Loopy Belief Propagation in the Presence of Determinism

David Smith
The University of Texas at Dallas
Richardson, TX, 75080, USA
dbs014200@utdallas.edu

Vibhav Gogate
The University of Texas at Dallas
Richardson, TX, 75080, USA
vgogate@hlt.utdallas.edu

Abstract

It is well known that loopy Belief propagation (LBP) performs poorly on probabilistic graphical models (PGMs) with determinism. In this paper, we propose a new method for remedying this problem. The key idea in our method is finding a reparameterization of the graphical model such that LBP, when run on the reparameterization, is likely to have better convergence properties than LBP on the original graphical model. We propose several schemes for finding such reparameterizations, all of which leverage unique properties of zeros as well as research on LBP convergence done over the last decade. Our experimental evaluation on a variety of PGMs clearly demonstrates the promise of our method – it often yields accuracy and convergence time improvements of an order of magnitude or more over LBP.

1 Introduction

Loopy Belief Propagation [15, 16] is a widely used algorithm for performing approximate inference in graphical models. The algorithm is exact on tree graphical models and often yields reasonably good poly-time approximations on graphical models having cycles. Its applicability has been demonstrated across a wide variety of fields, including error correcting codes [6, 12], computer vision [5], and combinatorial optimization [22].

Despite its popularity, LBP is known to display erratic behavior in practice. In particular, it is still not well understood when LBP will yield good approximations. Considerable progress has been made towards characterizing the approximations that LBP yields (as fixed points of approximate free energy), formalizing sufficient conditions for (i) ensuring the existence of fixed points [23]; (ii) ensuring LBP convergence [21]; and (iii) ensuring the uniqueness of the fixed point to which LBP converges to [8, 9, 14, 17]; and so on. Unfortunately, in practice, even the best bounds are too narrow to be used as an accurate predictor of convergence across the spectrum of graphical models.

The situation becomes even murkier when LBP is applied to problems that contain determinism. Many of the convergence results apply only to problems that model strictly positive distributions. Some work has been done on investigating the effect of determinism on the LBP family of algorithms [4], and some convergence results account for determinism in a restricted manner [14], but in general the existing literature avoids consideration of determinism because of the complications it engenders.

The goal of this work is to harness the presence of determinism in a PGM in order to help LBP converge more quickly and to more accurate results. In particular, we leverage the following unique property of graphical models (Markov networks) having zeros: if we can derive that the marginal probability of an entry in a potential is zero from other potentials in the Markov network, then we can change that entry to *any* non-negative real value without changing the underlying distribution. The key idea in this paper is to change such potential entries, namely re-parameterize the Markov network, in such a way that the LBP algorithm converges faster on the new parameterization than on the original Markov network.

To achieve this objective, we present two algorithms. The first algorithm leverages techniques from the Satisfiability (SAT) literature to identify a set of potential entries whose values can be altered without affecting the underlying distribution. The second algorithm uses a variety of heuristics to calculate a new set of values in order to create a reparameterization of the distribution in which the interactions between neighboring potentials are indicative of models on which LBP tends to be successful. We show experimentally that our new approach often allows LBP to converge in fewer iterations and to more accurate results.

Appearing in Proceedings of the 17^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2014, Reykjavik, Iceland. JMLR: W&CP volume 33.

2 Preliminaries and Notation

2.1 Markov Networks

Let $G = (V, E)$ be an undirected graph, where $V = \{1, \ldots, N\}$ is the set of vertices, and $E \subseteq \{(i, j) | i, j \in V; i \neq j\}$ is the set of edges. Let $X = \{X_i | i \in V\}$ denote the set of random variables associated with G and let $\text{dom}(X_i)$ denote the domain of X_i. (We restrict random variables to Boolean domains for the sake of simplicity of exposition. Our method is quite general and can be applied to multi-valued graphical models). Let $\Phi = \{\phi_1, \ldots, \phi_m\}$ be a set of real-valued functions, called potentials, where $\phi_i(X_j, \ldots, X_k) : \text{dom}(X_i) \times \ldots \times \text{dom}(X_k) \rightarrow \mathbb{R}_0^+$, and such that the vertex set $\{j, \ldots, k\}$ is a maximal clique in G. Then, the pair $M = \langle G, \Phi \rangle$ forms a Markov network. We denote the set of random variables associated with a potential ϕ_i by $\text{vars}(\phi_i)$.

A Markov network M, induces the following probability distribution:

$$P_M(x) = \frac{1}{Z} \prod_{1 \leq i \leq m} \phi_i(x_i)$$

where x is a full assignment to all variables in X, x_i is the projection of the assignment x on the variables in $\text{vars}(\phi_i)$, and Z is the normalization constant or the partition function, given by $Z = \sum_x \prod_{1 \leq i \leq m} \phi_i(x_i)$.

2.2 Factor Graphs

Factor graphs are special graph structures that enable us to systematically study and understand the properties of sum-product algorithms such as loopy belief propagation. A factor graph contains a corresponding vertex for each random variable and each potential in the Markov network. A potential vertex is connected (via an undirected edge) to a random variable vertex if the the potential contains the random variable in its scope. Formally, given a Markov network $M = \langle G = (V, E), \Phi \rangle$, the corresponding factor graph, denoted by $F = (V_F, E_F)$, is an undirected bipartite graph with vertex set $V_F = \{V_i | i \in V\} \cup \{P_i | \phi_i \in \Phi\}$ and edge set $E_F = \{(V_i, P_j) | X_i \in \text{vars}(\phi_j)\}$, where X_i is the random variable corresponding to V_i.

