Lifted Belief Propagation: Pairwise Marginals and Beyond

Babak Ahmadi and Kristian Kersting and Fabian Hadiji
Knowledge Dicovery Department, Fraunhofer IAIS
53754 Sankt Augustin, Germany
firstname.lastname@iais.fraunhofer.de

Abstract

Lifted belief propagation (LBP) can be extremely fast at computing approximate marginal probability distributions over single variables and neighboring ones in the underlying graphical model. It does, however, not prescribe a way to compute joint distributions over pairs, triples or k-tuples of distant random variables. In this paper, we present an algorithm, called *conditioned* LBP, for approximating these distributions. Essentially, we select variables one at a time for conditioning, running lifted belief propagation after each selection. This naive solution, however, recomputes the lifted network in each step from scratch, therefore often canceling the benefits of lifted inference. We show how to avoid this by efficiently computing the lifted network for each conditioning directly from the one already known for the single node marginals. Our experimental results validate that significant efficiency gains are possible and illustrate the potential for second-order parameter estimation of Markov logic networks.

1 Introduction

There has been much recent interest in methods for performing lifted probabilistic inference, handling whole sets of indistinguishable objects together, see e.g. (Milch et al., 2008; Sen et al., 2009) and references in there. Most of these lifted inference approaches are extremely complex, so far do not easily scale to realistic domains and hence have only been applied to rather small artificial problems. A remarkable exception are lifted versions of belief propagation (Singla and Domingos, 2008; Kersting et al., 2009). They grouped together random variables that have identical computation trees but now run a modified belief propagation (BP) on the resulting lifted, i.e., clustered network. Being instances of BP, they can be extremely fast at computing approximate marginal probability distributions over single variable nodes and neighboring ones in the underlying graphical model. Above all, they naturally scale to realistic domain sizes. Despite their success, however, lifted BP approaches do not provide a prescription to compute joint probabilities over pairs of non-neighboring variables in the graph. When the underlying graphical model is a tree, there is a single chain connecting any two nodes, and dynamic programming techniques might be developed for efficiently integrating out the internal variables. When cycles exist, however, it is not clear what approximate procedure is appropriate. The situation is even more frustrating when computing marginals over triples or k-tuples of distant nodes. As for the non-lifted case, sophisticated exact lifted inference algorithms are only tractable on rather small models and do not scale to realistic domain sizes. It is precisely this problem that we are addressing in this paper, that is we are interested in approximate lifted inference algorithms based on the conditioning idea that scale to realistic domain sizes. Specifically, we present *conditioned* LBP (CLBP), a scalable lifted inference algorithm for approximate inference based on conditioning. Essentially, we select variables one at a time for conditioning and run lifted belief propagation after each selection. This naive solution, however, recomputes the lifted network in each step from scratch, therefore often

canceling the benefits of lifted inference. We show how to avoid this by efficiently computing the lifted network for each conditioning directly from the one already known for the single node marginals. There has been some prior work for related problems. Delcher *et al.* (1996) propose a data structure that allows efficient queries when new evidence is incorporated in singly connected Bayesian networks and Acar *et al.* (2008) present an algorithm to adapt the model to structural changes using an extension of Rake-and-Compress Trees. The only lifted inference approach we are aware of is the work by Nath and Domingos (2010) that was independently developed in parallel. The authors essentially simulate their lifting procedure for a set of changed variables, obtaining the adapted lifted network.

We also consider the problem of determining the best variable to condition on in each iteration to stay maximally lifted over all iterations and propose a simple heuristic. Our experimental evaluation including experiments on second-order parameter estimation for Markov logic networks (Richardson and Domingos, 2006) shows that significant efficiency gains are obtainable compared to naively running (lifted) BP in each iteration. CLBP may also have future applications in more advanced relational learning tasks such as active learning.

We proceed as follows. We start off by briefly reviewing LBP. Then, we introduce CLBP, prove its soundness, and touch upon the problem of determining the best variable to condition on at each level of recursion. Before concluding, we present the results of our experimental evaluation.

