Poset Belief Propagation: Experimental Results

by

Jonathan Harel
with advisor Prof. Robert J. McEliece

A Senior Thesis Submitted to the Faculty of

Electrical Engineering

California Institute of Technology
Pasadena, CA 91125

May 28, 2003

Acknowledgments

I first wish to thank Prof. Robert J. McEliece, who originally introduced me to the subject and invited me to work on the problem. I also owe my gratitude to Jeremy Thorpe, who throughout my time at Caltech has been my academic mentor and good friend. The ideas he contributed to this research have proven extremely valuable. And finally, I thank Ravi Palanki for his help and Dr. Jonathan Yedidia for his contribution to initiating current interest in generalized belief propagation algorithms.

Abstract

Poset belief propagation (PBP), or Belief Propagation on Partially Ordered Sets, is a flexible generalization of ordinary belief propagation which can be used to exactly or approximately solve many probabilistic inference problems. In this paper, I introduce the problem of *Probabilistic Inference* and the algorithm of PBP as a solution to it. I summarize some experimental results comparing the performance of PBP to conventional techniques.

Table of Contents

Chapter 1. Introduction **5**
1.1. Background on Inference 5
1.2. An Exact Statement of Probabilistic Inference 6
1.3. Solutions to Inference Problems 6
1.3.1. Introduction 6
1.3.2. Graphical Models 7
1.3.3. Belief Propagation Algorithms 7

Chapter 2. Poset Belief Propagation: The Algorithm **9**
2.1. Posets 9
2.2. Graphs for Posets 9
2.3. Constructing a Poset for PI 10
2.4. The Algorithm 11
2.5. Examples 12
2.5.1. Exact Inference 12
2.5.2. Factor Graphs 12
2.5.3. Junction Graphs 13
2.5.4. Natural CV Graph 14

Chapter 3. Experiments **16**
3.1. Implementation 16
3.1.1. Introduction 16
3.1.2. C++ 16
3.1.3. C# 17
3.2. Toy Examples 17
3.2.1. Galaxy Plots 18
3.2.2. Energy Plots 20
3.2.3. Smoothed PBP 22
3.3. Larger Examples 26
3.3.1. Mastermind 26

Chapter 4. Conclusions **32**

Chapter 5. Appendix **33**
5.1. Connection to Statistical Physics 33
5.2. Complexity of PBP 34

References **36**

Chapter 1

INTRODUCTION

1.1 Background on Inference

Belief propagation, or any generalization or formulation of it such as PBP, is a method to approximately or exactly solve the problem of *Probabilistic Inference* (PI). In general, PI arises whenever one has a set of discrete random quantities, or variables, and the distribution on each of these variables (or groups of them) is unknown. For our purposes, a discrete random variable is completely characterized by its probability distribution, i.e. a table of values indicating with what probability it may take on each of some finite number of values. For example, the outcome of a fair coin toss is a random variable and it may be completely characterized by the following table: heads with probability $\frac{1}{2}$, and tails with probability of $\frac{1}{2}$. In order to "infer" these distributions, a collection of evidence, each piece of which bears on some of the variables, is given. For example, if the random variables are bits transmitted over a noisy channel, the evidence might be a received vector, or knowledge about the allowable patterns of bits. The task of PI is to "synthesize" the information given on each variable (or variables) to arrive at the correct *a posteriori* distribution on it (or the group). The term "*a posteriori*" may require some explanation. To quote from [14]: "A proposition is known *a priori* if it is known independently of empirical evidence. A proposition known on the basis of empirical evidence is known *a posteriori*." In terms of basic probability theory, an *a posteriori* distribution takes the form $\Pr\{X|Y = y\}$ and is related to the *a priori* distribution $\Pr\{X\}$ by Bayes's rule:

$$\Pr\{X|Y = y\} = \frac{\Pr\{Y = y|X\} \cdot \Pr\{X\}}{\Pr\{Y = y\}} = \frac{\Pr\{Y = y|X\} \cdot \Pr\{X\}}{\sum_x \Pr\{X = x, Y = y\}}.$$

We think of values of X causing Y to take on certain values, so that the evidence may take the form $\Pr\{Y = y|X = x\}$, which when combined with some *a priori* distribution, gives the *posterior* distribution $\Pr\{X|Y = y\}$. So Bayes's rule, in principle, is a direct formula to compute the desired distributions based on the evidence given. However, in practice, the collection of variables is huge, and so X is a vector of many unknowns. Then, direct application of Bayes's rule, which requires finding the "global" distribution, i.e. $\Pr\{X, Y\}$, means computing a table whose size grows exponentially with the number of components in X. This is kind of brute-force operation is impracticable in most cases.

1.2 An Exact Statement of Probabilistic Inference

For our purposes, PI shall formally be stated as follows: Given a set of discrete random variables

$$\{x_1, x_2, ..., x_n\},\ x_i \in A,$$

and a collection of subsets of $[n] = \{1, 2, ..., n\}$ (each subset indexing a group of these variables),

$$\mathcal{R} = \{R_1, R_2, ..., R_M\},$$

each subset R with some nonnegative "local kernel" (or "local potential") $\alpha_R(\mathbf{x}_R)$ associated with it (providing us with *evidence* about the variables indexed by R), compute

$$B_R(\mathbf{x}_R) = \frac{1}{Z} \sum_{\mathbf{v}:\mathbf{v}_R=\mathbf{x}_R} \prod_{S\in\mathcal{R}} \alpha_S(\mathbf{v}_S),\ \text{for each } R \in \mathcal{R}.$$

(We have introduced some notation. We have arranged the variables into a vector $\mathbf{x}$, which without subscript is $\in A^n$, and with subscript $\mathbf{x}_R$ denotes a new vector $\in A^{|R|}$ equal to only the components of $\mathbf{x}$ indexed by R). These are marginal probability distributions, or *marginals*, of a global distribution,

$$B(\mathbf{x}) = \frac{1}{Z} \prod_{R\in\mathcal{R}} \alpha_R(\mathbf{x}_R),$$

where Z is chosen so that $B(\mathbf{x})$ be normal, viz.,

$$Z = \sum_{\mathbf{x}\in A^n} \prod_{R\in\mathcal{R}} \alpha_R(\mathbf{x}_R).$$

We note that $B(\mathbf{x})$ is a table of A^n values. So, for example, if our variables are binary and we have 8,000 of them (as could possibly be the case when the variables correspond to data bits in a long, transmitted data block), then $A^n = 2^{8000} \approx 1.737 \times 10^{2408}$. We see how easy it is for the computation to become prohibitively complex. So directly computing $\{B_R(\mathbf{x}_R)\}$ from the formula is not always an option. More efficient solutions must be employed if PI is to be solved in these cases.

1.3 Solutions to Inference Problems

1.3.1 Introduction

Most methods to efficiently solve PI involve some kind of *Belief Propagation* (BP). Because the application of probabilistic reasoning is so far reaching, BP-like algorithms have been discovered many times and expressed in many forms. The first BP algorithms were of the kind introduced by Pearl [7], which solved PI exactly and only on cycle-free graphs.

1.3.2 Graphical Models

It is often desirable to somehow represent the probabilistic relationships between groups of associated variables in a graph. These graphs, both as mathematical objects and as visualizations, are useful in describing fast solutions to PI problems. In general, these graphs have nodes to represent the variables or groups of variables, and edges between nodes if they are not conditionally independent. When a graph contains a group of such nodes which forms a cycle, the graph is said to be "loopy". When the graph is connected and not "loopy" then it is "treelike".

1.3.3 Belief Propagation Algorithms

The trick to any belief propagation algorithm is the following: avoid computation of the global distribution $B(\mathbf{x})$ by iteratively moving estimate distributions $\{\widetilde{B}_R(\mathbf{x}_R)\}$ towards the desired distributions $\{B_R(\mathbf{x}_R)\}$. Maintain at each node in the graphical model a "guess" $\widetilde{B}$ as to what the actual distribution B is over its associated variables, and iteratively update these "belief distributions" or simply "beliefs" according to small messages passed about shared variables from connected nodes. In a treelike graphical model, the messages need only to be passed once in each direction, and the beliefs at the nodes become exactly equal to the desired marginals. At no point in the algorithm is a table of size more than A^m updated, where m is the number of variables in the largest node. In general, the complexity is approximately

$$\text{complexity} \approx O(I \cdot s \cdot A^m)$$

where the iteration count I is the number of times any one edge is updated, and s is number of nodes of size m. We apply BP when this complexity represents a significant improvement over $O(A^n)$.

Pearl's "ordinary belief propagation", or OBP, applied to loopy graphs performs only *approximate* inference. By "approximate", we mean that the marginals $\{B_R(\mathbf{x}_R)\}$ are never exactly found, but that approximations to them are provided. In the last ten years or so, it was empirically shown many times that despite a graphical model's loopiness, a belief propagation algorithm performing approximate inference on it could do quite well. Perhaps the most dramatic demonstration of this fact was the monumental success of "Turbo Codes" [13]. These error correcting codes made communication at rates astonishingly close to channel capacity (as originally defined and predicted by Claude Shannon in 1948) with extremely low probability of error a reality.

Although this success of loopy BP was accepted with much fervor, scientists could not explain (mathematically justify) the observed behavior. Partially, the formulation of Poset Belief Propagation given in [4] explains this phenomenon. The key to this understanding, which derives mainly from the work of Yedidia et. al. [10, 11], is that one can define an *energy function* over the set of possible answer distributions $\{\widetilde{B}_R(\mathbf{x}_R)\}$, and that this energy function is minimized when the possible distributions are exactly

equal to the desired marginals $\{B_R(\mathbf{x}_R)\}$. Every belief propagation algorithm, even one on a loopy graph, attempts to perform this minimization on some *approximation* to the actual energy surface. So it seems reasonable to expect a BP algorithm to achieve approximately good results when the energy approximation associated with it is sufficiently accurate. I leave a more rigorous discussion of this point to the Appendix.

