the subproblems into a solution to the larger problem. To solve each subproblem, we can recursively employ a similar strategy. This style of problem solving is typically referred to as *divide-and-conquer*.

Closely related to divide-and-conquer is the notion of *dynamic programming*. In dynamic programming, we start with an optimization problem that can expressed as an optimization over subproblems via some recurrence. Naively computing the recurrence can be costly; we may end up solving many of the same subproblems over and over again. Instead, we employ a bottom up strategy: we create a table containing the solutions to all of the subproblems by solving the smallest subproblems first, recording their solution in the table, and then using these solutions to compute the solutions to the next smallest subproblems. Local message passing algorithms such as belief propagation and min-sum are, in many ways, a natural by-product of the same problem solving strategies underlying dynamic programming. This connection is best illustrated by example:

Example 2.1 (Maximum weight independent set on a tree).
Let $G = (V, E)$ be a graph with associated weights w_i for each node $i \in V$. A set, $S \subseteq V$, of the vertices forms an independent set if no two vertices in the subset are joined by an edge in E. The weight of a set S, written $w(S)$, is the sum of the weights of all of the vertices in S. The maximum weight independent set problem is to find an independent set in G of maximum weight.

Computing the maximum weight independent set for an arbitrary graph is known to be NP-hard. However, in the special case that the graph G is a tree, the problem can be solved in polynomial time using standard dynamic programming techniques. To see this, let T be a tree rooted at a node r. Define the function $\text{mwis}(i, \text{inSet})$ to be the value of the maximum weight independent set in the subtree rooted at i formed by the descendants of node i where inSet is either zero or one to indicate whether or not node i should be taken as part of the independent set. The function mwis is defined

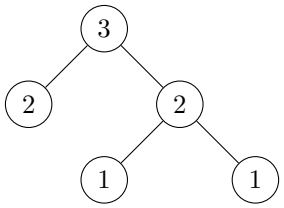

Figure 1: Example of the maximum weight independent set problem on a tree. The number inside of each node corresponds to the weight of that node.

via the following recursion:

$$\text{mwis}(i,1) = w_i + \sum_{c \text{ child of } i} \text{mwis}(c,0) \tag{7}$$

$$\text{mwis}(i,0) = \sum_{c \text{ child of } i} \max\{\text{mwis}(c,0), \text{mwis}(c,1)\} \tag{8}$$

We have that $w(S^*) = \max\{\text{mwis}(r,0), \text{mwis}(r,1)\}$ for any maximum weight independent set, S^*, on T. Notice that $\text{mwis}(i,1)$ and $\text{mwis}(i,0)$ only depend on the value of mwis for the children of node i. To see how we might turn this into a distributed algorithm, suppose we could perform computations at each node of the graph. We can have each leaf node l compute $\text{mwis}(l,1)$ and $\text{mwis}(l,0)$ and pass these results to their parents. The nodes directly above the leaf nodes can then perform a similar computation and pass the results to their parents and so on. Formally, let $p(i)$ be the parent of node i in the tree, and define a vector of messages, m, by setting $m_{i\to p(i)}(1) \equiv \text{mwis}(i,1)$ and $m_{i\to p(i)}(0) \equiv \text{mwis}(i,0)$. From this formulation, we can see that each node other than r needs to pass two messages to its parent. We then have that the maximum weight independent set has weight

$$\max\{w_r + \sum_{c \text{ child of } r} m_{c\to r}(0), \sum_{c \text{ child of } r} \max_{x_c \in \{0,1\}} m_{c\to r}(x_c)\}. \tag{9}$$

The algorithm described in Example 2.1 exactly computes the maximum weight independent set on a tree. We would like to answer the following question: if the graph is not a tree, under what conditions can we use a similar message passing strategy to find the maximum weight independent set?