2 Lifted Belief Propagation

Let $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ be a set of n discrete-valued random variables each having d states, and let x_i represent the possible realizations of random variable X_i. Graphical models compactly represent a joint distribution over $\mathbf{X}$ as a product of factors (Pearl, 1991), i.e.,

$$P(\mathbf{X} = \mathbf{x}) = Z^{-1} \prod_k f_k(\mathbf{x}_k) \ .$$

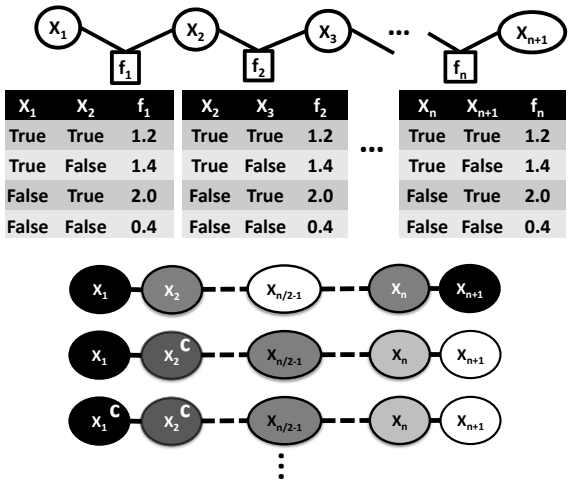


Figure 1: (**Top**) An example for a factor graph — a chain graph model with $n+1$ nodes — with associated potentials. Circles denote variables, squares denote factors. (**Bottom**) Supernodes, indicated by the shades of the original nodes, produced by repeatedly clamping nodes, indicated by "c",on a chain graph model with $n+1$ nodes. Factors have been omitted. The conditioning order is $\pi = \{2, 1, 3, 4, \ldots, n-2, n-1, n+1, n\}$. After clamping X_2 all subsequent LBP runs work on the fully grounded network.

Each factor f_k is a non-negative function of a subset of the variables $\mathbf{x}_k$, and Z is a normalization constant. If $P(\mathbf{X} = \mathbf{x}) > 0$ for all joint configurations $\mathbf{x}$, the distribution can be equivalently represented as a log-linear model: $P(\mathbf{X} = \mathbf{x}) = Z^{-1} \exp\left[\sum_i w_i \cdot g_i(\mathbf{x})\right]$, where the features $g_i(x)$ are arbitrary functions of (a subset of) the configuration $\mathbf{x}$. Each graphical model can be represented as a factor graph. A factor graph, cf. Fig 1 **(top)**, is a bipartite graph that expresses the factorization structure of the joint distribution. It has a variable node (denoted as a circle) for each variable X_i, a factor node (denoted as a square) for each f_k, with an edge connecting variable node i to factor node k if and only if X_i is an argument of f_k. We assume one factor $f_i(\mathbf{x}) = \exp\left[w_i \cdot g_i(\mathbf{x})\right]$ per feature $g_i(\mathbf{x})$.

An important (#P-complete) inference task is to compute the conditional probability of variables given the values of some others, the evidence, by summing out the remaining variables.

The belief propagation (BP) algorithm is an efficient way to solve this problem that is exact when the factor graph is a tree, but only approximate when the factor graph has cycles. Although this loopy BP has no guarantees of convergence or of giving the correct result, in practice it often does, and can be much more efficient than other methods. BP can be elegantly described in terms of sending messages within a factor graph. The message from a variable X to a factor f is

$$\mu_{X\to f}(x) = \prod_{h\in \mathrm{nb}(X)\setminus\{f\}} \mu_{h\to X}(x)$$

where $\mathrm{nb}(X)$ is the set of factors X appears in. The message from a factor to a variable is

$$\mu_{f\to X}(x) = \sum_{\neg\{X\}} \left(f(\mathbf{x}) \prod_{Y\in \mathrm{nb}(f)\setminus\{X\}} \mu_{Y\to f}(y) \right)$$

where $\mathrm{nb}(f)$ are the arguments of f, and the sum is over all of these except X, denoted as $\neg\{X\}$. The messages are usually initialized to 1, and the unnormalized belief of each variable X_i can be computed from the equation

$$b_i(x_i) = \prod_{f\in \mathrm{nb}(X_i)} \mu_{f\to X_i}(x_i) \ .$$

Evidence is incorporated by setting $f(\mathbf{x}) = 0$ for states $\mathbf{x}$ that are incompatible with it. Different schedules may be used for message-passing.