PBP is the most general statement of a solution to PI known thus far, and it includes as special cases [12] optimal, brute-force inference (direct application of Bayes's Rule), ordinary belief propagation [7], probability propagation [8], the generalized distributive law [1, 2], the sum-product algorithm [3], and generalized belief propagation [10, 11], as well as many other instances whose efficacy has not yet been carefully studied.

Chapter 2

Poset Belief Propagation: The Algorithm

Throughout this chapter, I will be using the notation provided in "An Exact Statement of Probabilistic Inference" (1.2). This exposition of Poset Belief Propagation (PBP) will read as a solution to the problem of PI stated in exactly that way.

2.1 Posets

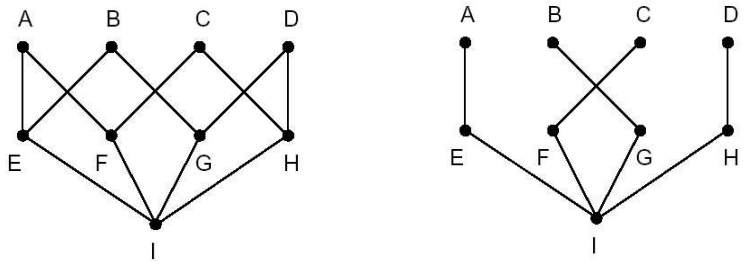
Before we can discuss Poset Belief Propagation, it seems prudent to define exactly what a poset is. A finite *partially ordered set* (or poset for short) is a set P together with a binary relation $\leq$ satisfying the following three axioms:

1. For all $\rho \in P$, $\rho \leq \rho$. $(reflexivity)$
2. If $\rho \leq \sigma$ and $\sigma \leq \rho$, then $\rho = \sigma$. $(antisymmetry)$
3. If $\rho \leq \sigma$ and $\sigma \leq \tau$, then $\rho \leq \tau$. $(transitivity)$

A technical note: we will use $\rho \geq \sigma$ to mean $\sigma \leq \rho$, $\rho < \sigma$ to mean $\rho \leq \sigma$ but $\rho \neq \sigma$, etc..

2.2 Graphs for Posets

The *Hasse diagram* $H = H(P)$ for a poset P is a graph with a vertex for every element $\rho \in P$, and edge for every pair (ρ, σ) such that $\rho > \sigma$ and there is no element $\tau \in P$ such that $\rho > \tau > \sigma$. We say P is *connected* if the graph $H(P)$ is connected. If $H(P)$ does not contain cycles and it is connected, we say P forms a *tree*, or is *treelike*. This distinction is shown by example in the following figure:



(a) *Connected* Hasse diagram (b) *Treelike* Hasse diagram

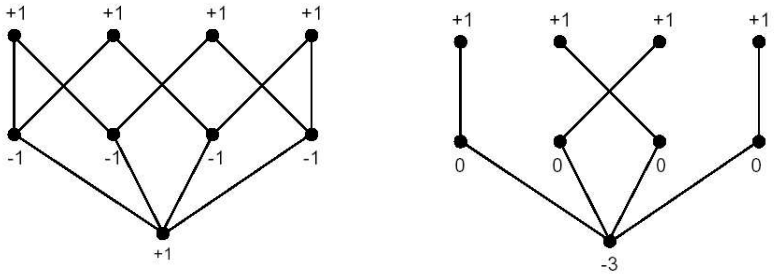
We represent an ordering of an edge $e = (\rho, \sigma)$ by placing ρ "above" σ in the Hasse diagram. We call ρ the *parent* and σ the *child*. The *ancestors* of ρ are then $\{\sigma : \rho \leq \sigma\}$ and the *descendants* of ρ are $\{\sigma : \sigma \leq \rho\}$. So, for example, in poset (a) the descendants of A are $\{A, E, F, I\}$ whereas in poset (b) the descendants of A are $\{A, E, I\}$.

2.3 Constructing a Poset for PI

First we introduce the concept of a *labelled poset*. Given a poset, we label each element $\rho \in P$ with some subset of $[n] = \{1, 2, ..., n\}$, and denote this label $L(\rho)$. The members of $L(\rho) \subseteq [n]$ index the variables with which ρ is associated. The labelling is called *consistent* if $L(\sigma) \subseteq L(\rho)$ when $\sigma \leq \rho$. We require that the labelling be consistent. We say that a variable x_i is assigned to an element ρ if $i \in L(\rho)$. Furthermore, we assign each element of a poset P an *overcounting number* $\phi(\rho)$ such that

$$\sum_{\rho:\rho\geq\sigma} \phi(\rho) = 1 \text{ for all } \sigma \in P.$$

The assignment may be carried out according to a numbering algorithm provided in [4]. For the example posets given in the previous section, the overcounting numbers are assigned as follows:



(a) $\phi(\rho)$ assigned to first poset (b) $\phi(\rho)$ assigned to second poset

Now, given the collection $\mathcal{R}$ of subsets of $[n]$, we assign each element $\rho \in P$ some $\mathcal{R}(\rho)$ $\subseteq \mathcal{R}$ and require:

$$\begin{aligned}
&(i) && \bigcup_{R\in\mathcal{R}(\rho)} R \subseteq L(\rho), && \text{for all } \rho \in P \\
&(ii) && \mathcal{R}(\sigma) \subseteq \mathcal{R}(\rho), && \text{if } \sigma \leq \rho \\
&(iii) && \sum_{\rho:i\in L(\rho)} \phi(\rho) = 1, && \text{for all } i \in \{1, 2, ...n\} \text{ ("1-variable balance"),} \\
&(iv) && \sum_{\rho:R\in\mathcal{R}(\rho)} \phi(\rho) = 1, && \text{for all } R \in \mathcal{R} \text{ ("conservation of energy").}
\end{aligned}$$

Such a poset P is called a *poset for* $\mathcal{R}$, and we will use it to solve the PI problem. One condition that is desirable but not necessary is k-variable balance:

$$(v) \quad \textstyle\sum_{\rho:T\subseteq L(\rho)} \phi(\rho) = 1 \quad \text{for all } T \subseteq [n] \text{ s.t. } |T| = k.$$

(Perhaps this is a good place to briefly motivate the use of overcounting numbers. They are used to keep track of the "multiplicity" of variables and potentials. Our goal is for each piece of evidence to be counted exactly once, overall. The more balance conditions are met with overcounting numbers in a poset, the better the corresponding energy approximation (see Appendix) is likely to be.) We have implicitly assigned the "local kernels" to the elements of P: a local kernel $\alpha_R(\mathbf{x}_R)$ is assigned to ρ simply if $R \subseteq L(\rho)$. We define the *local kernel at* ρ as:

$$\alpha_\rho(\mathbf{x}_\rho) = \prod_{R \in \mathcal{R}(\rho)} \alpha_R(\mathbf{x}_R).$$

To summarize, having been given the statement of PI, we have constructed a poset for $\mathcal{R}$ by assigning variables and kernels to elements ρ of a given poset P such that certain "balance" and "consistency" conditions are met.

2.4 The Algorithm

Throughout the algorithm, each element ρ (or corresponding node in the Hasse diagram) has associated with it a *belief distribution*, or simply *belief*, which is initialized to:

$$b_\rho(\mathbf{x}_\rho) = \frac{1}{N}\alpha_\rho(\mathbf{x}_\rho),$$

where N normalizes $\alpha_\rho(\mathbf{x}_\rho)$ over all choices for $\mathbf{x}_\rho$. If no local kernel is assigned to a node, we take the convention that its belief is initialized to the uniform distribution over its assigned variables. This is the best estimate of the actual distribution over its variables, given no evidence at all. An edge $e = (\rho, \sigma)$ in the Hasse Diagram for the poset $\mathcal{R}$ for is consistent if

$$\sum_{\mathbf{x}_\rho \setminus \mathbf{x}_\sigma} b_\rho(\mathbf{x}_\rho) = b_\sigma(\mathbf{x}_\sigma).$$

The algorithm proceeds as follows: For each inconsistent edge $e = (\rho, \sigma)$, define the following "correction table":

$$\Delta_e(\mathbf{x}_\sigma) = \frac{\sum_{\mathbf{x}_\rho \setminus \mathbf{x}_\sigma} b_\rho(\mathbf{x}_\rho)}{b_\sigma(\mathbf{x}_\sigma)},$$

and update the beliefs throughout P according to:

$$b_\tau(\mathbf{x}_\tau) \leftarrow b_\tau(\mathbf{x}_\tau) \cdot \Delta_e(\mathbf{x}_\sigma),$$

for each $\tau \in P$ such that $\sigma \leq \tau$, $\rho \not\leq \tau$. Stop after a sufficiently large number of iterations, or when no edge is inconsistent. The algorithm is successful if:

$$b_\rho(\mathbf{x}_\rho) \approx B_\rho(\mathbf{x}_\rho),$$

where B_ρ is shorthand for the global distribution $B(\mathbf{x})$ marginalized down to only the variables indexed by $L(\rho)$.

