Efficient Sequential Clamping for Lifted Message Passing

Fabian Hadiji, Babak Ahmadi, and Kristian Kersting

Knowledge Discovery Department, Fraunhofer IAIS
53754 Sankt Augustin, Germany
{firstname.lastname}@iais.fraunhofer.de

Abstract. Lifted message passing approaches can be extremely fast at computing approximate marginal probability distributions over single variables and neighboring ones in the underlying graphical model. They do, however, not prescribe a way to solve more complex inference tasks such as computing joint marginals for k-tuples of distant random variables or satisfying assignments of CNFs. A popular solution in these cases is the idea of turning the complex inference task into a sequence of simpler ones by selecting and clamping variables one at a time and running lifted message passing again 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 experiments show that significant efficiency gains are possible for lifted message passing guided decimation for SAT and sampling.

Keywords: Relational Probabilistic Models, Relational Learning, Probabilistic Inference, Satisfiability

1 Introduction

Recently, there has been much interest in methods for performing lifted probabilistic inference, handling whole sets of indistinguishable objects together, see e.g. [9, 15] 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. An exception are lifted versions of belief propagation (BP) [16, 6]. They group together random variables that have identical computation trees and now run a modified BP on the resulting lifted (compressed) 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. Still, lifted message passing approaches leave space for improvement. For instance, in BP-guided decimation for satisfiability and sampling [10, 8], which are common methods for two important AI tasks, one is essentially interested in probabilities of the same network but with changing evidence. Both taks are crucial to AI in general and have important applications. For example, sampling is often used in parameter learning and the idea of "reduction to SAT" is a powerful paradigm for solving problems in different areas. A popular solution in these cases is the idea of turning the complex inference task into a sequence of simpler ones by selecting and clamping variables one at a time and running lifted BP again after each selection. However, the naive solution recomputes the lifted network in each step from scratch, therefore often canceling the benefits of lifted inference.

This paper makes a number of important and novel contributions to both the WP and lifted BP literature. We present *Shortest-Paths-Sequence Lifting* for BP (SPS-LBP), a scalable lifted inference algorithm for approximate solutions of complex inference tasks. SPS-LBP avoids the lifting from scratch for each subtask by efficiently computing the corresponding lifted network directly from the one already known for the single node marginals. We demonstrate the powerful application of these techniques to two novel lifted inference tasks: lifted message passing guided sampling and SAT solving. The experimental results demonstrate that significant efficiency gains are obtainable compared to ground inference and naive lifting in each iteration.

Indeed, there has been some prior work for related problems. Delcher *et al.* [3] propose a data structure that allows efficient queries when new evidence is incorporated in singly connected Bayesian networks and Acar *et al.* [1] 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 [11]. They essentially memorize the intermediate results of previous liftings. For new evidence they make a warm start from the first valid intermediate lifting. So far their approach has not been used for lifted sampling and lifted satisfiability. Additionally, the algorithm is only defined for Markov Logic Networks (MLNs) whereas our approach applies to any factor graph. Furthermore, the approach presented in the present paper gives a clear characterization of the core information required for sequential clamping for lifted message passing: the shortest-paths connecting the variables in the network. We proceed as follows. We start off by briefly reviewing LBP and show how it can be carried over to WP. Then, we introduce SPS-LBP. Before concluding, we present our experimental results.

2 Lifted Message Passing

Let $\mathbf{X} = (X_1, \ldots, X_n)$ 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 [12], i.e., $P(\mathbf{X} = \mathbf{x}) = Z^{-1} \prod_k f_k(\mathbf{x}_k)$. Each factor f_k is a non-negative function of a subset of the variables $\mathbf{x}_k$, and Z is a normalization constant. Each graphical model can be represented as a factor graph, 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

