Revised version submitted to *Neural Computation*

Correctness of Local Probability Propagation in Graphical Models with Loops

Yair Weiss

Dept. of Brain and Cognitive Sciences

MIT E10-120, Cambridge, MA 02139

yweiss@psyche.mit.edu

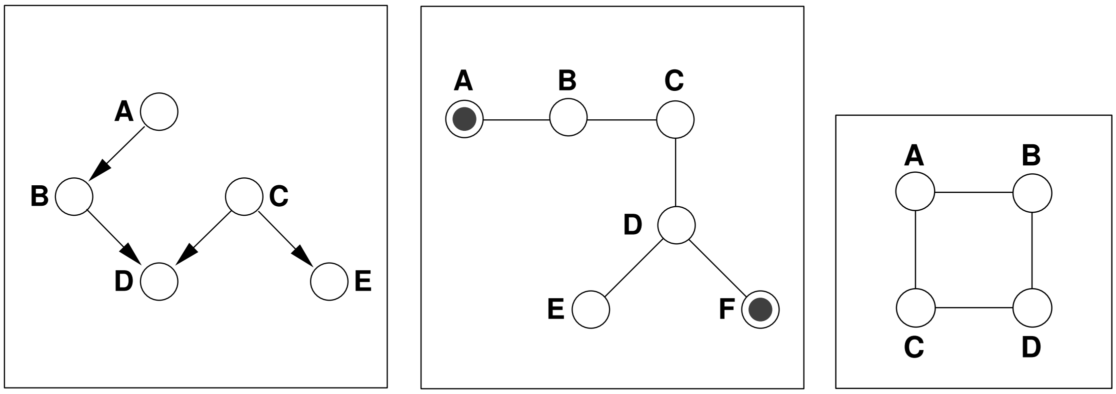
July 31 1998

Abstract

Graphical models, such as Bayesian networks and Markov networks, represent joint distributions over a set of variables by means of a graph. When the graph is singly connected, local propagation rules of the sort proposed by Pearl (1988) are guaranteed to converge to the correct posterior probabilities. Recently, a number of researchers have empirically demonstrated good performance of these same local propagation schemes on graphs with loops, but a theoretical understanding of this performance has yet to be achieved.

For graphical models with a single loop, we derive an analytical relationship between the probabilities computed using local propagation and the correct marginals. Using this relationship we show a category of graphical models with loops for which local propagation gives rise to provably optimal MAP assignments (although the computed marginals will be incorrect). We also show how nodes can use local information in the messages they receive in order to correct their computed marginals.

We discuss how these results can be extended to graphical models with multiple loops and show simulation results suggesting that some properties of propagation on single loop graphs may hold for a larger class of graphs. Specifically we discuss the implication of our results for understanding a class of recently proposed error-correcting codes known as "Turbo codes".



a b c

Figure 1: Examples of graphical models. Nodes represent variables and the qualitative aspects of the joint probability function are represented by properties of the graph. Shaded nodes represent observed variables. **a.** A singly connected Bayesian network. **b.** A singly connected Markov network (A and F are observed). **c.** A Markov network with a loop. This paper analyzes the behavior of local propagation rules in graphical models with a loop.

1 Introduction

Problems involving probabilistic belief propagation arise in a wide variety of applications including error correcting codes, speech recognition and medical diagnosis. Typically, a probability distribution is assumed over a set of variables and the task is to infer the values of the unobserved variables given the observed ones. The assumed probability distribution is described using a graphical model (Lauritzen, 1996) — the qualitative aspects of the distribution are specified by a graph structure. Figure 1 shows two examples of such graphical models, a Bayesian network (Pearl, 1988; Jensen, 1996) (Figure 1a) and Markov networks (Pearl, 1988; Geman and Geman, 1984) (Figure 1b-c).

If the graph is singly connected (i.e. there is only one path between any two given nodes) then there exist efficient local message- passing schemes to calculate the posterior probability of an unobserved variable given the observed variables. Pearl (1988) derived such a scheme for singly connected Bayesian networks and showed that this "belief propagation" algorithm is guaranteed to converge to the correct posterior probabilities (or "beliefs"). However, as Pearl noted, the same algorithm will not give the correct beliefs for multiply connected networks:

> When loops are present, the network is no longer singly connected and local propagation schemes will invariably run into trouble ... If we ignore the existence of

> loops and permit the nodes to continue communicating with each other as if the network were singly connected, messages may circulate indefinitely around the loops and the process may not converge to a stable equilibrium ... Such oscillations do not normally occur in probabilistic networks ... which tend to bring all messages to some stable equilibrium as time goes on. However, this asymptotic equilibrium is not coherent, in the sense that it does not represent the posterior probabilities of all nodes of the network (Pearl 1988, p. 195)

Despite these reservations, Pearl advocated the use of belief propagation in loopy networks as an approximation scheme (J. Pearl, personal communication) and one of the exercises in (Pearl, 1988) investigates the quality of the approximation when it is applied to a particular loopy belief network.

Several groups (Frey, 1998; Mackay and Neal, 1995; Weiss, 1996) have recently reported excellent experimental results by using this approximation scheme — by running algorithms equivalent to Pearl's algorithm on networks with loops. Perhaps the most dramatic instance of this performance is in an error correcting code scheme known as "Turbo Codes" (Berrou et al., 1993). These codes have been described as "the most exciting and potentially important development in coding theory in many years" (McEliece et al., 1995) and have recently been shown (Wiberg, 1996; Kschischang and Frey, 1998; McEliece et al., 1998) to utilize an algorithm equivalent to belief propagation in a network with loops. Although there is widespread agreement in the coding community that these codes "represent a genuine, and perhaps historic, breakthrough" (McEliece et al., 1995) a theoretical understanding of their performance has yet to be achieved.

Here we lay a foundation for an understanding of belief propagation in networks with loops. For networks with a single loop we derive an analytic expression relating the steady-state beliefs to the correct posterior probabilities. We discuss how these results can be extended to networks with multiple loops and show simulation results for networks with multiple loops. Specifically we discuss the implication of our results for understanding the performance of Turbo codes.

2 Formulation – Bayes nets, Markov nets and belief propagation

We begin by briefly reviewing the concepts of Bayesian and Markov networks. The reader is referred to chapter 3 of (Pearl, 1988) for a more exhaustive treatment.

Bayesian networks and Markov networks are graphs that represent the qualitative nature of a distribution over a set of variables — the structure of the graph implies a product form for the distribution.

A Bayesian Network is a *directed* acyclic graph in which the nodes represent variables, the arcs signify the existence of direct influences between the linked variables and the strength of these influences are expressed by forward conditional probabilities. Using the chain rule, the full probability distribution over the variables can be expressed as a product of conditional and prior probabilities. Thus for example for the network in Figure 1a we can write:

$$P(ABCDEF) = P(A)P(B|A)P(C)P(D|B,C)P(E|C) \tag{1}$$

A Markov network is an *undirected* graph in which the nodes represent variables and arcs represents compatability constraints between them. Assuming all probabilities are nonzero, the Hammersley-Clifford theorem (e.g. (Pearl, 1988)) guarantees that the probability distribution will factorize into a product of functions of the maximal cliques of the graph. Thus for example, in the network in Figure 1c we have:

$$P(ABCD) = \frac{1}{Z}\Psi(AB)\Psi(BC)\Psi(CD)\Psi(DA) \tag{2}$$

Different communities tend to prefer different graphical models (see (Smyth, 1997) for a recent review) — directed graphs are more common in AI, medical diagnosis and statistics while undirected graphs are more common in image processing, statistical physics and error correcting codes. In both cases, however, a full specification of the probability distribution involves specifying (1) the graph and (2) numerical values for the conditional probabilities (in Bayes nets) or the clique potentials (in Markov nets).

Given a full specification of the probability, the *inference* task in graphical models is simply to infer the values of the unobserved variables given the observed ones. Figure 1b shows an example. The observed nodes A, F are represented by shaded circles in the graph. We use $\mathcal{O}$ to denote the set of observed variables. There are actually three distinct subtasks for the inference problems:

- Marginalization. Calculating the marginal probability of a variable given the observed variables. For example $P(C = c|\mathcal{O})$.
- Maximum a posteriori assignment (MAP assignment). Finding assignments to the unobserved variables that are most probable given the observed variables. For example, find (b, c, d, e) such that $P(B = b, C = c, D = d, E = e|\mathcal{O})$ is maximized.
- Maximum marginal assignment (MM assignment). Finding assignments to the unobserved variables that maximize the marginal probability of the assignment. For example, (b, c, d, e) such that $P(B = b|\mathcal{O})$,$P(C = c|\mathcal{O})$,$P(D = d|\mathcal{O})$,$P(E = e|\mathcal{O})$ are all maximized.

2.1 Message-passing algorithms for Graphical models

For singly connected graphical models, there exist various message passing schemes for performing inference (see (Smyth et al., 1997) for a review). Here we present such a scheme for pairwise Markov nets — ones in which the maximal cliques are pairs of units. This choice is not as restrictive as it may seem. First, any singly connected Markov net is a pairwise Markov network and a Markov network with larger cliques can be converted into a pairwise Markov network by merging larger cliques into cluster nodes. Furthermore, as we show in the appendix any Bayesian network can be converted into a pairwise Markov net. When this conversion is performed, the update rules given here reduce to Pearl's original algorithm for inference in Bayesian networks. The main advantage of the pairwise Markov net formulation is that it enables us to write the message-passing scheme in terms of matrix and vector operations, and this makes the subsequent analysis simpler.

To derive the message-passing algorithm, consider first the naive way of performing inference — by exhaustive enumeration. Referring again to Figure 1b, calculating the marginal of B merely involves computing:

$$P(C = c|A = a, F = f) = \alpha \sum_{b,d,e} P(a, b, c, d, e, f) \qquad (3)$$

where α is a normalizing factor.

This exhaustive enumeration is of course exponential in the number of unobserved nodes, but for the particular factorized distributions of Markov nets we can write:

$$P(C=c|A=a,F=f) = \alpha \sum_c \sum_d \sum_e \Psi(ab)\Psi(bc)\Psi(cd)\Psi(df)\Psi(de) \quad (4)$$

$$= \alpha \left(\sum_b \Psi(ab)\Psi(bc) \right) \left(\sum_d \Psi(cd)\Psi(df) \sum_e \Psi(de) \right) \quad (5)$$

Note that the exponential enumeration is now converted into a series of enumerations over each variable separately and that many of the terms (e.g. $\sum_b \Psi(ab)\Psi(bc)$) are equivalent to matrix multiplication. This forms the basis for the message-passing algorithm.

The messages that nodes transmit to each other are vectors, and we denote by $\vec{v}_{XY}$ the message that node X sends to node Y. We define the "transition matrix" of an edge in the graph by:

$$M_{XY}(i,j) \stackrel{\text{def}}{=} \Psi(X=i, Y=j) \quad (6)$$

Note that the matrix going in one direction on an edge is equal by definition to the transpose of the matrix going in the other direction: $M_{XY} = M_{YX}^T$. We denote by $\vec{b}_X$ the belief vector at node X.

Following the terminology of Pearl (88) we call the message-passing scheme for calculating marginals *belief update*. In belief update, the message that node X sends to node Y is updated as follows:

- Combine all messages coming into X except for that coming from Y into a vector $\vec{v}$. The combination is done by multiplying all the message vectors element by element.
- Multiply $\vec{v}$ by the matrix M_{XY} corresponding to the link from X to Y.
- Normalize the product $M_{XY}\vec{v}$ so it sums to 1. The normalized vector is sent to Y.

The belief vector for a node X is obtained by combining all incoming messages to X (again by multiplying the message vectors element by element) and normalizing.

By introducing a symbol $\odot$ for the componentwise multiplication operator the updates can be rewritten:

$$\vec{v}_{XY} \leftarrow \alpha M_{XY} \bigodot_{Z \in N(X) \backslash Y} \vec{v}_{ZX} \tag{7}$$

$$\vec{b}_X \leftarrow \alpha \bigodot_{Z \in N(X)} \vec{v}_{ZX} \tag{8}$$

where $\vec{z} = \vec{x} \odot \vec{y} \leftrightarrow \vec{z}(i) = \vec{x}(i)\vec{y}(i)$. Throughout this paper $\alpha\vec{v}$ denotes normalizing the components of $\vec{v}$ so they sum to 1.

The procedure is initialized with all message vectors set to $(1, 1, \cdots 1)$. Observed nodes do not receive messages and they always transmit the same vector — if X is observed to be in state x then $\vec{v}_{XY}(i) = \Psi(X = x, Y = i)$. The normalization of $\vec{v}_{XY}$ in equation 7 is not theoretically necessary — whether the message are normalized or not, the belief vector $\vec{b}_X$ will be identical. Thus one could use an equivalent algorithm where the messages are not normalized. However, as Pearl has pointed out, normalizing the messages avoids numerical underflow and adds to the stability of the algorithm. Equation 7 does not specify the order in which the messages are updated. For simplicity, we assume throughout this paper that all nodes simultaneously update their messages in parallel.

For a singly connected network, the belief update equations (Equations 7–8) solve two of the three inference tasks mentioned earlier. It can be shown that the the belief vectors reach a steady state after a number of iterations equal to the longest path in the graph. Furthermore, at convergence the belief vectors are equal to the posterior marginal probabilities. Thus for singly connected networks: $\vec{b}_X(j) = P(X = j|\mathcal{O})$.

We define the belief update assignment (BU assignment) by assigning each variable X to the state j that maximizes $\vec{b}_X(j)$. Since $\vec{b}_X$ converges to the correct posteriors for a singly connected network, the BU assignment is guaranteed to give the MM assignment. Thus two of the three inference tasks mentioned earlier can be solved using the belief update procedure.

To calculate the MAP assignment, however, calculating the posterior marginals is insufficient. In order to calculate the MAP assignment, the nodes need to calculate the posterior when the other unobserved nodes are maximized rather than summed. In Figure 1 this corresponds to calculating:

$$P^*(C = c|A = a, F = f) = \alpha \max_{b,d,e} P(a, b, c, d, e, f) \tag{9}$$

(we use the notation $\P^*$ to denote the fact that P^* does not correspond to a marginal probability).

In complete analogy to Equation 4 the maximization over the unobserved variables can be replaced by a series of maximizations over individual variables. This leads to an alternative message-passing scheme, that we call, following (Pearl, 1988), *belief revision.*

The procedure is almost identical to that described earlier for belief update. The only difference is that the matrix multiplication in equation 7 needs to be replaced with a nonlinear matrix operator. For a matrix M and a vector $\vec{x}$ we define the operator $\overset{\infty}{M}$ such that $\vec{y} = \overset{\infty}{M} \vec{x}$ if:

$$\vec{y}(i) = \max_j M(i,j)\vec{x}(j) \tag{10}$$

Note that the $\overset{\infty}{M}$ operator is similar to regular matrix multiplication, with the sum replaced with the maximum operator.