Special note: the above gives the algorithm in only a serial update schedule. That is, every time an edge is updated, beliefs are modified, and the next edge update is possibly immediately affected by the previous change. In order to implement a different update schedule, we can keep at each node, in addition to a belief and a kernel, a "future correction factor" $\delta_\rho(\mathbf{x}_\rho)$, which is initialized to the uniform distribution at the start of each iteration. When an edge $e = (\rho, \sigma)$ is updated, instead of updating the beliefs at $\{\tau : \sigma \leq \tau,\ \rho \not\leq \tau\}$ immediately, we only update $\delta_\tau(\mathbf{x}_\tau)$:

$$\delta_\tau(\mathbf{x}_\tau) \leftarrow \delta_\tau(\mathbf{x}_\tau) \cdot \Delta_e(\mathbf{x}_\sigma).$$

Then, at an arbitrary time chosen, say at the end of an entire iteration for a completely parallel update schedule, or at the end of every edge update for a completely serial schedule, the beliefs at each node ρ are adjusted:

$$b_\tau(\mathbf{x}_\tau) \leftarrow b_\tau(\mathbf{x}_\tau) \cdot \delta_\tau(\mathbf{x}_\tau).$$

It seems to be experimentally true that the serial update schedule moves the algorithm to convergence quickest, but this added generality may be useful for analytical purposes.

2.5 Examples

I have stated that PBP is the most general formulation of a solution to PI to date. Here, I will show how to construct posets for $\mathcal{R}$ which correspond to different known solutions to PI.

2.5.1 Exact Inference

The brute-force computation of the marginals $\{B_R(\mathbf{x}_R)\}$ is an instance of PBP. To carry out this computation, we construct a poset with exactly one element ρ_o. We assign *every* kernel $\alpha_R(\mathbf{x}_R)$ to this one element. We have $\phi(\rho_o) = 1$. Clearly, conditions $(i) - (iv)$ are met and P is a poset for $\mathcal{R}$. Then, when PBP is initialized, we are forced to compute

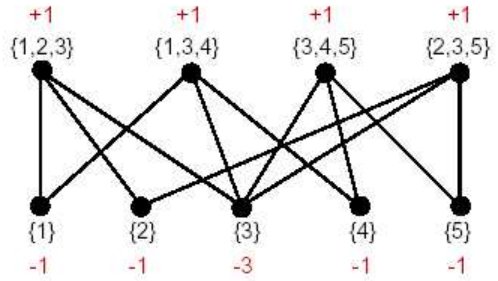
$$b_{\rho_o}(\mathbf{x}_{\rho_o}) = \frac{1}{N}\alpha_{\rho_o}(\mathbf{x}_{\rho_o}) = \frac{1}{N}\prod_{R\in\mathcal{R}} \alpha_R(\mathbf{x}_R),$$

which is exactly the global distribution $B(\mathbf{x})$. If we wish to extract $\{B_R(\mathbf{x}_R)\}$ from this, we perform the necessary marginalizations.

2.5.2 Factor Graphs

The *sum-product algorithm* [3], equivalent to ordinary belief propagation, can also be carried out as an instance of PBP. We construct a two-level poset P (meaning a poset

whose Hasse diagram consists of two levels of nodes), where there is a top-level node for each $R \in \mathcal{R}$, each one assigned the potential $\alpha_R(\mathbf{x}_R)$, and a bottom level node for each variable x_i. Such a poset, or its Hasse diagram, is called a *factor graph*. For example, if $\mathcal{R} = \{\{1,2,3\},\{1,3,4\},\{3,4,5\},\{2,3,5\}\}$, the the factor graph looks like this:

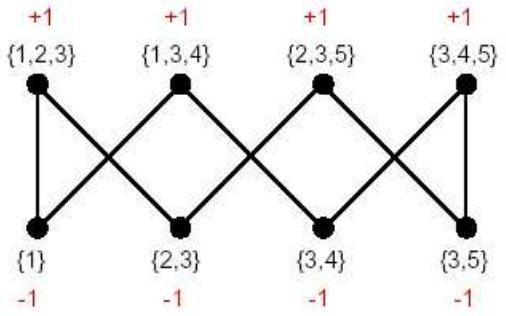


Note that it is possible that some of elements of $\mathcal{R}$ will consist of a single variable (number, technically). That is, it is possible that we are also given evidence $\alpha_i(x_i)$ about a single variable i, in addition to evidence about that variable in a group. In this case, the single variable node i is not in the top level, but remains in the bottom level, and its parents are each assigned at least two kernels, one for the group, and one for each of their variables also assigned a kernel.

2.5.3 Junction Graphs

PBP can also carry out the *generalized distributive law* algorithm [1, 2]. The poset P we construct to carry out the equivalent of this algorithm will be referred to as a "junction graph" or "junction poset". Here again, we construct a two-level poset, with a top level node for each $R \in \mathcal{R}$. The bottom level nodes are intersections of one or more top level nodes. (I've introduced the notion of "intersecting" two nodes. A node τ constructed from the intersection of ρ and σ is simply a node whose label $L(\tau) = L(\rho) \cap L(\sigma)$.) The intersections are allowed to be of size greater than 1 (allowing possible improvement over the factor graph construction). We impose one more key requirement that $P_i = \{\rho : i \in L(\rho)\}$ traces a tree in the Hasse diagram (or equivalently is a poset whose Hasse diagram is a tree), for every variable x_i. For the same example

set $\mathcal{R}$ given above, the "junction graph" may be:



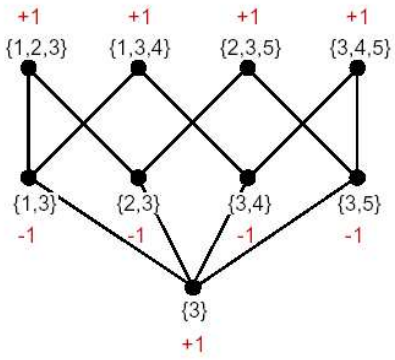
Note that there is more than one junction graph that can be constructed for every statement of PI. To construct a junction graph given a set $\mathcal{R}$ one may use the following algorithm [4]: First construct a graph G with vertex set V, consisting of a node ρ for each element of $\mathcal{R}$. We assign each ρ a unique $R_i \in \mathcal{R}$ and label it $L(\rho) = R_i$. We add an edge e between each two vertices ρ and σ and label this edge $L(e(\rho, \sigma)) = L(\rho) \cap L(\sigma)$. The set of all edges is E. $G = (V, E)$ forms a clique, and the subgraph $G_i = (V_i, E_i)$ where $V_i = \{\rho : i \in L(\rho)\}$ and $E_i = \{e(\rho, \sigma) : i \in L(e(\rho, \sigma))\}$ also forms a clique [4]. Now, for each $i \in [n]$, choose a spanning tree T_i of the subclique G_i, and delete i from the labels of all edges $e \in E$ not in T_i. The resulting graph is a "Junction Graph" as the one defined in [1, 2].To construct a junction poset from this, we define the following two sets: $Top = \mathcal{R}$, $Bottom = \{L : L = L(e(\rho, \sigma))$ for some $e(\rho, \sigma)\}$. The set $Top \cup Bottom$ together with the binary relation $S_1 \leq S_2$ iff $S_1 \subseteq S_2$ forms a poset whose Hasse diagram has two levels, with a top level node for each element of Top and a bottom level node for each element of $Bottom$. To run PBP on this poset, we assign each $\alpha_R(\mathbf{x}_R)$ to the top level node whose label is R.

2.5.4 Natural CV Graph

Given the set $\mathcal{R}$, let $\mathcal{S} = \{S_1, S_2, ..., S_K\}$ be the set of all intersections of one or more elements of $\mathcal{R}$. Together with the binary relation $S_1 \leq S_2$ iff $S_1 \subseteq S_2$, this forms a poset, ordered by "inclusion". For each $\rho \in \mathcal{S}$, we define

$$\begin{aligned} L(\rho) &= \rho \\ R(\rho) &= \{R \in \mathcal{R} : R \subseteq \rho\}. \end{aligned}$$

For the example set $\mathcal{R}$ given previously, the CV poset is:



For this kind of poset, not only are requirements $(i) - (iv)$ met, but there is also k-variable balance. This is because here, for any $T \subseteq [n]$, $\{\rho : T \subseteq L(\rho)\}$ takes the form $\{\rho : \rho \geq \rho(T)\}$, which is an "upset" that by definition must be assigned net overcounting 1. This construction is called the *Natural Cluster Variations (CV) Method* [10, 11], and the corresponding poset, or Hasse diagram, is called the natural CV graph. To run PBP on this poset, we assign each $\alpha_R(\mathbf{x}_R)$ to the top level node whose label is R. The resulting algorithm is equivalent to "Generalized Belief Propagation" (ibid.).

Chapter 3

Experiments

In this chapter, I summarize the results of my experimentation. I should explain the basic concepts and motivation. Besides the generality of PBP being mathematically elegant, and in a way summarizing all that is known about efficient solutions to PI, it gives the engineer a method of adjusting the complexity of the solution to the desired performance. Given the set $\mathcal{R}$ and the corresponding pieces of evidence, one can construct many posets. Typically the maximal elements of $\mathcal{R}$ will be chosen as top level nodes. Even with this degree of freedom removed, the algorithms which result from different poset choices have differing complexity (see Appendix). This kind of clarity on the problem is new. To be able to understand OBP, the generalized distributive law, generalized belief propagation, brute-force computation, and many other possibilities as all being points in a spectrum of possible BP solutions is something novel. Through experimentation, we hoped to reach a better understanding of PBP by comparing the performance of its instances on various concrete examples.

3.1 Implementation

3.1.1 Introduction

In this section, I briefly discuss the approach taken to implement PBP.