2.3 Loopy Belief Propagation

Given a Markov network M, loopy belief propagation (LBP) is an iterative algorithm that computes approximations to the set of marginal distributions of P_M, $\{P(X_i)\}_{i \in V}$. It does so by passing messages on the factor graph corresponding to M. On every iteration, each vertex passes a message to each of its neighbors. We denote the message from variable vertex V_i to a potential vertex P_j by $\mu_{V_i \rightarrow P_j}(x_i)$ and the message from P_j to V_i by $\mu_{P_j \rightarrow V_i}(x_i)$. Note that each message is a potential over a single variable. On each iteration it, the messages a node sends are determined by the messages received from its neighbors on the previous iteration $it - 1$, as well as by the associated potential ϕ_j if the node is a potential node P_j. They are calculated by the following update equations:

$$\mu^{(it+1)}_{V_i \rightarrow P_j}(x_i) \propto \prod_{P_k \in N_F(V_i) \setminus P_j} \mu^{(it)}_{P_k \rightarrow V_i}(x_i) \tag{1}$$

$$\mu^{(it+1)}_{P_j \rightarrow V_i}(x_i) \propto \sum_{\text{vars}(\phi_j) \setminus X_i} \phi(x_i) \prod_{V_k \in N_F(P_j) \setminus V_i} \mu^{(it)}_{V_k \rightarrow P_j}(x_k) \tag{2}$$

$N_F(V_i)$ and $N_F(P_j)$ are the sets of neighbors of V_i and P_j in the factor graph F respectively.

In practice, the messages are normalized in order to prevent underflow or overflow (hence $\propto$ instead of $=$). The loopy belief propagation algorithm iterates until it reaches some iteration it such that $\forall i, j, x_i \in \text{dom}(X_i), |\mu^{(it-1)}_{V_i \rightarrow P_j}(x_i) - \mu^{(it)}_{V_i \rightarrow P_j}(x_i)| < \epsilon$ and similarly, $\forall i, j, x_i \in \text{dom}(X_i) | \mu^{(it-1)}_{P_j \rightarrow V_i}(x_i) - \mu^{(it)}_{P_j \rightarrow V_i}(x_i)| < \epsilon$, for some small positive constant $\epsilon > 0$.

At this point, the algorithm is said to have converged and one can calculate the approximations to the single variable marginals (often called *beliefs*) via the following equation:

$$b_i(x_i) = N_i \prod_{P_j \in N(V_I)} \mu^{(it)}_{P_j \rightarrow V_I}(x_i) \tag{3}$$

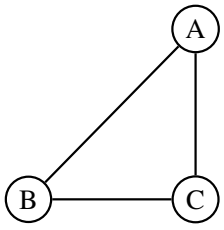
where N_i is a normalization constant, chosen so that $\sum_{x_i \in \text{dom}(X_i)} b_i(x_i) = 1$.

3 Zero-Based Reparameterizations of Markov Networks

The behavior of the LBP algorithm on models with determinism is, in general, poorly understood. In practice, LBP has been observed to yield sub-par results when applied to such models. The purpose of this work is to present an algorithm that takes a Markov network M with determinism (i.e., P_M is not strictly positive), and yields a Markov network M' such that $P_M = P_{M'}$ (i.e., our algorithm achieves a reparameterization), and the associated approximations to the single variable marginals that LBP yields on M' are more accurate than the ones it yields on M.

To this end, we are interested in defining a class of functions that take a Markov network M with zeros as input, leverage unique properties of zeros, and return a reparameterization M' of M (if M' is a reparameterization of M, we will say $M \equiv M'$). Also, in order to keep the number of possible reparameterizations manageable, we will only consider reparameterizations satisfying the following two properties: (i) M and M' are defined over the same set of variables; and (ii) for each potential ϕ in M there is a corresponding potential ϕ' in M' such that $\text{vars}(\phi) = \text{vars}(\phi')$.[1]

[1] Technically, if we express the Markov network as a set of



(a) A graph

A	B	C	P_M
0	0	0	0*
0	0	1	0*
0	1	0	0
0	1	1	1/3
1	0	0	1/3
1	0	1	0
1	1	0	0
1	1	1	1/3

(b) The joint probability distribution

A	B	$\phi_{A,B}$
0	0	2†
0	1	1
1	0	1
1	1	1

A	C	$\phi_{A,C}$
0	0	0
0	1	1
1	0	1
1	1	1

B	C	$\phi_{B,C}$
0	0	1
0	1	0
1	0	0
1	1	1

(c) A set of potentials

Figure 1: A Markov network and its associated joint probability distribution

We will use a unique property of zeros for defining our reparameterizing functions. We illustrate it using the following example.

Example 1. *Consider the Markov network given in Figure 1. The Markov network consists of 3 binary-valued random variables and 3 pairwise potentials with some zero valued tuples. The joint distribution of the network is shown in Figure 1(b). Notice that the entry in $\phi_{A,B}$ corresponding to $A = 0, B = 0$ (denoted with a †) has a non-zero weight of 2, but that all the tuples in the joint distribution consistent with this assignment (denoted with a $*$) have zero weight. Thus, even though $\phi_{A,B}(A = 0, B = 0) = 2$, the probability of the assignment $P(A = 0, B = 0) = 0$. As a consequence, we can create a Markov network $M' = \langle G', \Phi' \rangle$, in which $G' = G$ and $\Phi' = \{\phi'_{A,B}, \phi_{A,C}, \phi_{B,C}\}$, where:*

$$\phi'_{A,B}(a, b) = \begin{cases} r, & (a = 0) \wedge (b = 0) \\ \phi_{A,B}(a, b), & \textit{otherwise} \end{cases}$$

If $r \geq 0$, we are guaranteed that M' is well-formed as long as M is. And notice that $M \equiv M'$ for any *value we choose for r.*

We formalize the ideas presented in Example 1 starting with a required definition.