The belief revision update rules are:

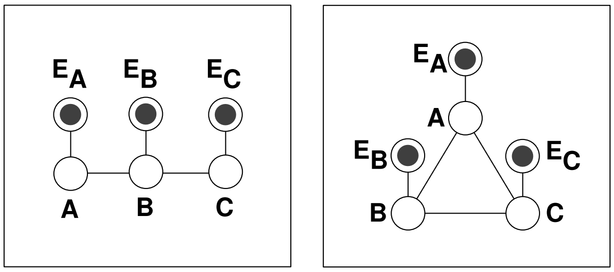
$$\vec{v}_{XY} \leftarrow \alpha \overset{\infty}{M}_{XY} \bigodot_{Z \in N(X) \backslash Y} \vec{v}_{ZX} \tag{11}$$

$$\vec{b}_X \leftarrow \alpha \bigodot_{Z \in N(X)} \vec{v}_{ZX} \tag{12}$$

with initialization identical to belief update.

Again, it can be shown that for singly connected networks the belief vector $\vec{b}_X(i)$ at a node will converge to the modified belief $BEL^*(X = i)$. We define the belief revision (BR) assignment as assigning each node X the value j that maximizes $\vec{b}_X(j)$ after belief revision has converged. From the definition of BEL^* it follows that the BR assignment is equal to the MAP assignment for singly connected networks. Thus all three inference tasks can be accomplished by local message passing.

Update rules such as Equations 7–8 and Equations 11–12 have appeared in many areas of applied mathematics and optimization. In the hidden Markov model literature, the belief update equations (Equations 7–8) are equivalent to the forward-backward algorithm, while the belief revision equations 11–12 are equivalent to the Viterby algorithm (Rabiner, 1989). In a general optimization context, equation 11–12 are a distributed implementation of standard dynamic programming (Bertsekas, 1987). If the nodes represent continuous Gaussian random



a b

Figure 2: The intuition behind loopy belief propagation. In singly connected networks (as in **a.**), the local update rules avoid double counting of evidence. In multiply connected networks, (e.g. **b.**) double counting is unavoidable. However, as we show in the text, certain structures lead to equal double counting.

variables, equations 7–8 are equivalent to the Kalman filter and optimal smoothing (Gelb, 1974). In error-correcting codes, both belief revision and belief update can be thought of as special cases of a general class of decoding algorithms for codes defined on graphs (Kschischang and Frey, 1998; Forney, 1997; Aji and McEliece, 1998).

In nearly all the contexts surveyed above, belief propagation has only been analyzed for singly connected graphs. Note, however, that these procedures are perfectly well defined for any pairwise Markov network. This raises questions including:

- How far is the steady-state belief from the correct posterior when the update rules (Equations 7–8) are applied in a loopy network?
- What are the conditions under which the BU assignment equals the MM assignment when the update rules are applied in a loopy network?
- What are the conditions under which the BR assignment equals the MAP assignment when the update rules are applied in a loopy network?

Answering these questions is the goal of this paper.

3 Intuition - why does loopy propagation work?

Before launching into the details of the analysis, let us consider the examples in Figure 2 and obtain some intuition. In both graphs, there is an observed node connected to every unobserved nodes and we will refer to the observed node connected to X as the local evidence at X. Intuitively, the message-passing algorithm can be thought of as a way of communicating local evidences between nodes such that all nodes calculate their beliefs given all the evidence.

As Pearl has pointed out, in order for a message-passing scheme to be successful, it needs to avoid "double counting" – a situation in which the same evidence is passed around the network multiple times and mistaken for new evidence. In a singly connected network (e.g. Figure 2a) this is accomplished by update rules such as those in Equation 7. Thus in Figure 2a, node B will receive from A a message that involves the local evidence at A and send that information to C. The message it sends to A will involve the local evidence at C but *not* the local evidence at A. Thus A never receives its own evidence back again, and double counting is avoided.

In a loopy graph (e.g. 2b) double counting cannot be avoided. Thus B will send A's evidence to C, but in the next iteration, C will send that same information back to A. Thus it seems that belief propagation in such a net will invariably give the wrong answer. How then, can we explain the good performance reported experimentally?

Intuitively, the explanation is that "double counting" may still lead to correct inference if all evidence is "double counted" in equal amounts. Because nodes mistake existing evidence for new evidence, they are overly confident in their beliefs regarding which values should be assigned to the unobserved variables. However, if all evidence is equally double counted, the assignment, although based on overly confident beliefs, may still be correct. Indeed, as we show in this paper, this intuition can be formalized for belief revision. For all networks with a single loop, the BR assignment gives the MAP assignment even though the numerical values of the beliefs are wrong.

The notion of "equal double counting" can be formalized by means of what we call the "unwrapped network" corresponding to a loopy network. The unwrapped network is a singly connected network constructed such that performing belief propagation in the unwrapped network is equivalent to performing belief propagation in the loopy network. The exact construction method of the unwrapped network is discussed in section 5, but the basic idea is to replicate the nodes and the transition matrix as shown in the examples in Figure 3. As we show, (see also (Weiss, 1996; Mackay and Neal, 1995; Wiberg, 1996; Frey et al., 1998)), for any number of iterations of belief propagation in the loopy network, there exists an unwrapped network such that the final messages received by a node in the unwrapped network are equivalent to those that would be received by a corresponding node in the loopy network. This concept is illustrated in Figure 3. The messages received by node B after $1, 2$ and 3 iterations of propagation in the loopy network are identical to the final messages received by node B in the unwrapped networks shown in Figure 3.

It seems that all we have done is convert a finite, loopy problem into an infinite network without loops. What have we gained? The importance of the unwrapped network is that since it is singly-connected, belief propagation on it is *guaranteed to give the correct beliefs.* Thus belief propagation on the loopy network gives the correct answer for the unwrapped network. The usefulness of this estimate now depends on the similarity between the probability distribution induced by the unwrapped problem and the original loopy problem.

In subsequent sections we formally compare the probability distribution induced by the loopy network to that induced by the original problem. Roughly speaking, if we denote the original probability induced on the hidden nodes in Figure 1b by:

$$P(a, b, c) = \alpha e^{-J(a,b,c)} \tag{13}$$

then MAP assignment for the unwrapped problem maximizes:

$$\tilde{P}(a, b, c) = \alpha e^{-kJ(a,b,c)} \tag{14}$$

for some positive constant k. That is, if we are considering the probability of assignment the unwrapped problem induces a probability which is a monotonic transformation of the true probability. In statistical physics terms, the unwrapped network has the same energy function but at different temperature. This is the intuitive reason that *belief revision* which finds the MAP assignment in the unwrapped network, also finds the MAP assignment in the loopy network. However, *belief update* which finds marginal distributions of particular nodes may not find the MM assignment — the marginals of $\tilde{P}$ are not necessarily a monotonic transformation of the marginals of P.

Since every iteration of loopy propagation gives the correct beliefs for a different problem, it is not immediately clear why this scheme should ever converge. Note, however, that the

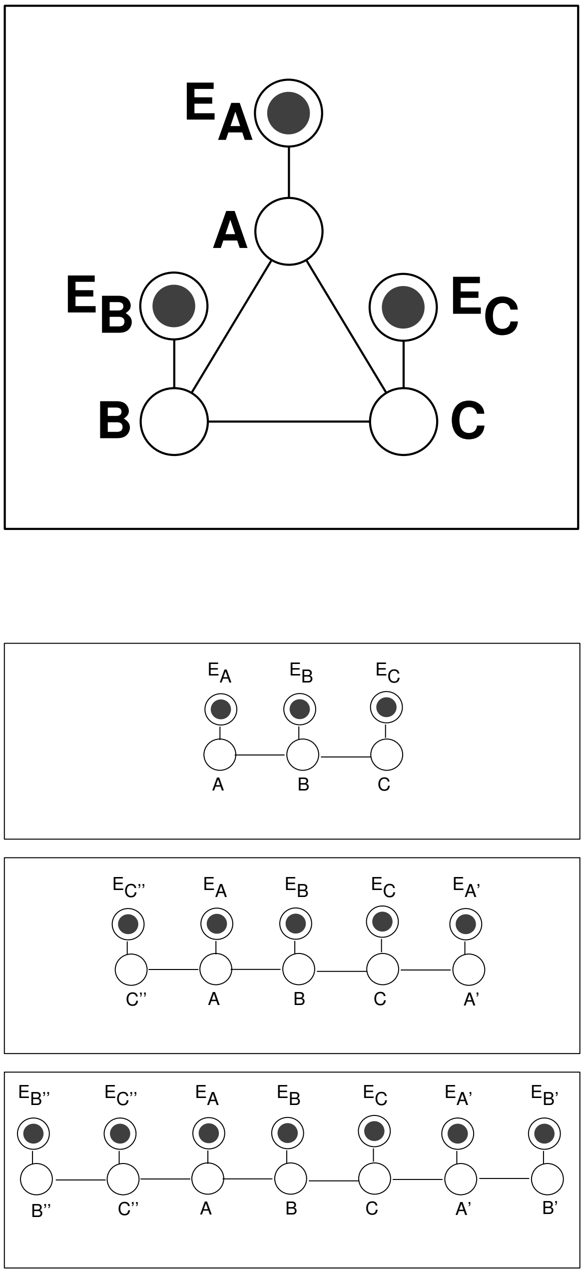


Figure 3: Top: A simple loop figure. **Bottom:** Unwrapped networks corresponding to the loopy network shown on top. The unwrapped networks are constructed by replicating the evidence and the transition matrices while preserving the local connectivity of the loopy network. They are constructed so that the messages received by node B after t iterations in the loopy network are equivalent to those that would be received by B in the unwrapped network. The unwrapped networks for the first three iterations are shown.

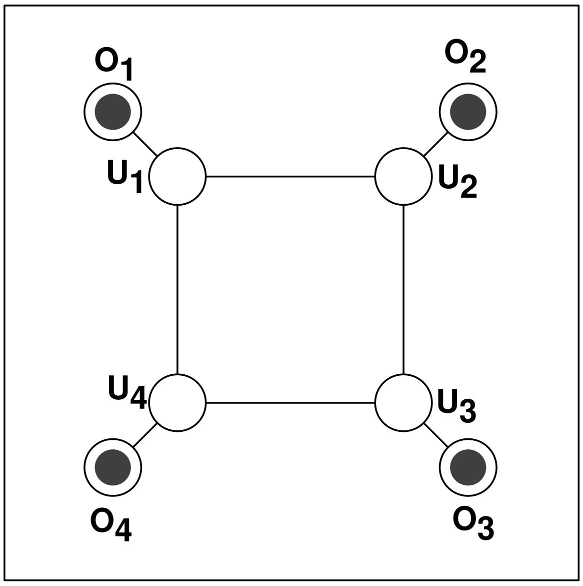


Figure 4: A simple network with a single loop. We label the unobserved nodes $U_1 \cdots U_N$ and the observed nodes $O_1 \ldots O_N$. In this section we derive an analytical relationship between the belief vector at each node in the loop (e.g. $\vec{b}_{U_1}$) and its correct marginal probability (e.g. $\vec{p}_{U_1}$).

unwrapped network of iteration $t+1$ is simply the unwrapped network of size t with an additional finite number of nodes added at the boundary. Thus loopy belief propagation will converge when the addition of these additional nodes at the boundary will not alter the posterior probability of the node in the center. In other words, convergence is equivalent to the independence of center nodes and boundary nodes in the unwrapped network. In the subsequent sections, we formalize this intuition.

4 Belief update in networks with a single loop

Consider a network composed of N unobserved nodes arranged in a loop and N observed nodes, one attached to each unobserved node (see Figure 4). We denote by $U_1, U_2 \cdots U_N$ the unobserved variables, and $O_1, O_2 \cdots O_N$ the corresponding observed variables. In this section we derive a relationship between the belief at a given node $\vec{b}_{U_1}$ and the correct posterior probability which we denote by $\vec{p}_{U_1}$.

First, let us find what the messages received by node U_1 converge to. By the belief update equations (equations 7), the message that U_N sends to U_1 depends on the message U_N receives from U_{N-1}:

$$\vec{v}_{U_N U_1} \leftarrow \alpha M_{U_N U_1} \left(\vec{v}_{O_N U_N} \odot \vec{v}_{U_{N-1} U_N}\right) \tag{15}$$

and similarly the message U_{N-1} sends to U_N depends on the message U_{N-1} receives from U_{N-2}:

$$\vec{v}_{U_{N-1} U_N} \leftarrow \alpha M_{U_{N-1} U_N} \left(\vec{v}_{O_{N-1} U_{N-1}} \odot \vec{v}_{U_{N-2} U_{N-1}}\right) \tag{16}$$

We can continue expressing each message in terms of the one received from the neighbor until we go back in the loop to U_1:

$$\vec{v}_{U_1 U_2} \leftarrow \alpha M_{U_1 U_2} \left(\vec{v}_{O_1 U_1} \odot \vec{v}_{U_N U_1}\right) \tag{17}$$

Thus the message U_N sends to U_1 at a given time step depends on the message U_N sent to U_1 at a previous time step (N time steps ago):

$$\vec{v}^{(t+N)}_{U_N U_1} = \alpha C_{N1} \vec{v}^{(t)}_{U_N U_1} \tag{18}$$

where the matrix C_{N1} summarizes all the operations performed on the message as it is passed around the loop:

$$C_{N1} = M_{U_N U_{N-1}} D_N M_{U_{N-1} U_{N-2}} D_{N-1} \cdots M_{U_1 U_2} D_1 \tag{19}$$

and we define the matrix D_i to be a diagonal matrix whose elements are the constant messages sent from observed node O_i to unobserved U_i. A diagonal matrix is used because it enables us to rewrite the componentwise multiplication of two vectors as a multiplication of a matrix and a vector. For any vector $\vec{x}$, $D_i \vec{x} = \vec{v}_{O_i U_i} \odot \vec{x}$.

Using the matrix C_{N1} we can formally state the claims of this section.