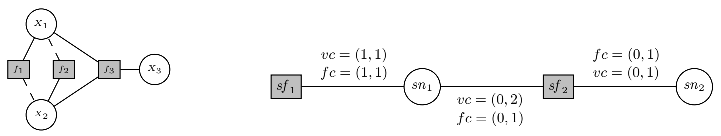


Fig. 1. (Left) $(X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_2) \wedge (X_1 \vee X_2 \vee X_3)$ represented as a factor graph. Circles denote variables, squares denote factors. A dashed line indicates that the variable appears negated in a clause **(Right)** Lifted factor graph after running CP.

if and only if X_i is an argument of f_k. 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. BP 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 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 $\mu_{X\to f}(x) = \prod_{h\in \text{nb}(X)\setminus\{f\}} \mu_{h\to X}(x)$ where nb(X) is the set of factors X appears in. The message from a factor to a variable is $\mu_{f\to X}(x) = \sum_{\neg\{X\}} \Big(f(\mathbf{x}) \prod_{Y\in \text{nb}(f)\setminus\{X\}} \mu_{Y\to f}(y)\Big)$ where nb(f) are the arguments of f, and the sum is over all of these except X, denoted as $\neg\{X\}$. The unnormalized belief of each variable X_i can be computed from the equation $b_i(x_i) = \prod_{f\in \text{nb}(X_i)} \mu_{f\to X_i}(x_i)$. Evidence is incorporated by setting $f(\mathbf{x}) = 0$ for states $\mathbf{x}$ that are incompatible with it.

Although already quite efficient, many graphical models produce factor graphs with a lot of symmetries not reflected in the structure. Lifted BP (LBP) can make use of this fact by essentially performing 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 will now briefly review the first step of LBP. For more details we refer to [6].

The lifting step can be viewed as a color-passing (CP) approach. This view abstracts from the type of messages sent and, hence, highlights one of the main insights underlying the present paper: *CP can be used to lift other message-passing approaches besides BP as well.* Since a large part of our experiments focuses on Boolean satisfiability problems, we will here explain some of the specifics when lifting Boolean formulas. We assume that a Boolean formula is represented in Conjunctive Normal Form (CNF). Every CNF can be represented as a factor graph, see e.g. [7]. More importantly, as we will show now, CP can actually be simplified in the case of CNFs. A CNF is a conjunction of disjunctions of Boolean literals. A literal is either a negated or unnegated propositional variable. Specifically, a CNF consists of n variables with $x_i \in \{0,1\}$ and m clauses constraining the variables. Every clause is represented by a factor f_k. A solution to a CNF is an assignment to all variables satisfying all constraints f_k. As an example, consider the following CNF: $(X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_2) \wedge (X_1 \vee X_2 \vee X_3)$. The factor graph representing this CNF is shown in Fig. 1 **(left)**. We use a dashed edge between a variable and a clause whenever the variable appears negated in a clause, otherwise a full line.

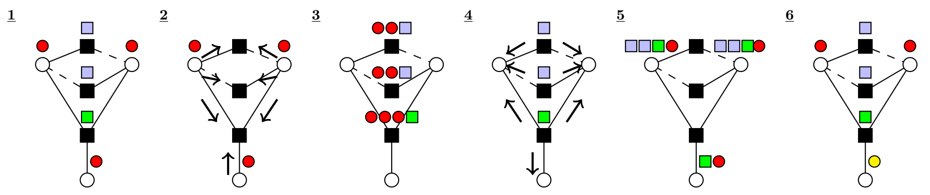


Fig. 2. From left to right, the steps of running CP on the factor graph in Fig. 1 **(left)** (assuming no evidence). The colored small circles and squares denote the groups and signatures produced running CP. (Best viewed in color.)