Definition 1 (Inferable Zero). Let M be a Markov network and let x_i be an assignment to all variables in a potential ϕ_i of M. We say that $\phi_i(x_i)$ is an inferable zero of M iff for all possible extensions x of x_i to all variables X of M, there exists a potential ϕ_j in M, $i \neq j$ such that $\phi_j(x_j) = 0$. In other words, $\phi_i(x_i)$ is an inferable zero iff $P_M(x_i) = 0$ can be inferred from other potentials in M.

weighted features, we are only considering transformations that change the weights of features.

In our running example, we have only one inferable zero: $\phi_{A,B}(a = 0, b = 0)$. It turns out that we can always safely change an inferable zero to any non-negative real number without changing the underlying distribution. We consider a class of functions f that achieve this, and we show that they yield a reparameterization of M.

Theorem 3.1. *Given a Markov network $M = \langle G, \Phi \rangle$, an inferable zero of M $\phi_i(x_i)$, and an element $r \in \mathbb{R}_0^+$, let f be a function of the form $f(M, \phi_i(x_i), r) = M'$, where $M' = \langle G, \Phi' \rangle$, $\Phi' = \Phi \setminus \phi_i \cup \phi'_i$ and*

$$\phi'_i(x'_i) = \begin{cases} r, & x'_i = x_i \\ \phi_i(x'_i), & \textit{otherwise} \end{cases} \tag{4}$$

Then, $f(M, \phi_i(x_i), r)$ is a reparameterization of M. Namely, $f(M, \phi_i(x_i), r) \equiv M$.

Proof. We will show that for any assignment x, $\prod_{\phi_j \in \Phi} \phi_j(x_j) = \prod_{\phi'_j \in \Phi'} \phi'_j(x_j)$. We consider two cases.

Case 1: When $\phi_i(x_i)$ is not an inferable zero, by definition, $\phi_j(x_j) = \phi'_j(x_j)$ for all j and therefore $\prod_{\phi_j \in \Phi} \phi_j(x) = \prod_{\phi'_j \in \Phi'} \phi'_j(x)$.

Case 2: When $\phi_i(x_i)$ is an inferable zero, by definition, there exists a potential ϕ_k such that $\phi_k(x_k) = \phi'_k(x_k) = 0$. Therefore, $\prod_{\phi_j \in \Phi} \phi_j(x_j) = \prod_{\phi'_j \in \Phi'} \phi'_j(x_j) = 0$. □

The class of functions denoted by f in Theorem 3.1 define our reparameterizing functions. f can be simultaneously applied to all potential values which are non-zero without altering the underlying distribution because f will not remove any zeroes from the network M. However, more care must be taken when applying f to potential weights which are zero-valued, because f can potentially replace these zeros with positive real numbers, thus changing the set of inferable zeros in M. We illustrate this in the following example.

Example 2. *Consider the Markov network shown in Figure 1 with the following change: $\phi_{A,B}(A = 0, B = 0) = 0$ instead of 2. Notice that the new Markov network has two inferable zeros: $\phi_{A,B}(A = 0, B = 0)$ and $\phi_{A,C}(A = 0, C = 0)$. Out of these, we can either change $\phi_{A,B}(A = 0, B = 0)$ or $\phi_{A,C}(A = 0, C = 0)$ to an arbitrary positive number without affecting the underlying distribution, but we cannot change both.*

In other words, the order in which inferable zeros with associated zero weights are changed to positive numbers matters. Since we are interested in replacing multiple inferable zeros by positive numbers, we next define an algorithmic process that achieves this objective. We will refer to this process as $\mathcal{F}$.

Process $\mathcal{F}$ applies f along a particular (heuristically chosen) ordering of inferable zeros in M. As previously mentioned, if $\phi_i(x_i) \neq 0$, then we can safely apply f to the

Algorithm 1 Process $\mathcal{F}$

1: **Input:** A Markov network M
2: **Output:** A Markov Network M' such that $M \equiv M'$, a set Ψ of configurable parameters of M'
3: Let $\Phi_a = \{\phi_i(x_i) \mid \phi_i(x_i)$ is an inferable zero of $M\}$
4: Let $\pi(\Phi_a)$ be an ordering of the elements of Φ_a
5: $\Psi = \varnothing$
6: $M' = M$
7: **for** i = 1 to $|\Phi_a|$ **do**
8: Let $\phi_i(x_i)$ be the i-th entry in $\pi(\Phi_a)$
9: Let $\phi'_i(x_i)$ be the potential entry in M' corresponding to $\phi_i(x_i)$
10: **if** $\phi_i(x_i) \neq 0$ **then**
11: $M' = f(M', \phi'_i(x_i), 1)$
12: $\Psi = \Psi \cup \{\phi_i(x_i)\}$
13: **else if** $\phi'_i(x_i)$ is an inferable zero of M' **then**
14: $M' = f(M', \phi'_i(x_i), 1)$
15: $\Psi = \Psi \cup \{\phi_i(x_i)\}$
16: **end if**
17: **end for**
18: **return** $\langle M', \Psi \rangle$

Markov network (Steps 9 and 10). However, if $\phi_i(x_i) = 0$ then care must be taken to ensure that x_i is also an inferable zero of the current Markov network M' (Steps 12 and 13). $\mathcal{F}$ sets the value of each inferable zero to 1 (as a placeholder), and it collects the inferable zeros in the set Ψ.

