Choosing a Variable to Clamp:
Approximate Inference Using Conditioned Belief Propagation

Frederik Eaton and **Zoubin Ghahramani**
Department of Engineering
University of Cambridge, UK

Abstract

In this paper we propose an algorithm for approximate inference on graphical models based on belief propagation (BP). Our algorithm is an approximate version of Cutset Conditioning, in which a subset of variables is instantiated to make the rest of the graph singly connected. We relax the constraint of single-connectedness, and select variables one at a time for conditioning, running belief propagation after each selection. We consider the problem of determining the best variable to clamp at each level of recursion, and propose a fast heuristic which applies backpropagation to the BP updates. We demonstrate that the heuristic performs better than selecting variables at random, and give experimental results which show that it performs competitively with existing approximate inference algorithms.

1 INTRODUCTION

Belief propagation, also known as the sum-product algorithm, is an algorithm for approximate inference in graphical models. It is related to the Bethe approximation (Bethe, 1935; Yedidia et al., 2000) which was developed in statistical physics. BP was first used as an inference algorithm by Gallager in an application to error correcting codes (Gallager, 1963).

BP is the basis for a number of other algorithms, both exact and approximate. Approximate extensions of BP include Generalized Belief Propagation (Yedidia

et al., 2001) (which can be used for the Cluster Variation Method (Kikuchi, 1951; Pelizzola, 2005)), Expectation Propagation (Minka, 2001), and Loop Corrected Belief Propagation (Mooij et al., 2007).

A popular exact extension of BP is called Cutset Conditioning (Pearl, 1988), which is based on the idea of turning a complex graphical model into a simpler one by conditioning on a set of variables. Other exact extensions of the conditioning idea have been developed (Darwiche, 2001).

Even sophisticated exact inference algorithms are only tractable on reasonably small graphs, and in this article we are interested in *approximate* inference algorithms based on the conditioning idea.

The organization of this article is as follows. First we give a theoretical background for our algorithm, with brief descriptions of BP, conditioning, and backpropagation, and then a description of the algorithm itself. Then we present the results of numerical experiments which compare the performance of our algorithm with other popular algorithms.

2 THEORY

2.1 GRAPHICAL MODELS

Given N random variables $x = \{x_i\}_{i \in \mathcal{V}}$, where $x_i \in \mathcal{X}_i$. For $\alpha \subseteq \mathcal{V}$ where $\alpha = \{i_1, \ldots, i_m\}$, write $x_\alpha = (x_{i_1}, \ldots, x_{i_m})$. We define a distribution over x using a set of functions $\psi_\alpha : \mathcal{X}_\alpha \to \mathbb{R}_+$ for each $\alpha \in \mathcal{F}$ for some $\mathcal{F} \subseteq 2^\mathcal{V}$:

$$P(x_1, \ldots, x_N) = \frac{1}{Z_P} \prod_{i \in \mathcal{V}} \psi_i(x_i) \prod_{\alpha \in \mathcal{F}} \psi_\alpha(x_\alpha) \qquad (1)$$

where

$$Z_P = \sum_x \prod_{i \in \mathcal{V}} \psi_i(x_i) \prod_{\alpha \in \mathcal{F}} \psi_\alpha(x_\alpha) \qquad (2)$$

We write $\alpha \setminus i$ for $\alpha \setminus \{i\}$. Dually, we can consider each variable as the set of factors containing it, so

we will write $i \setminus \alpha$ for $\{\beta \in \mathcal{F} \mid i \in \beta, \beta \neq \alpha\}$, and $|i| = |\{\alpha \in \mathcal{F} \mid i \in \alpha\}|$. Membership $i \in \alpha$ will also be written $i \sim \alpha$ or $\alpha \sim i$. Where there is no danger of confusion, we will drop the index set and write e.g. $\prod_\alpha$ for $\prod_{\alpha \in \mathcal{F}}$ or $\prod_i$ for $\prod_{i \in \mathcal{V}}$.

Associated with $\mathcal{F}$ and $\mathcal{V}$ is a bipartite graph G with vertex set $\mathcal{V} \cup \mathcal{F}$ and edges (i, α) for each $i \in \alpha$, called a *factor graph*.

A probability distribution q will often be defined by normalizing a given measure m:

$$q(x) = \frac{m(x)}{\sum_{x'} m(x')} \qquad (3)$$

When there is no room for confusion, for each such q we will define the *normalization constant* $Z_q \equiv \sum_x m(x)$ and *unnormalized measure* $\tilde{q}(x) \equiv m(x) = Z_q q(x)$.

2.2 BELIEF PROPAGATION

Belief propagation (BP) is a well-known algorithm for approximate inference in graphical models. It is exact on models represented by singly-connected factor graphs, but often gives good results on other models.

BP consists of propagating the following messages along edges of G, until convergence:

$$n_{i\alpha}(x_i) \leftarrow \frac{1}{Z_{n_{i\alpha}}} \psi_i(x_i) \prod_{\beta \sim i \setminus \alpha} m_{\beta i}(x_i) \qquad (4)$$

$$m_{\alpha i}(x_i) \leftarrow \frac{1}{Z_{m_{\alpha i}}} \sum_{x_{\alpha \setminus i}} \psi_\alpha(x_\alpha) \prod_{j \sim \alpha \setminus i} n_{j\alpha}(x_j) \qquad (5)$$

This yields estimates for variable and factor marginals:

$$P_i(x_i) \approx b_i(x_i) = \frac{1}{Z_{b_i}} \psi_i(x_i) \prod_{\alpha \sim i} m_{\alpha i}(x_i) \qquad (6)$$

$$P_\alpha(x_\alpha) \approx b_\alpha(x_\alpha) = \frac{1}{Z_{b_\alpha}} \psi_\alpha(x_\alpha) \prod_{i \sim \alpha} n_{i\alpha}(x_i) \qquad (7)$$

BP also provides an estimate of Z_P using the *Bethe free energy*:

$$
\begin{aligned}
-\log Z_P \approx F_{\text{Bethe}} \equiv & \sum_\alpha \sum_{x_\alpha} b_\alpha(x_\alpha) \log\left(\frac{b_\alpha(x_\alpha)}{\psi_\alpha(x_\alpha)}\right) \\
& + \sum_i \sum_{x_i} b_i(x_i) \log\left(\frac{b_i(x_i)}{\psi_i(x_i)}\right) \\
& - \sum_i \sum_{x_i} |i|\, b_i(x_i) \log\left(b_i(x_i)\right) \quad (8)
\end{aligned}
$$

2.3 CONDITIONING

One technique for improving the performance of BP (and indeed any inference algorithm providing an estimate of Z_P) is based on divide-and-conquer: either apply BP to a model, or write the model as a sum of simpler models, recurse, and combine the resulting approximate marginals using the Z_P estimates. We can express such a decomposition using a "condition" variable c:

$$\tilde{P}(x) = \tilde{P}(x)(I_c + I_{\neg c}) \equiv \tilde{P}(x|c) + \tilde{P}(x|\neg c) \quad (9)$$

$$Z_P = Z_{P(|c)} + Z_{P(|\neg c)} \qquad (10)$$

Dividing equation 9 by equation 10 yields the more familiar

$$P(x) = P(x|c)P(c) + P(x|\neg c)P(\neg c) \qquad (11)$$

where $P(c) \equiv \frac{Z_{P(|c)}}{Z_P}$ and $P(\neg c) \equiv \frac{Z_{P(|\neg c)}}{Z_P}$.

In general we will refer to any combination of BP and conditioning using divide-and-conquer as "conditioned BP" or CBP.

The *cutset conditioning* algorithm is a special case of CBP, conditioning on a set of variables (the *cutset*) whose removal makes G singly connected. Since BP is exact on trees, the cutset conditioning algorithm is also exact. The drawback of cutset conditioning is that its complexity is exponential in the cutset size, so it is only applicable to small or tree-like graphs.

In this paper, we will only consider conditions c of the form $\{x_i = \hat{x}_i\}$ for some variable i and state $\hat{x}_i$. Then $I_c(x) = \delta_{\hat{x}_i}(x_i)$ and $I_{\neg c}(x) = 1 - \delta_{\hat{x}_i}(x_i)$, so the two submodels replace the original factor $\psi_i(x_i)$ with $\delta_{\hat{x}_i}\psi_i(x_i)$ and $(1 - \delta_{\hat{x}_i})\psi_i(x_i)$, respectively. In the first submodel, the variable x_i is "clamped" to the state $\hat{x}_i$, and in the second it is required to take any state *but* $\hat{x}_i$ (there may be more than one). Each sub-model has fewer states than the original, without extending the original factor graph; so we might hope that combining the submodels would yield more accurate approximate marginals, in analogy to Rao-Blackwellization which applies to sampled estimates. Empirically, the combined BP estimates are usually but not always more accurate (section 3).