Let G be an arbitrary 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. In the case of CNFs, $d = 2$. All factor nodes with the same associated potentials also fall into one group represented by a shade. For CNFs, two clauses fall into the same group if both have the same number of positive and negative literals. For the factor graph in Fig. 1 the CP steps are depicted in Fig. 2. As shown on the left-hand side, assuming no evidence, all variable nodes are unknown, here: red. Now, each variable node sends a message to its neighboring factor nodes (step 2). 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, represented as light blue, respectively green, squares in Fig. 2 (step 3). Based on these signatures, the colors of the factors are newly determined and sent back to the neighboring variables nodes (step 4). The variable nodes stack the incoming signatures and, hence, form unique signatures of their one-step message history (step 5). Variable nodes with the same stacked signatures are grouped together. To indicate this, we assign a new color to each group (step 6). In our example, only variable node X_3 changes its color from red to yellow. This process is iterated until no new colors are created anymore. The final lifted graph $\mathfrak{G}$ is constructed by grouping nodes (factors) with the same color into *supernodes* (*superfactors*). Supernodes (resp. superfactors) are sets of nodes (resp. factors) that send and receive the same messages at each step of carrying out message-passing on G. In our case, variable nodes X_1, X_2 and factor nodes f_1, f_2 are grouped together into supernode sn_1 and superfactor sf_1. (Fig. 1 **(right)**). On this lifted network, LBP runs an efficient modified message passing. We refer to [16, 6] for details. However, in the case of CNFs, we can employ a more efficient simplified color signature coding scheme. The factor nodes do not have to sort incoming colors according to the positions of the variables, instead only the sign of a variable matters, i.e. only two positions exist.

3 Lifted Decimation

Many complex inference tasks, such as finding a satisfying assignment to a Boolean formula or computing joint marginals of random variables, can be cast into a sequence of simpler ones by selecting and clamping variables one at a time and running lifted inference after each selection. This is sometimes called dec-

Algorithm 1: Lifted Decimation using SPS

```
input: A factor graph $fg$, list of query vars $L$
output: List of clamped variables and values
1  cfg ← compress(fg);
2  distances ← calcDistances(fg, cfg);
3  varList = $\emptyset$;
4  while $L \neq \emptyset$ do
5      marginals ← runInfernce(cfg);
6      var, val = pickVarToClamp(marginals);
7      clampVariable(cfg, var, val);
8      adaptLifiting(cfg, dist);
9      varList += (var, val);
10     remove var from $L$;
11 return varList
```

imation and is essentially summarized in Alg. 1[1]. SAT problems can be solved using a decimation procedure based on BP (Montanari *et al.* 2007). Here, decimation is the process of iteratively assigning truth values to variables of formula F. This results in a factor graph which has these variables clamped to a specific truth value. Now, we repeatedly decimate the formula in this manner, until all variables have been clamped. In fact, when we are interested in solving CNFs, we make use of a second message passing algorithm in every iteration because some of the variables may be implied directly by unit clauses. Unit clauses are clauses consisting of a single literal only, i.e. factors with an edge to exactly one variable X. A unit clause essentially fixes the truth value of X. The process of fixing all variables appearing in unit clauses and simplifying the CNF correspondingly is called *Unit Propagation* (UP). Cast into the message passing framework, it is known under the name of *Warning Propagation (WP)* [2]. Intuitively, a message $\mu_{f \to X} = 1$ sent from a factor to a variable says: "Warning! The variable X must take on the value satisfying the clause represented by the factor f."

In the lifted case, however, we need to extend the WP equations by counts, to ensure that we simulate the ground messages. First, there is the *factor count* fc. It represents the number of equal ground factors that send messages to the variable at a given position. Because there are only two positions — a variable may occur at any position in *negated* and *unnegated* form — we store two values: counts for the negated position and for the unnegated position. Second, there is the *variable count* vc. It corresponds to the number of ground variables that send messages to a factor at each position. Reconsider Fig. 1 **(right)**. Here, the factor count associated with the edge between sn_1 and sf_1 is $(1,1)$ because sn_1 represents ground variables that appeared positive and negative in ground factors represented by sf_1. In the ground network, the clause f_3 is connected to three positive variables, but two of them are represented by a single supernode in the lifted network. Hence, we have variable count $vc = (0,2)$ for