2.2 Factorizations and Factor Graphs

The basic observation of the max-sum algorithm is that, even though the original maximization problem may be difficult, if f can be written as a sum of functions depending on only a small subset of the variables, then we may be able to maximize the global function by performing a series of maximizations over (presumably easier) sub-problems. To make this concrete, let $\mathcal{A} \subseteq 2^{\{1,\ldots,n\}}$. We say that f *factorizes* over $\mathcal{A}$ if we can write f as a sum (or product depending on the application) of real valued potential functions $\phi_i : \mathcal{X}_i \to \mathbb{R} \cup \{\infty, -\infty\}$ and $\psi_\alpha : \mathcal{X}_\alpha \to \mathbb{R} \cup \{\infty, -\infty\}$ as follows:

$$f(x) = \sum_{i=1}^{n} \phi_i(x_i) + \sum_{\alpha \in \mathcal{A}} \psi_\alpha(x_\alpha) \tag{10}$$

This factorization is by no means unique. For example, suppose we are given the objective function $f(x_1, x_2) = x_1 + x_2 + x_1 x_2$. There are many different ways that we can factorize f:

$$f(x_1,x_2) = x_1 + x_2 + x_1x_2 \tag{11}$$

$$= x_1 + (x_2 + x_1x_2) \tag{12}$$

$$= (x_1 + x_2 + x_1x_2) \tag{13}$$

$$= x_1 + x_2 + \frac{x_1x_2}{2} + \frac{x_1x_2}{2} \tag{14}$$

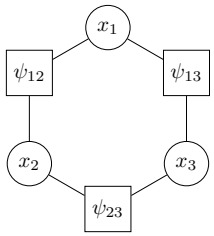

Figure 2: Factor graph corresponding to $f(x_1, x_2, x_3) = \phi_1 + \phi_2 + \phi_3 + \psi_{12} + \psi_{23} + \psi_{13}$. By convention, variable nodes are represented as circles and factor nodes are represented as squares.

Each of these rewritings represents a different factorization of f (the parenthesis indicate a single potential function). All of these factorizations can be captured by the above definitions, except for the last. Recall that $\mathcal{A}$ was taken to be a subset of $2^{\{1,\ldots,n\}}$. In order to accommodate the factorization given by equation 14, we will allow $\mathcal{A}$ to be a multiset whose elements are members of the set $2^{\{1,\ldots,n\}}$.

The set of all factorizations of the objective function $f(x)$ over the set $\mathcal{A}$ forms a is defined as:

$$\mathcal{F}_{\mathcal{A}}(f) \quad \triangleq \quad \{(\phi, \psi) | \sum_{i=1}^{n} \phi_i(x_i) + \sum_{\alpha \in \mathcal{A}} \psi_\alpha(x_\alpha) = f(x) \text{ for all x}\} \tag{15}$$

If $(\phi, \psi) \in \mathcal{F}_{\mathcal{A}}(f)$ and $(\phi', \psi') \in \mathcal{F}_{\mathcal{A}}(f)$, then (ϕ, ψ) is called a *reparameterization* of (ϕ', ψ') and vice versa.

Every factorization of f has a corresponding graphical representation known as a *factor graph*. Factor graphs provide a visual representation of the relationship among the potential functions. The factor graph consists of a variable node i for each variable x_i, a factor node α for each of the potentials ψ_α, and an edge joining the factor node corresponding to α to the variable node representing x_i if $i \in \alpha$. For a concrete example, see Figure 2.

2.3 Message Passing Algorithms

The max-sum, sum-product, max-product, and min-sum algorithms are local message passing algorithms over a factor graph. Here, we will describe the max-sum algorithm. Suppose f factorizes over $\mathcal{A}$ as

$$f(x) = \sum_{i=1}^{n} \phi_i(x_i) + \sum_{\alpha \in \mathcal{A}} \psi_\alpha(x_\alpha) \tag{16}$$

The max-sum algorithm is a local message passing algorithm that attempts to compute an assignment that maximizes f. During the execution of the max-sum algorithm, messages are passed back and forth between adjacent nodes of the graph. In the algorithm, there are two types of messages: messages passed from variable nodes to factor nodes and messages passed from factor nodes to variable nodes. On the t^{th} iteration of the algorithm, messages are passed along each edge of the factor graph as follows:

$$m_{i \to \alpha}^{t}(x_i) \quad = \quad \kappa + \phi_i(x_i) + \sum_{\beta \in \partial i \setminus \alpha} m_{\beta \to i}^{t-1}(x_i) \tag{17}$$

$$m_{\alpha \to i}^{t}(x_i) \quad = \quad \kappa + \max_{x_{\alpha \setminus i}} \left[\psi_\alpha(x_\alpha) + \sum_{k \in \alpha \setminus i} m_{k \to \alpha}^{t-1}(x_k) \right] \tag{18}$$

where ∂i denotes the set of all $\alpha \in \mathcal{A}$ such that $i \in \alpha$ (intuitively, this is the set of neighbors of variable node x_i in the factor graph), x_α is the vector formed from the entries of x by selecting only the indices in α, and $\alpha \setminus i$ is abusive notation for the set-theoretic difference $\alpha \setminus \{i\}$.