Since the Markov network M' at each iteration in $\mathcal{F}$ is either the same or obtained by applying the function f to the Markov network in the previous iteration, it follows from Theorem 3.1 that M' is a reparameterization of the Markov network in the previous iteration. Therefore, by using the principle of induction, it is straight-forward to show that:

Theorem 3.2. *$\mathcal{F}$ yields a reparameterization of M. Namely, if $\mathcal{F}(M) = \langle M', \Psi \rangle$, then $M' \equiv M$.*

Now, notice that each of the potential weights Ψ of M' can be set to *any* element $r \in \mathbb{R}_0^+$ without affecting the distribution P_M. Let $\Delta(M, \Psi)$ be any function that assigns any element of $\mathbb{R}_0^+$ to each free parameter in Ψ. Then:

Theorem 3.3. *$\Delta(M, \Psi)$ yields a reparameterization of M. Namely, if $\mathcal{F}(M) = \langle M', \Psi \rangle$, then $\Delta(M, \Psi) \equiv M$.*

Thus, we refer to the parameters of Ψ as the *free parameters* of M.

Three practical issues remain with regard to using $\mathcal{F}$ for improving the convergence and accuracy of LBP:

- How to find the inferable zeros of M?
- What value of $\mathbb{R}_0^+$ should be assigned to each inferred zero in Ψ?
- How to choose an ordering for inferable zeros?

We will answer these three questions in the next two sections

4 Finding Inferable Zeros

The computation of f requires inferring some global information about the given Markov network for every entry of the given potential ϕ_i. It is well-known that such a task is NP-Hard for arbitrary Markov networks [3]. Fortunately, we do not need to infer the actual probability of each entry; it suffices to know whether or not the probability of each entry is equal to zero. Even so, such a task is NP-Complete for arbitrary Markov networks. However, with the help of modern SAT solving techniques (cf. [20]), even problems with a large number of variables and clauses can often be handled easily.

It is straightforward to encode the problem of finding the zero-valued marginals of a Markov network M with binary random variables as a SAT instance. For each random variable in M, add a Boolean variable to the SAT instance. For each $\phi_i(x_i) = 0$ in M, add the clause $\neg c_i$ to the SAT instance, where c_i is the conjunctive clause corresponding to the assignment x_i.

Example 3. *Let A, B and C be the Boolean variables associated with three random variables in the Markov network in Figure 1. Then the conjunctive clause $c_{(A=0,C=0)} = (\neg A \wedge \neg C)$, and $\neg c_{(A=0,C=0)} = (A \vee C)$.*

The conjunction of all such clauses forms a SAT instance in conjunctive normal form that is satisfiable iff M is well-defined (i.e., contains any full assignment x such that $P(x) \neq 0$). The problem is only slightly more difficult for Markov networks with multi-valued random variables (cf. Sang et al. [19]).

Example 4. *Let A, B and C be the Boolean variables associated with three random variables in the Markov network in Figure 1. The SAT instance corresponding to the Markov network is $(A \vee C) \wedge (B \vee \neg C) \wedge (\neg B \vee C)$.*

We can use such an encoding in order to determine if any potential entry is an inferable zero of M. Algorithm 2 details this procedure:

Algorithm 2 Check Inferable Zero

1: **Input:** A Markov network M, an inferable zero candidate, $\phi_i(x_i)$
2: **Output:** A member of $\{True, False\}$
3: Let S = ConstructSAT(M)
4: Let c_i be the conjunctive clause corresponding to x_i
5: **if** $\phi_i(x_i) \neq 0$ **then**
6: $S = S \wedge c_i$
7: **else**
8: $S = (S \setminus \neg c_i) \wedge c_i$ [2]
9: **end if**
10: **return** $\neg$ isSatisfiable(S)

Example 5. *Consider again M from Figure 1. Suppose we want to to determine if $\phi_{A,B}(A = 0, B = 0)$ is an inferable zero of M. Since $\phi_{A,B}(A = 0, B = 0) = 2 \neq 0$, we construct the SAT instance as shown in Example 4 and conjoin it with the assignment $(\neg A) \wedge (\neg B)$ to yield the SAT*

instance $S = (A \vee C) \wedge (B \vee \neg C) \wedge (\neg B \vee C) \wedge (\neg A) \wedge (\neg B)$. *$S$ is not satisfiable; hence Algorithm 2 returns True. Thus,* $\phi_{A,B}(A = 0, B = 0)$ *is an inferable zero of* M.

Example 6. *Consider the same modification to the Markov network in Figure 1 as detailed in Example 2; namely that* $\phi_{A,B}(A = 0, B = 0) = 0$ *instead of* 2*. Again, suppose we want to to determine if* $\phi_{A,B}(A = 0, B = 0)$ *is an inferable zero of* M*. Since* $\phi_{A,B}(A = 0, B = 0) = 0$*, we construct the SAT instance* $S = (A \vee B) \wedge (A \vee C) \wedge (B \vee \neg C) \wedge (\neg B \vee C)$*, remove the clause corresponding to* $\neg(\neg A \wedge \neg B) = (A \vee B)$*, and add* $(\neg A) \wedge (\neg B)$*, yielding a SAT instance* $S = (A \vee C) \wedge (B \vee \neg C) \wedge (\neg B \vee C) \wedge (\neg A) \wedge (\neg B)$*. Again,* S *is not satisfiable; hence Algorithm 2 again returns True. Thus,* $\phi_{A,B}(A = 0, B = 0)$ *is an inferable zero of* M.

There are a few notes we would like to briefly mention at this point. First, in the case in which the Markov network M is pairwise-binary, its corresponding SAT encoding is an instance of 2-SAT. Since each clause added by Algorithm 2 contains just a single variable, each inferable zero test reduces to a 2-SAT satisfiability test, which can be done in linear time using the approach of mapping the problem to one of checking for strongly connected components [1]. Second, note that it is not necessary to find all inferred zeroes in order to apply Algorithm 1. If it is infeasible to apply Algorithm 2 to all $\phi_i(x_i) \in M$, one can simply check only those potential entries that are troublesome for LBP. We propose a method for heuristically ranking potentials for this purpose in Section 5.