Claims: Consider a pairwise Markov network consisting of a single loop of unobserved variables $\{U_i\}_{i=1}^N$ each of which is connected to an observed variable O_i. Define C_{N1} as in equation 19 where D_i a diagonal matrix whose entries are the messages $\vec{v}_{O_i U_i}$. If all elements of C_{N1} are nonzero then:

1. $\vec{v}_{U_N U_1}$ converges to the principal eigenvector of C_{N1}
2. $\vec{v}_{U_2 U_1}$ converges to the principal eigenvector of $D_1^{-1} C_{N1}^T D_1$.
3. The convergence rate of the messages is governed by the ratio of the largest eigenvalue of C_{N1} to the second largest eigenvalue.

4. The diagonal elements of C_{N1} give the correct posteriors: $\vec{p}_{U_1}(i) = \alpha C_{N1}(i,i)$.

5. The steady-state belief $\vec{b}_{U_1}$ is related to the correct posterior marginal $\vec{p}_{U_1}$ by: $\vec{b}_{U_1} = \beta \vec{p}_{U_1} + (1-\beta)\vec{q}_{U_1}$ where β is the ratio of the largest eigenvalue of C_{N1} to the the sum of all eigenvalues and $\vec{q}_{U_1}$ depends on the eigenvectors of C_{N1}.

The first claim follows from the recursion (equation 18). Recursions of this type form the basis of the power method for finding eigenvalues of matrices (Strang, 1986). The method is based on the following lemma.

Power method lemma: (Strang, 1986) Let C be a matrix with eigenvalues $\lambda_1 \ldots \lambda_n$ sorted by decreasing magnitude and eigenvectors $\vec{u_1} \ldots \vec{u_n}$. If $|\lambda_1| > |\lambda_2|$, then the recursion $\vec{x}^{(t+1)} = \alpha C \vec{x}^{(t)}$ converges to a multiple of $\vec{u_1}$ from any initial vector $\vec{x}^{(0)} = \sum_i \beta_i \vec{u}_i$ s.t. $\beta_1 \neq 0$. The convergence factor r is given by $r = |\lambda_2/\lambda_1|$ (the distance to the steady-state vector decreases by $r\%$ at every iteration).

If $\vec{v}$ is a fixed point for the recursion (equation 18) then $\vec{v} = \alpha C_{N1} \vec{v}$ or $C_{N1}\vec{v} = 1/\alpha \vec{v}$ so that $\vec{v}$ is an eigenvector of C_{N1}. The power method lemma guarantees that, in general, the method will converge to the principal eigenvector, i.e. the one with largest eigenvalue.

It is not trivial to check for a given matrix and initial condition whether the assumptions of the power method lemma are satisfied. A sufficient (but not necessary) condition is that all elements of the matrix C and the initial vector $\vec{x}^{(0)}$ be positive. The Perron-Frobenius theorem (Minc, 1988) guarantees that for such matrices $\lambda_1 > \lambda_2$ and that C will have only one eigenvector whose elements are non-negative, namely $\vec{u}_1$. Thus for C with positive elements, the iterations will converge to $\alpha \vec{u_1}$ for any initial vector $\vec{x}^{(0)}$ with positive elements. If the matrix contains non-negative values, conditions for guaranteed convergence from arbitrary initial condition are more complicated. (Minc, 1988).

In the context of graphical models, the matrix C_{N1} is a product of matrices with non-negative elements (the elements are either conditional probabilities or potential functions) but it may contain zeros. Indeed, it is possible to construct examples of graphical models for which the messages do not converge. These examples, however, are extremely rare (e.g. when all the transition matrices are deterministic).

To summarize, if the matrix C_{N1} has only positive elements, then $\vec{v}_{U_N U_1}$ converges to the principal eigenvector of C_{N1}. $\Box$

To prove the second claim, we define:

$$C_{21} = M_{U_2U_1} D_2 M_{U_3U_2} D_3 \cdots M_{U_1U_N} D_1 \tag{20}$$

and in complete analogy to the previous proof, this defines a recursion for $\vec{v}_{U_2U_1}$ and therefore the steady-state message is the principal eigenvector of C_{21}. Furthermore, C_{21} can be expressed as:

$$C_{21} = D_1^{-1} C_{N1}^T D_1 \tag{21}$$

where we have used the fact that for any two nodes X, Y $M_{XY} = M_{YX}^T$. $\Box$

The third claim, regarding the convergence rate of the messages also follows from the power method lemma. Thus the convergence rate of $\vec{v}_{U_NU_1}$ is governed by the ratio of the largest eigenvalue to the second largest eigenvalue of C_{N1} and that of $\vec{v}_{U_2U_1}$ by the ratio of eigenvalues of C_{21}. Furthermore, by equation 21, C_{21} and C_{N1} have the same eigenvalues. $\Box$

To prove the fourth claim, we denote by $\vec{e}_i$ the vector that is zero everywhere except for a 1 at the ith component then:

$$\begin{aligned}
\vec{p}_{U_1}(i) &= \alpha \sum_{U_2 \cdots U_N} P(U_1 = i, U_2, \cdots U_N) && (22)\\
&= \alpha \sum_{U_2 \cdots U_N} \Psi(i, U_2)\Psi(i, O_1)\Psi(U_2, U_3) && (23)\\
&\quad \Psi(U_2, O_2) \cdots \Psi(U_N, i)\Psi(U_N, O_N) \\
&= \alpha \sum_{U_2} \Psi(i, U_2)\Psi(U_2, O_2) \sum_{U_3} \Psi(U_2, U_3) && (24)\\
&\quad \cdots \sum_{U_N} \Psi(U_N, O_N)\Psi(U_N, i)\Psi(i, O_1) \\
&= \alpha \vec{e}_i^T M_{U_2U_1} D_2 M_{U_3U_2} D_3 \cdots M_{U_1U_N} D_1 \vec{e}_i && (25)\\
&= \alpha \vec{e}_i^T C_{21} \vec{e}_i && (26)\\
&= \alpha C_{21}(i, i) && (27)
\end{aligned}$$

Note that we can also calculate the normalizing factor α in terms of the matrix C_{21}, thus:

$$\vec{p}_{U_1}(i) = \frac{C_{21}(i, i)}{trace(C_{21})} \tag{28}$$

Again, by equation 21 C_{21} and C_{N1} have the same diagonal elements so $\vec{p}_{U_1}(i) = \alpha C_{N1}(i,i)$. □

To prove the fifth claim, we denote by $\{\vec{w}_i, \lambda_i\}$ the eigenvectors and eigenvalues of the matrix C_{N1} and by $\{\vec{s}_i, \lambda_i\}$ the eigenvectors and eigenvalues of matrix C_{21}. Now, recall from the belief update rules 7 that:

$$\vec{b}_{U_1} = \alpha \vec{v}_{U_2 U_1} \odot \vec{v}_{U_N U_1} \odot \vec{v}_{O_1 U_1} \tag{29}$$

Using the first two claims, we can rewrite this in terms of the diagonal matrix D_1 and the principal eigenvectors of C_{21} and C_{N1}:

$$\vec{b}_{U_1} = \alpha D_1 \vec{s_1} \odot \vec{w}_1 \tag{30}$$

Or in component notation:

$$\vec{b}_{U_1}(i) = \alpha D_1(i,i) \vec{s_1}(i) \vec{w}_1(i) \tag{31}$$

However, to obtain an expression analogous to equation 28 we want to rewrite $\vec{b}_{U_1}$ solely in terms of C_{21}.

First we write $C_{21} = S \Lambda S^{-1}$ where the columns of S contain the eigenvectors of C_{21} and the diagonal elements of Λ contain the corresponding eigenvalues. We order S such that the principal eigenvector is in the first column, thus $S(i,1) = \beta \vec{s}_1(i)$. Surprisingly, the first *row* of S^{-1} is related to the product $D_1 \vec{w}_1$, where $\vec{w}_1$ is the principal eigenvector of C_{N1}:

$$C_{21} = S \Lambda S^{-1} \tag{32}$$

$$C_{N1} = D_1^{-1} C_{21}^T D_1 = D_1^{-1} (S^{-1})^T \Lambda S^T D_1 \tag{33}$$

Thus the first row of $S^{-1} D_1^{-1}$ gives the principal eigenvector of C_{N1} or:

$$D_1(i,i) \vec{w}_1(i) = \gamma S^{-1}(1,i) \tag{34}$$

where the constant γ is independent of of i. Substituting into equation 31 gives:

$$\vec{b}_{U_1}(i) = \alpha S_(i,1) S^{-1}(1,i) \tag{35}$$

where α is again a normalizing constant. In fact, this constant is equal to unity since for any invertible matrix S:

$$\sum_i S(i,1) S^{-1}(1,i) = 1 \tag{36}$$

Finally, we can express the relationship between $\vec{p}_{U_1}$ and $\vec{b}_{U_1}$:

$$\vec{p}_{U_1}(i) = \frac{\vec{e}_i^T C_{21} \vec{e}_i}{trace(C_{21})} \tag{37}$$

$$= \frac{\vec{e}_i^T S \Lambda S^{-1} \vec{e}_i}{\sum_j \lambda_j} \tag{38}$$

$$= \frac{\sum_j S(i,j)\lambda_j P^{-1}(j,i)}{\sum_j \lambda_j} \tag{39}$$

$$= \frac{\lambda_1 \vec{b}_{U_1}(i) + \sum_{j=2} S(i,j)\lambda_j S^{-1}(j,i)}{\sum_j \lambda_j} \tag{40}$$

Thus $\vec{p}_{U_1}$ can be written as a weighted average of $\vec{b}_{U_1}$ and a second term, which we will denote by $\vec{q}_{U_1}$

$$\vec{p}_{U_1} = \frac{\lambda_1}{\sum_j \lambda_j}\vec{b}_{U_1} + (1 - \frac{\lambda_1}{\sum_j \lambda_j})\vec{q}_{U_1} \tag{41}$$

with:

$$\vec{q}_{U_1} = \frac{\sum_{j=2} S(i,j)\lambda_j S^{-1}(j,i)}{\sum_{j=2} \lambda_j} \tag{42}$$

□

The weight given to $\vec{b}_{U_1}$ is proportional to the maximum eigenvalue λ_1. Thus the error in loopy belief propagation is small when the maximum eigenvalue dominates the eigenvalue spectrum:

$$\vec{p}_{U_1} - \vec{b}_{U_1} = (1 - \frac{\lambda_1}{\sum_j \lambda_j})(\vec{q}_{U_1} + \vec{p}_{U_1}) \tag{43}$$

Note again the importance of the ratio between the subdominant eigenvalue and the dominant one. When this ratio is small loopy belief propagation converges rapidly, and furthermore the approximation error is small.

The preceding discussion has made no assumption about the range of possible values that variables in the networks can take. If we now assume that the variables are *binary* we can show that the belief update assignment is guaranteed to give the maximum marginal assignment.

Claim: Consider a pairwise Markov network consisting of a single loop of unobserved variables U_i each of which is connected to an observed variable O_i. Assume the unobserved variables are binary, then the belief update assignment gives the maximum marginal assignment.

Note that in this case:

$$\vec{p}_{U_1}(i) = \frac{\lambda_1 S(i,1)S^{-1}(1,i) + \lambda_2 S(i,2)S^{-1}(2,i)}{\lambda_1 + \lambda_2} \tag{44}$$

$$= \frac{\lambda_1 \vec{b}_{U_1}(i) + \lambda_2(1 - \vec{b}_{U_1}(i))}{\lambda_1 + \lambda_2} \tag{45}$$

Thus:

$$\vec{p}_{U_1}(i) - \vec{p}_{U_1}(j) = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2}(\vec{b}_{U_1}(i) - \vec{b}_{U_1}(j)) \tag{46}$$

Thus $\vec{p}_{U_1}(i) - \vec{p}_{U_1}(j)$ will be positive if and only if $\vec{b}_{U_1}(i) - \vec{b}_{U_1}(j)$ is positive ($\lambda_1 > \lambda_2$, $\lambda_1 > 0$ and $trace(C_{12}) > 0$). In other words the calculated beliefs may be wrong, but are guaranteed to be on the correct side of 0.5. □

4.1 Correcting the beliefs using locally available information

Equation 40 suggests that if node U_1 was able to estimate the higher order eigenvalues and eigenvectors of the matrix C_{21} from the messages it receives, it would be able to correct the belief vector. Indeed, there exist several numerical algorithms to estimate all eigenvectors of a matrix using recursions similar to equation 18 (Strang, 1986).

Claim: Consider a pairwise Markov network consisting of a single loop of unobserved binary variables $\{U_i\}_{i=1}^N$ each of which is connected to an observed variable O_i. For node U_1 define the temporal difference $\vec{d}_i^{(t)} = \vec{v}_{U_2U_1}^{(t)} - \vec{v}_{U_2U_1}^{(t-N)}$, and the ratio r as the solution to $\vec{d}_i^{(t)} = r\vec{d}_i^{(t-N)}$. Then $\vec{p}_{U_i} = \frac{1}{1+r}\vec{b}_{U_i} + \frac{r}{1+r}(1 - \vec{b}_{U_i})$

The proof is based on the following lemma:

Temporal difference lemma: Let C be a matrix with eigenvalues $\lambda_1 \cdots \lambda_n$ sorted by decreasing magnitude and eigenvectors $\vec{u_1} \cdots \vec{u_n}$. Assume that the eigenvectors are normalized so they sum to 1. Define the recursion $\vec{x}^{(t+1)} = \alpha C \vec{x}^{(t)}$ and the temporal difference $\vec{d}^{(t)} = \vec{x}^{(t)} - \vec{x}^{(t-1)}$. Assume the conditions of the power method lemma hold and in addition $|\lambda_2| > |\lambda_3|$, then $\vec{d}^{(t)}$ approaches a multiple of $\vec{u}_2 - \vec{u}_1$ and furthermore $\vec{d}^{(t)}$ approaches $\frac{\lambda_2}{\lambda_1}\vec{d}^{(t-1)}$.

This lemma can be proven by linearizing the relationship between $\vec{x}^{(t+1)}$ and $\vec{x}^{(t)}$ around the steady-state value $\vec{x}^{\infty}$ and then applying the power-method lemma.

The temporal difference lemma guarantees that each component of $\vec{d}$ in the claim will decrease geometrically in magnitude and the rate of decrease gives the ratio $r = \lambda_2/\lambda_1$. Substituting $r = \lambda_2/\lambda_1$ into equation 45 gives:

$$\vec{p}_{U_1} = \frac{1}{1+r}\vec{b}_{U_1} + \frac{r}{1+r}(1 - \vec{b}_{U_1}) \tag{47}$$

□

More complicated correction schemes are possible. In the case of ternary nodes, the nodes can use the direction of the vector $\vec{d}$ to calculate the second eigenvector and thus obtain a better estimate of the correct probability. In general, all eigenvectors and eigenvalues can be calculated by performing operations on the stream of messages a node receives.