The other message passing algorithms can be obtained from the one described here by substituting the appropriate operations. For example, the sum-product updates are given by:

$$m_{i\to\alpha}^t(x_i) = \kappa\phi_i(x_i)\prod_{\beta\in\partial i\setminus\alpha} m_{\beta\to i}^{t-1}(x_i) \tag{19}$$

$$m_{\alpha\to i}^t(x_i) = \kappa\sum_{x_{\alpha\setminus i}}\Big[\psi_\alpha(x_\alpha)\prod_{k\in\alpha\setminus i} m_{k\to\alpha}^{t-1}(x_k)\Big] \tag{20}$$

where f factorizes over $\mathcal{A}$ as

$$f(x) = \prod_{i=1}^n \phi_i(x_i)\prod_{\alpha\in\mathcal{A}}\psi_\alpha(x_\alpha) \tag{21}$$

When the graph is a tree, these message updates can be derived using the same dynamic programming techniques we saw in Example 2.1. When the graph is not a tree, the same updates are used as if the graph was a tree. Understanding when these updates work for arbitrary graphs is the central question underlying the study of local message passing algorithms.

Each message update has an arbitrary normalization factor κ. Because κ is not a function of any of the variables, it only affects the value of the maximum and not where the maximum is located. As such, we are free to choose it however we like for each message and each time step. In practice, these constants are used to avoid numerical issues that may arise during the execution of the algorithm.

Definition 2.1. *A vector of messages $m = (\{m_{\alpha\to i}\},\{m_{i\to\alpha}\})$ is **real-valued** if for all $\alpha\in\mathcal{A}$, $\forall i\in\alpha$, and $\forall x_i\in\mathcal{X}$, $m_{\alpha\to i}(x_i)$ and $m_{i\to\alpha}(x_i)$ are real-valued functions (i.e., they do not take the value ∞ for any $x_i\in\mathcal{X}$).*

We will think of the messages as a vector of functions indexed by the edge over which the message is passed. Any vector of real-valued messages is a valid choice for the vector of initial messages m^0, but the choice of initial messages can greatly affect the behavior of the algorithm. A typical assumption for the max-sum algorithm is that the initial messages are chosen such that $m_{\alpha\to i}^0\equiv 0$ and $m_{i\to\alpha}^0\equiv 0$. This uniformity assumption is often useful when we need to analyze the evolution of the algorithm over time, but ideally, we would like to design message passing schemes that perform well independent of initialization.

In Example 2.1, we saw that the maximum weight independent set could be computed by looking at all of the messages passed to the root node. $\mathrm{mwis}(r,\mathrm{inSet})$ is special in that it recursively depends on all of the problem variables. Because of this, we can compute the value of the maximum weight independent set simply by trying both possible values of inSet. For general objective functions, this phenomenon is captured by a marginal computation. A max-marginal of f is a function of one or more variables obtained by fixing a subset of the variables and maximizing the function f over all of the remaining variables. For example, the max-marginal for the variable x_i would be the function $f_i(x_i) = \max_{x':x'_i=x_i} f(x')$.

We can use the messages in order to construct an estimate of the max-marginals of f. Given any vector of messages, m^t, we can construct a set of beliefs that are intended to approximate the max-marginals of f:

$$b_i^t(x_i) = \kappa+\phi_i(x_i)+\sum_{\alpha\in\partial i} m_{\alpha\to i}^t(x_i) \tag{22}$$

$$b_\alpha^t(x_\alpha) = \kappa+\psi_\alpha(x_\alpha)+\sum_{i\in\alpha} m_{i\to\alpha}^t(x_i) \tag{23}$$