3.1.2 C++

Because C++ compiles in most every environment, and because C++ is both fast and object-oriented, I chose it initially to describe the algorithm in computer language. I defined a Poset class, which contained a list of Edges and Nodes. Nodes were objects containing a kernel and a belief. Nodes also had pointers to parent Nodes and children Nodes. The trickiest part of the implementation was the description of a "Function" class. PBP requires the manipulation of many discrete functions – beliefs and kernels are both instances of this. A Function is simply a table, and a list of variables which imply the meaning of each entry of the table. One must implement the capability to multiply two such Functions together. The set of variables over which the product is a Function is the union of the multiplicands' sets, for example. Also, for a given poset, each edge $e = (\rho, \sigma)$ has a unique set of nodes $\{\tau : \sigma \leq \tau,\ \rho \not\leq \tau\}$. This set of Nodes is computed once at the beginning for each Edge, and the Edge stores a list of pointers to those Nodes. Those are the Nodes which are updated when the edge e is made consistent. Once this is all defined, the implementation of the actual update rule is obvious. The

output of a simulation was universally a `.m` file which could easily be used by `MATLAB` to interpret the data.

3.1.3 C#

Eventually, my experiments involved the declaration and deletion of a huge number of Posets in any one simulation. Also, Posets began to grow larger as demands for more interesting examples arose. Most experienced programmers know that unless seriously planned for, pushing code to its limits like this usually results in unforeseen errors. My implementation in C++ was especially vulnerable to memory leaks since data deletion had to be done manually. Furthermore, both debugging and editing of code was difficult with the unavailability of good debugging tools and `emacs` my only editor. This all grew rather cumbersome, and I decided to port all the simulation code into C#. C# is a language which, like Java, is more abstract than earlier languages such as C and C++. It is entirely object oriented and hides many details from the programmer. Notably, it carries out garbage collection of unused data. Although C# is a Microsoft product, which makes it slightly less portable than C++, the development environment and debugging tools, combined with the inherent power and flexibility of the language, have made it possible for me to overcome some of the barriers I encountered coding in C++. I would not recommend implementing PBP in all its generality in any language not well suited for object oriented design. It should be noted that if PBP is ever to become useful in a practical sense, it will not need to be implemented in all its generality. Rather, a highly optimized equivalent of exactly one instance will be used.

3.2 Toy Examples

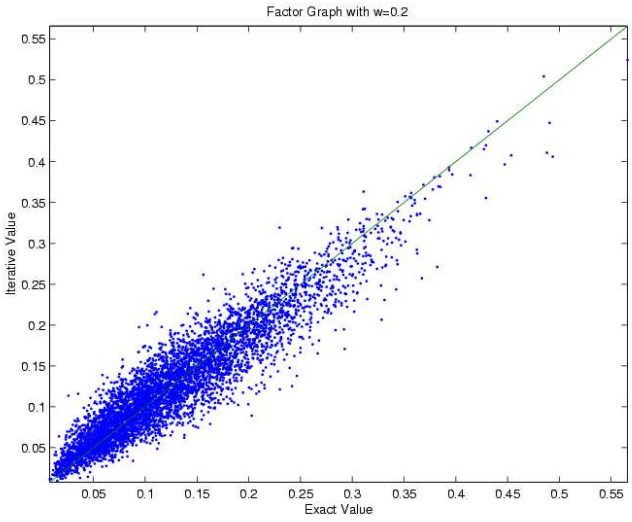
This work began with the goal of comparing the performance of PBP run on a factor graph, junction graph, and natural CV graph for the set

$$\mathcal{R} = \{\{1,2,3\},\{1,3,4\},\{3,4,5\},\{2,3,5\}\}.$$

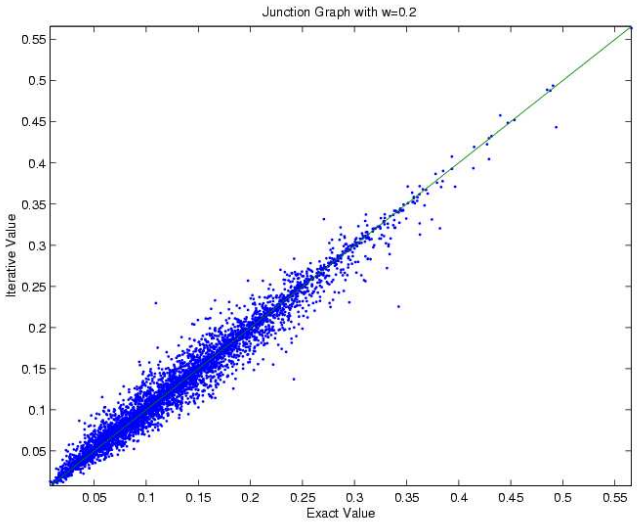
The junction graph chosen was the one shown above (section 2.5.3). There is only one factor graph and one natural CV graph associated with this set, and they are also shown above (2.5.2, 2.5.4, resp.). To run PBP on each of these three posets, it is necessary to associate evidence kernels with the elements of $\mathcal{R}$. The evidence kernels were chosen at random. That is, each unique element of the discrete kernel function was chosen from a uniform distribution, typically on $[0,1]$. Once the evidence was chosen and the posets determined, PBP was run until either edges were consistent or the maximum number of iterations was exceeded. Typically, on these posets, edges were almost entirely consistent after 10 or so iterations.

3.2.1 Galaxy Plots

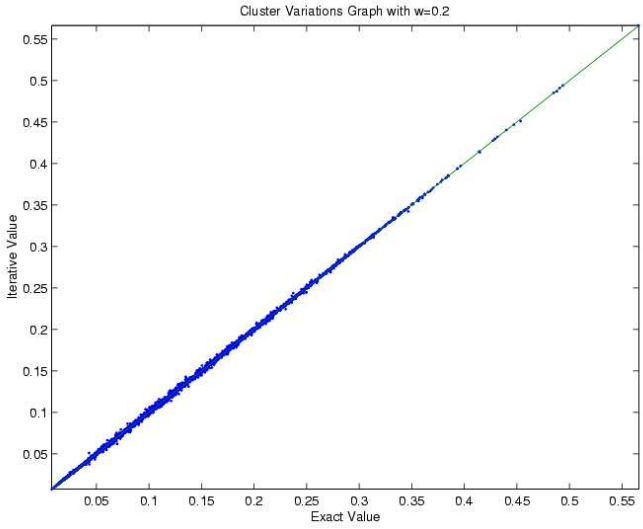
Introduction One idea for comparing the performance of these three small examples was to make "galaxy plots" (so termed because of their appearance). In this paper, a galaxy plot consists of the points $\{(B_R(\mathbf{x}_R), b_R(\mathbf{x}_R))\}$ plotted in the $x-y$ plane. Recall that the goal of PBP is to compute, exactly or approximately, the discrete functions $\{B_R(\mathbf{x}_R)\}$, and so, for each entry of this function, there is an "exact value" $B_R(\mathbf{x}_R)$ which can be computed by brute-force marginalization of the true global probability function, and some "iterative value" $b_R(\mathbf{x}_R)$, which is computed using one of the above three instances of PBP. The closer the points $\{(B_R(\mathbf{x}_R), b_R(\mathbf{x}_R))\}$ fall to the line $y = x$, the better the performance of the inference algorithm. We choose the initial conditions (evidence kernels) at random many times, and add to the galaxy plot at the termination of each small experiment. These are the results:



"Factor Graph" with $w = 0.2$



"Junction Graph" with $w = 0.2$



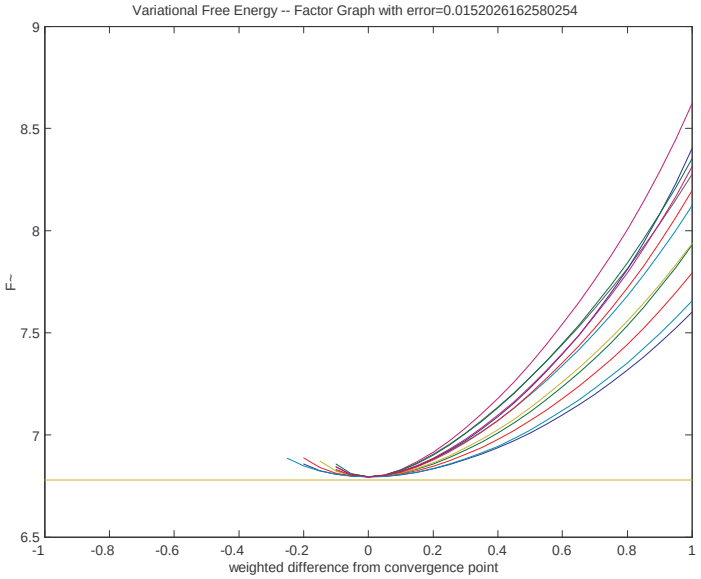
"Natural CV Graph" with $w = 0.2$

Discussion The first obvious conclusion is that, for this instance of PI, the factor graph (FG) performs worst, with the natural CV graph (NCVG) doing best, and the junction graph (JG) in the middle. One possible explanation is this: JG outperforms FG because the bottom level nodes contain more than one variable. So, when information coming from the evidence of one top level node is passed to the others, it is passed over more than one variable (the "correction table" is a function of more than one variable). Thus a belief is always updated according to more complete information. It is feasible that this would evolve the beliefs to a point more close to the actual marginals of the global. In terms

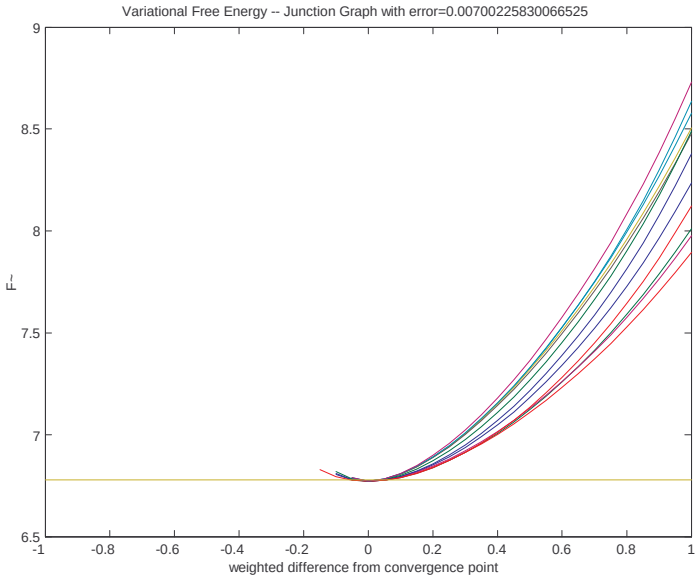
of statistical physics, the beliefs are moving towards a minimum of an energy function which more closely approximates the actual one. The approximation is probably better because the bottom level nodes are larger. Similarly, NCVG outperforms JG because updates are based on even more complete information, and the evolution of the beliefs takes place on a energy surface which even more closely approximates the real one.