Although already quite efficient, many graphical models produce factor graphs with a lot of symmetries not reflected in the graphical structure. Consider the factor graph in Fig. 1(**top**). The associated potentials are identical. Lifted BP (LBP) can make use of this fact. It essentially performs two steps: Given a factor graph G, it first computes a compressed factor graph $\mathfrak{G}$ and then runs a modified BP on $\mathfrak{G}$. We use fraktur letters such as $\mathfrak{G}$, $\mathfrak{X}$, and $\mathfrak{f}$ to denote the lifted, i.e., compressed graphs, nodes, and factors. For the present paper, only the first step is important, which we will now briefly review.

Step 1 of LBP — Lifting by Color-Passing (CP): Let G be a given factor graph with variable and factor nodes. Initially, all variable nodes fall into $d+1$ groups (one or more of these may be empty) — known states $s_1, \ldots, s_d$, and *unknown* — represented by colors. All factor nodes with the same associated potentials also fall into one group represented by a shade. Now, each variable node sends a message to its neighboring factor nodes saying "I am of color C". A factor node sorts the incoming colors into a vector according to the order the variables appear in its arguments. The last entry of the vector is the factor node's own color. This color signature is sent back to the neighboring variables nodes, essentially saying "You have communicated with these kinds of nodes". The variable nodes stack the incoming signatures together and, hence, form unique signatures of their one-step message history. Variable nodes with the same stacked signatures are grouped together, and a new color is assigned to each group. The factors are grouped in a similar fashion based on the incoming color signatures of neighboring nodes. This CP process is iterated until no new colors are created anymore. As the effect of the evidence propagates through the factor graph, more groups are created. The final lifted graph $\mathfrak{G}$ is constructed by grouping nodes (factors) with the same color (signatures) into *supernodes* (*superfactors*). Supernodes (superfactors) are sets of nodes (factors) that send and receive the same messages at each step of carrying out BP on G and form a partition of the nodes in G. On this lifted network, LBP runs an efficient modified BP (MBP). We refer to (Singla and Domingos, 2008; Kersting et al., 2009) for details.

3 Lifted Conditioning

We are often faced with the problem of repeatedly answering slightly modified queries on the same network. Consider e.g. computing a joint distribution $P(X_1, X_2, \ldots, X_k)$ using LBP. A simple method is the following *conditioning* procedure that we call *conditioned* LBP (CLBP). Let π define a *conditioning order* on the nodes, i.e., a permutation on the set $\{1, 2, \ldots, k\}$ and its i-th element be denoted as $\pi(i)$. The simplest one is $\pi(i) = i$. Now, we select variables one at a time for conditioning, running LBP after each selection, and combine the resulting marginals. More precisely,

1. Run LBP to compute the prior distribution $P(X_{\pi(1)})$.

2. Clamp $X_{\pi(1)}$ to a specific state $x_{\pi(1)}$. Run LBP to compute the conditional distribution $P(X_{\pi(2)}|x_{\pi(1)})$.

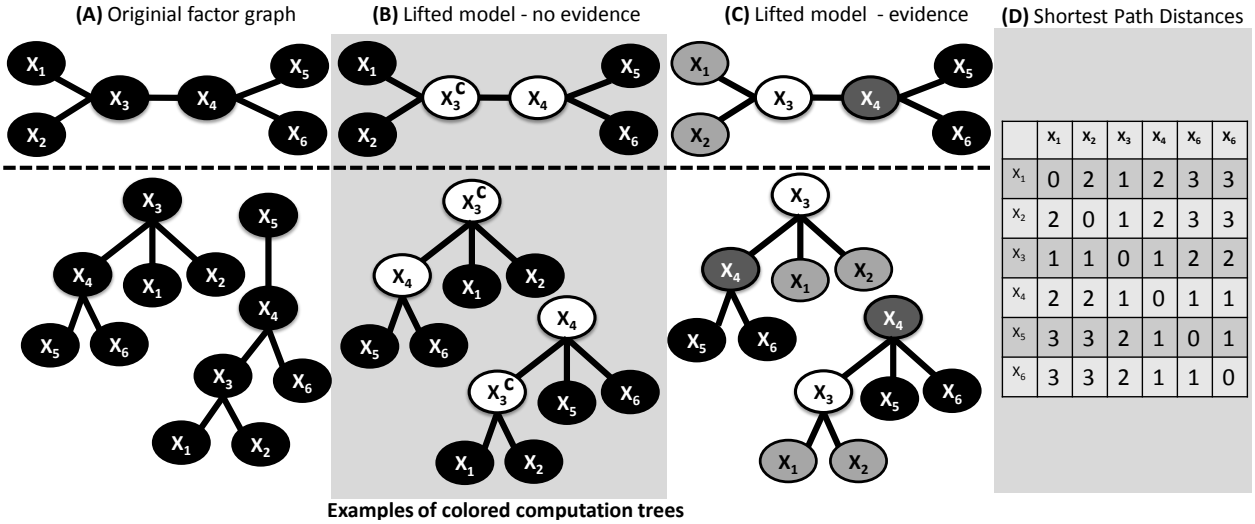
3. Do this for all states of $X_{\pi(1)}$ to obtain all conditional distributions $P(X_{\pi(2)}|X_{\pi(1)})$. The joint distribution is now $P(X_{\pi(2)}, X_{\pi(1)}) = P(X_{\pi(2)}|X_{\pi(1)}) \cdot P(X_{\pi(1)})$.