At each level of recursion, a CBP implementation has to decide which variable to clamp to which value. How do we go about making this decision? One simple method is to choose a random variable and value (perhaps restricting our choice to states with positive BP marginal). As far as we know, this paper is the first to explore other algorithms for choosing condition variables.

Our proposal is based on the following idea: because BP uses local messages, it poorly represents correlations between distant variables. Since CBP must rely on conditioning to capture distant correlations, it should condition on variables whose potentials have

the greatest influence on the rest of the graph. Conditioning on a set $\mathcal{X}_i' \subseteq \mathcal{X}_i$ has the same effect as setting to zero any value of $\psi_i(x_i)$ with $x_i \notin \mathcal{X}_i'$. We propose that the notion of "influence" of a variable i and value x_i can be usefully approximated as the effect of *infinitesimal* variation of $\psi_i(x_i)$ on some function of the BP beliefs, call it $V(b)$. Although we can measure the change in V directly by clamping each variable i in the graph to each possible value x_i and running BP, such a procedure would have time complexity quadratic in the number of graph variables. If we can make do with querying infinitesimal changes in V, then we only need the derivatives $\frac{\mathrm{d}V}{\mathrm{d}\psi_i(x_i)}$. We can compute a full set of such derivatives in linear time complexity using a standard technique called *back-propagation*.

2.4 BACK-PROPAGATION AND BP

In this section, we will apply back-propagation to belief propagation, deriving an iterative algorithm for estimating the gradient of any differentiable objective function of the BP beliefs. We will refer to this algorithm as *back-belief-propagation* or BBP.

The BP updates may not converge when initialized at a given point in the configuration space of factors and initial messages, but if they do, then typically there will be a smooth function between some open ball in that space, and the BP approximate marginals. The derivatives of this function are well-defined, and are what we seek to calculate.

Back-propagation computes the derivatives of a common objective function V with respect to various quantities y, which we will abbreviate (not following any existing convention) as

$$\not{d}y \equiv \frac{\mathrm{d}V}{\mathrm{d}y} \qquad (12)$$

This is called the *adjoint* of y (with respect to V). Given $V(f(y), g(y))$, we can apply the chain rule to compute $\not{d}y = \frac{\partial f}{\partial y}\not{d}f + \frac{\partial g}{\partial y}\not{d}g$. This process can be extended to general function compositions and is called *back-propagation* or *reverse-mode automatic differentiation* (Griewank, 1989).

Given a normalized distribution $q(x) = \frac{1}{Z_q}\tilde{q}(x)$ as in equation 3, we can calculate $\not{d}\tilde{q}$ from $\not{d}q$ and Z_q:

$$\not{d}\tilde{q}(x) = \frac{1}{Z_q}\left(\not{d}q(x) - \sum_{x'} q(x')\not{d}q(x')\right) \qquad (13)$$

(provided that V doesn't depend explicitly on Z_q) We can assume that the BP messages are updated in parallel, and index them with a variable $t \in [0, T]$. Then, given an objective function specified in terms of BP
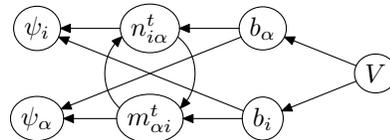

Figure 1: Functional dependencies of BP

beliefs $V(b)$, we can compute the adjoints of the factors ψ_i and ψ_α by following the BP messages backwards through time. The functional dependencies are depicted in figure 1. Using the chain rule, we can derive equations for the back-propagation of BP message and factor adjoints (see appendix[1], section 5.1).

To use these equations, we must save the message normalizers from the BP run, since we need to calculate unnormalized adjoints from the normalized adjoints with equation (13) (e.g. to calculate $\not{d}\tilde{n}_{i\alpha}$ from $\not{d}n_{i\alpha}$ requires $Z_{n_{i\alpha}}$).

We should be able to take $T \to \infty$ and get sensible answers. But the factor adjoints involve a sum of the message adjoints over time, which would diverge if the message adjoints did not converge to zero. In fact, for non-degenerate problems BP converges to attractive fix-points, which means that to a certain extent it is insensitive to the initial values of messages. This means that if we go far enough back in time, the messages have diminishing contribution to the final beliefs, and their adjoints should converge to zero as required. In this way, the BBP algorithm is both sensitive to initial conditions (which specify the objective function), and convergent to a stable fixed point.

The equations can be simplified. Since BP will have converged, we can assume the messages are constant with respect to time, and drop the t superscripts from the BP messages (and their normalizers).

Furthermore, the factor adjoints $\not{d}\psi_i(x_i)$ and $\not{d}\psi_\alpha(x_\alpha)$ are expressed as initial values plus a sum involving message adjoints over time. We can compute such quantities incrementally, by making sure that each time we update a message adjoint, we also update the appropriate factor adjoint.

This yields the following algorithm:

Algorithm (BPP)

Input: The beliefs and messages (and normalizers thereof) of a BP run, also an objective function $V(b)$ defining initial adjoints $\not{d}b_i$ and $\not{d}b_\alpha$

---

[1]If the appendix is not at the end of this paper, an extended paper with appendix can be downloaded from `http://mlg.eng.cam.ac.uk/frederik/aistats2009_choosing.php`

Output: $\not{d}\psi_i(x_i)$, $\not{d}\psi_\alpha(x_\alpha)$

Precompute the following quantities:

$$T_{i\alpha}(x_i) = \prod_{\beta \sim i \backslash \alpha} m_{\beta i}(x_i) \tag{14}$$

$$U_{\alpha i}(x_\alpha) = \prod_{j \sim \alpha \backslash i} n_{j\alpha}(x_j) \tag{15}$$

$$S_{i\alpha j}(x_i, x_j) = \sum_{x_{\alpha \backslash i \backslash j}} \psi_\alpha(x_\alpha) \prod_{k \sim \alpha \backslash i \backslash j} n_{k\alpha}(x_k) \tag{16}$$

$$R_{\alpha i \beta}(x_i) = \psi_i(x_i) \prod_{\gamma \sim i \backslash \alpha \backslash \beta} m_{\gamma i}(x_i) \tag{17}$$

Initialize:

$$\not{d}\psi_i(x_i) \leftarrow \left(\prod_{\alpha \sim i} m_{\alpha i}(x_i) \right) \not{d}\tilde{b}_i(x_i) \tag{18}$$

$$\not{d}\psi_\alpha(x_\alpha) \leftarrow \left(\prod_{i \sim \alpha} n_{i\alpha}(x_i) \right) \not{d}\tilde{b}_\alpha(x_\alpha) \tag{19}$$

$$\not{d}n_{i\alpha}(x_i) \leftarrow \sum_{x_{\alpha \backslash i}} \psi_\alpha(x_\alpha) \prod_{j \sim \alpha \backslash i} n_{j\alpha}(x_j) \not{d}\tilde{b}_\alpha(x_\alpha) \tag{20}$$

$$\not{d}m_{\alpha i}(x_i) \leftarrow \psi_i(x_i) \prod_{\beta \sim i \backslash \alpha} m_{\beta i}(x_i) \not{d}\tilde{b}_i(x_i) \tag{21}$$

Then, apply the following updates in parallel (for every factor α and variable i) until the message adjoints converge to zero:

$$\not{d}\psi_i(x_i) \leftarrow \not{d}\psi_i(x_i) + T_{i\alpha}(x_i)\not{d}\tilde{n}_{i\alpha}(x_i) \tag{22}$$

$$\not{d}n_{i\alpha}(x_i) \leftarrow \sum_{j \sim \alpha \backslash i} \sum_{x_j} S_{i\alpha j}(x_i, x_j)\not{d}\tilde{m}_{\alpha j}(x_j) \tag{23}$$

$$\not{d}\psi_\alpha(x_\alpha) \leftarrow \not{d}\psi_\alpha(x_\alpha) + U_{\alpha i}(x_\alpha)\not{d}\tilde{m}_{\alpha i}(x_i) \tag{24}$$

$$\not{d}m_{\alpha i}(x_i) \leftarrow \sum_{\beta \sim i \backslash \alpha} R_{\alpha i \beta}(x_i)\not{d}\tilde{n}_{i\beta}(x_i) \tag{25}$$

The individual updates, which must be performed for each edge in the factor graph until convergence, are (assuming a bounded state-space for each variable) of complexity quadratic in the largest number of variables in a given factor, and in the largest number of factors containing a given variable.

2.4.1 Sequential Updates

The above parallel algorithm occasionally suffers from numerical stability problems. It is straightforward (but slightly more involved) to derive a sequential algorithm which computes the same quantities, one message at a time, by ensuring that $m_{\alpha i}^{t+1} = m_{\alpha i}^t$ for all but one (α, i) (see appendix, figure 5). The sequential algorithm has the same time complexity as the above, but it allows us to fine-tune the order in which updates are performed. In particular, we can record the order in which BP messages are sent (which may be according to a dynamic schedule as in RBP (Elidan et al., 2006) or its refinements (Sutton & McCallum, 2007)), and send BBP messages in the reverse of this order. Empirically, this method yields slightly better convergence than the parallel algorithm, and is therefore used throughout the experiments (section 3), with BP messages scheduled as in RBP. However, only the results for the "alarm" graph (figure 4) were noticeably improved by the use of sequential BBP updates.