4.2 Illustrative examples

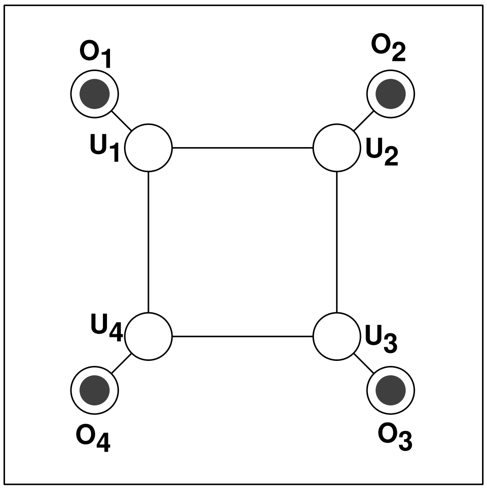
Figure 5a shows a network of four nodes, connected in a single loop. Figure 5b shows the local probabilities, i.e. the probability of a node being in state 1 or 2 given only its local evidence. The transition matrices, were set to:

$$M = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix} \tag{48}$$

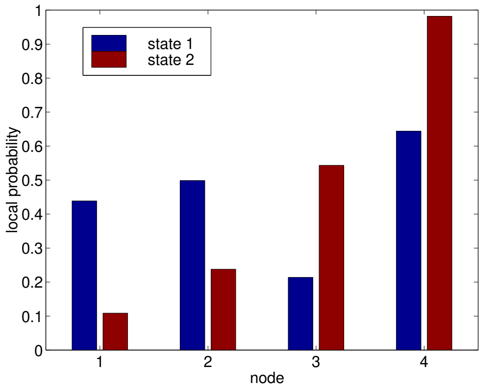
Figure 5c shows the convergence rate of one of the messages as a function of iteration. We plot $||\vec{d}^{(t)}||$ as a function of iteration and the magnitude can be seen to decrease geometrically (or linearly in the log plot in Figure 5d). The rate at which the components of $\vec{d}^{(t)}$ decrease gives the value of r that the nodes can use to correct the beliefs.

Figure 6a shows the correct posteriors (calculated by exhaustive enumeration) and the steady-state beliefs estimated by loopy propagation. Note that the estimated beliefs are on the correct side of 0.5 as guaranteed by the theory, but are numerically wrong. In particular, the estimated beliefs are overly confident (closer to 1 and 0, and further from 0.5). Intuitively, this is due to the fact that the loopy network involves counting each evidence multiple times, rather than a single time, thus leading to an overly confident estimate. More formally, this is a result of λ_2 being positive (equation 46). Figure 6c shows the estimated beliefs after the correction of equation 47 has been added. The beliefs are identical to the correct beliefs up to machine precision.

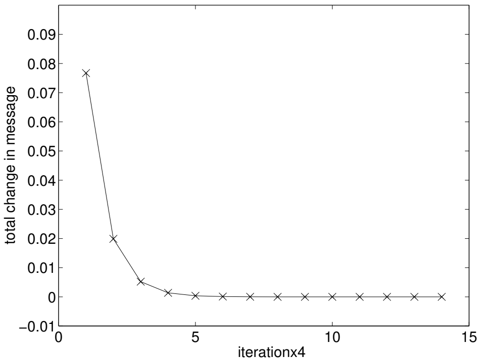
Figure 7 illustrates the relationship between the asymptotic convergence ratio $|r| = ||\vec{d}^{(t+4)}||/||\vec{d}^{(t)}||$ and the approximation error $||\vec{b}_X - \vec{p}_X||$ at a given node. Each datapoint represents the results on a simple loop network (Figure 5a) with randomly selected transition matrices. Consistent with equation 40 the error is small when convergence is rapid and large when convergence is



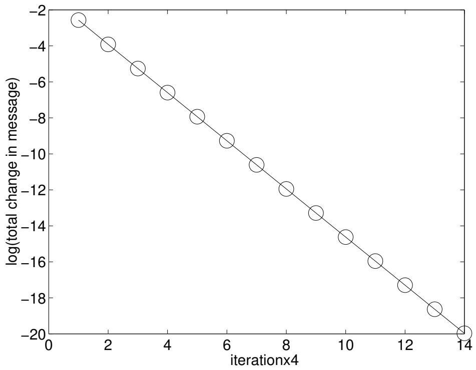
a



b



c



d

Figure 5: a. A simple loop structure. **b.** The local evidence probabilities ($\vec{v}_{O_iU_i}$) used in the simulations. **c.** The differences between messages at 4 iterations ($|\vec{v}^{(t)}_{U_2U_1} - \vec{v}^{(t-4)}_{U_2U_1}|$). Note the geometric convergence. **d.** The log of the magnitude of the differences plotted in c. The slope of this line determines the ratio of the second and first eigenvalues.

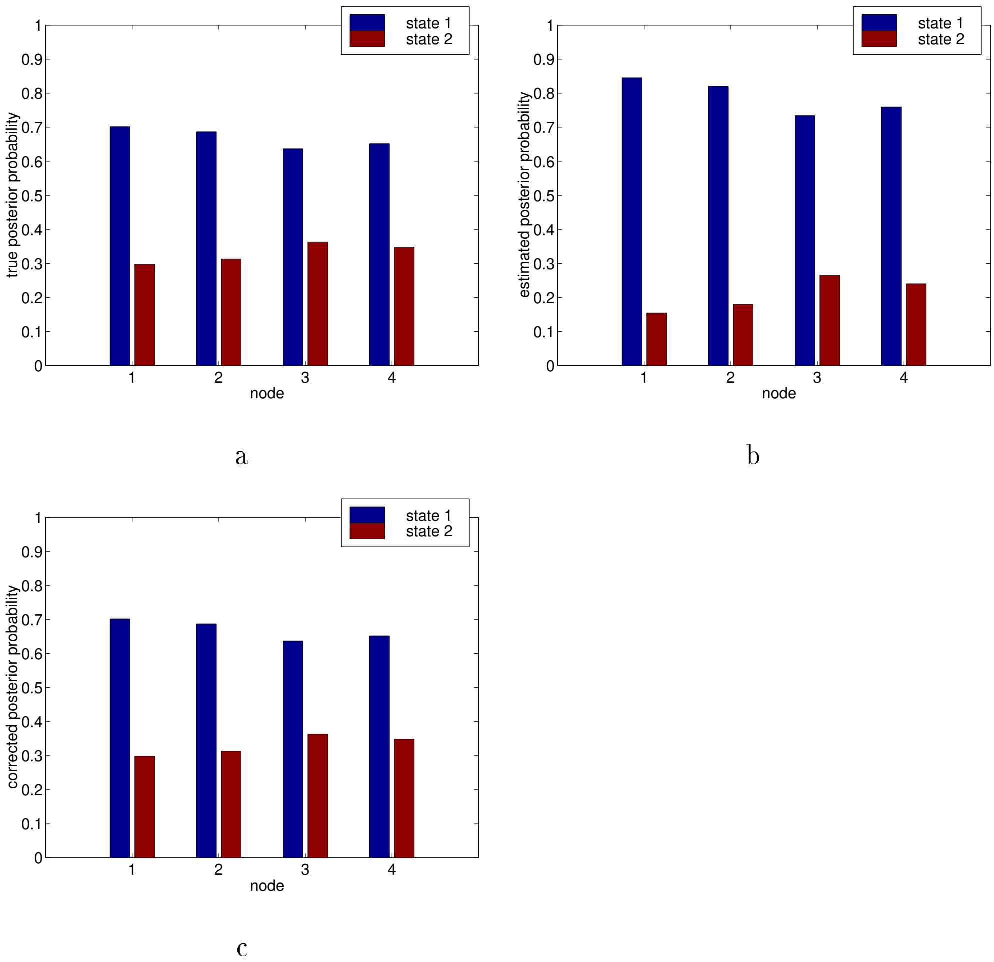


Figure 6: **a.** The correct marginals (calculated by exhaustive enumeration) for the data in 5. **b.** The beliefs estimated at convergence by the loopy belief propagation algorithm. Note that the estimated beliefs are on the correct side of 0.5 as guaranteed by the theory, but are numerically wrong. In particular, the estimated beliefs are overly confident (closer to 1 and 0, and further from 0.5). **c.** The estimated beliefs after correction based on the convergence rates of the messages (equation 47). The beliefs are identical to the correct beliefs up to machine precision.

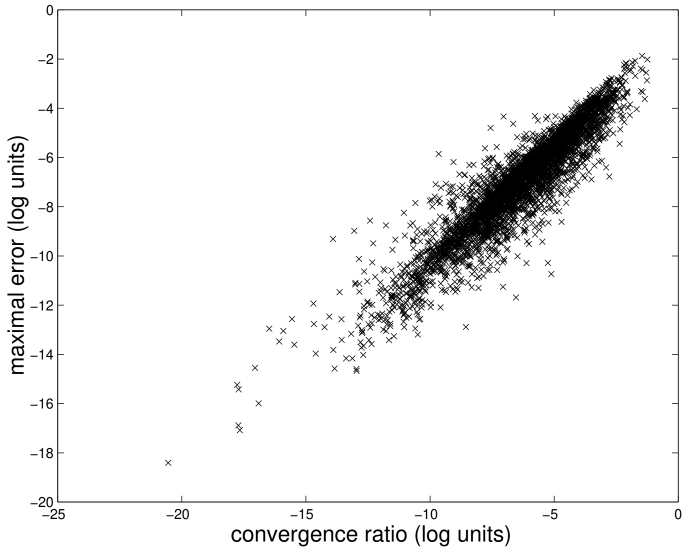


Figure 7: The maximal error between the beliefs estimated using loopy belief update and the true beliefs, plotted as a function of the convergence ratio. As predicted by the theory, the error is small when convergence is rapid.

slow.

4.3 Networks containing a single loop and additional trees

The results of the previous section can be extended to networks that contain a single loop and additional trees. An example is shown in Figure 8a. These networks include arbitrary combinations of chains and trees, except for a single loop. The nodes in these networks can be divided into two categories: those in the loop and those that are not. For example in Figure 8 nodes 1–4 are part of the loop while nodes 5–7 are not.

Let us focus first on nodes inside the loop. If all we are interested in is obtaining the correct beliefs for these nodes, then the situation is equivalent to a graph that only has the loop and evidence nodes. That is, after a finite number of iterations, the messages coming into the loop nodes from the non loop nodes will converge to a steady-state value. These messages can be thought of as messages from observed nodes and the analysis of the previous section holds.

For nodes outside the loop, however, the relationship derived in the previous section between $\vec{p}_X$ and $\vec{b}_X$ does not hold. In particular, there is no guarantee that the BU assignment will give the MM assignment, even in the case of binary nodes. Consider node 5 in the figure. Its posterior probability can be factored into three independent sources — the probability given the evidence at node 6, at node 7 and the evidence in the loop. The messages node 5 receives from nodes 6 and 7 in the belief update algorithm will correctly give the probability given the evidence at these nodes. However, the message it receives from node 4 will be based on the messages passed around the loop multiple times, i.e. evidence in the loop will be counted more times than evidence outside the loop.

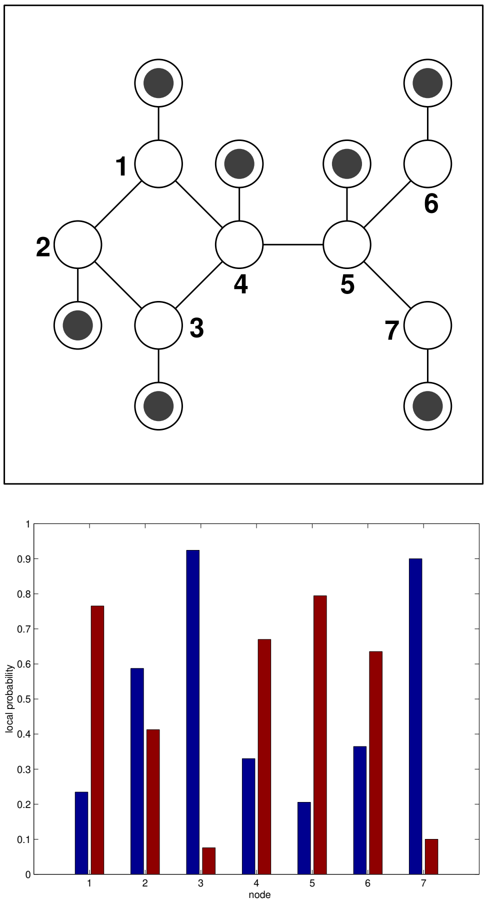
Nevertheless, if nodes in the loop use the convergence rate to correct their beliefs, they can also correct the messages they send out to nodes outside the loop in a similar way. One way to do this is to calculate outgoing messages $\vec{v}_{XY}$ by dividing the belief vector $\vec{b}_X$ componentwise by the vector $\vec{v}_{YX}$ and multiplying the result by M_{XY}. If $\vec{b}_X$ is as defined by equation 8 then this procedure gives the exact same messages as equation 7. However, if $\vec{b}_X$ represents the corrected beliefs, then the outgoing messages will also be corrected and the beliefs in all nodes will be correct.

To illustrate these ideas, consider figures 8–9. Figure 8b shows the local probabilities for each of the seven nodes. The transition matrix was identical to that used in the previous example (equation 48). Figure 9a shows the correct posterior probabilities calculated using exhaustive enumeration. Figure 9b shows the beliefs estimated using regular belief update on the loopy network. Note that the beliefs for the nodes inside the loop are on the correct side of 0.5 but overly confident. Note however that the belief of node 6 is *not* on the correct side of 0.5. Finally, Figure 9c shows the results of the corrected belief propagation algorithm, by using equation 47 for nodes in the loop. The results are identical to the correct posteriors up to machine precision.

5 Belief revision in networks with a single loop

For belief revision the situation is simpler than belief update. In belief update we had to distinguish between binary nodes versus nonbinary nodes, and between networks consisting of a single loop versus networks with a loop and additional trees. For belief revision we can prove the optimality of the BR assignment for any net with a single loop.