By iterating steps 2) and 3) and employing the chain rule we have $P(X_1, \ldots, X_k) = P(X_{\pi(1)}, \ldots, X_{\pi(k)}) = \prod_{i=1}^{k} P(X_{\pi(i)}|X_{\pi(i-1)}, \ldots, X_{\pi(1)})$. CLBP is simple and even exact for tree-structured models. Indeed, it is common to apply (L)BP to graphs with cycles as well. In this case the beliefs will in general not equal the true marginals, but often provide good approximations in practice. Moreover, Welling and Teh (2003) report that conditioning performs surprisingly well in terms of accuracy for estimating the covariance[1]. In the lifted case, however, the naive solution of repeatedly calling LBP may perform poorly in terms of running time. We are repeatedly answering slightly modified queries on the same graph. Because LBP generally lacks the opportunity of adaptively changing the lifted graph and using the updated lifted graph for efficient inference, it is doomed to lift the original model in each iteration again from scratch. Each CP run scales $\mathcal{O}(n \cdot m)$ where n is the number of nodes and m is the length of the longest path without loop. Hence, CLBP essentially spends $\mathcal{O}(k \cdot n \cdot m)$ time just on lifting. Moreover, in contrast to the propositional case, the conditioning order has an effect on the sizes of the lifted networks produced and, hence, the running time of MBP. It may even cancel out the benefit of lifted inference. Reconsider our chain example[2] from Fig. 1. Fig. 1(**bottom**) sketches the lifted networks produced over time when using the conditioning order $\pi = \{2, 1, 3, 4, \ldots, n-2, n-1, n+1, n\}$. As one can see, clamping X_2 dooms all subsequent iterations to run MBP on the fully grounded network, canceling the benefits of lifted inference. In contrast, the order $\pi = \{1, n+1, 2, n, \ldots, n/2-1\}$ produces lifted and fully grounded networks alternatingly, the best we can achieve for chain models. We now address both issues.

Shortest-Paths Lifting: Consider the situation depicted in Fig. 2. Given the network in **(A)** and the prior lifted network, i.e., the lifted network when no evidence has been set **(B)**, we want to compute $P(X|x_3)$ as shown in **(C)**. To do so, it is useful to describe BP in terms of its *computation tree* (CT), see e.g. (Ihler et al., 2005). The CT is the unrolling of the (loopy) graph structure where each level i corresponds to the i-th iteration of message passing. Similarly we can view CP, i.e., the lifting procedure as a *colored computation tree* (CCT). More precisely, one considers for every node X the computation tree rooted in X but now each node in the tree is colored according to the nodes' initial colors, cf. Fig. 2**(bottom)**. Each CCT encodes the root nodes' local communication patterns that show all the colored paths along which node X communicates in the network. Consequently, CP groups nodes with respect to their CCTs: nodes having the same set of rooted paths of colors (node and factor names neglected) are clustered together. For instance, Fig. 2**(A)** shows the CCTs for X_3 and X_5. Because their set of paths are different, X_3 and X_5 are clustered into different supernodes as shown in Fig. 2**(B)**. The prior lifted network can be encoded as the vector $l = (0, 0, 1, 1, 0, 0)$ of node colors. Now, when we clamp a node, say X_3, to a value x_3, we change the communication pattern of every node having a path to X. Specifically, we change X_3's (and only X_3's) color in all CCTs X_3 is involved. This is illustrated in Fig. 2**(B)**. For the prior lifted network, the dark and light nodes in Fig. 2**(B)** exhibit the

[1]The symmetrized estimate of the covariance matrix is typically not positive semi-definite and marginals computed from the joint distributions are often inconsistent with each other.