If $b_i(x_i) = \max_{x':x'_i=x_i} f(x')$, then for any $y_i\in\arg\max_{x_i} b_i(x_i)$ there exists a vector x^* such that $x_i^* = y_i$ and x^* maximizes the function f. If the $|\arg\max_{x_i} b_i(x_i)| = 1$ for all i, then we can take $x^* = y$, but, if the objective function has more than one optimal solution, then we may not be able to construct such an x^* so easily. For this reason, one typically assumes that the objective function has a unique global maximum. Unfortunately, because the beliefs are not necessarily the

true max-marginals, we can only approximate the optimal assignment by computing an estimate of the argmax:

$$x_i^t \quad \in \quad \arg\max_{x_i} b_i^t(x_i) \tag{24}$$

Definition 2.2. *A vector, $b = (\{b_i\}, \{b_\alpha\})$, of beliefs is **locally decodable** to x^* if $b_i(x_i^*) < b_i(x_i)$ for all i, $x_i \neq x_i^*$. Equivalently, each b_i has a unique maximum at x_i^*.*

If the algorithm converges to a vector of beliefs that are locally decodable to x^*, then we hope that the vector x^* is a global maximum of the objective function. This is indeed the case when the factor graph contains no cycles. Informally, this follows from the correctness of dynamic programming on a tree. We will defer a formal proof of this result until later. For now, consider using the max-sum algorithm to compute the maximum weight independent set on a tree:

Example 2.2 (Maximum weight independent set on a tree revisited).
Consider the maximum weight independent set problem on the graph $G = (V, E)$ with weights w_i from Example 2.1. Let $x \in \{0, 1\}^n$ be an indicator vector for a set $S \subseteq V$ where $x_i = 1$ if $i \in S$ and zero otherwise. We can construct an objective function for the maximum weight independent set problem as follows:

$$f(x) = \sum_{i \in V} w_i x_i + \sum_{(i,j) \in E} \log\{x_i + x_j \leq 1\} \tag{25}$$

where $\{x_i + x_j \leq 1\}$ is one if $x_i + x_j \leq 1$ and zero otherwise. Fix a vector $\overline{x}$ and let $\overline{S}$ be the corresponding set. We can check that $f(\overline{x}) = w(\overline{S})$ if $\overline{S}$ is an independent set in G and $f(x) = \infty$ otherwise. Consequently, any x that maximizes f must correspond to a maximum weight independent set. For the natural factorization given by equation 25, the max-sum message passing updates can be written as:

$$m_{i \to (i,j)}^t(x_i) = w_i x_i + \sum_{k \in \partial i \setminus j} m_{(i,k) \to i}^{t-1}(x_i) \tag{26}$$

$$m_{(i,j) \to i}^t(x_i) = \max_{x_j} [\log\{x_i + x_j \leq 1\} + m_{j \to (i,j)}^{t-1}(x_j)] \tag{27}$$

By substituting equation 27 into equation 26, we have

$$m_{i \to (i,j)}^t(x_i) \quad = \quad w_i x_i + \sum_{k \in \partial i \setminus j} \max_{x_k} [\log\{x_i + x_k \leq 1\} + m_{k \to (i,k)}^{t-2}(x_k)] \tag{28}$$

which is equivalent to the recurrence, equations 7 and 8, derived using dynamic programming. If G is a tree with root $r \in V$, then a simple proof by induction can be used to show that the messages will converge to fixed values after at most $|V|$ time steps. Further, for all $t > |V|$, $m_{i \to (i,p(i))}^t(x_i) = \text{mwis}(i, x_i)$ where $p(i) \in V$ is the parent of i in G. From this observation, we can conclude that $\max_{x_r} b_r(x_r)$ corresponds to the value of the maximum weight independent set.

Notice that, unlike the dynamic programming solution, there is no fixed root node, and the max-sum algorithm does not pass messages only in one direction in the tree. The max-sum algorithm actually computes all of the max-marginals simultaneously, instead of just the one at the root.