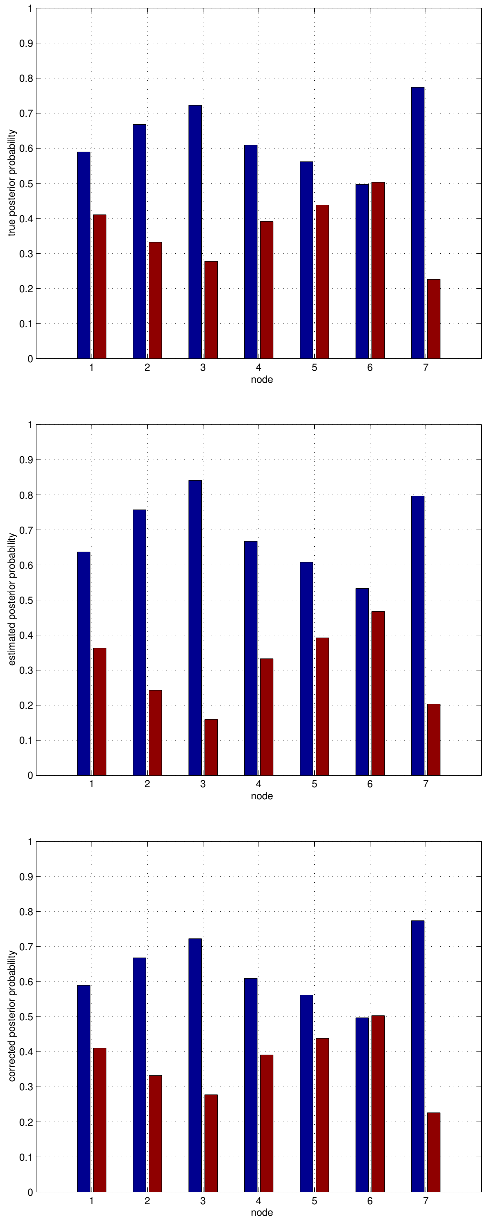
Claim: Consider a pairwise Markov net with a single loop. If all potentials Ψ are positive and belief revision converges then the BR assignment gives the MAP assignment.



a

b

Figure 8: a. A network including a tree and a loop. **b.** The probability of each node given its local evidence used in the simulations.



a

b

c

Figure 9: a. The correct posterior probabilities calculated using exhaustive enumeration. **b.** The beliefs estimated using regular belief update on the loopy network. Note that the beliefs for the nodes inside the loop are on the correct side of 0.5 but overly confident. Note however that the belief of node 6 is *not* on the correct side of 0.5.**c.** The results of the corrected belief propagation algorithm by using equation 47 for nodes in the loop. The results are identical to the correct posteriors up to machine precision.

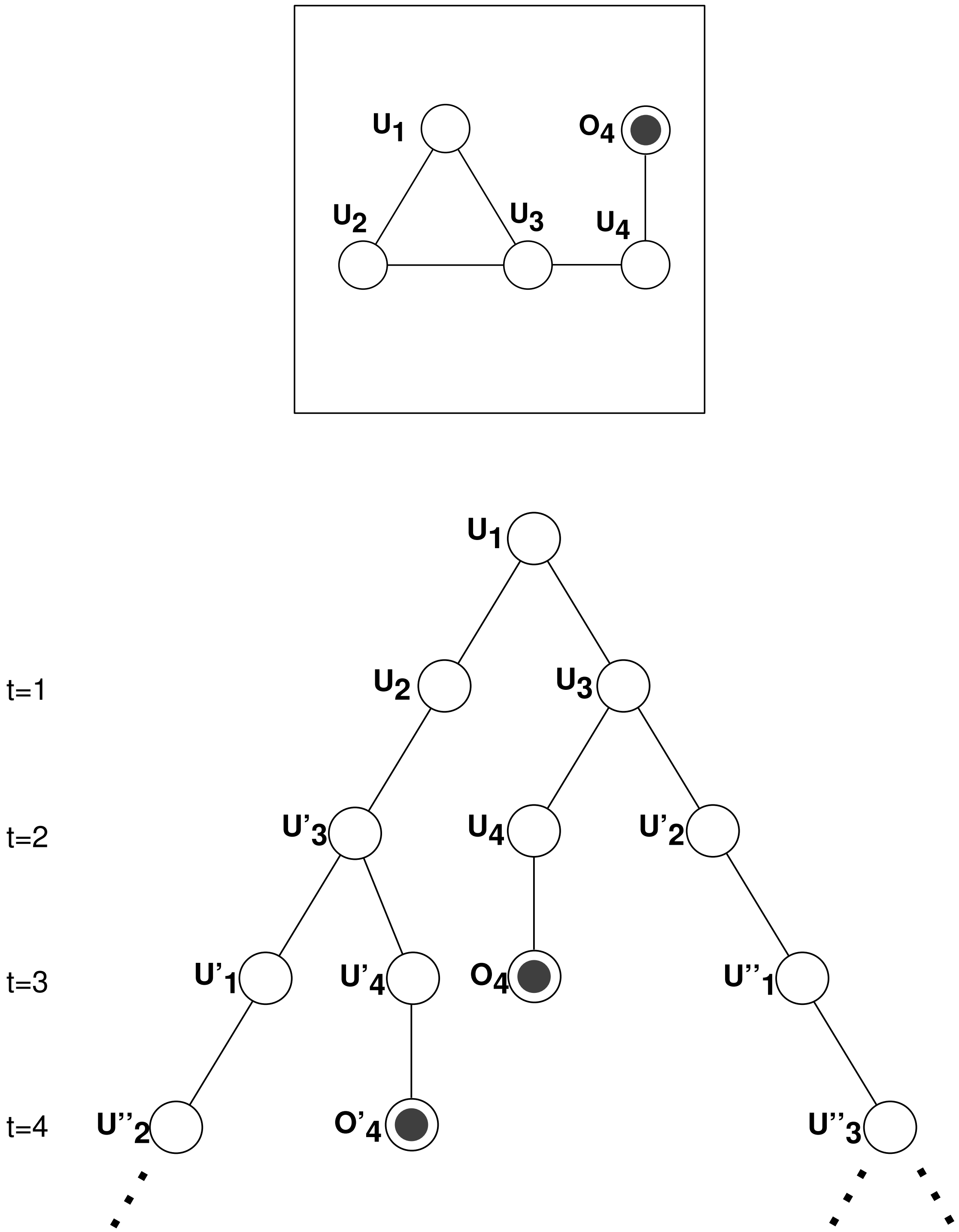
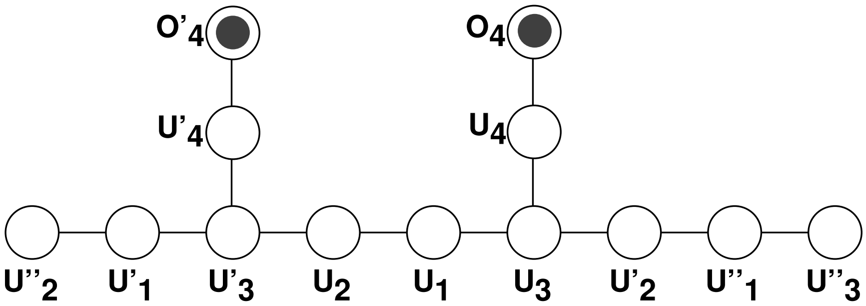
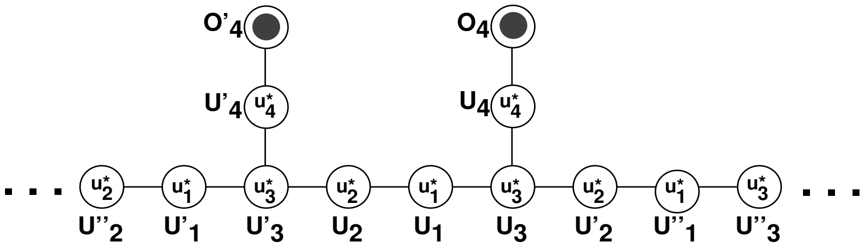


Figure 10: An illustration of the procedure for generating an unwrapped network. At every iteration, we examine all leaf nodes in the tree and find the neighbors of the corresponding node in the loopy network. We add nodes as children to each leaf corresponding to all neighbors except the parent node.



a



b

Figure 11: a. For networks with a single loop, the unwrapped network is equivalent to an increasingly long chain (corresponding to the loop nodes) and additional trees connected at the corresponding nodes. **b.** When belief revision converges, the MAP assignment of the unwrapped network contains periodic replicas of the MAP assignment in the loopy network.

The proof uses the notion of the "unwrapped" network introduced in section 3. Recall from section 3 that for every loopy net, node X and iteration t, there exists an unwrapped network such that the messages received by node X in the loopy net after t iterations are identical to the final messages received by the corresponding node in the unwrapped network after convergence. This means that the belief revision (BR) assignment at node X after t iterations corresponds to the MAP assignment for the corresponding X in the unwrapped network. It does not, however, tell us anything directly about the MAP assignment for other replicas of X in the unwrapped network. When belief revision converges, however, the BR assignment at X gives us the MAP assignment at nearly all replicas of X in the unwrapped network. This is shown in the following two lemmas.

Unwrapped network lemma: For every node X in a loopy network and iteration time t we can construct an unwrapped network such that all messages $\vec{v}_{YX}^{(t)}$ that X receives from a neighboring node Y at time t in the loopy network are equal to the final messages $\vec{v}_{YX}$ that X receives from the corresponding node in the unwrapped network.

Proof: The proof is by construction. We construct an unwrapped tree by setting X to be the root node and then iterating the following procedure t times:

- find all leafs of the tree (nodes that have no children).
- For each leaf, find all k nodes in the loopy graph that neighbor the node corresponding to this leaf.
- Add $k-1$ nodes as children to each leaf, corresponding to all neighbors except the parent node.

The transition matrices are set to be identical to those in the loopy network. Likewise, if a node X in the loopy network is observed to be in state i then all replicas of X are also set to be observed in state i. Thus any node in the interior of the unwrapped tree has the same neighbors as the corresponding node in the loopy network and the same transition matrices. It is easy to verify that the messages the root node receives in the unwrapped tree are identical to those received by the corresponding node in the loopy network after t iterations. $\square$

Figure 10 illustrates the construction of the unwrapped network for a network with a single loop. Note that although we have used a tree structure to generate the unwrapped network, for the case of a single loop network the unwrapped network is equivalent to an increasingly long chain (corresponding to the loop nodes) and additional tree nodes connected to the chain at the corresponding nodes. Figure 11a shows the same unwrapped network as in figure 10 but drawn as a long chain. We define the endpoint nodes of the unwrapped network as the two loop nodes that are most distant from the root. Thus in figure 11a U_2'' and U_3'' are endpoint nodes.

Any unwrapped network defines a joint probability distribution and we can therefore define the MAP assignment of the unwrapped network. Although the unwrapped network is constructed so that all replicas of X have identical neighbors, there is no constraint that their MAP assignment is identical. Thus in figure 11a the MAP assignment in the unwrapped network requires assigning values to 11 variables. There is no guarantee that replicas of the same node (e.g. U_1, U_1') will have the same assignment. However, as we show in the following lemma, if belief revision *converges* the MAP assignment for the unwrapped network is guaranteed to be semiperiodic — all replicas of a given node in the unwrapped network will have the same

MAP assignment as long as they are far away from the endpoint nodes.

Periodic assignment lemma: Assume all messages in a network with a single loop converge to their steady-state values after t_c iterations: $\vec{v}^{(t)}_{XY} = \vec{v}^{\infty}_{XY}$ for $t > t_c$. If Y is a node in the unwrapped network such that the distance of Y to both endpoints is greater than t_c then the MAP assignment at Y is equal to the BR assignment of the corresponding node in the loopy network.

Proof: We first show that all messages transmitted in the unwrapped network are equal to the steady-state messages in the loopy network. To show this, we divide the nodes in the unwrapped network into two categories - "chain nodes" form part of the infinite chain (corresponding to nodes in the loop in the loopy network) and "nonchain" nodes (corresponding to nodes outside the loop in the loopy network). This gives four types of messages in the unwrapped network. "leftward" and "rightward" messages are the two directions of propagation inside the chain, "outward" are propagated in the nonchain nodes away from the chain nodes and "inward" messages go toward the chain nodes. Thus in Figure 11a, $\vec{v}_{U_2U_1}$ is a rightward message, $\vec{v}_{U_3U_1}$ is a leftward message, $\vec{v}_{U_3U_4}$ is an outward message and $\vec{v}_{U_4U_3}$ is an inward message.

The "inward" messages depend only on other inward messages and the evidence at the nonchain nodes. Thus they are identical at all replicas of the nonchain nodes. Since the transition matrices and evidences are replicated from the loopy network, the inward messages are identical to the corresponding messages in the loopy network.

The leftward messages depend only on other leftward messages to the right of them and the inward message. By construction of the unwrapped network, if X is a node in the unwrapped network whose distance from the rightmost endpoint is d, and Y is its neighbor on the left, then $\vec{v}_{XY} = \vec{v}^{(d)}_{XY}$, i.e. the message is identical to the message that X sends to Y in the loopy network at time d. Since $d > t_c$ $\vec{v}_{XY} = \vec{v}^{\infty}_{XY}$.

The proof for the rightward messages is analogous to that of the leftward messages. Finally, the outward messages that a chain node sends to a nonchain node depends only on the leftward and rightward messages. Thus if both leftward and rightward messages that it receives are equal to the steady-state messages in the loopy network, so will the outward message it sends out.

Therefore if X is a node in the unwrapped tree whose distance from the endpoint is more than t_c all incoming messages to X are identical to the steady-state messages in the loopy network in the corresponding node. Therefore the belief vector at X is identical to the corresponding steady-state belief in the loopy network and the MAP assignment at X is equal to the BR assignment in the corresponding node. $\square$

The periodic assignment lemma guarantees that if belief revision converges then the MAP assignment in the unwrapped tree will have replicas of the BR assignment for all nodes whose distance from the endpoints is greater than t_c. The importance of this result is that we can now relate optimality in the unwrapped tree to optimality in the loopy network.

It is easier here to work with log probabilities rather than probabilities. We define $J(X,Y) = -\log \Psi(X,Y)$. For any Markov network with observed variables, we refer to the "network cost function" J as the negative log posterior of the unobserved variables. Thus in Figure 10a:

$$J(\{u_i\}) = J(u_1, u_2) + J(u_2, u_3) + J(u_3, u_1) + J(u_3, u_4) + J(u_4, o_4) \tag{49}$$

where o_4 is the observed value of O_4.

Optimality relationship lemma: Assume belief revision converges in a network with a single loop and u_i^* is the BR assignment in the loopy network. Assume furthermore that all potentials $\Psi(X,Y)$ are nonzero and define J as the negative log posterior of the loopy network. For any n, u_i^* must minimize:

$$J_n(\{u_i\}) = nJ(\{u_i\}) + \tilde{J} \tag{50}$$

where $\tilde{J}$ is finite and independent of n.