2.5 CBP-BBP

We have not yet addressed the question of which objective function to use with BBP. There are several possibilities, and the following proposal seems to perform well. The performance of some other objective functions is shown in the appendix (figure 7).

We use a brief run of Gibbs sampling to select a random state $x^\star$ of the model. Then we define

$$V^{\mathrm{G},x^\star}(\{b_\alpha\}_\alpha) = \sum_\alpha b_\alpha(x_\alpha^\star) \tag{26}$$

The intuition behind this choice is that we want to find a condition which "pushes" the model's beliefs in a certain direction. If the model's probability mass is concentrated in several modes, then we expect Gibbs sampling to find a sample $x^\star$ from one of them; the objective function $V^{\mathrm{G},x^\star}$ then helps us find a variable which pushes the beliefs in the direction of that mode.

The complete CBP-BBP algorithm becomes

Algorithm (CBP-BBP)

Input: A graphical model, and a maximum number of variables to clamp

Output: A set of approximate beliefs, and an approximate Z

1. Run BP. If the maximum number of variables have been clamped, return the BP beliefs and estimated Z. Otherwise,

2. Run Gibbs sampling to get a state $x^\star$

3. Run BBP with $V = V^{\mathrm{G},x^\star}$

4. Find the pair (i, x_i) with largest $\not{d}\psi_i(x_i)$

5. Clamp variable i to x_i and recurse

6. Clamp variable i to $\mathcal{X}_i \backslash x_i$ and recurse

7. Combine the results from steps 5 and 6 as described in section 2.3

An implementation of this algorithm based on libDAI (Mooij, 2008) can be downloaded with the extended version of the paper.

Random regular graph, 25 variables and 30 factors size 3
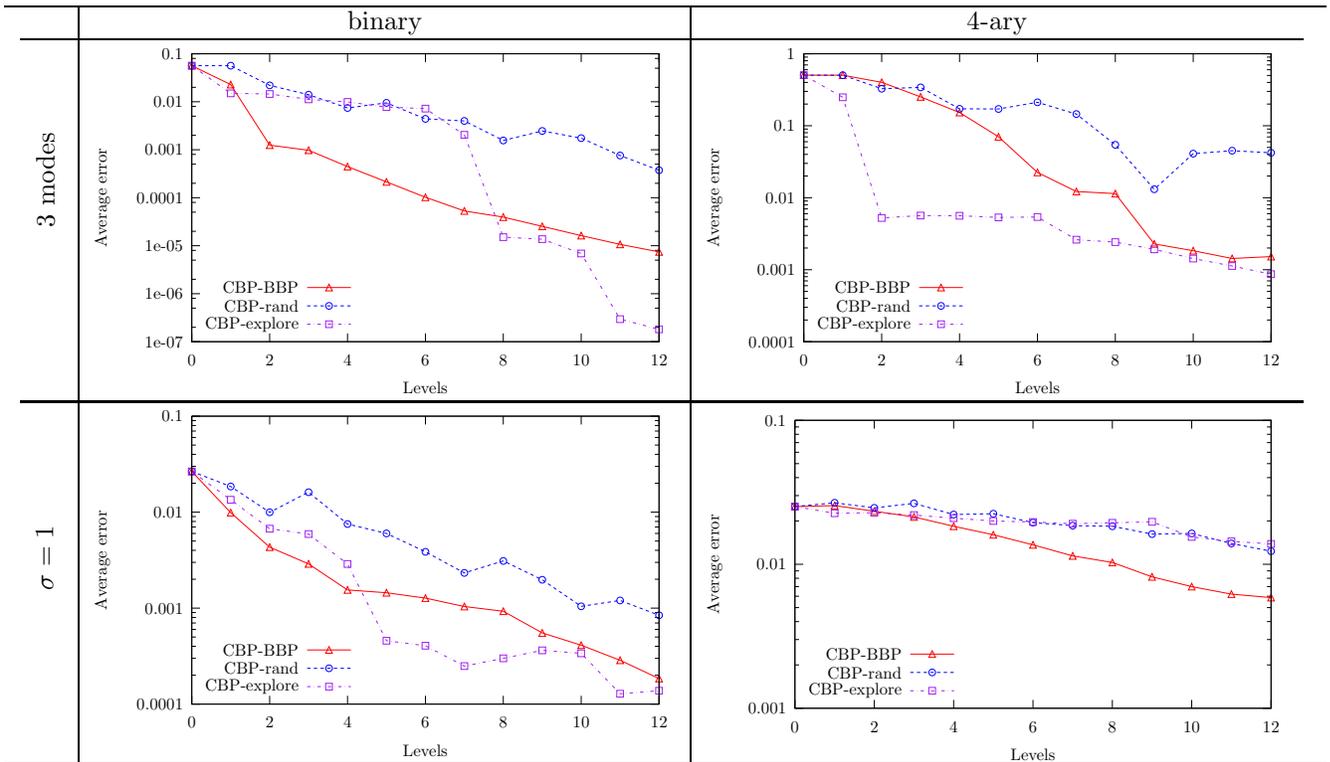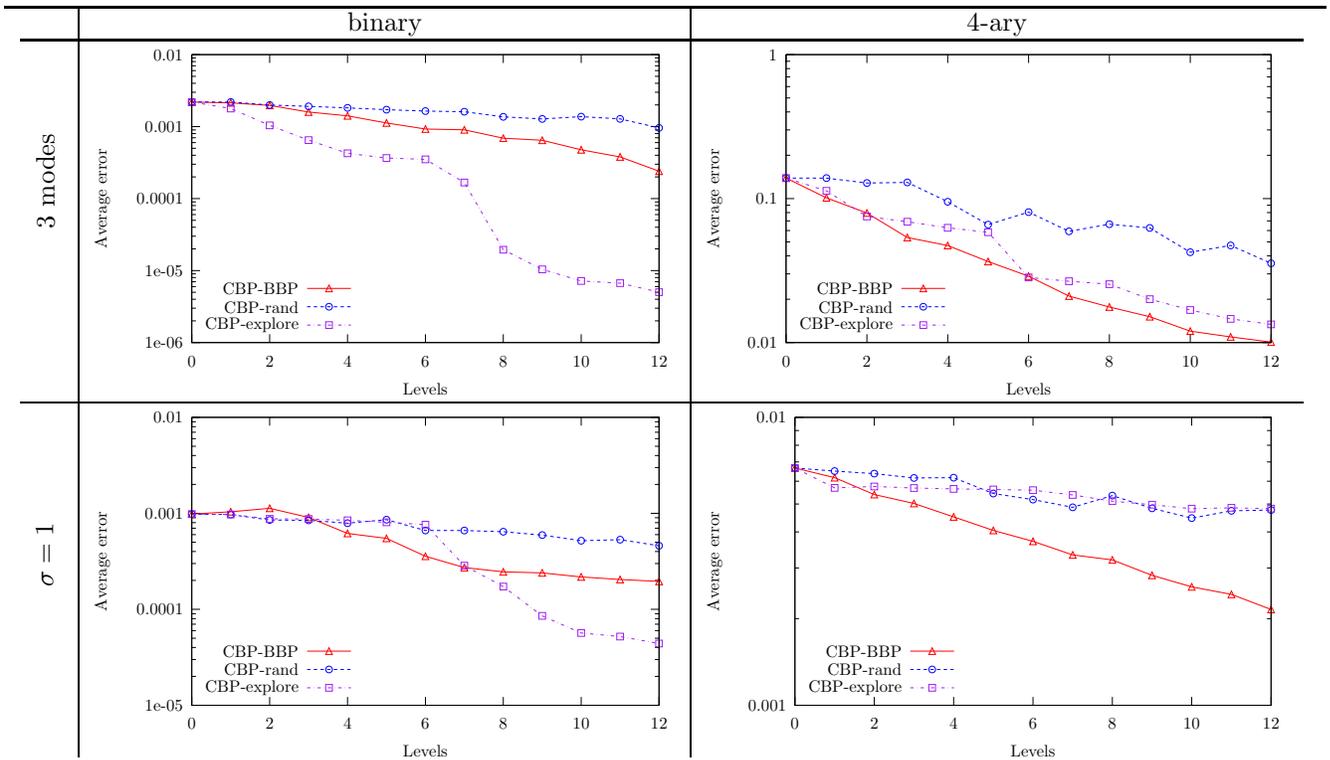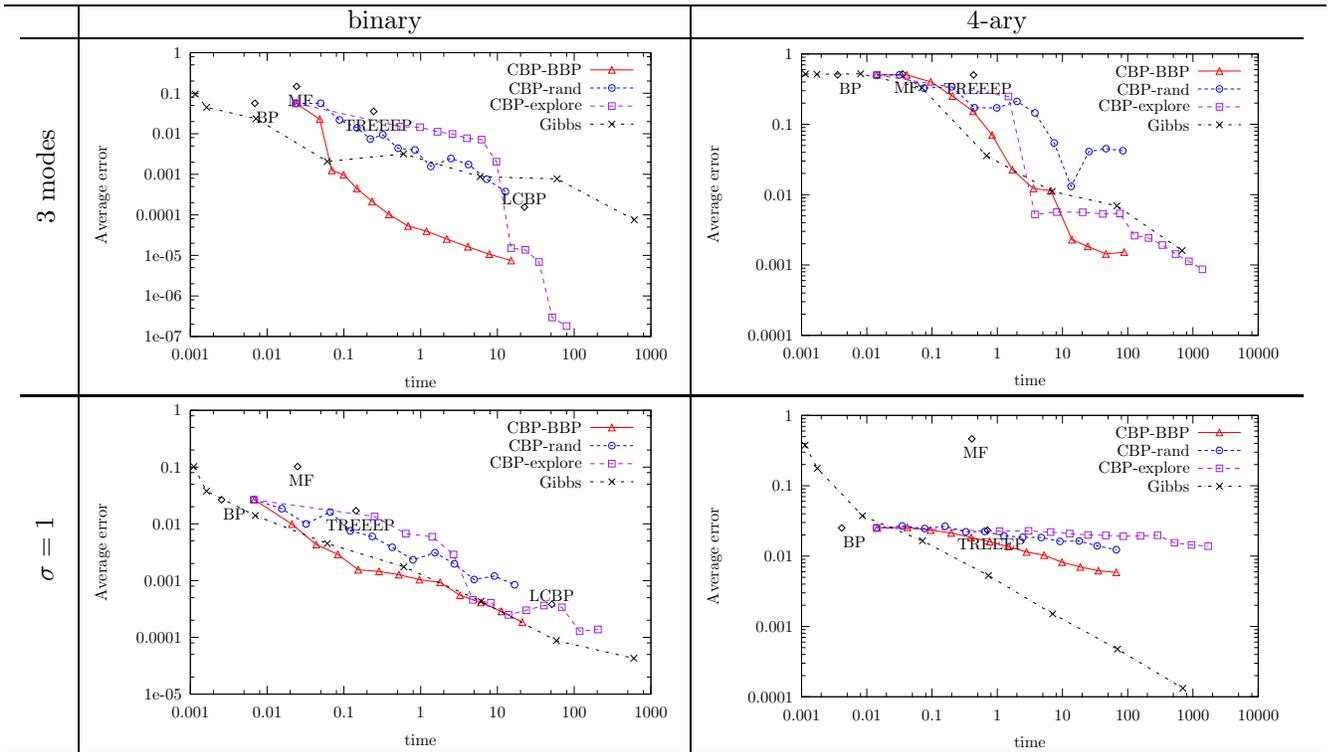