5 Setting the Free Parameters of M

Now that we can discover which potential entries can be safely changed without affecting the overall distribution, we must decide *how* to change them. There are many possible approaches to modifying a Markov network. In this work, we have chosen to take advantage of the substantial research that establishes sufficient conditions for the convergence of LBP to a fixed point [8, 9, 14, 17, 21, 23].

Many of the results on the convergence of LBP rely on defining some notion of the 'strength' of the potential and then guaranteeing convergence if some relationship among these potential strengths holds on the factor graph associated with the given Markov network. These notions are often defined exclusively on Markov networks that yield strictly positive distributions, thus making them unsuitable for our purposes. However, Mooij and Kappen [14] have defined a notion of potential strength that is well-defined on many distributions that are not strictly positive, and so we have chosen to use the formula offered in their work as the basis for our choice of parameters. We reprint it here for convenience:

$$N(\phi_i, j, k) := \sup_{\alpha \neq \alpha'} \sup_{\beta \neq \beta'} \sup_{\gamma, \gamma'} \frac{\sqrt{\phi_i(\alpha\beta\gamma)\phi_i(\alpha'\beta'\gamma')} - \sqrt{\phi_i(\alpha'\beta\gamma)\phi_i(\alpha\beta'\gamma')}}{\sqrt{\phi_i(\alpha\beta\gamma)\phi_i(\alpha'\beta'\gamma')} + \sqrt{\phi_i(\alpha'\beta\gamma)\phi_i(\alpha\beta'\gamma')}} \quad (5)$$

Convergence to a unique fixed point, irrespective of initial messages, is guaranteed if:

$$\max_{P_j \to V_j} \sum_{P_i \in N(P_j) \setminus P_j} \sum_{V_i \in N(P_i) \setminus V_j} N(P_i, i, j) < 1 \quad (6)$$

Our approach is to develop heuristics that attempt to minimize this quantity. We present several methods for performing this optimization.

Given a Markov network M and some set of free parameters Ψ, one has several choices as to how to optimize elements of Ψ for LBP. One can optimize each parameter independently, partition the parameters of Ψ into disjoint sets and optimize each set of parameters independently, or one can optimize the entire set of parameters in Ψ jointly. We have found that optimizing the parameters independently yields relatively little gain on the majority of Markov networks. Conversely, optimizing all the parameters jointly tends to be infeasible on the majority of Markov networks. Hence we have chosen to optimize the free parameters in each potential jointly.

We present a formal framework for these optimizing functions. Define $\Delta(M, \Psi) = M'$, where $M' = \langle G, \Phi' \rangle$ and $\Phi' = \{\delta(\phi_i, \psi_i) | \phi_i \in \Phi, \psi_i = \{\phi_i(x_i) | \phi_i(x_i) \in \Psi\}\}$, and $\delta(\phi_i, \psi_i)$ is any function that takes a potential ϕ and its free parameters ψ as input, and returns a new potential ϕ' that has the same scope ($vars(\phi) = vars(\phi')$), but in which all free parameters have (possibly) different weights.

Thus, $\Delta(\mathcal{F}(M)) = M'$ and $M \equiv M'$. That is, a Δ function takes a Markov network M and a set of free parameters in M, and it calls δ on each potential of M along with the free parameters of that potential, assigns values to all free parameters, and returns a Markov network M' with an equivalent joint distribution.

5.1 The Zero Heuristic- δ_0

Perhaps the most obvious heuristic is to simply set all the free parameters in each potential to 0. The intuitive idea is that this choice would give LBP the maximum knowledge about the global distribution at each potential. In the context of Equation 5, such a choice of heuristic turns out to be problematic, because it forces an evaluation of 1 in more cases. Equation 5 implies that such evaluations will hurt the performance of LBP, and our experiments corroborate this hypothesis in many cases, although this heuristic does, in general, help the performance of LBP (see Section 6).

5.2 The Mean Heuristic- δ_μ

Another computationally inexpensive method is to simply set all the weights of the free parameters to the mean value of the weights of the non-free parameters in their respective

[2] We abuse notation slightly. Take $S \setminus c$ to mean the SAT instance S with the clause c removed.

potentials. Clearly, given a graphical model with uniform pseudo-distributions at each potential (and hence an overall distribution that is uniform), Equation 6 will have a value of 0, assuring convergence to a unique fixed point. The idea behind this simple modification is that it brings the pseudo-distribution at each potential closer to the uniform one.

5.3 The Local Min Strength Heuristic- δ_{LMS}

This heuristic directly minimizes the quantity in Equation 5 via a nonlinear optimization routine. We optimize each potential independently. Let $x \in \mathbb{R}_0^{+n}$, where n is the number of variable values in potential ϕ_i. We formulate the optimization problem as:

$$\min_{t \in \mathbb{R}_0^+, x \in \mathbb{R}_0^{+n}} t \textbf{ subject to } \underset{k \in \{1,\ldots,m\}}{g_k(x) - t \leq 0}$$

where m is equal to the total number of constraints of the form:

$$g_k(x) = \frac{\sqrt{\phi_i(\alpha\beta\gamma)\phi_i(\alpha'\beta'\gamma')} - \sqrt{\phi_i(\alpha'\beta\gamma)\phi_i(\alpha\beta'\gamma')}}{\sqrt{\phi_i(\alpha\beta\gamma)\phi_i(\alpha'\beta'\gamma')} + \sqrt{\phi_i(\alpha'\beta\gamma)\phi_i(\alpha\beta'\gamma')}}$$

taken over all possible values for α,α',β,β', γ, and γ', such that $\alpha \neq \alpha'$, $\beta \neq \beta'$, and the at least one of the members of the set $\{\phi_i(\alpha\beta\gamma), \phi_i(\alpha'\beta'\gamma'), \phi_i(\alpha'\beta\gamma), \phi_i(\alpha\beta'\gamma')\}$ is free parameter and hence equals x_i for $i \in \{1, \ldots, n\}$. This formulation is equivalent to minimizing the maximum value for $g_k(x)$, and hence is equivalent to minimizing the strength of each potential independently (see Equation 5).