[2]When the graph is a chain or a tree there is a single chain connecting any two nodes and LBP together with dynamic programming can be used to efficiently integrate out the internal variables. When cycles exist, however, it is unclear what approximate procedure is appropriate.



	X_1	X_2	X_3	X_4	X_6	X_6
X_1	0	2	1	2	3	3
X_2	2	0	1	2	3	3
X_3	1	1	0	1	2	2
X_4	2	2	1	0	1	1
X_5	3	3	2	1	0	1
X_6	3	3	2	1	1	0

Figure 2: **(A):** Original factor graph. **(B):** Prior lifted network, i.e., lifted factor graph with no evidence. **(C):** Lifted factor graph when X_3 is set to some evidence. Factor graphs are shown **(top)** with corresponding colored computation trees **(bottom)**. For the sake of simplicity, we assume identical factors (omitted here). Ovals denote variables/nodes. The shades in **(B)** and **(C)** encode the supernodes. **(D):** Shortest-path distances of the nodes. The i-th row will be denoted d_i.

same communication pattern in the network. Consequently, X_3 appears at the same positions in all corresponding CCTs. When we now incorporate evidence on node X_3, we change its color in all CCTs as indicated by the "c" in Figs. 2**(B)** and **(C)**. This effects nodes X_1 and X_2 differently than X_4 respectively X_5 and X_6 for two reasons: (1) they have different communication patterns as they belong to different supernodes in the prior network; more importantly, (2) they have different paths connecting them to X_3 in their CCTs. The shortest path is the shortest sequence of factor colors connecting two nodes. Since we are not interested in the paths but whether the paths are identical or not, these sets might as well be represented as colors. Note that in Fig. 2 we assume identical factors for simplicity. Thus in this case path colors reduce to distances. In the general case, however, we compare the paths, i.e. the sequence of factor colors.

We only have to consider the vector d_3 of shortest-paths distances to X_3, cf. Fig. 2**(D)**, and refine the initial supernodes correspondingly. Recall that the prior lifted network can be encoded as the vector $l = (0, 0, 1, 1, 0, 0)$ of node colors. This is equivalent to (1) $l \oplus d_3$, the element-wise concatenation of two vectors, and (2) viewing each resulting number as a new color. $(0, 0, 1, 1, 0, 0) \oplus (1, 1, 0, 1, 2, 2) =_{(1)} (01, 01, 10, 11, 02, 02) =_{(2)} (0, 0, 1, 2, 3, 3)$, the lifted network for $P(X|x_3)$ as shown in Fig. 2**(C)**. Thus, we can directly update the prior lifted network in linear time without taking the detour through running CP on the ground network. Now, let us compute the lifted network for $P(X|x_4, x_3)$. Essentially, we proceed as before: compute $l \oplus (d_3 \oplus d_4)$. However, the resulting network might be suboptimal. It assumes $x_3 \neq x_4$ and, hence, X_3 and X_4 cannot be in the same supernode. For $x_4 = x_3$, they could be placed in the same supernode, if they are in the same supernode in the prior network. This can be checked by $d_3 \odot d_4$, the element-wise sort of two vectors. In our case, this yields $l \oplus (d_3 \odot d_4) = l \oplus l = l$: the prior lifted network. In general, we compute $l \oplus (\bigoplus_s (\bigoplus_v d_{s,v}))$ where $d_{s,v} = \bigodot_{i \in s : x_i = v} d_i$, s and v are the supernodes and the truth value respectively. For an arbitrary network, however, the shortest paths might be identical although the nodes have to be split, i.e. they differ in a longer path, or in other words, the shortest paths of other nodes to the evidence node are

different. Consequently we iteratively apply the shortest paths lifting. Let SN_S denote the supernodes given the set S as evidence. By applying the shortest path procedure we compute $SN_{\{X_1\}}$ from $SN_\emptyset$. This step might cause initial supernodes to be split into newly formed supernodes. To incorporate these changes in the network structure the shortest paths lifting procedure has to be iteratively applied. Thus in the next step we compute $SN_{\{X_1\}\cup\Gamma_{X_1}}$ from $SN_{\{X_1\}}$, where Γ_{X_1} denotes the changed supernodes of the previous step. This procedure is iteratively applied until no new supernodes are created. This essentially sketches the proof of the following theorem.