8 by 8 square grid

Figure 2: Comparisons of BBP clamping (CBP-BBP) to random clamping (CBP-rand) and "exploratory" clamping (CBP-explore), on eight example graphs.

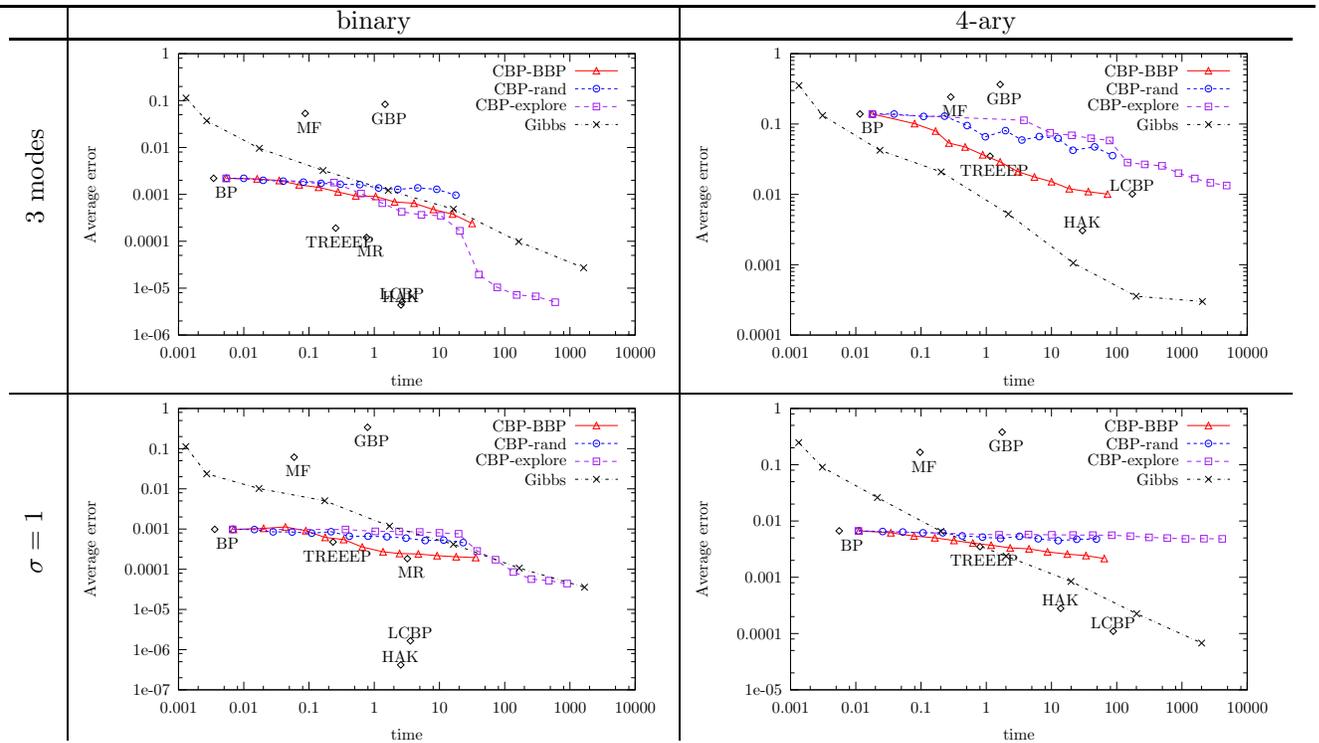**Random regular graph, 25 variables and 30 factors size 3**

8 by 8 square grid

Figure 3: Performance for different versions of CBP and other standard approximate inference algorithms. Some algorithms, such as LCBP and MR, required too much time or memory to run on some graphs (such as those with 4-ary variables and random regular topology).

3 EXPERIMENTS

Our experiments use 8 graphical models, representing 2^3 combinations of topology (square grid, random regular), variable arity (2, 4), and potential initialization (modes, random). The "random" potentials are created by setting each factor entry to $\exp(\sigma W)$ where W is a standard normal deviate and $\sigma = 1$. The "modes" potentials are created by choosing 3 random configurations $x^{\star,1\ldots3}$ for the graph variables, and setting each factor entry $\psi_\alpha(x_\alpha)$ to some constant c if it is consistent with one or more of them (i.e. $x_\alpha = x_\alpha^{\star,k}$ for some k) and to 1 otherwise. We arbitrarily choose c to be 4. This is a way of creating graphs with long-distance correlations. Most of the model's probability mass will be concentrated in the 3 selected "modes".

Our first task is to establish that the CBP-BBP algorithm chooses better conditioning variables than random ("CBP-rand"), for a fixed number of variables ("clamping level"). These results are shown in figure 2. In all the plots in this paper, each CBP-BBP and CBP-rand data point shows the result of averaging error and timing data for 5 runs of the algorithm. The errors are computed as total variation distance[2] between our single-node marginals, and exact marginals (calculated by the junction tree algorithm with HUGIN updates (Jensen et al., 1990)).

In each model, CBP-BBP is usually better than CBP-rand at a given level, but the difference is sometimes small. Also included for comparison is an algorithm "CBP-explore" which (as suggested at the end of 2.3) prospectively clamps each variable to each value, runs BP each time, and chooses the (variable,value) pair which produces marginals that are maximally different (L^1 distance) from the current marginals. This is slower than CBP-rand or CBP-BBP, being quadratic in the size of the graph, but gives an interesting comparison and often produces more accurate results. We are not sure why CBP-BBP is sometimes more accurate than CBP-explore.

Next, we compare the performance of CBP-BBP to other algorithms.

Gibbs - Gibbs sampling, using runs with from 1000 to 1e7 samples

BP - Belief Propagation

MF - Mean Field

TreeEP - algorithm of Minka and Qi (Minka & Qi, 2004)

GBP - Generalized Belief Propagation (Yedidia et al., 2001), using loops of size 4

HAK - algorithm of Heskes, Albers, and Kappen (Heskes et al., 2003), using loops of size 4

LCBP - Loop Corrected Belief Propagation (full cavities) (Mooij et al., 2007)

MR - algorithm of (Montanari & Rizzo, 2005)

The last two algorithms are based on propagating *cavity distributions* and have complexity exponential in cavity size. The random regular graphs of arity > 2 have cavities which are too large, so these algorithms can only be tested on the other graphs. Also, MR requires binary variables and could not be run on the 4-ary graphs.