5.4 The Local Min Sum Strength Heuristic- δ_{LMSS}

The previous heuristic has the disadvantage that it only attempts to minimize the maximum value of the set of constraints. Even after the minimax is found, we would like to continue minimizing the $g_k(x)$ for all values of k. One way to approximate this behavior is to minimize the sum of the constraints. We formulate the problem as follows:

$$\min_{t \in \mathbb{R}_0^+, x \in \mathbb{R}_0^{+n}} t \textbf{ subject to } t - \sum_{k=1}^{m} t_k \leq 0$$

$$\underset{k \in \{1,\ldots,m\}}{g_k(x) - t_k \leq 0}$$

where the set of constants are the same as those from the previous heuristic.

5.5 The Local Min Exponential Sum Strength Heuristic- δ_{LMESS}

Although the previous heuristic attempts to minimize over all constraints, it has the disadvantage that it assigns equal priority to the minimization of all sums, regardless of their size. We would like to give larger sums a higher priority for minimization. One way to approximate this behavior is to minimize the exponential sum of the constraints. We formulate the problem as follows:

$$\min_{t \in \mathbb{R}_0^+, x \in \mathbb{R}_0^{+n}} t \textbf{ subject to } t - \sum_{k=1}^{m} e^{t_k} \leq 0$$

$$\underset{k \in \{1,\ldots,m\}}{g_k(x) - t_k \leq 0}$$

where the set of constants are the same as those from the previous heuristic. Under this formulation, the largest values for $g_k(x)$ are assigned weights that decrease exponentially faster than the smaller weights, and hence are the priority for minimization, but after their mins have been found, the overall sum can still be minimized by optimizing over the remaining constraints.

5.6 Using the N function as an Ordering Heuristic

The N function can also be used to rank the strength of potentials in order to find an ordering for the application of f. Our approach is to order the set of potentials in descending order according to their Nscore. We then apply f to each entry of each potential along this ordering (we do no sorting of the entries in a given potential).

The choice of this heuristic is motivated by the fact that those potentials with higher Nscores are potentially more problematic for LBP, and hence adding variable values to them first is more likely to yield a reparameterization closer to the convergence guarantee.

6 Experiments

The implementation of this algorithm requires a SAT-solver and a nonlinear optimizer. We used Minisat [20] and NLOpt [10] for these tasks, respectively. We experimented with the following networks: (1) Ising Models, (2) random Markov networks, and (3) random 3-SAT instances. The potential values of the randomly generated Ising models and Markov networks were set as follows: we draw a random real number between 0 and 1 from the uniform distribution; if the number is smaller than the percent zero value for the network, we assign the potential entry a weight of 0; else a real number x is drawn from a Gaussian distribution with $\mu = 0$ and variance equal to σ, an input parameter. Then we take e^x and assign it as the potential value.

6.1 Grid Networks (Ising Models)

To demonstrate the soundness of the concept, we first generated 10×10 Ising models with increasing levels of σ = $\{1, 3, 5\}$. For each value for σ, we generated networks with determinism between 0% to 20% (at 2.5% intervals, 1000 graphs at each interval). On each graph we calculated the exact values of the single variable marginals; we then applied a variety of heuristics and ran LBP. Figure 2 shows

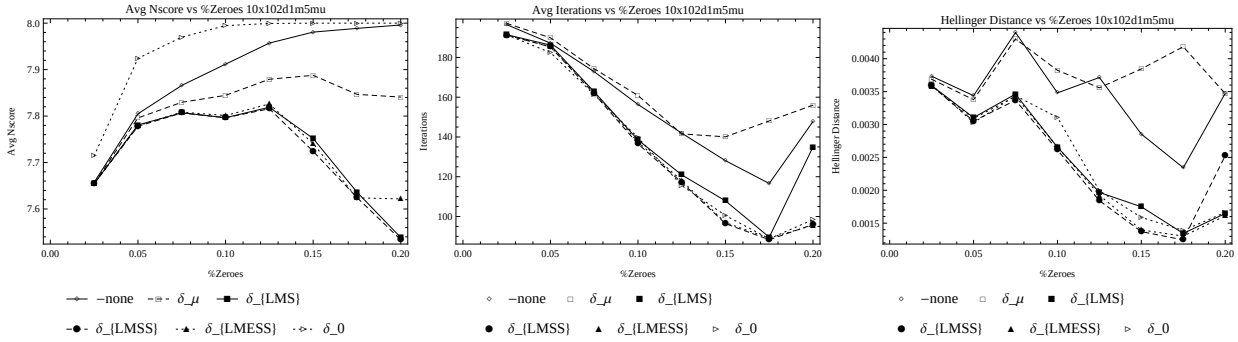


Figure 2: Nscore, iterations to convergence, and Hellinger distance for 10x10 grids of binary variables, generated from an exponential Gaussian distribution with $\sigma = 5$.

the results for networks generated with $\sigma = 5$. The results indicate a correlation between the Nscore of a graph and many desirable properties under LBP. Networks with a lower Nscore converge more often, in fewer iterations, and to more accurate single variable marginals. Thus, even if it is not possible to find an Nscore low enough to guarantee convergence for a particular graph, optimizing for its minimum value will still give a higher chance of obtaining an accurate approximation.