Theorem 1. *If the shortest-path colors among all nodes and the prior lifted network are given, computing the lifted network for $P(X|X_i,\ldots,X_1)$, $i>0$, takes $\mathcal{O}(i\cdot n\cdot s)$, where n is the number of nodes, s is the number of supernodes. Running MBP produces the same results as running BP on the original model.*

Proof. For a Graph $G=(V,E)$, when we set new evidence for a node $X\in V$ then for all nodes within the network the color of node X in the $CCTs$ is changed. If two nodes $Y_1,\ Y_2\in V$ were initially clustered together (denoted as $sn_0(Y_1)=sn_0(Y_2)$), i.e. they belong to the same supernode, they have to be split if the $CCTs$ differ. Now we have to consider two cases: If the difference in the $CCTs$ is in the shortest path connecting X with Y_1 and Y_2, respectively, then shortest-path lifting directly provides the new clustering. If the coloring along the shortest paths is identical the nodes' $CCTs$ might change in a longer path. Since $sn_0(Y_1)=sn_0(Y_2)$ there exists a mapping between the paths of the respective $CCTs$. In particular $\exists Z_1,Z_2$, s.t. $sn_0(Z_1)=sn_0(Z_2)$ from a different supernode, i.e. $sn_0(Z_i)\neq sn_0(Y_i)$, and $Y_1,\ldots,\underbrace{Z_1,\ldots,X}_{\Delta_1}\in CCT(Y_1)$, $Y_1,\ldots,\underbrace{Z_2,\ldots,X}_{\Delta_2}\in CCT(Y_2)$ and $\Delta_1\in CCT(Z_1)\neq\Delta_2\in CCT(Z_2)$ are the respective shortest paths for Z_1 and Z_2. Thus, by iteratively applying shortest-path lifting as explained above, the evidence propagates through and we obtain the new clustering. □

On Finding a Conditioning Order: Clearly, CLBP will be most efficient for estimating the probability of a joint state when it produces the smallest lifted networks. This calls for the task of finding the most efficient[3] conditioning order. Here, we provide a generically applicable strategy based on the nodes' shortest-path colors to all other nodes. That is, in each conditioning iteration, we add that node having the smallest number of unique paths to all other nodes and, if possible, is a member of a supernode of one of the already clamped nodes. Intuitively, we select nodes that are expected to create the smallest number of splits of existing supernodes in each iteration. Therefore, we call it *min-split*. Although this increases the running time — each conditioning iteration now has an additional $\mathcal{O}(n^2)$ step — our experiments show that there are important cases such as computing pairwise joint marginals where the efficiency gains achievable due to a better lifting can compensate this overhead.

4 Experimental Evaluation

Our intention here is to illustrate the performance of CLBP compared to naively running LBP and BP. We implemented CLBP and its variants in Python and using libDAI library (Mooij, 2009) and evaluated the algorithms on a number of Markov logic networks.

In our first experiment, we compared CLBP to naively running LBP, i.e. lifting the network each time from scratch, and BP for computing pairwise probabilities. We generated the "Friends-and-Smokers" Markov logic network (Singla and Domingos, 2008) with 2, 5, 10, 15, 20, and 25 people, resulting in networks ranging from 8 to 675 nodes. The shortest-path lifting clearly pays out in terms of the total messages sent (including CP and shortest-path mes-

[3]This question is different from the more common question of finding highly accurate orders. The latter question is an active research area already for the ground case see e.g. (Eaton and Ghahramani, 2009), and is also related to the difficult question of convergent BP variants, see e.g. (Mooij et al., 2007).