Figure 3 shows the results of these experiments. Notice that CBP-BBP still typically dominates CBP-rand, not just by clamping level but by runtime as well, even though it is usually somewhat slower due to the overhead of BBP[3]. Gibbs sampling eventually performs better than our algorithm, for long runs. Figure 4 shows per-level comparisons and performance plots for the "alarm" graph, which is part of libDAI. There were convergence problems for BP and BBP on this graph, which may explain the poor results.

Our implementation of both CBP-BBP and CBP-rand include the optimization of not clamping a variable whose BP marginal is already close to 0 or 1. The number of levels of recursion is otherwise fixed. We have experimented with heuristics for controlling recursion depth automatically, for instance recursing until the current Z estimate is smaller than a certain fraction of the top-level Z, but these did not have notably different performance than fixed-level recursion (appendix, figure 6).

4 DISCUSSION

We have presented an approximate inference algorithm which combines BP and variable conditioning. The time complexity of the algorithm is proportional to the cost of a BP run, the square of the maximum variable or factor degree, and is exponential in the number of clamped variables. This "clamping level" can be specified by the user to achieve a desired speed/accuracy trade-off. One advantage of our algorithm is that it can be applied to models with large or densely connected graphs, unlike LCBP or GBP. It seems particularly promising on models with long-range correlations between variables.

---

[2]$\frac{1}{2} \sum_x |P(x) - Q(x)|$

[3]In our experiments, CBP-rand is up to 2.1 times faster than CBP-BBP for a fixed level of clamping. Sometimes, however, due to faster convergence of the BP runs in CBP-BBP, CBP-rand is slower (by up to about 1.5 times).

References

Bethe, H. (1935). Statistical Theory of Superlattices. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, *150*, 552–575.

Darwiche, A. (2001). Recursive conditioning. *Artificial Intelligence*, *126*, 5–41.

Elidan, G., McGraw, I., & Koller, D. (2006). Residual belief propagation: Informed scheduling for asynchronous message passing. *Proceedings of the Twenty-second Conference on Uncertainty in AI (UAI), Boston, Massachussetts* (pp. 6–4).

Gallager, R. (1963). Low Density Parity Check Codes. Number 21 in Research monograph series.

Griewank, A. (1989). On automatic differentiation. *Mathematical Programming: Recent Developments and Applications*, 83–108.

Heskes, T., Albers, K., & Kappen, B. (2003). Approximate inference and constrained optimization. *Uncertainty in Artificial Intelligence* (pp. 313–320).

Jensen, F., Olesen, K., & Andersen, S. (1990). An algebra of Bayesian belief universes for knowledge-based systems. *Networks*, *20*.

Kikuchi, R. (1951). A Theory of Cooperative Phenomena. *Physical Review*, *81*, 988–1003.

Minka, T. (2001). Expectation propagation for approximate Bayesian inference. *Uncertainty in Artificial Intelligence* (pp. 362–369).

Minka, T., & Qi, Y. (2004). Tree-structured approximations by expectation propagation. *Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference* (p. 193).

Montanari, A., & Rizzo, T. (2005). How to compute loop corrections to the Bethe approximation. *Journal of Statistical Mechanics: Theory and Experiment*, *10*, P10011.

Mooij, J. (2008). libDAI 0.2.2: A free/open source C++ library for Discrete Approximate Inference methods. `http://mloss.org/software/view/77/`.

Mooij, J., Wemmenhove, B., Kappen, H., & Rizzo, T. (2007). Loop corrected belief propagation. *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS-07)*.

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

Pelizzola, A. (2005). Cluster variation method in statistical physics and probabilistic graphical models. *J. Phys. A: Math. Gen*, *38*, R309–R339.

Sutton, C., & McCallum, A. (2007). Improved dynamic schedules for belief propagation. *Uncertainty in Artifical Intelligence (UAI)*.

Yedidia, J., Freeman, W., & Weiss, Y. (2000). Bethe free energy, Kikuchi approximations and belief propagation algorithms. *Advances in Neural Information Processing Systems*, *13*.

Yedidia, J., Freeman, W., & Weiss, Y. (2001). Generalized belief propagation. *Advances in neural information processing systems*, 689–695.
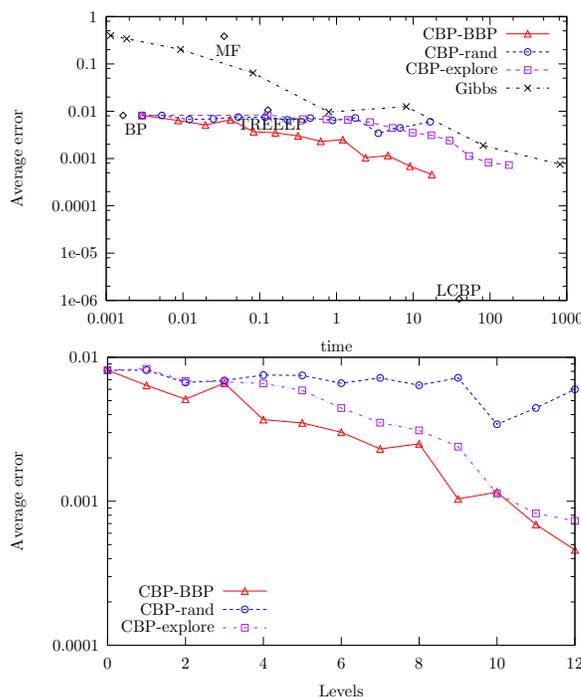
Figure 4: Performance on "alarm" graph; level comparison to CBP-rand on same graph

5 APPENDIX

5.1 BBP DERIVATION

Applying the chain rule to the message representation in section 2.2 yields the following equations:

$$\not{d}\psi_i(x_i) = \left(\prod_{\alpha\sim i} m_{\alpha i}^T(x_i)\right)\not{d}\tilde{b}_i(x_i) + \sum_{t=1}^{T}\sum_{\alpha\sim i}\left(\prod_{\beta\sim i\backslash\alpha} m_{\beta i}^{t-1}(x_i)\right)\not{d}\tilde{n}_{i\alpha}^t(x_i) \tag{27}$$

$$\not{d}\psi_\alpha(x_\alpha) = \left(\sum_{i\sim\alpha} n_{i\alpha}^T(x_i)\right)\not{d}\tilde{b}_\alpha(x_\alpha) + \sum_{t=1}^{T}\sum_{i\sim\alpha}\left(\prod_{j\sim\alpha\backslash i} n_{j\alpha}^{t-1}(x_j)\right)\not{d}\tilde{m}_{\alpha i}^t(x_i) \tag{28}$$

$$\not{d}n_{i\alpha}^t(x_i) = \delta_t^T\sum_{x_{\alpha\backslash i}}\left(\psi_\alpha(x_\alpha)\prod_{j\sim\alpha\backslash i} n_{j\alpha}^T(x_j)\right)\not{d}\tilde{b}_\alpha(x_\alpha) + (1-\delta_t^T)\sum_{j\sim\alpha\backslash i}\sum_{x_j}\left(\sum_{x_{\alpha\backslash i\backslash j}}\psi_\alpha(x_\alpha)\prod_{k\sim\alpha\backslash i\backslash j} n_{k\alpha}^t(x_k)\right)\not{d}\tilde{m}_{\alpha j}^{t+1}(x_j) \tag{29}$$

$$\not{d}m_{\alpha i}^t(x_i) = \delta_t^T\left(\psi_i(x_i)\prod_{\beta\sim i\backslash\alpha} m_{\beta i}^T(x_i)\right)\not{d}\tilde{b}(x_i) + (1-\delta_t^T)\sum_{\beta\sim i\backslash\alpha}\left(\psi_i(x_i)\prod_{\gamma\sim i\backslash\alpha\backslash\beta} m_{\gamma i}^t(x_i)\right)\not{d}\tilde{n}_{i\beta}^{t+1}(x_i) \tag{30}$$

To derive a sequential update rule, we use a different message representation, which sends only a single message $E^t = (\alpha, i)$ at time t. This representation also includes a damping factor λ, which should be 1 for no damping.