3.2.2 Energy Plots

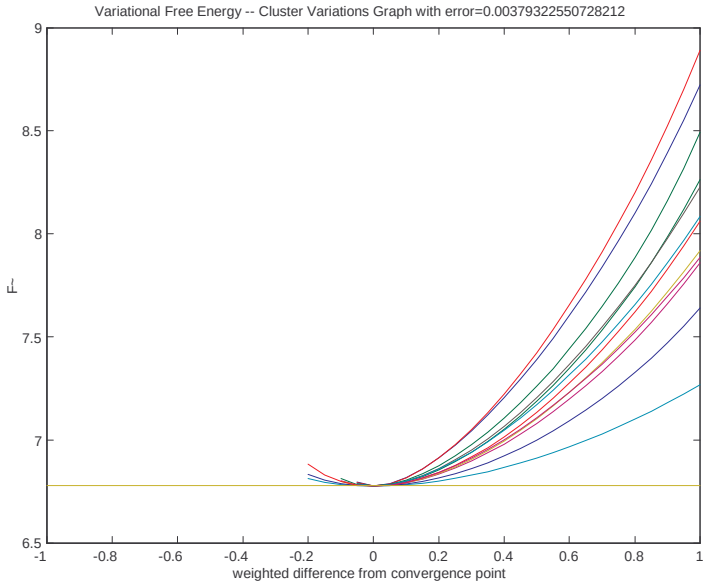
To illustrate the point that when PBP converges the set of beliefs are at a minimum of an energy function, the following plots were produced:



Energy Function for "Factor Graph"



Energy Function for "Junction Graph"



Energy Function for "Natural CV Graph"

It is necessary to read the first section of the Appendix in order to understand this section. The Helmholtz free energy is shown above as the flat line. It is the minimum of the actual variational free energy function. We attempt to illustrate the approximation to this actual energy function by plotting points along it from different "directions". The minimum of each colored curve occurs when the belief set is equal to the belief set PBP converged to in each instance. We can move away from this point by picking another consistent belief set at random, and taking linear combinations of these two belief sets,

and computing the approximate variational free energy along the way. We notice that the value of the approximation always increases as we move away from the convergence point in each direction, meaning that we are probably at a local minimum. The difference between the minimum at which the algorithm converges and the Helmholtz free energy is shown in the title of the plots, summarized here:

Algorithm (Instance of PBP)	$\left\|\widetilde{F}(b) - F\right\|$, at convergence
"Factor Graph"	0.015203 bits
"Junction Graph"	0.007002 bits
"Natural CV Graph"	0.003793 bits

Now, these are the results for exactly one set of initial conditions. However, similar results are obtained for a different choice of kernels on these posets. These results were chosen because they illustrate the following point well: the FG approximation is worse than the JG approximation which is worse than the NCVG approximation. For this example, we notice the error at the local minimum decreases by a factor of about two between successive algorithms – a substantial improvement.

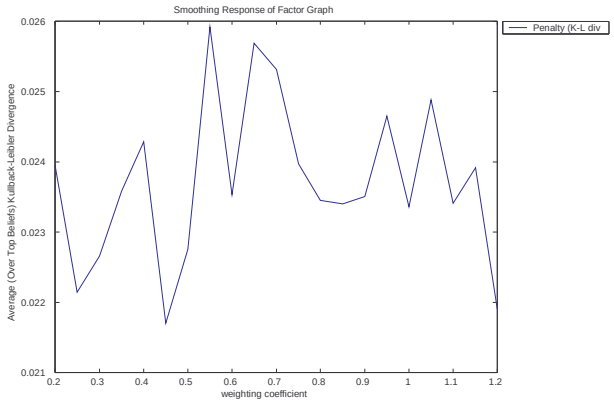
3.2.3 Smoothed PBP

The reader may have noticed that the galaxy plots indicate a value of w. It was noticed very early on that the algorithm stated exactly as it has been in this paper does not always work. Sometimes, the belief updates are too aggressive and the algorithm does not converge. In these cases, we apply a "smoothing" or "damping" coefficient w to the update rule:

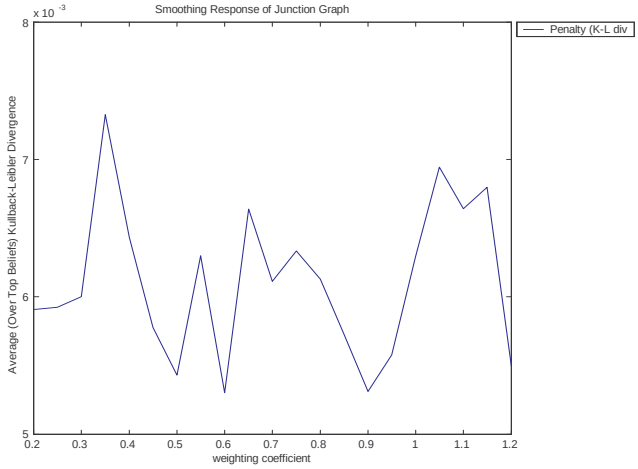
$$b_\tau(\mathbf{x}_\tau) \leftarrow b_\tau(\mathbf{x}_\tau)^{1-w} \cdot (b_\tau(\mathbf{x}_\tau) \cdot \Delta_e(\mathbf{x}_\sigma))^w = b_\tau(\mathbf{x}_\tau) \cdot \Delta_e(\mathbf{x}_\sigma)^w.$$

Instead of moving the beliefs completely to their new position, they are taken to a weighed multiplicative, or geometric, mean between their current position and the one predicted by the original algorithm. It should be emphasized that the only effect of this is to help convergence. If the algorithm is already converging, it will not do any better with smoothing. The beliefs are still moving along the same approximation to the energy function, however bad, only more carefully. With the intention of perhaps

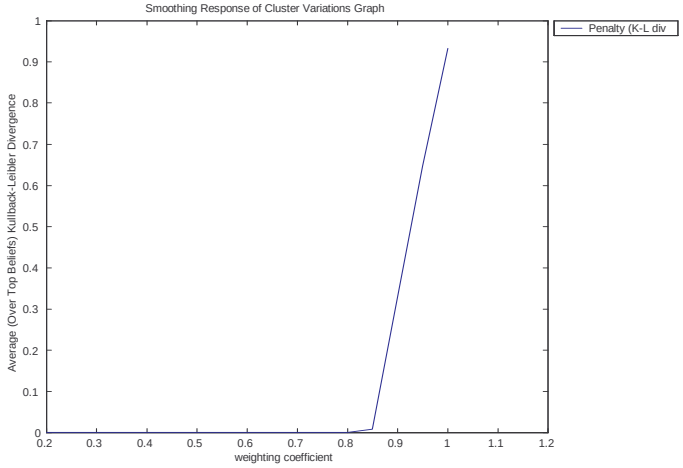
validating this hypothesis the following plots were made:



Performance Dependence on w for "Factor Graph"

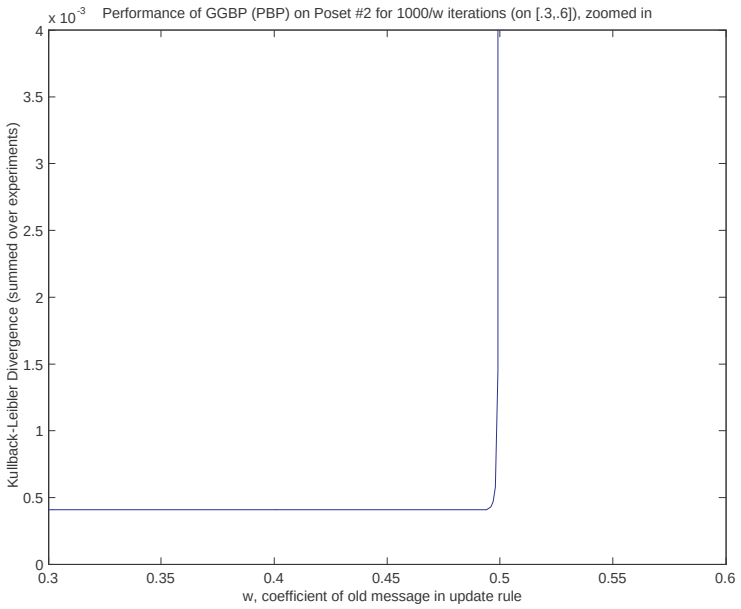


Performance Dependence on w for "Junction Graph"