To prove this, note that for any n there exists a t such that the unwrapped network at iteration t contains a subnetwork with n copies of all links in the loopy network. Furthermore, by choosing a sufficiently large t the distance of all nodes in the subnetwork from the endpoints can be made sufficiently large. By the periodic assignment lemma, the MAP assignment will have replicas of the BR assignment for all nodes within the subnetwork. Furthermore, by the definition of the MAP assignment, the assignment within the subnetwork maximizes the posterior probability when nodes outside the subnetwork are clamped to their MAP assignment

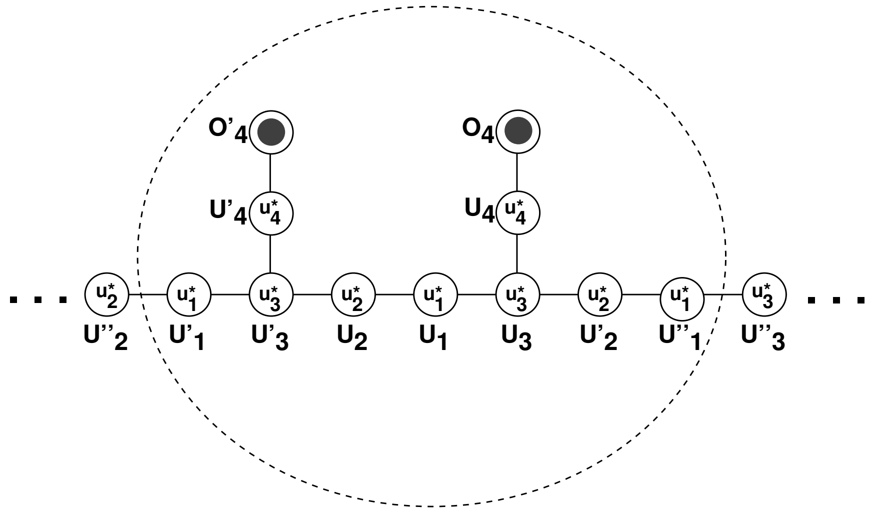


Figure 12: The proof of the optimality relationship lemma for networks with a single loop. For any n there exists a t such that the unwrapped network of size t includes a subnetwork that contains exactly n copies of all links in the original loopy network. observed.

values. We denote by J_n the cost function that the assignment of the subnetwork minimizes. The cost function decomposes into a sum of pairwise costs, one for each link in the subnetwork. For a periodic assignment, the cost function can be rewritten:

$$J_n(\{u_i\}) = nJ(\{u_i\}) + \tilde{J} \tag{51}$$

where the $\tilde{J}$ term captures the existence of links between the boundaries of the chain and its observed neighbors. Note that this term is independent of n (the distance of the neighboring nodes to the endpoints is also greater than t_c). Since periodic copies of $\{u_i^*\}$ form the MAP assignment for the subnetwork, $\{u_i^*\}$ must minimize J_n. $\Box$

Figure 12 illustrates the optimality relationship lemma for $n = 2$. The subnetwork encircled by the dotted line includes two copies of all links in the original loopy network. Therefore the subnetwork cost J_n is given by:

$$J_n(\{u_i\}) = 2J(\{u_i\}) + \tilde{J}(\{u_i\}) \tag{52}$$

with $\tilde{J}(\{u_i\})$ a sum of the two links that connect the subnetwork to the larger unwrapped network: $J(u_2^*, u_1) + J(u_1, u_3^*)$.

The optimality relationship lemma essentially proves the main claim of this section. Since the BR assignment minimizes J_n for arbitrarily large n and $\tilde{J}$ is independent of n, the BR assignment must minimize J and hence is the MAP assignment for the loopy network. $\Box$

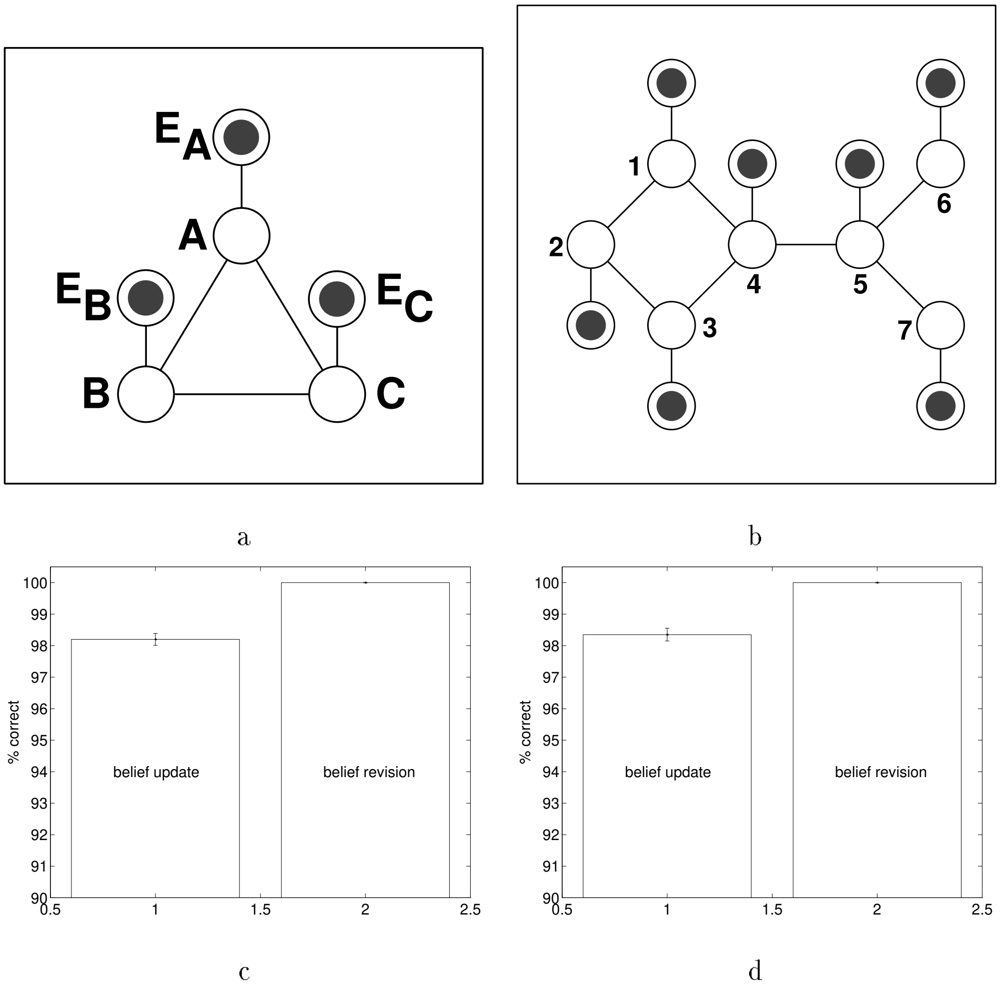


Figure 13: a-b. Two structures for which the BU assignment need not coincide with the MM assignment, but the BR assignment is guaranteed to give the MAP assignment. **a.** A simple loop structure with ternary nodes. **b.** A loop adjoined to a tree. **c-d.** The number of correct assignments using belief update and belief revision in 5000 networks of these structure with randomly generated transition matrices. Consistent with the analysis, the BU assignment may be incorrect but an incorrect BR assignment is never observed.

To illustrate this analysis, Figure 13 shows two network structures for which there is no guarantee that the BU assignment will coincide with the MM assignment. The loop structure has ternary nodes, while the added tree on the right means that the BU assignment even for binary variables will not necessarily give the MM assignment.

However, due to the claim proven above, the BR assignment should always coincide with the MAP assignment. We selected $5,000$ random transition matrices for the two structures, and calculated the correct beliefs using exhaustive enumeration. We then counted the number of correct assignments using belief update and belief revision by comparing the BU assignment to the MM assignment and the BR assignment to the MAP assignment. We only considered assignment when belief revision converged (about 95% of the simulations).

The results are shown in Figure 13. In both of these structure, the BU assignment is not guaranteed to give the MM assignment and indeed the wrong assignment is sometimes observed. However, an incorrect BR assignment is never observed.

6 Discussion - single loop

To summarize, the analysis enables us to categorize a given network with a loop into one of two categories - (1) structures for which both belief update and belief revision are guaranteed to give the correct assignment, or (2) those for which only belief revision is guaranteed to give the correct assignment.

The crucial element determining the performance of belief update is the eigenspectrum of the matrix C_{N1}. Note that this matrix depends both on the transition matrices and the observed data. Thus belief update may give very exact estimates for certain observed data even in structures for which it is not guaranteed to give the correct assignment.

We have primarily dealt here with discrete state spaces, but the analysis can also be carried out for continuous spaces. For the case of Gaussian posterior probabilities, it is easy to show that belief update assignment will give the correct MAP assignment for all Gaussian Markov networks with single loop (this is simply due to the fact that belief update and revision are identical for a Gaussian).

We emphasize again that our choice of Markov nets with pairwise potentials as the primary object of analysis was based mainly on notational convenience. Due to the equivalence between Pearl's algorithm on Bayesian nets and our update rules on Markov nets, our results also hold for applications of Pearl's original update rules on Bayesian nets with a single loop. In particular this means that one can perform exact inference on a Bayesian net with a single loop by running Pearl's propagation algorithm and then using a correction analogous to equation 47. More generally, if the Bayesian network can be converted into a Markov network with a single loop, our results show how to perform exact inference. Exact inference in networks containing a single loop, however, can also be performed using cut-set conditioning (Pearl, 1988). Our goal in analyzing belief propagation in networks with a single loop is primarily to understand loopy belief propagation in general, with single loop networks being the simplest special case.

Independently of our work, several groups working in the context of error correcting codes have recently obtained results on probability propagation in networks with a single loop. Aji et al. (1998) have shown that iterative decoding (equivalent to belief update) of a single loop code will converge to a correct decoding for binary nodes. They also showed that the messages will converge to the principal eigenvectors of a matrix analogous to our matrix C_{21}. Forney et al. (1998) have also shown the convergence of the messages to the principal eigenvectors and have additionally analyzed the "max-sum" decoding algorithm (analogous to belief revision) on a network consisting of a single loop showing that it will converge to the "dominant pseudocodeword" – a periodic assignment on the unwrapped network such that the average cost per period is minimal. To the best of our knowledge the analytical relationship between the correct beliefs and the steady-state ones, the analysis of networks including a single loop and arbitrary trees, and the correction of beliefs using locally available information, have not been discussed elsewhere.

One of the important distinctions that arise from analyzing networks containing a single loop and arbitrary trees is the difference in performance between belief update and belief revision when they both converge. The BR assignment is always guaranteed to give the MAP assignment while the BU assignment is guaranteed to give the MM assignment only when there are no additional trees attached to the loop and the nodes are binary. As we show in subsequent sections, this difference in performance is also apparent in networks with multiple loops.

7 Networks with multiple loops

The basic intuition that explains why belief propagation works in networks with a single loop is the notion of "equal double counting". Essentially, while evidence is double counted, all evidence is double counted equally as can be seen in the unwrapped network and hence belief revision is guaranteed to give the correct assignment. Furthermore, by using recursion expressions for the messages we could quantify the "amount" of double counting, and derive a relationship between the estimated beliefs and the true ones.

For networks with multiple loops, the situation is more complex. In particular, the relationship between the messages sent at time t and at time $t+n$ is not as simple as in the single loop case, and tools such as the power method lemma or the temporal difference lemma are of no immediate use. However, the notion of the unwrapped tree is equally well applied to networks with multiple loops. Figure 14 shows an example. The unwrapped network lemma of the previous section holds — i.e. the messages received by node A in the loopy network after 4 iterations are identical to the final messages received by node A in the unwrapped network. Furthermore, a version of the periodic assignment lemma of the previous section holds for any network; if belief revision converges then the MAP assignment for arbitrarily large unwrapped networks contains periodic replicas of the BR assignment.

Thus for a network with multiple loops, if BR converges, then one can construct arbitrarily large problems such that periodic copies of the BR assignment give the optimal assignment for that problem. For example, for the network in Figure 14 the periodic assignment lemma guarantees that if $\{u_i^*\}$ is the BR assignment then it minimizes:

$$J_n(\{u_i\}) = nJ(\{u_i\}) + \tilde{J}_n \tag{53}$$

for arbitrarily large n. As in the single loop case, $\tilde{J}_n$ is a boundary cost reflecting the constraints imposed on the endpoints of the unwrapped network. Note that unlike the single loop case, the boundary cost here may increase with increasing n.

Because the boundary cost may grow with n, we *cannot* automatically assume that if an assignment minimizes J_n it also minimizes J, i.e. the assignment may be optimal only because

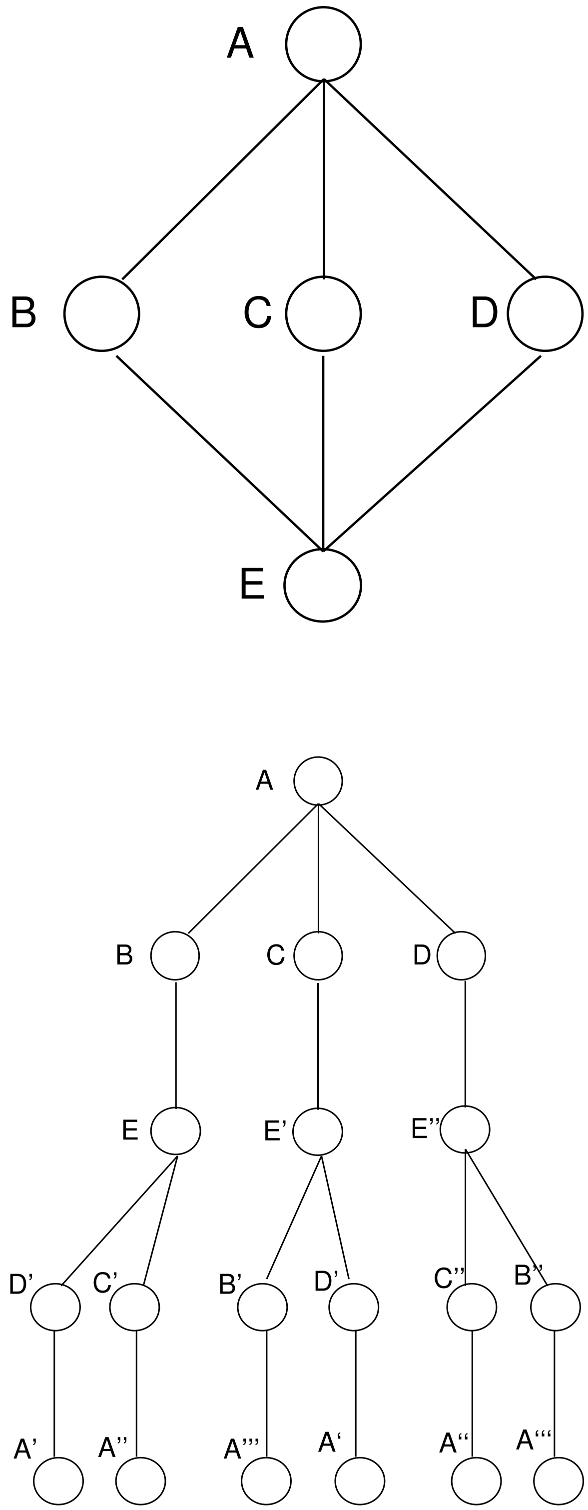


Figure 14: Top: A Markov network with multiple loops. **. Bottom:** The unwrapped network corresponding to this structure. Note that all nodes and connections in the loopy structure appear equal numbers of time in the unwrapped network, except the boundary nodes A.