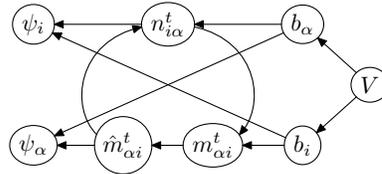$$\tilde{b}_\alpha(x_\alpha) = \psi_\alpha(x_\alpha)\prod_{i\sim\alpha} n_{i\alpha}(x_i) \tag{31}$$

$$\tilde{b}_i(x_i) = \psi_i(x_i)\prod_{\alpha\sim i} m_{\alpha i}(x_i) \tag{32}$$

$$\tilde{n}_{i\alpha}^t(x_i) = \psi_i(x_i)\prod_{\beta\sim i\backslash\alpha} m_{\beta i}^t(x_i) \tag{33}$$

$$\tilde{\hat{m}}_{\alpha i}^t(x_i) = \sum_{x_{\alpha\backslash i}}\psi_\alpha(x_\alpha)\prod_{j\sim\alpha\backslash i} n_{j\alpha}^t(x_j) \tag{34}$$

$$m_{\alpha i}^{t+1}(x_i) = (\hat{m}_{\alpha i}^t(x_i))^{\lambda\delta_{E^t}^{(\alpha,i)}}(m_{\alpha i}^t(x_i))^{1-\lambda\delta_{E^t}^{(\alpha,i)}} \tag{35}$$

Here is a diagram of the dependencies in the new message equations:

Computing adjoints and eliminating t superscripts for messages (having converged), and noting that at convergence $\hat{m}_{\alpha i}(x_i) = m_{\alpha i}(x_i)$, and using pre-computed quantities T, U, S, and R (equations 14, 15, 16, 17) yields:

$$\not{d}\psi_i(x_i) = \not{d}\tilde{b}_i(x_i)\prod_{\alpha\sim i} m_{\alpha i}(x_i) + \sum_{t=0}^{T}\sum_{\alpha\sim i}\not{d}\tilde{n}_{i\alpha}^t(x_i)T_{i\alpha}(x_i) \tag{36}$$

$$\not{d}\psi_\alpha(x_\alpha) = \not{d}\tilde{b}_\alpha(x_\alpha)\prod_{i\sim\alpha} n_{i\alpha}(x_i) + \sum_{t=0}^{T}\sum_{i\sim\alpha}\not{d}\tilde{\hat{m}}_{\alpha i}^t(x_i)U_{\alpha i}(x_\alpha) \tag{37}$$

$$\not{d}n_{i\alpha}^t(x_i) = \delta_t^T\sum_{x_{\alpha\backslash i}}\not{d}\tilde{b}_\alpha(x_\alpha)\psi_\alpha(x_\alpha)\prod_{j\sim\alpha\backslash i} n_{j\alpha}(x_j) + \sum_{j\sim\alpha\backslash i}\sum_{x_j}\not{d}\tilde{\hat{m}}_{\alpha j}^t(x_j)S_{i\alpha j}(x_i, x_j) \tag{38}$$

$$\not{d}m_{\alpha i}^t(x_i) = \delta_t^T\not{d}\tilde{b}_i(x_i)\psi_i(x_i)\prod_{\beta\sim i\backslash\alpha} m_{\beta i}^t(x_i) + (1-\lambda\delta_{E^t}^{(\alpha,i)})\not{d}m_{\alpha i}^{t+1}(x_i) + \sum_{\beta\sim i\backslash\alpha}\not{d}\tilde{n}_{i\beta}^t(x_i)R_{\alpha i\beta}(x_i) \tag{39}$$

$$\not{d}\hat{m}_{\alpha i}^t(x_i) = \lambda\delta_{E^t}^{(\alpha,i)}\not{d}m_{\alpha i}^{t+1}(x_i) \tag{40}$$

Which updates must be performed when $E^t = (\alpha, i)$? Note that $E^t = (\alpha, i) \iff d\hat{m}^t_{\alpha i}(x_i) \neq 0$. In that case, $dn^t_{j\alpha}(x_j) > 0$ for $j \sim \alpha \setminus i$. Then $dm^t_{\beta j}(x_j)$ incremented for $j \sim \alpha \setminus i$ and $\beta \sim j \setminus \alpha$.

By eliminating $dn_{i\alpha}$, which is mostly zero after initialization, we only need keep quantities $dm_{\alpha i}(x_i)$, $d\psi_i(x_i)$, $d\psi_\alpha(x_\alpha)$ between messages. The final algorithm is shown in figure 5.

Definitions:

$$T_{i\alpha}(x_i) = \prod_{\beta \sim i \setminus \alpha} m_{\beta i}(x_i) \tag{41}$$

$$U_{\alpha i}(x_\alpha) = \prod_{j \sim \alpha \setminus i} n_{j\alpha}(x_j) \tag{42}$$

$$S_{i\alpha j}(x_i, x_j) = \sum_{x_{\alpha \setminus i \setminus j}} \psi_\alpha(x_\alpha) \prod_{k \sim \alpha \setminus i \setminus j} n_{k\alpha}(x_k) \tag{43}$$

$$R_{\alpha i\beta}(x_i) = \psi_i(x_i) \prod_{\gamma \sim i \setminus \alpha \setminus \beta} m_{\gamma i}(x_i) \tag{44}$$

Routines:
Send-$n_{i\alpha}$ $(f(x_i)) =$
  $\tilde{f}(x_i) = \frac{1}{Z_{n_{i\alpha}}} \left(f(x_i) - \sum_{x'_i} n_{i\alpha}(x'_i) f(x'_i) \right)$
  $d\psi_i(x_i) \leftarrow d\psi_i(x_i) + \tilde{f}(x_i) T_{i\alpha}(x_i)$
  For each $\beta \sim i \setminus \alpha$ do:
    $dm_{\beta i}(x_i) \leftarrow dm_{\beta i}(x_i) + \tilde{f}(x_i) R_{\beta i\alpha}(x_i)$
Send-$m_{\alpha i} =$
  $d\psi_\alpha(x_\alpha) \leftarrow d\psi_\alpha(x_\alpha) + \lambda d\tilde{m}_{\alpha i}(x_i) U_{\alpha i}(x_\alpha)$
  For each $j \sim \alpha \setminus i$ do:
    Send-$n_{j\alpha} \left(\lambda \sum_{x_i} d\tilde{m}_{\alpha i}(x_i) S_{j\alpha i}(x_j, x_i) \right)$
  $dm_{\alpha i}(x_i) \leftarrow (1 - \lambda) dm_{\alpha i}(x_i)$
Initialization:

$$d\psi_i(x_i) \leftarrow d\tilde{b}_i(x_i) \prod_{\alpha \sim i} m_{\alpha i}(x_i) \tag{45}$$

$$d\psi_\alpha(x_\alpha) \leftarrow d\tilde{b}_\alpha(x_\alpha) \prod_{i \sim \alpha} n_{i\alpha}(x_i) \tag{46}$$

$$dm_{\alpha i}(x_i) \leftarrow d\tilde{b}_i(x_i) \psi_i(x_i) \prod_{\beta \sim i \setminus \alpha} m_{\beta i}(x_i) \tag{47}$$

For each i, α do:
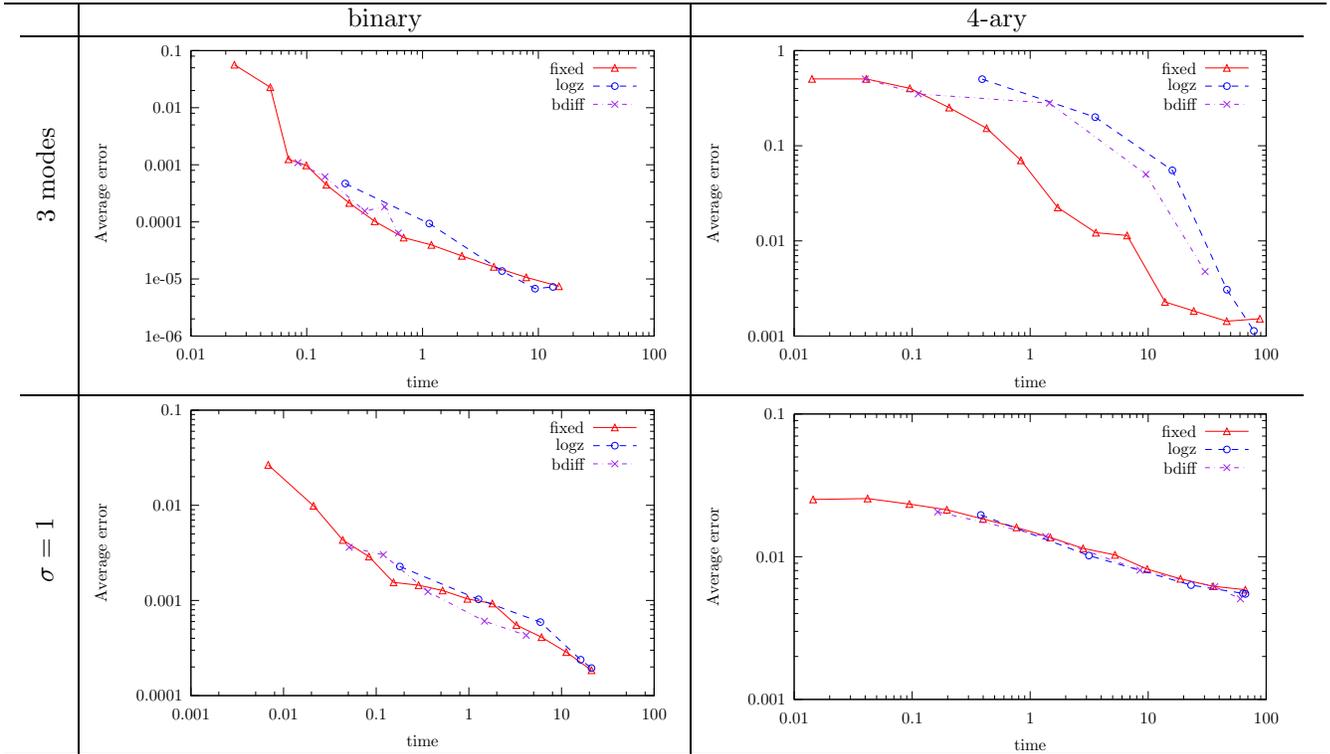 Call Send-$n_{i\alpha} \left(\sum_{x_{\alpha \setminus i}} d\tilde{b}_\alpha(x_\alpha) \psi_\alpha(x_\alpha) \prod_{j \sim \alpha \setminus i} n_{j\alpha}(x_j) \right)$
Main loop:
Call Send-$m_{\alpha i}$ for each message (α, i) sent by BP