1 Presented already using SPS that is devised in the next section

the edge between sn_1 and sf_2. The lifted equations are as follows [2]: $\mu_{f\to X,p} = \prod_{Y\in \mathrm{nb}(f)} \prod_{p'\in P(f,Y)} \theta(n_{Y\to f,p'} s_{p'})^{\mathrm{vc}(f,Y,p')-\delta_{XY}\delta_{pp'}}$ with $\theta(x) = 0$ if $x \leq 0$ and 1 otherwise, $s_{p'} = 1$ if $p' = 0$ and $s_{p'} = -1$ if $p' = 1$, $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. Here, $P(f, Y)$ only runs over positions with a variable count greater than zero. Reconsidering the factor graph in Fig. 1 **(right)**, P contains both positions for the edge $sn_1 - sf_1$, while for $sn_2 - sf_2$ it only contains the unnegated position. The Kronecker deltas reduce counts when a message is sent to the node itself that means it prevents double counting of messages. Due to space limitations we are not stating the proof that this is correct. Instead we illustrate using an example. Applying the formulas on the ground network shown in Fig. 1 **(left)** shows that the lifted formula indeed simulates WP on the ground network. In this case, there is exactly one position with a variable count greater than zero for each neighbor Y of factor f. In other words, $P(f, Y)$ is a singleton. In turn, the second product can be dropped, and both formulas coincide. Similarly, the lifted message from a supervariable to a superfactor is: $\mu_{X\to f,p} = (\sum_{\substack{h\in \mathrm{nb}(X),\\ \mathrm{fc}(h,X,1)>0}} \mu_{h\to X}(\mathrm{fc}(h, X, 1) - \delta_{fh}\delta_{p1})) - (\sum_{\substack{h\in \mathrm{nb}(X),\\ \mathrm{fc}(h,X,0)>0}} \mu_{h\to X}(\mathrm{fc}(h, X, 0) - \delta_{fh}\delta_{p0}))$. We can prove that lifted warning propagation[3] (LWP) gives the same results as WP applied to the ground network similar to the proof for LPB [16].

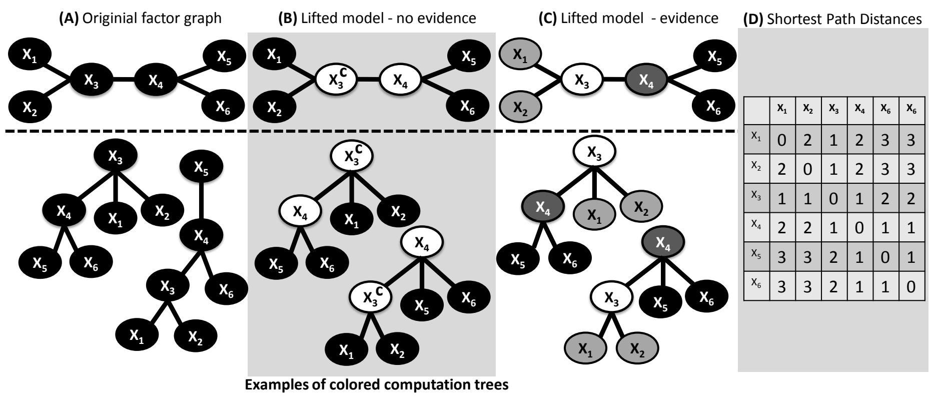
In our sequential setting, first (L)WP is run on the factor graph to clamp directly implied variables. Based on these implications the lifting is updated and (L)BP is used to fix the next variable. Additionally, we use (L)WP to detect possible contradictions . When (L)WP finds a contradiction, the algorithm stops and does not return a satisfying configuration. Indeed, lifted inference can already speed up the decimation procedure. The naive solution, however, recomputes the lifted network in each step from scratch, therefore often canceling the benefits of lifted inference. As we will show, we can do better.

4 Lifted Sequential Inference

When we turn a complex inference task into a sequence of simpler tasks, we are repeatedly answering slightly modified queries on the same graph. Because lifted BP/WP 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 of the k iterations 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 a loop. Hence, we essentially spend $\mathcal{O}(k\cdot n\cdot m)$ time just on lifting. Consider now BP-guided sampling (although the same argument applies to BP-guided decimation). When we want to sample from the joint distribution over k

[2] We define $0^0 = 1$ in cases where $\theta(x) = 0$ and $\mathrm{vc}(a, j, p') - \delta_{ij,pp'} = 0$.