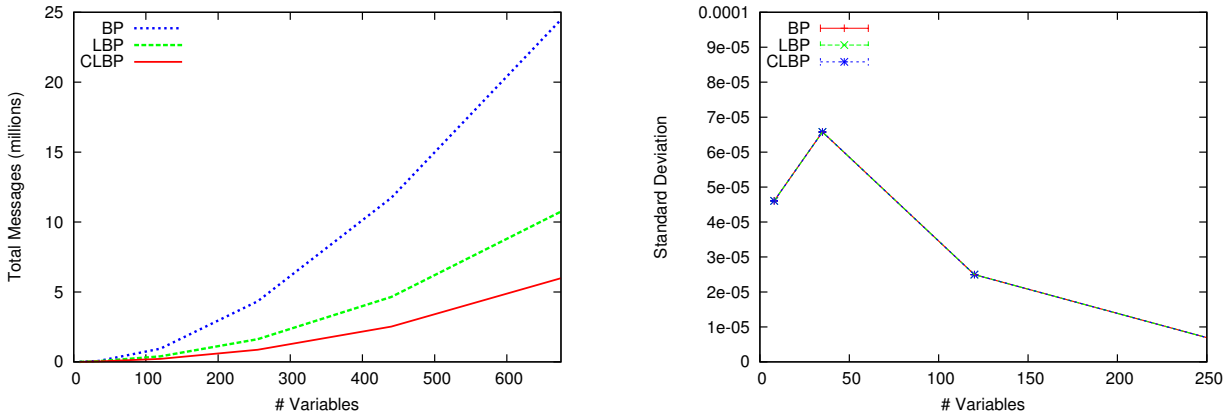


Figure 3: Pairwise Probability Estimates: **(Left)** Comparison of the total number of messages sent for BP, Lifted BP and "min-split" order CLBP for "Friends-and-Smokers" MLNs (including clustering messages for LBP and CLBP). **(Right)** The Standard Deviation of the error compared to the exact solution computed using the Junction Tree Algorithm.

sages) as shown in Fig. 3 (**left**). Moreover, the accuracy estimates are surprisingly good and confirm Welling and Teh (2003); Fig. 3 (**right**) shows the Standard Deviation of the difference compared to the exact solution computed using the Junction Tree (JT). The maximal error we got was below 10^{-4}. Note, however, that running JT with more than 20 persons was impossible due to memory and time restrictions.
In our second experiment we investigated CLBP for computing joint marginals. For the "Friends-and-Smokers" MLN with 20 people we randomly chose 1, 2, ..., 10 "cancer" and "friends" nodes as query nodes. The joint state was randomly chosen. The results are averaged over 10 runs. Fig. 4 shows the cumulative number of messages (including CP messages). "Min-split" is indeed better. By chosing the order following our heuristic the cumulative number of supernodes and in turn messages is reduced compared to a random elimination order.

Finally, we learnt paramters for the "Friends-and-Smokers" MLN with 10 perople, maximizing the conditional marginal log-likelihood (CMLL). Therefore we sampled 5 data cases from the joint distribution. We compared conjugate gradient (CG) optimization using Polak-Ribiere with Newton conjugate gradient (NCG) optimization using the covariance matrix of MLN clauses computed using CLBP. The gradient was computed as described in (Richardson and Domingos, 2006) but normalized by the number of groundings of each clause. The results summarized in Fig. 5 confirm that information about dependencies among clauses is indeed useful: the second order method exhibits faster convergence.

5 Conclusion

We presented *conditioned lifted BP*, the first approach for computing arbitrary joint marginals using lifted BP. It relates conditioning to computing shortest-paths. Exploiting this link in order to establish runtime bounds is an interesting avenue for future work. By combining lifted BP and variable conditioning, it can readily be applied to models of realistic domain size. As our results show significant efficiency gains are obtainable, sometimes order of magnitude, compared to naively running (lifted) BP in each iteration. An interesting avenue for future work is to apply CLBP within important AI tasks such as finding the MAP assignment, sequential forward sampling, and structure learning. Furthermore, our results suggest to develop lifted cutset conditioning algorithms, see e.g. (Bidyuk and Dechter, 2007), and to lift Eaton and Ghahrmani's (2009) fast heuristic for selecting nodes to be clamped to improve CLBP's accuracy.