Figure 5: A sequential BBP algorithm

Random regular graph, 25 variables and 30 factors size 3
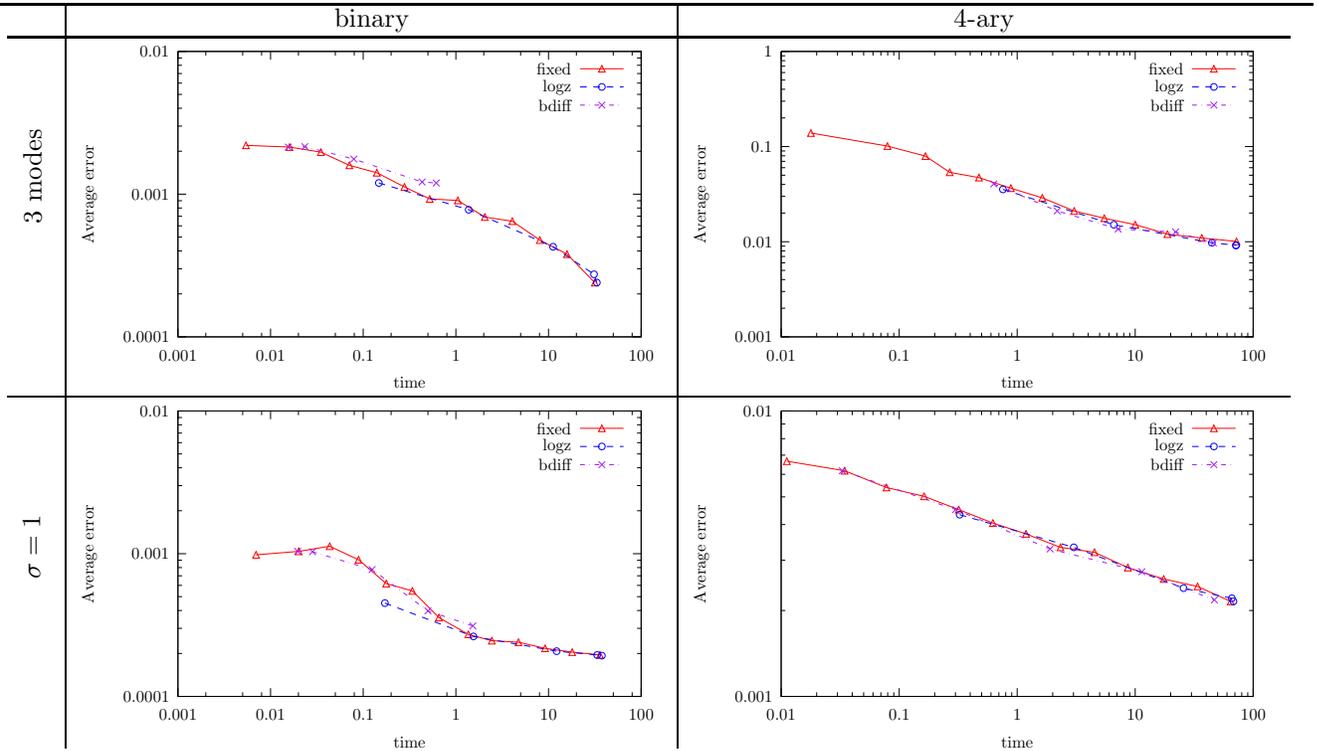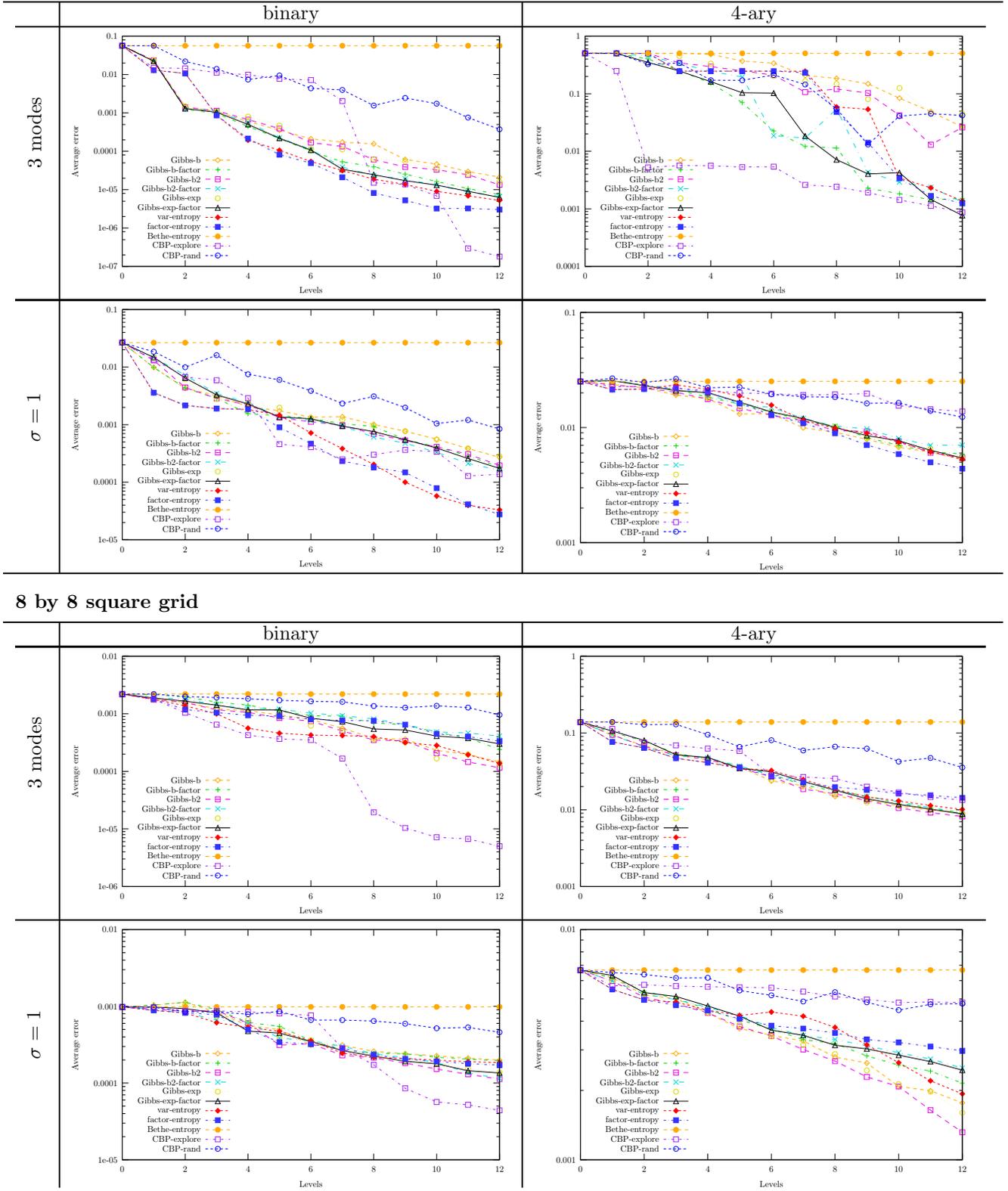

8 by 8 square grid

Figure 6: Performance comparison of variable-level to fixed-level clamping. "Logz" recurses until fraction of current Z to original Z is below specified tolerance. "Bdiff" is the same, but fraction is scaled by difference between new and original beliefs. The tolerance runs from 1e-1 to 1e-5.