2.3.1 Computation Trees

An important tool in the analysis of the local message passing algorithms is the notion of a computation tree. Intuitively, the computation tree is an unrolled version of the original graph that captures the evolution of the messages passed by the max-sum algorithm needed to compute the belief at time t at a particular node of the factor graph. Computation trees describe the evolution of the beliefs over time, which, in some cases, can help us prove correctness and/or convergence of

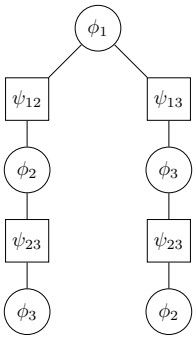

Figure 3: The computation tree at time $t = 4$ rooted at the variable node x_1 of the factor graph in Figure 2. The variable nodes have been labeled with their potentials for emphasis.

the message passing updates. For example, the convergence of the max-sum algorithm on graphs containing a single cycle can be demonstrated by analyzing the computation trees produced by the max-sum algorithm at each time step [4].

The depth t computation tree rooted at node i contains all of the length t non-backtracking walks in the factor graph starting at node i. A walk is non-backtracking if it does go back and forth successively between two vertices. For any node v in the factor graph, the computation tree at time t rooted at v, denoted by $T_v(t)$, is defined recursively as follows: $T_v(0)$ is just the node v, the root of the tree. The tree $T_v(t)$ at time $t > 0$ is generated from $T_v(t-1)$ by adding to each leaf of $T_v(t-1)$ a copy of each of its neighbors in G (and the corresponding edge), except for the neighbor that is already present in $T_v(t-1)$. Each node of $T_v(t)$ is a copy of a node in G, and the potentials on the nodes in $T_v(t)$, which operate on a subset of the variables in $T_v(t)$, are copies of the potentials of the corresponding nodes in G. The construction of a computation tree for the graph in Figure 2 is pictured in Figure 3. Note that each variable node in $T_v(t)$ represents a distinct copy of some variable x_j in the original graph.

Given any initialization of the messages, $T_v(t)$ captures the information available to node v at time t. At time $t = 0$, node v has received only the initial messages from its neighbors, so $T_v(0)$ consists only of v. At time $t = 1$, v receives the round one messages from all of its neighbors, so v's neighbors are added to the tree. These round one messages depend only on the initial messages, so the tree terminates at this point. By construction, we have the following lemma:

Lemma 2.1. *The belief at node v produced by the max-sum algorithm at time t corresponds to the exact max-marginal at the root of $T_v(t)$ whose boundary messages are given by the initial messages.*

Proof. See, for example, [2] and [5]. □

2.3.2 Fixed Point Properties

Computation trees provide us with a dynamic view of the max-sum algorithm. After a finite number of time steps, we hope that the beliefs on the computation trees stop changing and that the message vector converges to a fixed point of the message update equations (in practice, when the beliefs change by less than some small amount, we say that the algorithm has converged). For any real-valued objective function f (i.e. $|f(x)| < \infty$ for all x), there always exists a fixed point of the message update equations (see Theorem 2 of [3]).

Ideally, the beliefs constructed from any fixed point of the message update equations would be the true max-marginals of the function f. If the beliefs are the exact max-marginals, then the estimate corresponding to our beliefs would indeed be a global maximum. Unfortunately, the algorithm is only known to produce the exact max-marginals on special factor graphs (e.g. when the factor graph is a tree). Instead, we will show that the fixed point beliefs are almost like max-marginals. Like the messages, we will think of the beliefs as a vector of functions indexed by the nodes of the factor graph. Consider the following definitions:

Definition 2.3. *A vector of beliefs, b, is **admissible** for a function f if*

$$f(x) = \kappa + \sum_i b_i(x_i) + \sum_\alpha \Big[b_\alpha(x_\alpha) - \sum_{k \in \alpha} b_k(x_k) \Big]$$

Beliefs satisfying this property are said to reparameterize the objective function.

Definition 2.4. *A vector of beliefs, b, is **max-consistent** if for all α and all $i \in \alpha$:*

$$\max_{x_{\alpha \setminus i}} b_\alpha(x_\alpha) = \kappa + b_i(x_i)$$

Any vector of beliefs that satisfies these two properties provides a meaningful reparameterization of the original objective function. Any vector of beliefs obtained from a fixed point of the message updates does indeed satisfy these two properties:

Theorem 2.2. *For any vector of fixed point messages of the max-sum algorithm, the corresponding beliefs are admissible and max-consistent.*