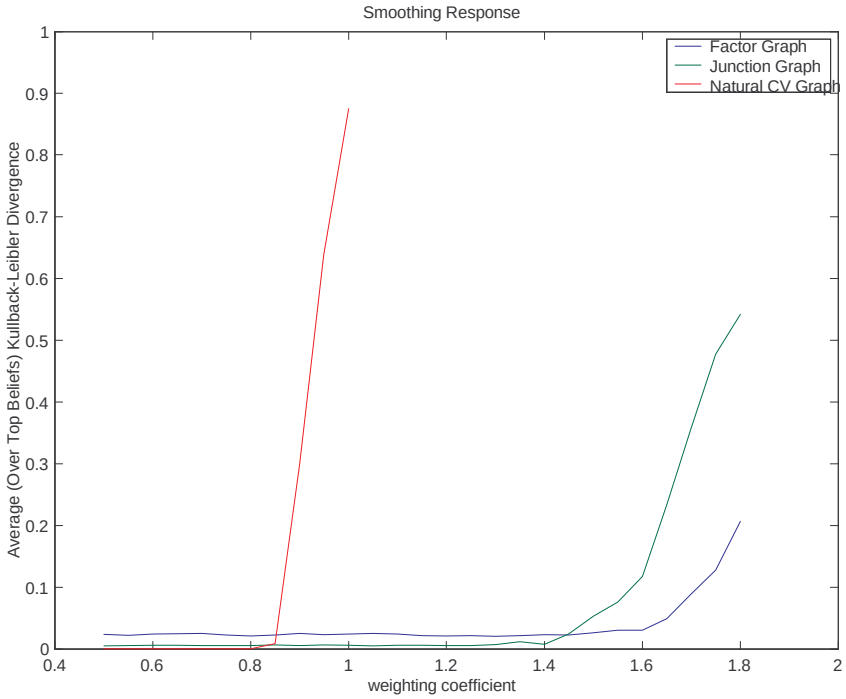
Performance Dependence on w for "Natural CV Graph"

On the $x-$axis of these plots is the coefficient w used in the update rule. On the $y-$axis is the average Kullback-Leibler divergence between the actual beliefs and beliefs converged to by the various algorithms, averaged over both many choices of initial kernels, and over the four different top level beliefs. The bumps and dips in the first two plots are simply artifacts of noise, the performance is the same over all choices of $w \in [0, 1]$. This is because the algorithm always converged in this region. However, for the natural CV graph, there is a breaking point in the convergence. A certain $w^* \approx 0.85$ exists below which convergence is achieved and above which the algorithm diverges. The exact value of w^* depends greatly on exactly which order the beliefs are updated in, however its existence is always present. For the same natural CV graph, but a different update schedule, the following similar plot was generated:



Performance Dependence on w for "Natural CV Graph" (again)

Here $w^* \approx 0.5$, but it is especially clear that the performance once $w < w^*$ is constant, as expected. It was interesting to know if the algorithm could be made even more efficient with w. Smoothing slows the algorithm down if $w < 1$, but speeds the algorithm up if $w > 1$, assuming the computation of the multiplicative means does not asymptotically affect performance. For the factor and junction graphs, it was actually possible to use $w > 1$ and still obtain convergence. Here is another coefficient response plot (using the same update schedule as the one used for the first set of plots):

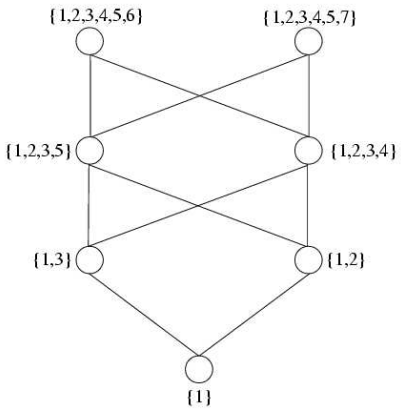


Performance for $w > 1$

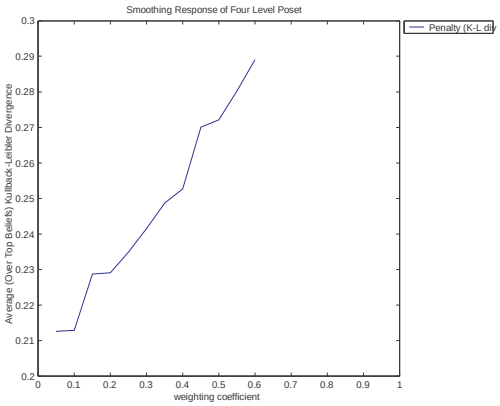
w^* occurs at about 1.4 for the JG, and 1.6 for the FG. So the algorithm as stated originally (with $w = 1$) was too "conservative". It is natural to ask "Will smoothing always work?". That is, for every poset, for each update scheduling rule, does there exist some w^* below which convergence is guaranteed? It is believed that the answer to this is no. For example, given the set

$$\mathcal{R} = \{\{1,2,3,4,5,6\},\{1,2,3,4,5,7\}\},$$

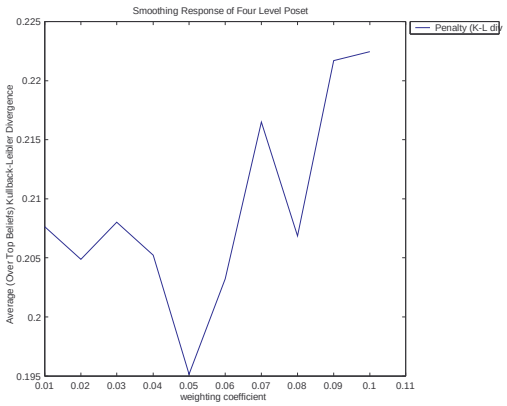
each member of which is associated with some evidence kernel, and the four-level poset:



the experiment shows that for seemingly no value of w^* and no update schedule will PBP converge. Consider the following coefficient response plots for PBP on this poset:



(a) performance on $[.05, 1]$



(b) performance on $[0.01, .1]$

In the first plot, we see that apparently w^* lies to the left of .05, since the performance starts out bad and only gets worse, resembling the response of earlier posets in the $w > w^*$ region. Then, pushing w back onto the smaller interval $[0.01, .1]$, we find that the performance is still bad, and probably getting worse with larger values of w, despite the noisy bumps (note: running PBP with such small values of w takes many, many actual iterations to equate the movement of one step of the original algorithm, so the plots are noisy because a relatively small amount of data was generated per point on the plot). We can confidently say that for a serial update schedule, $w^* < 0.01$ for this four-level poset. Whether in this case w^* is arbitrarily close to 0 (implying that sometimes no amount of smoothing will help convergence) or some tiny value close to 0 is irrelevant from the engineering standpoint, because an algorithm which proceeds so slowly is too computationally expensive to be useful.

3.3 Larger Examples

3.3.1 Mastermind

Introduction In the children's game of Mastermind, player A selects a random pattern of pegs of various colors, and player B attempts to discover this pattern, which is hidden from view. To discover the pattern, B will provide A with guess patterns, one at a time. In return, A will tell B some limited information about the goodness of each of his guesses. In this way, B iteratively improves his guesses until one is correct. The goal is for B to arrive at A's pattern in as few guesses as possible. We define a simplified version of the game which can be solved using belief propagation. This general kind of "data-fusion" problem has wide-reaching applicability. See [15] for a more complete explanation.

The Problem Assume we have n binary random variables, taking values uniformly in the set $S = \{-1, 1\}$, which we will arrange into a vector $\mathbf{x}$. In terms of the Mastermind analogy, think of there being two kinds of pegs, say black and white (corresponding to -1 and 1). The vector $\mathbf{x}$ is a pattern of "pegs" in locations $1, 2, ..., n$. Player A selects this pattern at random with no bias for black or white pegs. The problem is to infer the value of $\mathbf{x}$, given a set $\{\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_N\}$ of test vectors, and their dot product with $\mathbf{x}$, with each $\mathbf{g}_i \in \{-1, 0, 1\}^n$. Each $\mathbf{g}_i$ is thought of as a "guess" player B makes about some of the pegs A chose, and each $y_i = \mathbf{x} \cdot \mathbf{g}_i$ is the information A provides B about his guess. Because of the nature of the dot product, only the non-zero components of $\mathbf{g}_i$ will help B infer $\mathbf{x}$. For simplicity, we will assume each $\mathbf{g}_i$ contains exactly s non-zero components, so that each $y_i \in \{-s, -s+1, ..., 0, ..., s-1, s\}$. Also, to borrow notation from [15], let $\mathbf{v}^i$ denote the components of $\mathbf{v}$ corresponding to the nonzero components of $\mathbf{g}_i$. In this notation, $y_i = \mathbf{x} \cdot \mathbf{g}_i = \mathbf{x}^i \cdot \mathbf{g}_i^i$.

Solutions To infer the value of $\mathbf{x}$, in this context, means to provide a discrete probability distribution on $\mathbf{x}$, or each of its components. One correct solution is this: each pair $(\mathbf{g}_i\ y_i)$ limits the possible patterns for $\mathbf{x}^i$ to only those which result in $y_i = \mathbf{x}^i \cdot \mathbf{g}_i^i$. Thus, starting with the set of all possible values for $\mathbf{x}$, and eliminating the impossible ones one-by-one as we iterate through each of the guesses $\mathbf{g}_i$, leaves us with a certain finite number of possible causes which are consistent with all the observations y_i. The correct distribution to be provided is one which is uniform over those causes, and zero elsewhere. However, this brute-force approach is exponential in n, so we look for a faster solution, via belief propagation. We associate an evidence kernel $\alpha_i(\mathbf{x}^i)$ with $(\mathbf{g}_i, y_i)$ which is uniform over the possible patterns it permits, and zero elsewhere. The correct distributions are given by:

$$\begin{aligned} P(\mathbf{x}) &= \frac{1}{Z} \prod_{i \in [N]} \alpha_i(\mathbf{x}^i) \\ P(x_j = 1) &= \sum_{\mathbf{x}: x_j = 1} P(\mathbf{x}), \end{aligned}$$

but cannot be computed directly. To run PBP, we start by constructing an instance of PI. Let

$$\begin{aligned} p_i &= \{j : \text{the } j\text{th component of } \mathbf{g}_i \text{ is nonzero}\} \\ \mathcal{R} &= \{p_i : i \in [N]\}. \end{aligned}$$