The heuristics that require optimization outperform those that do not by a substantial margin. In general, δ_{LMESS} yields the best results on these types of models and δ_{LMSS} is next best. This result is expected; these two heuristics require the most constraints of any the heuristics tested. The δ_0 heuristic is an anomaly. While it usually creates networks with poor Nscores, these networks generally yields good approximations, particularly as the value of σ increases. This might suggest that our use of $0/0 = 1$ in the calculation of the Nscore is incorrect. However, in many other varieties of networks, δ_0 performs poorly; in many cases, the application of δ_0 forces networks that converge under standard LBP to never converge.

We also calculate results for 40×40 grids generated via the same procedure, with $\sigma = 3$. These models are too large to run exact inference so we calculate only Nscore and iterations to convergence (shown in Figure 3). Because of the size of the model our algorithm would take a long time to finish its preprocessing step, so we limited our algorithm to 5 minutes for finding variable entries in the model, and 5 minutes to run its optimization routine. Although we cannot calculate the Hellinger distance from the true marginals, we observe similar improvements in Nscore and iterations to convergence, indicating that the heuristics are improving accuracy on models of this size as well.

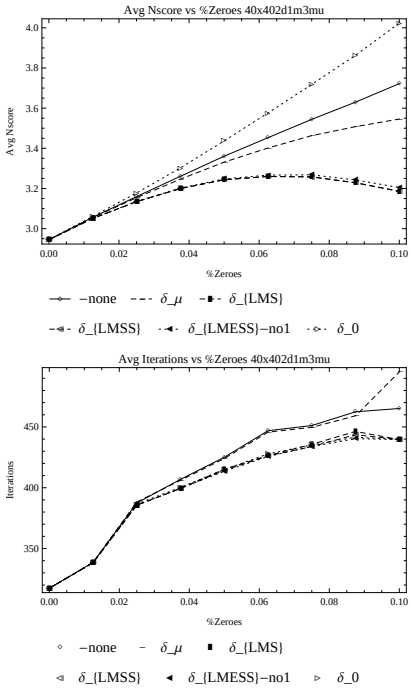


Figure 3: Nscore and iterations to convergence for 40x40 grids of binary variables, generated from an exponential Gaussian distribution with $\sigma = 3$.

6.2 Networks with Random Structure

We also applied our method to networks with random structure. We generated models with 100 binary-valued variables and 80 potentials, each with 3 variables in its scope, chosen at random from the uniform distribution. For these networks, we chose $\sigma = 3$. Row 1 of Figure 4 shows the results. The results here are similar to those found with the grid models. δ_0 performs well again, outperforming even the expensive optimizing heuristics. Again, in results with a smaller value of σ, the performance of δ_0 suffers.

6.3 3-SAT Problems

Because inference in graphical models can be viewed as an algorithm for weighted model counting [7], our algorithm can also be used to find approximate solution counts to satisfiability problems. We generated random 3-SAT problems in conjunctive normal form (100 variables, 380 to 480 clauses) and applied LBP to approximate their model

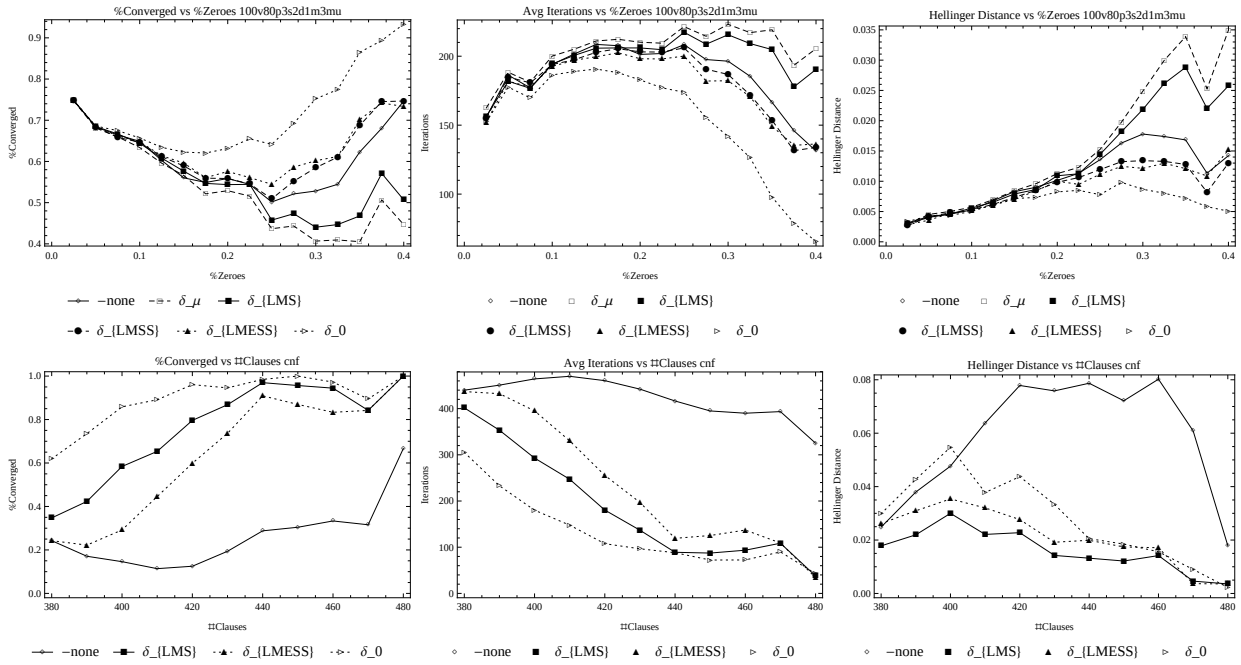


Figure 4: **Row 1:** Results for randomly generated Markov networks. The x-axis denotes the amount of determinism in the models. The y-axis shows (left to right) the ratio of converged graphs, the average iterations until convergence, and the Hellinger distance of the single variable marginal approximations from their true values.
Row 2: Results for randomly generated SAT instances. The x-axis denotes the number of clauses. The y-axis shows (from left to right) the ratio of converged graphs, the average iterations until convergence, and the Hellinger distance of the single variable marginal approximations from their true values.