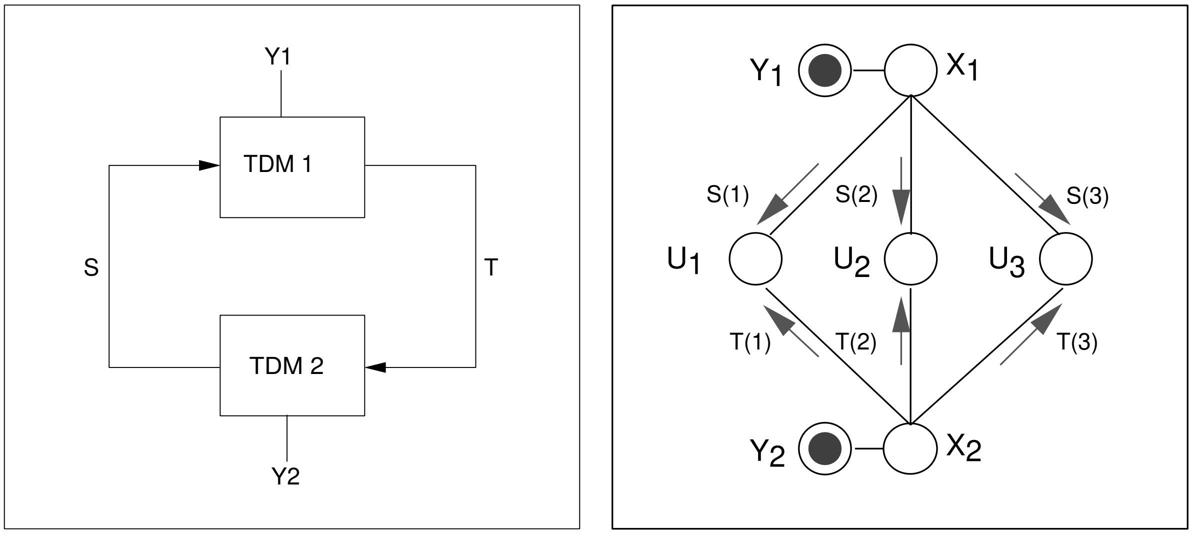
of the boundary cost. We have not been able to analytically determine the conditions under which the steady-state assignment is due to $\tilde{J}$ as opposed to J. However, the fact that equation holds 53 holds for arbitrarily large subnetworks (such that the boundary can be made arbitrarily far from the center node), leads us to hypothesize that for positive potentials Ψ loopy belief revision will give the correct assignment for the network in Figure 14.

Although we have not been able to prove this hypothesis, extensive simulation results are consistent with it. We performed loopy belief revision on the structure shown in Figure 14 using randomly selected transition matrices and local evidence probabilities. We repeated the experiment 10000 times and counted the number of correct assignments. As in the previous section, a correct assignment for belief revision is one that is identical to the MAP assignment and a correct assignment for belief update is one that is identical to the MM assignment. Again, we only considered runs in which belief revision converged. We never observed a convergence of belief revision to an incorrect assignment for this structure, while belief update gave 247 incorrect assignments (error rate 2.47%). Since these rates are based on $10,000$ trials, the difference is highly significant. While by no means a proof, these simulation result suggest that some of our analytic results for single loop networks may also hold for a subclass of multiple loop networks. Trying to extend the proofs for a broader class of networks is a topic of current research.

7.1 Turbo codes

"Turbo codes" are a class of error-correcting codes whose decoding algorithm has recently been shown to be equivalent to belief propagation on a network with loops. To make the connection to the update rules (equations 7–8) explicit, we briefly review this equivalence.

We use the formulation of Turbo decoding presented in (McEliece et al., 1995). An unknown binary vector $\vec{U}$ is encoded using two codes, each of which is easy to decode by itself, and the results are transmitted over a noisy channel. The task of Turbo decoding is to infer the values of every bit of $\vec{U}$ from two "noisy versions" $\vec{Y}_1, \vec{Y}_2$ that are received after transmission. The decoder knows the prior distribution over $\vec{U}$ (which we assume here is uniform) and the two conditional probabilities $p_i(\vec{y}|\vec{u}) = P(\vec{Y}_i = \vec{y}|\vec{U} = \vec{u})$.



a b

Figure 15: a. The Turbo decoding algorithm (after (McEliece et al., 1995)) consists of two Turbo decision modules such that the output of each module serves as the input to the other one. The posterior likelihood ratio is approximated by multiplying the outputs S, T at convergence. **b.** The Turbo decoding algorithm as belief update in a loopy Markov network. Each unknown bit is represented by a node U_i. The belief of a node is approximated by multiplying the two incoming messages (corresponding to S and T). When the link matrices are set in the way detailed in the text, the update rules for the messages (equation 7–8) reduce to the Turbo decoding algorithm.

Since both noisy versions are assumed to be conditionally independent, the marginal probability ratio is simply:

$$\frac{P(U(i)=1)}{P(U(i)=0)} = \frac{\sum_{\vec{U}:U(i)=1} P(\vec{Y_1}|\vec{U})P(\vec{Y_2}|\vec{U})}{\sum_{\vec{U}:U(i)=0} P(\vec{Y_1}|\vec{U})P(\vec{Y_2}|\vec{U})} \tag{54}$$

In the typical error-correcting code application, the sum in equation 54 is intractable. Rather, the Turbo decoder approximates the ratio by the iterative algorithm outlined in Figure 15.

The algorithm consists of two "Turbo Decision Modules" each of which only considers one "noisy version" $\vec{Y_i}$. Each module receives a likelihood ratio vector as input and outputs a modified likelihood ratio vector that takes into account the observed $\vec{Y_i}$. If $\vec{T} = TDM_1(\vec{S})$, then:

$$\vec{T}(i) = \frac{\sum_{\vec{u}:\vec{u}(i)=1} p(\vec{y_1}|\vec{u}) \prod_{j\neq i} \vec{S}(j)^{\vec{u}(j)}}{\sum_{\vec{u}:\vec{u}(i)=0} p(\vec{y_1}|\vec{u}) \prod_{j\neq i} \vec{S}(j)^{\vec{u}(j)}} \tag{55}$$

The output of module 2 is identical with $\vec{y_1}$ replaced by $\vec{y_2}$. Note that each module still requires summing over all $\vec{u}$ but the codes are constructed so that $p_i(\vec{y}|\vec{u})$ have a factorized structure that enables calculation of $TDM_i(\vec{S})$ efficiently.

The Turbo decoding iterations can be written as:

$$\vec{T} \leftarrow TDM_1(\vec{S}) \quad (56)$$

$$\vec{S} \leftarrow TDM_2(\vec{T}) \quad (57)$$

and the likelihood ratio is approximated as:

$$\vec{R} = \vec{T} \odot \vec{S} \quad (58)$$

To see the equivalence of this scheme to belief update in Markov nets, consider Figure 15b. We assume that the unknown vector $\vec{U}$ is of length 3 and represent the value of the ith bit with a node U_i in the graph. Each U_i is connected to exactly two nodes, X_1, X_2 that are in turn connected to the received "noisy versions" Y_1, Y_2. By the belief update equations (equation 8), the belief of node U_i is equal to the pointwise multiplication of the messages it receives from X_1 and X_2. Note the similarity to equation 58 where the likelihood ratio for $U(i)$ is obtained by multiplying $S(i)$ and $T(i)$. Indeed, we now show that when the potentials in the graph in Figure 15b are set in a particular form, $S(i)$ is equivalent to the message that X_1 sends to U_i and $T(i)$ is equivalent to the message that X_2 sends to U_i.

X_i are meant to represent copies of the unknown binary message $\vec{U}$ and hence in the example (of message length 3), X_i can take on 8 possible values. We set $\Psi(\vec{x}_i, u_j) = 1$ if $\vec{x}_i(j) = u_j$ and zero otherwise. Furthermore, we set $\Psi(\vec{x}_i, \vec{y}_i) = p_i(\vec{y}|\vec{x})$.

When the potentials are set in this form, the belief update equations (equation 7) gives the Turbo decoding algorithm. Denote the message $\vec{v}_{X_2 U_i}$ sent by node X_2 to U_i by the vector $\alpha(T_i, 1)$. The message $\vec{v}_{U_i X_1}$ that U_i sends to X_1 is a vector of length 8. It is obtained by taking the message U_i receives from X_2 and multiplying it by the transition matrix $M_{U_i X_1}$:

$$\begin{aligned} \vec{v}_{U_1 X_1}(x_1) &= \alpha \sum_{u_1} \Psi(x_1, u_i) \vec{v}_{X_2 U_i}(u_i) && (59) \\ &= \alpha T_i^{X_1(i)} && (60) \end{aligned}$$

Now, the message $\vec{v}_{X_1 U_i}$ sent from X_1 to U_1 is a vector of length 2 that is obtained by multiplying all incoming messages except the one coming from U_i and then passing the result through the transition matrix $M_{X_1 U_i}$:

$$\vec{v}_{X_1 U_i}(u_i) = \sum_{x_1} \Psi(x_1, u_i) p_1(y_1|x_1) \prod_{j \neq i} T_j^{x_1(j)} \quad (61)$$

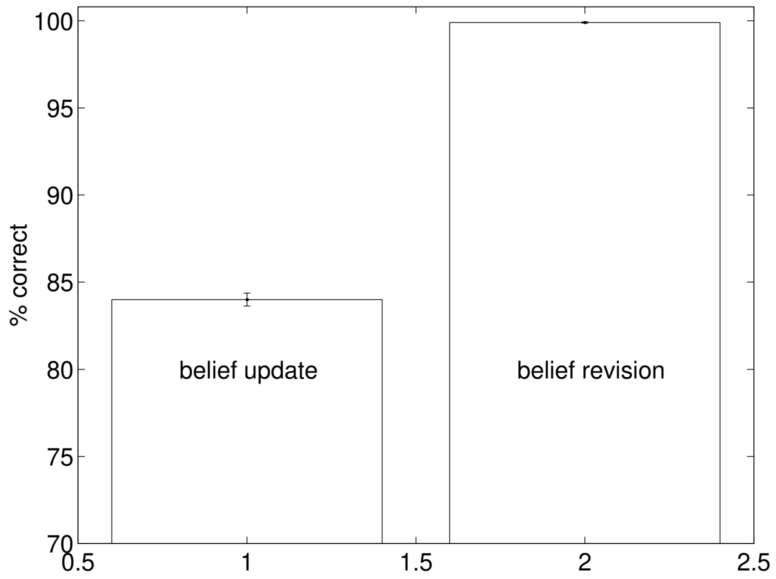


Figure 16: The results of belief revision and update on the Turbo code problem.

If we denote by $\vec{S}(i)$ the ratio of components of $\vec{v}_{X_1 U_i}$, then equation 61 is equivalent to equation 55. Thus the belief update rules in equations 7–8 give the Turbo decoding algorithm.

McEliece et al. (1995) showed several examples where the turbo decoding algorithm converges to an incorrect decoding. Since the Turbo decoding algorithm is equivalent to belief update, we wanted to see how well a decoding algorithm equivalent to *belief revision* will perform.

We randomly selected the probability distributions $p_1(u)$ and $p_2(u)$ and calculated the MAP assignment (often referred to as the ML decoding in coding applications) and the MM assignment (referred to as the optimal bitwise decoding, or OBD, in coding applications). We compared these assignments to the BU and BR assignments. As in previous sections, the BU assignment was judged as correct if it agreed with the MM assignment. and the BR assignment was judged as correct if it agreed with the MAP assignment. We repeated the experiment for $10,000$ randomly selected probability distributions. Only trials in which belief revision converged were considered.

Results are shown in Figure 16. As observed by McEliece et al. turbo decoding is quite likely to converge to an incorrect decoding (16% of the time). In contrast, belief revision converges to an incorrect answer less than 0.1% of the runs. Note that this difference is based on $10,000$

trials and is highly significant. Note, however, that the results only considered cases where belief revision converged, and in general belief revision converged less frequently than belief update.

A Turbo decoding algorithm that used maximization rather than summation in equation 55 was presented in in (Benedetto et al., 1996), where the maximization was presented as an approximation to the full sum in equation 55. The decoding performance of the approximate algorithm was shown to be slightly worse than standard Turbo decoding. Note, however, that in our simulations we only considered decodings where belief revision converged, whereas in (Benedetto et al., 1996) the decoding obtained after a fixed number of iterations was used, regardless of whether the algorithm converged. Our motivation in considering only decodings at convergence was based on our analysis of single loop networks. The simulation results suggest that Turbo codes exhibit a similar difference in performance between belief update and belief revision.

From a practical point of view, of course, errors due to failures in convergence are as troubling as errors due to a convergence to a wrong answer. Thus understanding the behavior of the decoding algorithms when they do not converge is an important question for further research. Forney et al. (98) have obtained some encouraging results in this regard.

Note that unlike the networks considered in the previous sections, the Turbo code Markov net has deterministic links (between the X_i and U_j) and hence the log of the potential functions Ψ may be infinite . We believe this may be the reason for the existence of a few incorrect decodings with the belief revision. The deterministic links make it difficult to relate optimality in the unwrapped tree to optimality in the original loopy network.

7.2 Discussion

Turbo codes are not the only error-correcting code whose decoding is equivalent to belief propagation. Indeed one of the first derivations of an algorithm equivalent to belief propagation appears in (Gallager, 1963) as a means of decoding codes known as "low density parity check codes". Gallager's decoding algorithm is equivalent to probability propagation on a network with loops, and Gallager discussed the fact that his decoding algorithm is not guaranteed to give the optimal decoding because it considers replicas of the same evidence as if they were independent. Gallager also showed that his algorithm will give the MAP decoding for a "computation" tree, equivalent to our unwrapped network. He noted that if the number of iterations is small compared to the size of the loops in the graph, the computation tree will not contain replicas of the same evidence. He presented a method of constructing codes such that the size of the loops in the network will be large.