Clearly, the kernel associated with each p_i is $\alpha_i(\mathbf{x}^i)$. Then, restricting ourselves to only posets with a top-level node for each kernel, we can try applying the factor graph or natural CV graph constructions. It defeats the purpose of belief propagation to construct from the beliefs the global distribution, so we require the various algorithms to return

beliefs on the individual variables (components of $\mathbf{x}$). Each algorithm will have at termination consistent beliefs

$$b_i(\mathbf{x}^i) \approx B_i(\mathbf{x}^i) = \sum_{\mathbf{v}:\mathbf{v}^i=\mathbf{x}^i} P(\mathbf{v})$$

which can be marginalized to obtain the desired beliefs (approximately):

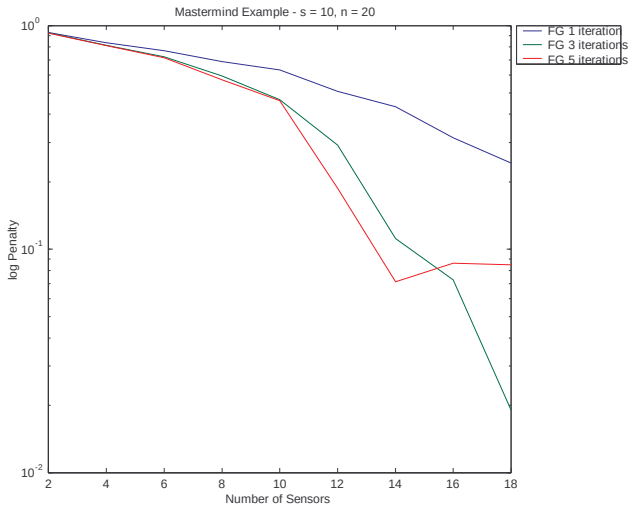
$$\text{assuming } j \in p_i,\ b(x_j = 1) = \sum_{\mathbf{x}^i:x_j=1} b_i(\mathbf{x}^i) \approx P(x_j = 1).$$

Results To evaluate the performance of each algorithm, we define a penalty function that quantifies the extent to which the inference was wrong. We use

$$\begin{aligned} \Delta_i &= -\log_2 b(x_i = x_i) \\ \Delta &= \frac{1}{n}\sum_{i=1}^{n} \Delta_i \end{aligned}$$

so that the penalty associated with a belief that is exactly correct ($b(x_j = x_j) = \Pr\{x_j = x_j\} = 1$) is 0, and the penalty associated with a completely incorrect belief is positive infinity. We average this penalty over the beliefs on each variable. The motivation for this definition is given in the Appendix of [15].

At first, in order to confirm older results, we wished to run PBP on a factor graph, so that the resulting algorithm would be equivalent to ordinary belief propagation. This was the result:

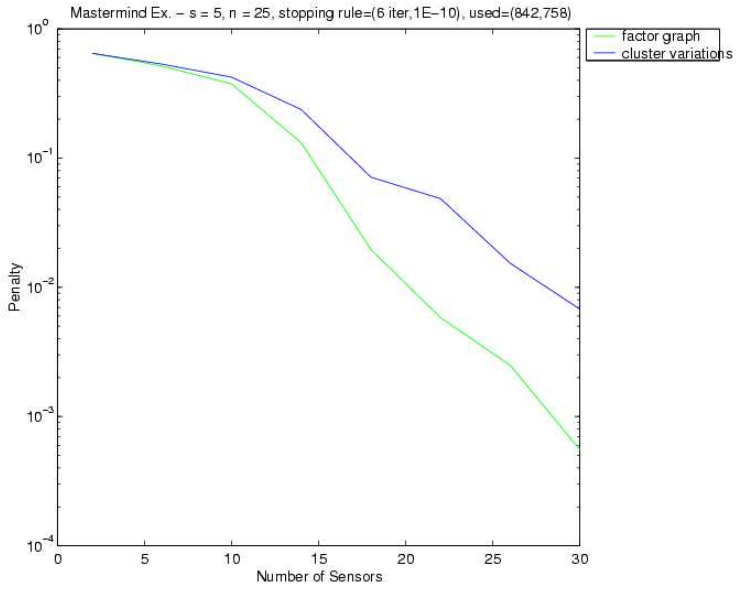


Performance using only Factor Graphs

We plot the average penalty per run against the number of "sensors", or guess vectors $\mathbf{g}_i$, N. For each unique N, many sets $\{\mathbf{g}_i\}$ were chosen, and for each choice a poset was

constructed, and PBP run, resulting in a single penalty value. The point on the plot for a single N represents the average penalty over many different choices $\{\mathbf{g}_i\}$. We notice that the performance seems to improve (penalty decreases more rapidly), with more iterations of the algorithm. Ultimately, the performance observed here was found to be superficially better than in [15], but it is suspected this is due to the belief update scheduling.

Next, it seemed reasonable to compare the FG construction to the natural cluster variations construction, which far outperformed FG in the toy examples. Our preliminary result was this:

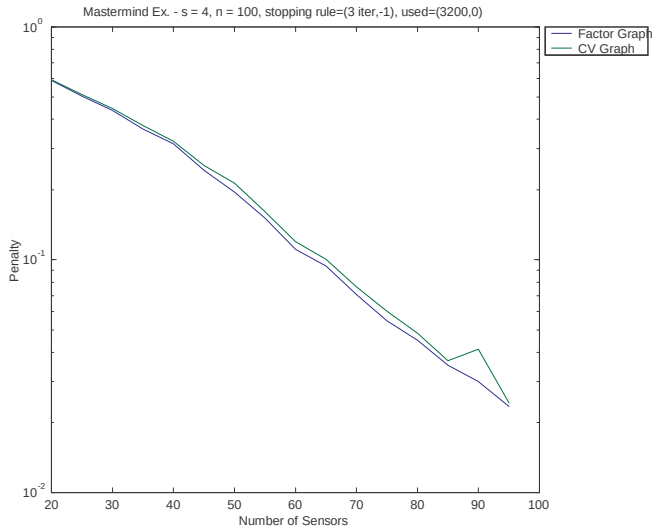


Wrong Natural Cluster Variations Definition #1

Something had obviously gone awry. The NCVG performs BP on an energy surface which better approximates the actual energy surface, its belief updates are based on more complete information, and on smaller examples, it was far more accurate an inference algorithm than OBP. It did not seem reasonable that the NCVG could perform so much worse than the FG in this case, which was not specially contrived in order to prove that generalized belief propagation could be outperformed by OBP. It was discovered that the code we were using to generate the NCVG was logically flawed. The problem was this: Given the set $\mathcal{R}$, the correct natural CV graph is the Hasse diagram for the poset consisting of all the intersections of elements of $\mathcal{R}$ (and their intersections), ordered by inclusion. It turns out this is not equivalent to constructing a graph, one level at a time as follows: start with a top level node for each element of $\mathcal{R}$. Take intersections of any two of these and introduce a second level node for each intersection. Remove the intersection if it is a subset of one already existing on that level, and proceed by

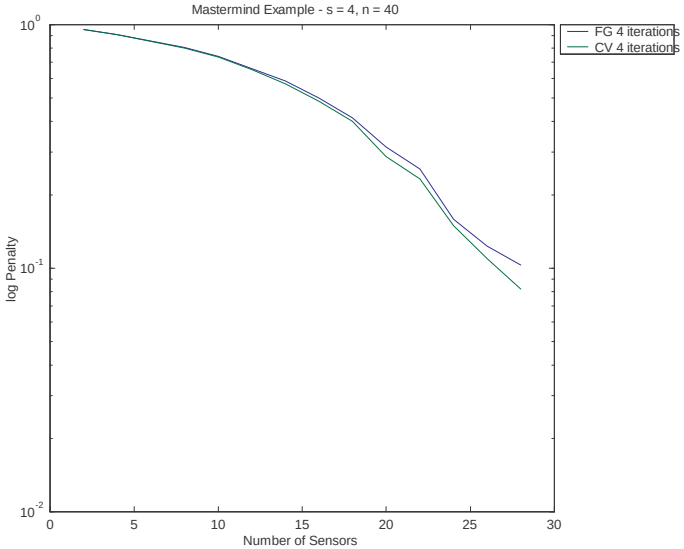
similarly introducing new levels until there are no more intersections. It was not obvious at the time that these two constructions were not equivalent.

Next, we modified the code which generated the NCVG to *not* remove a node on a certain level if it was a subset of another. The result was the following performance:



Wrong Cluster Variations Definition #2

Again, the NCVG construction performed worse than the FG, though much better than before. Finally, it was discovered that neither of these implementations were correct, and so the code was modified to reflect the actual definition:



Factor Graph vs. Natural CV Graph

And so the natural cluster variations graph finally outperformed the factor graph, which was a relief. It is believed that, in almost all cases, generalized belief propaga-

tion should outperform ordinary belief propagation, and this inference problem was no exception. It is interesting to note that the performance of PBP is so very sensitive to such minor changes in the graph structure of the underlying poset.

Chapter 4

Conclusions

One interesting and unexpected result of this work was that smoothing sometimes helps convergence, but it does not change the output, given that the algorithm converges. We verified that GBP (generalized belief propagation, equiv. to running PBP on NCVG) remains probably the most powerful inference tool besides brute-force computation. It outperforms the junction graph algorithm which outperforms ordinary belief propagation, at least on toy examples. So we were able to observe the complexity-performance trade-offs on the BP algorithms already known. However, on one larger example, though OBP is outperformed by GBP, it outperforms several other algorithms deriving from several similar poset constructions. This fact, that altering the definition of the NCVG, even slightly, made it perform even worse than the simplest form of belief propagation, is critical. It suggests that posets chosen at random will not yield good algorithms. This kind of result has been found in related fields as well. It may be interesting to find another class of posets, like the junction graph, or the natural cluster variations graph, which satisfy some balance requirements that could possibly lead to good performance somewhere in between those two. For example, imposing a "2-variable" balance condition may yield interesting results.