counts. Although the problems have relatively few variables, they have large treewidth in general (on average, the treewidth found by the minfill heuristic is $\sim$67). We again limited our algorithm to a 5 minute time limit. We calculated the exact solutions using the Cachet model counting package [18]. Row 2 of Figure 4 shows the results.

We found that our heuristics yield the most significant improvement in those cases where the generated SAT instances have the highest uncertainty as to their satisfiability status. Roughly half the generated problems have a solution at 430 clauses, and the results show that it is in this neighborhood that the heuristics yield the greatest improvement. These problems are well known to be the hardest SAT problems to solve [2, 13]; it is not surprising that LBP, which relies on only arc-consistency to enforce hard constraints, struggles to give accurate approximations. Providing each potential the global satisfiability information allows the counting problem to be approximated much more accurately in these cases.

Note that we have excluded results for the δ_μ heuristic from the plots because it yields results almost identical to δ_{LMS}.

7 Conclusions and Future Work

In this paper, we proposed a general method for improving the convergence and accuracy of LBP in presence of determinism. Our method relies on finding alternate parameterizations of the given graphical model such that LBP converges faster on them than on the original graphical model. We proposed several heuristic methods that leverage extensive previous work on LBP convergence for finding such parameterizations and showed experimentally that our new method is superior to LBP.

Directions for future work include: the development of more uniformly successful heuristics across PGMs with different graph structures and potential strengths; and the application of the zero-finding method to other algorithms in the LBP family (e.g., max-product BP, IJGP [11], etc.).

Acknowledgments

This research was partly funded by ARO MURI grant W911NF-08-1-0242, by the AFRL under contract number FA8750-14-C-0021 and by the DARPA Probabilistic Programming for Advanced Machine Learning Program under AFRL prime contract number FA8750-14-C-0005. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, AFRL, ARO or the US government.

References

[1] Bengt Aspvall, Michael F Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 14(4):195, 1982.

[2] Peter Cheeseman, Bob Kanefsky, and William M Taylor. Where the really hard problems are. In *IJCAI*, volume 91, pages 331–337, 1991.

[3] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, March 1990.

[4] Rina Dechter and Robert Mateescu. A simple insight into iterative belief propagation's success. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 175–183. Morgan Kaufmann Publishers Inc., 2003.

[5] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient belief propagation for early vision. *International journal of computer vision*, 70(1):41–54, 2006.

[6] Marc PC Fossorier, Miodrag Mihaljevic, and Hideki Imai. Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *Communications, IEEE Transactions on*, 47(5):673–680, 1999.

[7] V. Gogate and P. Domingos. Formula-based probabilistic inference. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pages 210–219, 2010.

[8] Tom Heskes. On the uniqueness of loopy belief propagation fixed points. *Neural Computation*, 16(11):2379–2413, 2004.

[9] Alexander T Ihler, John W Fisher III, Alan S Willsky, and David Maxwell Chickering. Loopy belief propagation: convergence and effects of message errors. *Journal of Machine Learning Research*, 6(5), 2005.

[10] Steven G Johnson. The nlopt nonlinear-optimization package, 2010.

[11] R. Mateescu, K. Kask, V. Gogate, and R. Dechter. Iterative Join Graph Propagation algorithms. *Journal of Artificial Intelligence Research*, 37:279–328, 2010.

[12] Robert J. McEliece, David J. C. MacKay, and Jung-Fu Cheng. Turbo decoding as an instance of pearl's belief propagation algorithm. *Selected Areas in Communications, IEEE Journal on*, 16(2):140–152, 1998.

[13] David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of sat problems. In *AAAI*, volume 92, pages 459–465. Citeseer, 1992.

[14] Joris Mooij and Hilbert Kappen. Sufficient conditions for convergence of loopy belief propagation. *arXiv preprint arXiv:1207.1405*, 2012.

[15] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 467–475, 1999.

[16] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[17] Tanya Gazelle Roosta, Martin J Wainwright, and Shankar S Sastry. Convergence analysis of reweighted sum-product algorithms. *Signal Processing, IEEE Transactions on*, 56(9):4293–4305, 2008.

[18] Tian Sang, Fahiem Bacchus, Paul Beame, Henry A Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. *SAT*, 4:7th, 2004.

[19] Tian Sang, Paul Beame, and Henry Kautz. Solving bayesian networks by weighted model counting. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, volume 1, pages 475–482, 2005.

[20] Niklas Sorensson and Niklas Een. Minisat v1. 13-a sat solver with conflict-clause minimization. *SAT*, 2005:53, 2005.

[21] Sekhar C Tatikonda and Michael I Jordan. Loopy belief propagation and gibbs measures. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 493–500. Morgan Kaufmann Publishers Inc., 2002.

[22] Chen Yanover and Yair Weiss. Finding the ai most probable configurations using loopy belief propagation. *Advances in Neural Information Processing Systems*, 16:289, 2004.

[23] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *Information Theory, IEEE Transactions on*, 51(7):2282–2312, 2005.