Indeed, if double counting is to be avoided, we would expect performance of belief propagation in loopy nets to be best when (1) the size of the loops is large or (equivalently) (2) the number of iterations is small. Turbo codes, however, contradict both of these expectations. The Markov network corresponding to Turbo decoding (e.g. Figure 15b) contains many loops of size 4. Furthermore, it has been well established (e.g. (McEliece et al., 1995; Benedetto et al., 1996)) that Turbo decoding performance *improves* as the number of iterations is increased.

The performance of Turbo codes is consistent, however, with the expectation based on the notion of "equal double counting" that we have emphasized throughout this paper. As we saw in the analysis of networks with a single loop, the BR assignment at convergence is guaranteed to give the MAP assignment regardless of the size of the loop. Furthermore, as we discussed in section 3, for any finite number of iterations the unwrapped network will not contain an equal number of replicas of all nodes and connections. It is only as the size of the unwrapped network approaches infinity that the influence of the boundary nodes can be neglected. Thus additional iterations are expected, based on our analysis, to increase the decoding performance.

Despite the importance of Turbo codes, one should use caution in generalizing from their performance to general loopy belief propagation. In the typical Turbo decoding setting, the probabilities $P(\vec{y}_i|\vec{u})$ in equation 55 are highly peaked around the correct $\vec{u}$. Thus the sum in equation 55 is typically dominated by a single term and therefore belief revision (in which the sum is approximated by its largest value) and belief update behave quite similarly. As we have shown throughout this paper, this is not the case for general loopy belief propagation, nor is it the case for Turbo decoding when the probabilities $P(\vec{y}_i|\vec{u})$ are chosen randomly in simulations. We believe that additional insights into Turbo decoding will be obtained by comparing the performance of the algorithm at different probability regimes.

8 Conclusion

As many authors have observed, in order for belief propagation to be exact, we must find a way to avoid double counting. Since double counting is unavoidable in networks with loops, belief propagation in such networks may seem to be a bad idea. The excellent performance of Turbo codes motivates a closer look at belief propagation in loopy networks. Here we have shown a class of networks for which, even though the evidence is double counted, all evidence is equally double counted. For networks with a single loop we have obtained an analytical expression relating the beliefs estimated by loopy belief propagation and the correct beliefs. This allows us to find structures in which belief update may give the wrong beliefs but the BU assignment is guaranteed to give the correct MM assignment. We have also shown that for networks with a single loop the BR assignment is guaranteed to give the MAP assignment. For networks with multiple loops we have presented simulation results suggesting that some of the results we have obtained analytically for single loop networks may hold for a large class of networks. These results are an encouraging first step towards understanding belief propagation in networks with loops.

Acknowledgments

Supported by NEI R01 EY11005 to E. H. Adelson. I thank the anonymous reviewers, E. Adelson, M. Jordan, P. Dayan, M. Meila, Q. Morris and J. Tenenbaum for helpful discussions and comments. After a preliminary version of this work appeared (Weiss, 1997), I had the pleasure of several discussions with D. Forney and F. Kschischang regarding related work they have been pursuing independently. I thank them for their comments and suggestions.

References

Aji, S., Horn, G., and McEliece, R. (1998). On the convergence of iterative decoding on graphs with a single cycle. In *Proc. 1998 ISIT*.

Aji, S. M. and McEliece, R. (1998). The generalized distributive law. submitted to IEEE Transactions on Information Theory.

Benedetto, S., Montorsi, G., Divsalar, D., and Pollara, F. (1996). Soft-output decoding algorithms in iterative decoding of Turbo codes. Technical Report 42–124, JPL TDA.

Berrou, C., Glavieux, A., and Thitimajshima, P. (1993). Near Shannon limit error-correcting coding and decoding: Turbo codes. In *Proc. IEEE International Communications Conference '93*.

Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models.* Prentice Hall.

Forney, G. (1997). On iterative decoding and the two-way algorithm. In *Intl. Symp. Turbo Codes and Related Topics*, Brest, France.

Forney, G., Kschischang, F., and Marcus, B. (1998). Iterative decoding of tail-biting trellisses. preprint presented at 1998 Information Theory Workshop in San Diego.

Frey, B. J. (1998). *Bayesian Networks for Pattern Classification, Data Compression and Channel Coding.* MIT Press.

Frey, B. J., Koetter, R., and Vardy, A. (1998). Skewness and pseudocodewords in iterative decoding. In *Proc. 1998 ISIT*.

Gallager, R. (1963). *Low Density Parity Check Codes.* MIT Press.

Gelb, A., editor (1974). *Applied Optimal Estimation.* MIT Press.

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. PAMI*, 6(6):721–741.

Jensen, F. (1996). *An Introduction to Bayesian Networks.* Springer.

Kschischang, F. R. and Frey, B. J. (1998). Iterative decoding of compound codes by probability propagation in graphical models. *IEEE Journal on Selected Areas in Communication*, 16(2):219–230.

Lauritzen, S. (1996). *Graphical Models.* Oxford University Press.

Mackay, D. and Neal, R. M. (1995). Good error-correcting codes based on very sparse matrices. In *Cryptography and Coding: 5th IAM Conference. Lecture Notes in Computer Science No. 1025*, pages 100–111. Springer-Verlag.

McEliece, R., MackKay, D., and Cheng, J. (1998). Turbo decoding as as an instance of Pearl's 'belief propagation' algorithm. *IEEE Journal on Selected Areas in Communication*, 16(2):140–152.

McEliece, R., Rodemich, E., and Cheng, J. (1995). The Turbo decision algorithm. In *Proc. 33rd Allerton Conference on Communications, Control and Computing*, pages 366–379, Monticello, IL.

Minc, H. (1988). *Nonnegative Matrices.* John Wiley and Sons.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann.

Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2):257–286.

Smyth, P. (1997). Belief networks, hidden Markov models, and Markov random fields: a unifying view. *Pattern Recognition*, 18(11):1261–1268.

Smyth, P., Heckerman, D., and Jordan, M. I. (1997). Probabilistic independence networks for hidden Markov probability models. *Neural Computation*, 9(2):227–269.

Strang, G. (1986). *Introduction to Applied Mathematics.* Wellesley-Cambridge.

Weiss, Y. (1996). Interpreting images by propagating Bayesian beliefs. In Mozer, M., Jordan, M., and Petsche, T., editors, *Advances in Neural Information Processing Systems 9.*

Weiss, Y. (1997). Belief propagation and revision in networks with loops. Technical Report 1616, MIT AI lab.

Wiberg, N. (1996). *Codes and decoding on general graphs.* PhD thesis, Department of Electrical Engineering, U. Linkoping, Sweden.

Appendix — Converting a Bayesian net to a pairwise Markov net

There are various ways to convert a Bayesian network into a Markov network that represents the same probability distribution. Indeed one of the most popular inference algorithms for Bayesian networks is to convert them into a Markov network known as the join tree or the junction tree (Pearl, 1988; Jensen, 1996)) and perform inference on the junction tree. Here we present a method to convert a Bayesian network into a Markov net with pairwise potentials, such that the update rules described in the paper reduce to Pearl's algorithm on the original Bayesian network. In particular, if the Bayesian network has loops in it, so will the corresponding Markov network constructed using this method. This should be contrasted with the "junction tree" construction in (Jensen, 1996) which constructs a *singly connected* Markov network for any Bayesian network.

For concreteness, consider the Bayesian network in figure 17a. The joint probability is given by:

$$P(ABCDEF) = P(A)P(B|A)P(C|A)P(D|BC)P(F|C)P(E|D) \tag{62}$$

The procedure is simple: for any node that has multiple parents, we create a compound node into which the common parents are clustered. This compound node is then connected to the individual parent nodes as well as the child. For all nodes that have single parents, we simply drop the arrows and leave the topology unchanged. Figure 17 shows an example – since d has two parents b and c we create a compound node $z = (b, c)$ and connect it to d, b and c.

Having created a new graph, we need to set the pairwise potentials so that the probability distributions are identical. The potentials are simply the conditional probabilities of children given parents, with the exception of the potentials between the parent nodes and a compound node. These are set to one if the nodes have a consistent estimate of the parent node and zero otherwise. Thus in the example, if $Z = (B', C')$ then $\Psi(BZ) = 1$ if $B' = B$ and zero otherwise.

To complete the example, we set $\Psi(AB) = P(A)P(B|A)$, $\Psi(AC) = P(C|A)$, $\Psi(FC) = P(F|C)$, $\Psi(ZD) = P(D|B'C')$ and then the joint probability of the Markov net:

$$P(ABCDEFZ) = \Psi(AB)\Psi(AC)\Psi(FC)\Psi(BZ)\Psi(DE)\Psi(ZB)\Psi(ZC) \tag{63}$$

which contains only pairwise potentials but is identical to the probability distribution of the

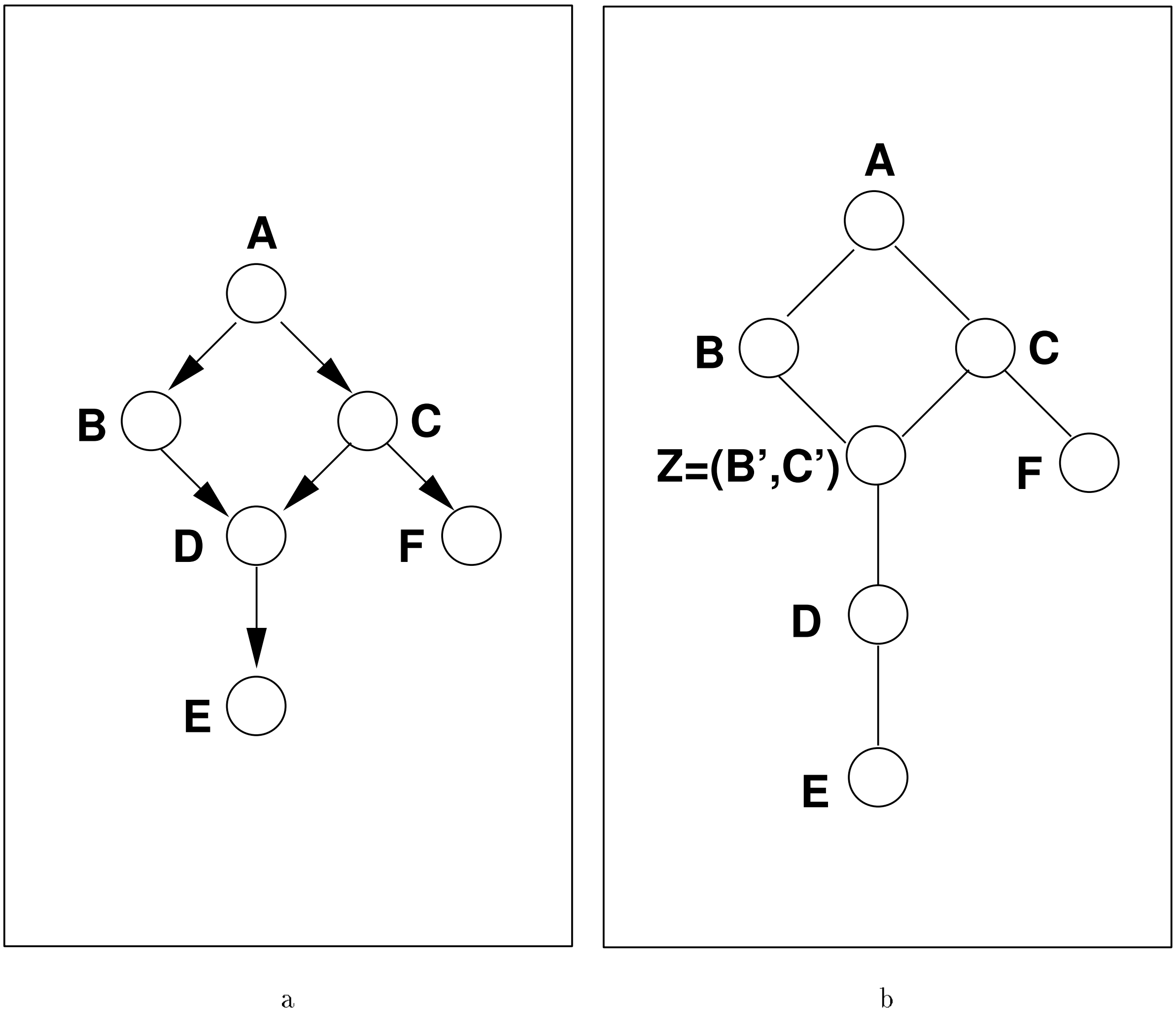


Figure 17: Any Bayesian network can be converted into a pairwise Markov network by adding cluster nodes for all parents that share a common child. **a.** A Bayesian network. **b.** The corresponding pairwise Markov network. A cluster node for (B, C) has been added. When the potentials are set in the way detailed in the text, the update rules presented in this paper reduce to Pearl's propagation rules in the original Bayesian network.

original Bayes net. (equation 62).

We now describe the equivalence between the messages passed in the Markov net and those that are passed in the Bayes net using Pearl's algorithm. If X is a single parent of Y then message X sends to Y in the Bayes net is given by $\pi_x(y)$. The message that X sends to Y in the Markov net is given by $\vec{v}_{XY}$. The messages are related by multiplication by the link matrix: $\vec{v}_{XY} = \alpha M_{XY} \pi_x(y)$. Thus in Pearl's formulation the matrix multiplication is performed at the child node, while in the algorithm described here it is performed in the parent node. The message that Y sends to X in Pearl's algorithm is denoted by $\lambda_Y(x)$ and is equal to $\vec{v}_{YX}$ in the Markov net algorithm up to normalization. Thus in Pearl's algorithm the π messages are normalized but the λ ones are not, while in the algorithm presented here all messages are normalized. If X is one of many parents of Y then in the Markov net, X is connected to Y by means of a cluster node Z. As before, $\vec{v}_{XZ} = M_{XZ}\pi_X(y)$ but since the M_{XZ} matrix contains only ones or zeros, the message in the Markov net is simply a replication of elements in the Bayesian net message. As before $\lambda_Y(x)$ is given by $\alpha\vec{v}_{ZX}$.

Thus every message in Pearl's algorithm in the Bayes net can be identified with an equivalent message in the Markov net using the algorithm presented here. Furthermore, by substituting the equivalent messages into the update rules presented here, one can derive Pearl's update rules.