However, there remains at least one serious limitation to PBP. Although it is used as an "efficient" solution, real-world applications of belief propagation are usually highly optimized versions of ordinary belief propagation, in which message passing rules take advantage of the structure of the kernel functions themselves. It is unclear at the moment how these kinds of optimizations may be applied to more general BP algorithms. Until such optimizations are discovered, PBP and its various instances can probably not be applied to engineering problems requiring real-time decoding. However, not very many large examples of inference problems have been tried, and it is possible that one exists for which a new kind of belief propagation algorithm may prove truly useful.

Chapter 5

Appendix

5.1 Connection to Statistical Physics

In this section, I explain in slightly more mathematical rigor the connection of PBP to statistical physics. We start by thinking of each variable as a particle, and its taking on a value as its being in a particular one of a finite number of states. The groups $\mathcal{R}$ shall be thought of as "regions" of particles, and the kernels $\{\alpha\}$ as potentials expressing the *a priori* likelihood of finding related particles in certain combinations of states. Recall Z is given by

$$Z = \sum_{x \in A^n} \prod_{R \in \mathcal{R}} \alpha_R(\mathbf{x}_R).$$

The *Helmholtz free energy* of the system is defined as

$$F = -\ln Z.$$

The *energy* of a given configuration (assignment to the variables) $\{x_1, x_2, ..., x_n\}$ is

$$E(x_1, x_2, ..., x_n) = E(\mathbf{x}) = -\sum_{R \in \mathcal{R}} \ln \alpha_R(\mathbf{x}_R).$$

Define $b(\mathbf{x})$ as a trial probability distribution over $\mathbf{x}$. Then the average (expected) energy is:

$$U = \sum_{x \in A^n} b(\mathbf{x}) E(\mathbf{x}).$$

and the *entropy*:

$$H = -\sum_{x \in A^n} b(\mathbf{x}) \ln b(\mathbf{x}).$$

The *variational free energy* of the system is defined as:

$$\widetilde{F}(b) = U - H.$$

It is turns out that

$$\widetilde{F}(b) = D(b||B) + F,$$

so that

$$\widetilde{F}(b) \geq F,$$

and

$$\widetilde{F}(b) = F \text{ iff } b(\mathbf{x}) = B(\mathbf{x}) = \frac{1}{Z} e^{-E(\mathbf{x})}.$$

Physicists call $B(\mathbf{x})$ the *Boltzmann-Gibbs* density Thus,

$$\begin{aligned} F &= \min_{b(\mathbf{x})} \widetilde{F}(b) \\ B(\mathbf{x}) &= \arg\min_{b(\mathbf{x})} \widetilde{F}(b). \end{aligned}$$

The key is this: every time we construct a poset, we are constructing an approximation to the variational free energy of the system. The approximation is made in the entropy term:

$$H_{approx} = \sum_{\rho \in P} \phi(\rho) H(b_\rho),$$

where

$$H(b_\rho) = -\sum_{\mathbf{x}_\rho} b_\rho(\mathbf{x}_\rho) \ln b_\rho(\mathbf{x}_\rho).$$

In general, variable balance conditions and large "regions" help better the approximation $H_{approx} \approx H$. When we run PBP, we attempt to minimize along this approximation to the real variational free energy surface. It was shown in [4] that when the beliefs are consistent, b is at a stationary point on this surface. If the point is a minimum, then the value of $\widetilde{F}(b)$ is approximately F, and we hope that b is approximately B. It is critical to note that, in practice, $b(\mathbf{x})$ is defined only in principle, but its direct computation would defeat the purpose of running the algorithm in the first place. The approximation to the variational free energy can be completely expressed in terms of the much smaller belief distributions at the elements ρ:

$$\widetilde{F}_{approx}(b) = \sum_{\rho \in P} \phi(\rho) \left[\underbrace{-\sum_{\mathbf{x}_\rho} b_\rho(\mathbf{x}_\rho) \ln \prod_{R \in \mathcal{R}(\rho)} \alpha_R(\mathbf{x}_R)}_{} \underbrace{+\sum_{\mathbf{x}_\rho} b_\rho(\mathbf{x}_\rho) \ln b_\rho(\mathbf{x}_\rho)}_{} \right]$$

where the first term is exactly the energy component and the second is the entropy approximation. This is called the Kikuchi approximation. For more details, see [4].

5.2 Complexity of PBP

Here, I examine closely the time complexity of PBP run on a given poset P and establish a rough upper bound. We'll say P has $|P|$ elements. I will assume that the algorithm runs for I iterations, and that every edge is inconsistent during each of the I iterations, so that it is updated. When an edge $e = (\rho, \sigma)$ is updated, the following happens:

1. The parent belief consisting of $|\rho|$ variables is marginalized down to a function over $|\sigma|$ variables. I use $|\cdot|$ as shorthand for $|L(\cdot)|$.

2. The parent belief is divided by the child belief, point-wise.

3. The resulting quotient function, of size $|\sigma|$, is then used to correct the beliefs at related nodes $\{\tau : \sigma \leq \tau,\, \rho \not\leq \tau\}$.

Let's examine the complexity of each step.

1. Marginalizing requires iterating through $A^{|\rho|}$ entries, and performing an addition at each one.

2. Dividing beliefs requires iterating through $A^{|\sigma|}$ entries, and performing a division at each one.

3. Updating the belief at each τ requires iterating through $A^{|\tau|}$ entries, and performing a multiplication at each one.

Thus, the complexity is approximately:

$$\begin{aligned} T(P) &\approx I \cdot \sum_{(\rho,\sigma)\in E} \left(A^{|\rho|} + A^{|\sigma|} + \sum_{\tau:\sigma\leq\tau,\rho\not\leq\tau} A^{|\tau|} \right) \\ &\leq 2I \cdot |P| \cdot A^{|\rho_{\max}|} + I \cdot |P| \cdot A^{|\rho_{\max}|} \cdot S(\rho)_{\max} \\ &= I \cdot |P| \cdot A^{|\rho_{\max}|} \cdot (2 + S(\rho)_{\max}) \\ &= O(I \cdot |P| \cdot A^{|\rho_{\max}|} \cdot S(\rho)_{\max}) \end{aligned}$$

Notes: the 2 introduces the approximation that every node is both a parent and a child, $S(\rho) \triangleq |\{e = (\upsilon, \sigma) : \rho \in \{\tau : \sigma \leq \tau, \upsilon \not\leq \tau\}\}|$, $|\rho_{\max}| = \max_{\rho\in P} |\rho|$ and $S(\rho)_{\max} = \max_{\rho\in P} S(\rho)$.

So to a really rough approximation, we notice that $|\rho_{\max}|$ is the most important factor in determining the complexity, because the complexity grows exponentially with this value. However, the number of nodes $|P|$ linearly affects complexity for a fixed $|\rho_{\max}|$. $S(\rho)_{\max}$, which counts the maximum number of times any one node is affected by remote updates, also linearly increases complexity, and is a function of the Hasse diagram geometry. It should remain relatively small if the degree of all nodes is kept small, and the poset is nowhere very deep.

References

[1] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Trans. Inform. Theory*, vol. 46, no. 2 (March 2000), pp. 325-343.

[2] S. M. Aji and R. J. McEliece, "The generalized distributive law and free energy minimization," Proc. 2001 Allerton Conf. Comm. Control and Computing (Oct. 2001)

[3] F. R. Kschichang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, no.2 (Feb. 2001), pp. 498-519.

[4] R. J. McEliece and M. Yildirim, "Belief Propagation on Partially Ordered Sets," to appear in the IMA Volume "Mathematical Systems Theory in Biology, Communication, Computation, and Finance," D. Gilliam and J. Rosenthal, eds.

[5] P. Pakzad and V. Anantharam, "Belief propagation and statistical physics," Proc. 2002 Conf. Inform. Sciences and Systems, Princeton U. March 2002

[6] P. Pakzad and V. Anantharam, "Minimal graphical representation of Kikuchi regions," preprint.

[7] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*. San Francisco: Morgan-Kaufmann, 1988.

[8] G. R. Shafer and P. P. Shenoy, "Probability propagation," *Ann. Mat. Art. Intell.*, vol. 2 (1990), pp. 327-352.

[9] R. P. Stanley, *Enumerative Combinatorics, vol. I* (Cambridge Studies in Advanced Mathematics 49) Cambridge: Cambridge University Press, 1997.

[10] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Generalized belief propagation," pp. 689-695 in *Advances in Neural Information Processing Systems 13*, (2000) eds. Todd K. Leen, Thomas G. Dietterich, and Volker Tresp.

[11] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Constructing free energy approximations and generalized belief propagation algorithms," submitted to IEEE Trans. Inform. Theory, Sept. 2002

[12] J. Harel, R. J. McEliece, and R. Palanki, "Poset Belief Propagation - Experimental Results", abstract submitted to the *International Symposium on Information Theory*, Jun 2003.

[13] R. J. McEliece, D. MacKay, and J. Cheng. "Turbo decoding as an instance of Pearl's 'belief propagation' algorithm. *IEEE J. on Sel. Areas in Comm.*, 16(2)140-152, 1998.

[14] "The Factasia Glossary",
http://www.rbjones.com/rbjpub/philos/glossary/aprior.htm

[15] Robert McEliece and Jeremy Thorpe. "Data Fusion Algorithms for Collaborative Robotic Exploration." *IPN Progress Reports* Vol. 24-149 Jan 2002. pp. 1-14.