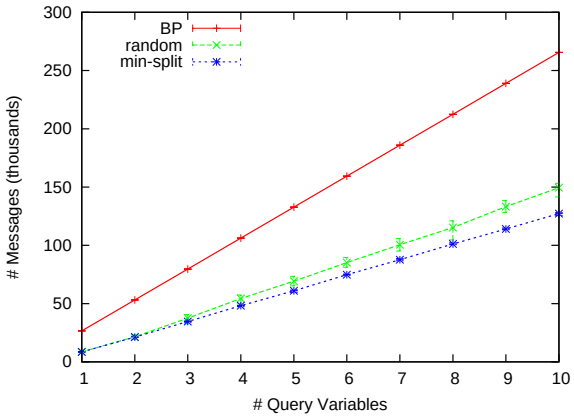


Figure 4: Number messages sent for computing joint marginals of varying size for BP, "random" and "min-split" order CLBP.

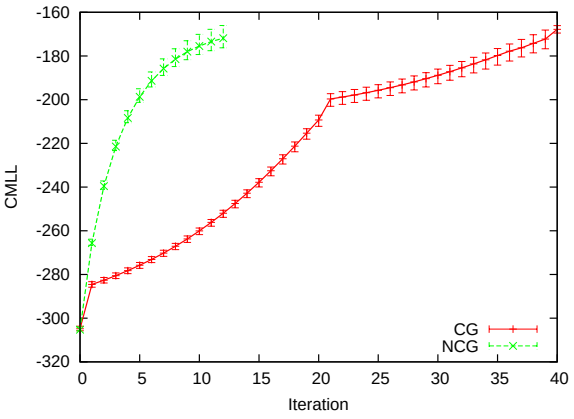


Figure 5: Learning curves for "Friends-and-Smokers" MLN. Optimization using clause covariances shows faster convergence.

Acknowledgements. This work was supported by the Fraunhofer ATTRACT fellowship STREAM and by the European Commission under contract number FP7-248258-First-MM.

References

U. Acar, A. Ihler, R. Mettu, and O. Sumer. 2008. Adaptive inference on general graphical models. In *Proc. of the Twenty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-08)*, Corvallis, Oregon. AUAI Press.

B. Bidyuk and R. Dechter. 2007. Cutset sampling for bayesian networks. *JAIR*, 28.

A. L. Delcher, A. J. Grove, S. Kasif, and J. Pearl. 1996. Logarithmic-time updates and queries in probabilistic networks. *JAIR*, 4:37–59.

F. Eaton and Z. Ghahramani. 2009. Choosing a variable to clamp: Approximate inference using conditioned belief propagation. In *Proc. of the 12th International Conference on Artificial Intelligence and Statistics (AIStats-09)*.

A.T. Ihler, J.W. Fisher III, and A.S. Willsky. 2005. Loopy belief propagation: Convergence and effects of message errors. *Journal of Machine Learning Research*, 6:905–936.

K. Kersting, B. Ahmadi, and S. Natarajan. 2009. Counting belief propagation. In J. Bilmes A. Ng, editor, *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI–09)*, Montreal, Canada, June 18–21.

B. Milch, L. Zettlemoyer, K. Kersting, M. Haimes, and L. Pack Kaelbling. 2008. Lifted Probabilistic Inference with Counting Formulas. In *Proc. of the 23rd AAAI Conf. on Artificial Intelligence (AAAI-08)*, July 13-17.

J. Mooij, B. Wemmenhove, H. Kappen, and T. Rizzo. 2007. Loop corrected belief propagation. In *Proc. of the 11th International Conference on Artificial Intelligence and Statistics (AIStats-09)*.

Joris M. Mooij. 2009. libDAI 0.2.3: A free/open source C++ library for Discrete Approximate Inference. http://www.libdai.org/.

A. Nath and P. Domingos. 2010. Efficient lifting for online probabilistic inference. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*.

J. Pearl. 1991. *Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 2. edition.

M. Richardson and P. Domingos. 2006. Markov Logic Networks. *MLJ*, 62:107–136.

P. Sen, A. Deshpande, and L. Getoor. 2009. Bisimulation-based approximate lifted inference. In J. Bilmes A. Ng, editor, *Proc. of the 25th Conference on Uncertainty in Artificial Intelligence (UAI–09)*, Montreal, Canada, June 18–21.

P. Singla and P. Domingos. 2008. Lifted First-Order Belief Propagation. In *Proc. of the 23rd AAAI Conf. on Artificial Intelligence (AAAI-08)*, pages 1094–1099, July 13-17.

M. Welling and Y.W. Teh. 2003. Linear response for approximate inference. In *Proc. of NIPS-03*, pages 191–199.