Proof. Let m be a fixed point of the message update equations:

$$m_{i \to \alpha}(x_i) = \kappa + \phi_i(x_i) + \sum_{\beta \in \partial i \setminus \alpha} m_{\beta \to i}(x_i) \tag{29}$$

$$m_{\alpha \to i}(x_i) = \kappa + \max_{x_{\alpha \setminus i}} \Big[\psi_\alpha(x_\alpha) + \sum_{k \in \alpha \setminus i} m_{k \to \alpha}(x_k) \Big] \tag{30}$$

First, we will show that m produces max-consistent beliefs. Take $\alpha \in \mathcal{A}$ and choose some $i \in \alpha$. Without loss of generality, we can assume that the constants are uniformly equal to zero.

$$\max_{x_{\alpha \setminus i}} b_\alpha(x_\alpha) = \max_{x_{\alpha \setminus i}} \psi_\alpha(x_\alpha) + \sum_{k \in \alpha} m_{k \to \alpha}(x_k) \tag{31}$$

$$= m_{i \to \alpha}(x_i) + \max_{x_{\alpha \setminus i}} \psi_\alpha(x_\alpha) + \sum_{k \in \alpha \setminus i} m_{k \to \alpha}(x_k) \tag{32}$$

$$= m_{i \to \alpha}(x_i) + m_{\alpha \to i}(x_i) \tag{33}$$

$$= \phi_i(x_i) + \Big[\sum_{\beta \in \partial i \setminus \alpha} m_{\beta \to i}(x_i) \Big] + m_{\alpha \to i}(x_i) \tag{34}$$

$$= b_i(x_i) \tag{35}$$

Next, we can check that the beliefs are admissible. Again, we can assume that the constants are uniformly equal to zero.

$$f(x) = \sum_i \phi_i(x_i) + \sum_\alpha \psi_\alpha(x_\alpha) \tag{36}$$

$$= \sum_i \Big[\phi_i(x_i) + \sum_{\alpha \in \partial i} m_{\alpha \to i}(x_i) \Big] + \sum_\alpha \Big[\psi_\alpha(x_\alpha) - \sum_{i \in \alpha} m_{\alpha \to i}(x_i) \Big] \tag{37}$$

$$= \sum_i b_i(x_i) + \sum_\alpha \Big[\psi_\alpha(x_\alpha) - \sum_{i \in \alpha} m_{\alpha \to i}(x_i) \Big] \tag{38}$$

$$= \sum_i b_i(x_i) + \sum_\alpha \Big[\psi_\alpha(x_\alpha) + \sum_{i \in \alpha} m_{i \to \alpha}(x_i) - \sum_{i \in \alpha} m_{i \to \alpha}(x_i) - \sum_{i \in \alpha} m_{\alpha \to i}(x_i) \Big] \tag{39}$$

$$= \sum_i b_i(x_i) + \sum_\alpha \Big[b_\alpha(x_\alpha) - \sum_{i \in \alpha} \Big[m_{i \to \alpha}(x_i) + m_{\alpha \to i}(x_i) \Big] \Big] \tag{40}$$

$$= \sum_i b_i(x_i) + \sum_\alpha \Big[b_\alpha(x_\alpha) - \sum_{i \in \alpha} \Big[\phi_i(x_i) + \Big[\sum_{\beta \in \partial i \setminus \alpha} m_{\beta \to i}(x_i) \Big] + m_{\alpha \to i}(x_i) \Big] \Big] \tag{41}$$

$$= \sum_i b_i(x_i) + \sum_\alpha \Big[b_\alpha(x_\alpha) - \sum_{i \in \alpha} b_i(x_i) \Big] \tag{42}$$

□

References

[1] S.M. Aji and R.J. McEliece. The generalized distributive law. *Information Theory, IEEE Transactions on*, 46(2):325 –343, mar 2000.

[2] S. Tatikonda and M. I. Jordan. Loopy belief propagation and gibbs measures. In *In Uncertainty in Artificial Intelligence*, pages 493–500. Morgan Kaufmann, 2002.

[3] M. Wainwright, T. Jaakkola, and A. S. Willsky. Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. *Statistics and Computing*, 14(2):143–166, 2004.

[4] Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Comput.*, 12(1):1–41, 2000.

[5] Y. Weiss and W.T. Freeman. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *Information Theory, IEEE Transactions on*, 47(2):736 –744, feb 2001.