Random regular graph, 25 variables and 30 factors size 3
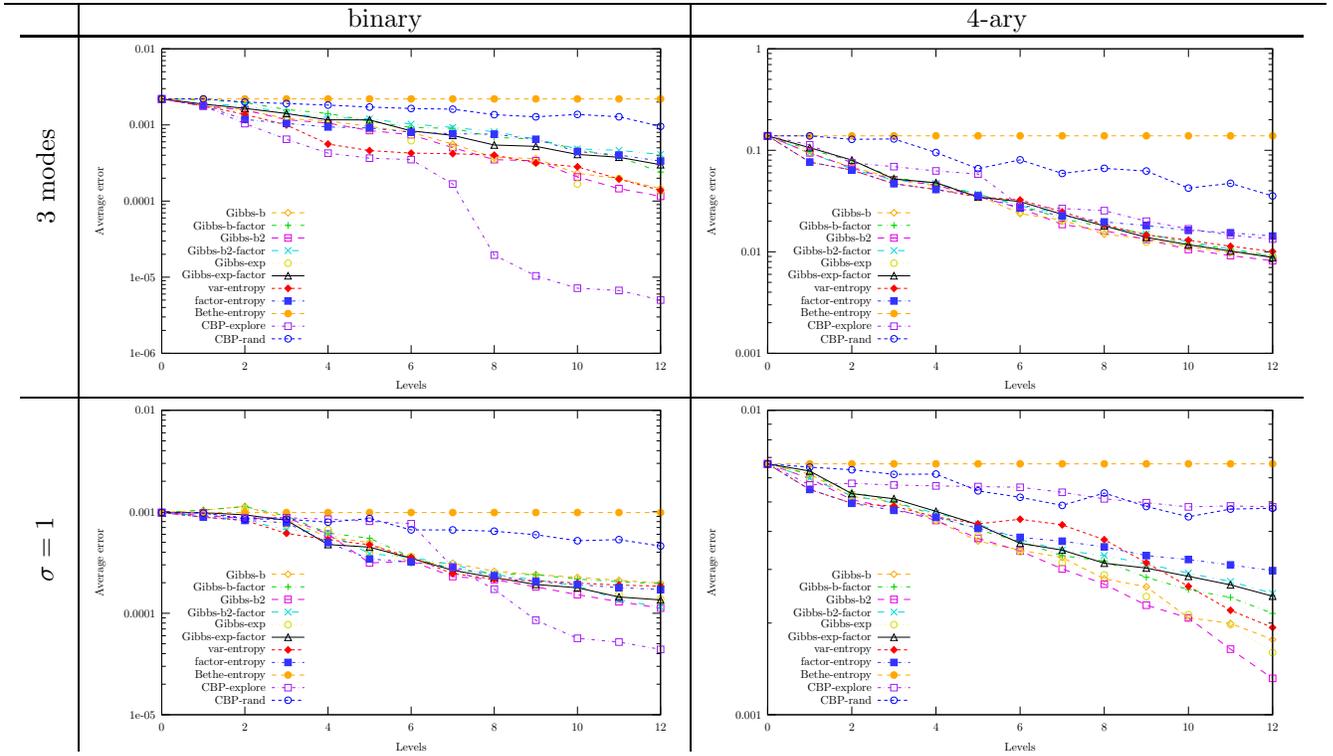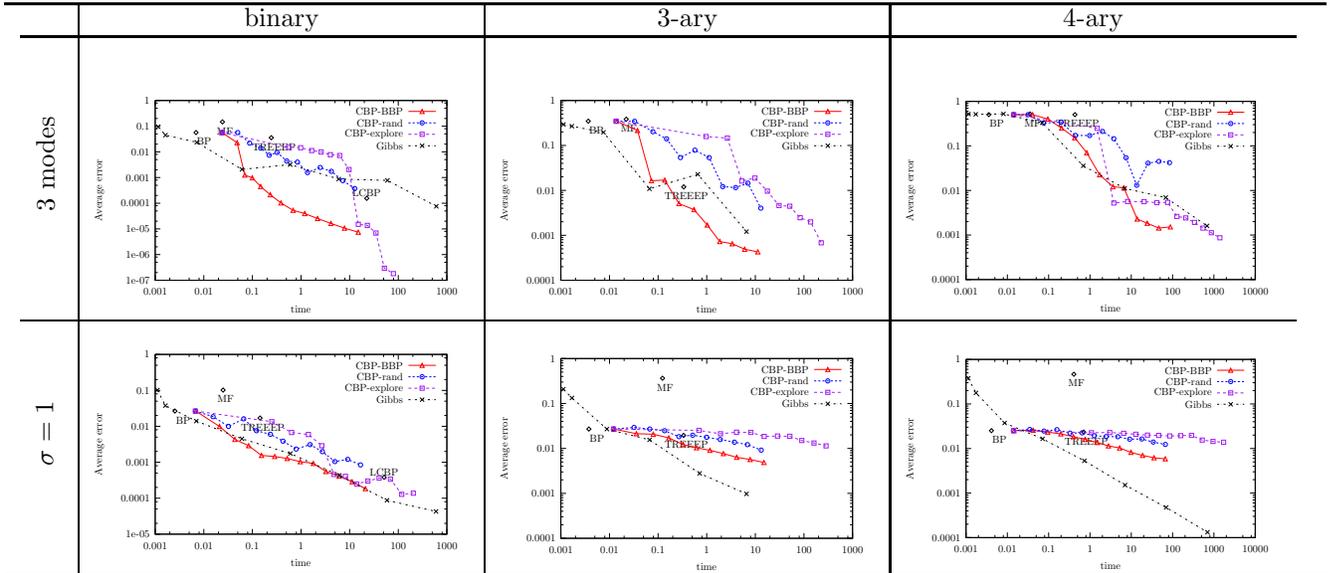

8 by 8 square grid

Figure 7: Comparison of different cost functions by clamping level: Gibbs-b (CBP-BBP): $V = \sum_i b_i(x_i^\star)$. Gibbs-b2: $V = \sum_i b_i(x_i^\star)^2$. Gibbs-exp: $V = \sum_i \exp(b_i(x_i^\star))$. Gibbs-b-factor: $V = \sum_\alpha b_\alpha(x_\alpha^\star)$ (similarly for Gibbs-b2-factor, Gibbs-exp-factor). var-entropy: $V = \sum_i H(b_i)$. factor-entropy: $V = \sum_\alpha H(b_\alpha)$. Bethe-entropy: $V = F_{\text{Bethe}}$.

Random regular graph, 25 variables and 30 factors size 3

	binary	3-ary	4-ary
3 modes			
$\sigma = 1$			

8 by 8 square grid

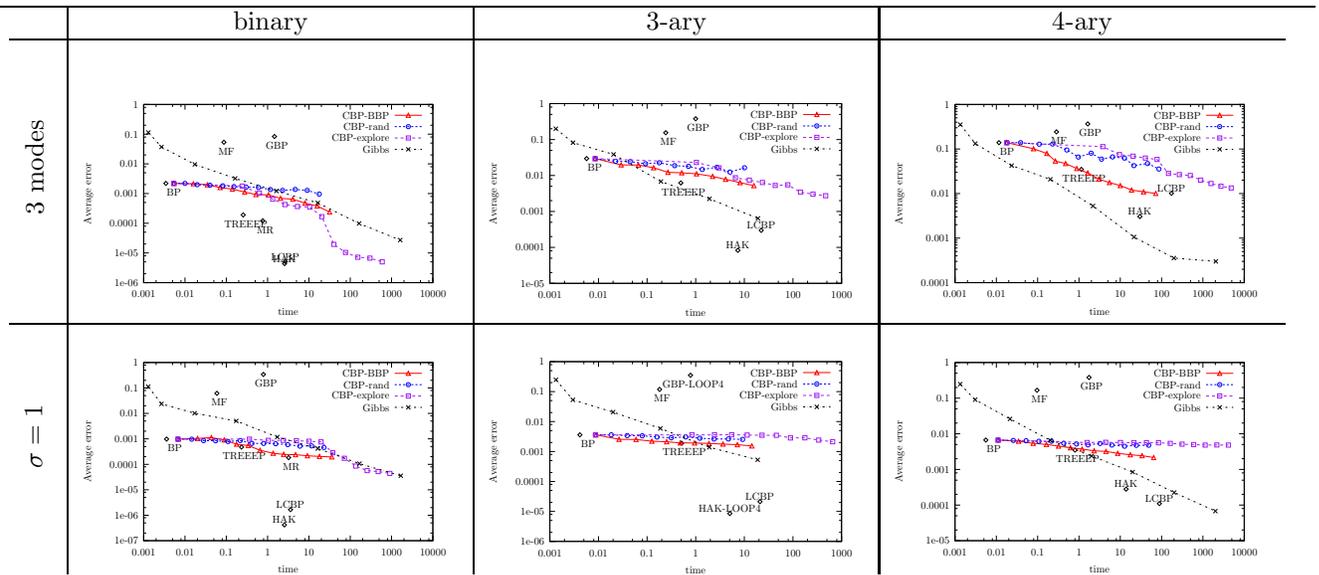|	binary	3-ary	4-ary
3 modes			
$\sigma = 1$			

Figure 8: Comparison of different variable arities