[3] For the sake of simplicity, we focussed on the evidence-free updates. In the experiments, we used similar looking updates involving clamped variables, i.e., evidence. Clamping variables has effects on the WP equations, since variables can possibly not satisfy clauses anymore due to their clamped value. Additionally, we can skip messages for variables that have already been set.


(A) Originial factor graph
(B) Lifted model - no evidence
(C) Lifted model - evidence
(D) Shortest Path Distances
Examples of colored computation trees

	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

Fig. 3. (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 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.

variables, this can be reduced to a sequence of one-variable samples conditioned on a subset of the other variables [8]. Thus, to get a sample for $X = X_1, \ldots, X_k$, we first compute $P(X_1)$, then $P(X_2|X_1)$, $\ldots$, $P(X_k|X_1, \ldots X_{k-1})$ (Alg.1).

Assume we want to sample from the joint distribution $P(X_1, X_2, X_3)$, given the network in Fig. 3 **(A)**. We first compute $P(X_3)$ from the prior lifted network, i.e., the lifted network when no evidence has been set **(B)** and sample a state x_3. Now, 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. [5]. The CT is the unrolling of the graph structure where each level i corresponds to the i-th iteration of message passing. Similarly we can view CP 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. 3**(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. 3**(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. 3**(B)**. Now, when we clamp the node X_3 to a value x_3 we change the communication pattern of every node having a path to X_3. Specifically, we change X_3's (and only X_3's) color in all CCTs X_3 is involved, as indicated by the "c" in Fig. 3**(B)**. 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. 3 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.
The prior lifted network can be encoded as the vector $l = (0, 0, 1, 1, 0, 0)$ of node colors. Thus, to get the lifted network for $P(X|x_3)$ as shown in Fig. 3**(C)**, we only have to consider the vector d_3 of shortest-paths distances to X_3, cf. Fig. 3**(D)**, and refine the initial supernodes correspondingly. This is done by (1) $l \oplus d_3$, the element-wise concatenation of two vectors, and (2) viewing each resulting number as a new color. $(0, 0, 1, 1, 0, 0) \oplus (1, 1, 0, 1, 2, 2) =_{(1)} (01, 01, 10, 11, 02, 02) =_{(2)} (3, 3, 4, 5, 6, 6)$, Thus, we can directly update the prior lifted network in linear time without taking the detour through running CP on the ground network. Now, we sample a state $X_4 = x_4$ and compute the lifted network for $P(X|x_4, x_3)$. to draw a sample for $P(X_1|x_4, x_3)$. 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.*

5 Experiments

Our intention here is to investigate whether lifting improves sequential inference approaches **(Q1)** and if SPS can be even more beneficial **(Q2)**. Therefore, we run

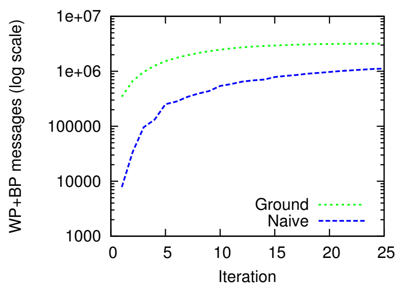
CNF Name	Iters	Ground	Naive	SPS	Walksat
ls8-normalized	26	3.17	1.12	0.95	540
ls9-normalized	13	5.47	1.65	1.47	1,139
ls10-normalized	14	10.27	1.84	1.59	1,994
ls11-normalized	26	38.82	11.51	10.64	4,500
ls12-normalized	35	60.83	13.15	11.57	10,351
ls13-normalized	21	55.39	9.99	8.21	30,061
ls14-normalized	22	83.30	10.22	8.30	104,326
2bitmax_6	55	2.35	1.25	1.05	379
5_100_sd_schur	53	111.19	75.98	64.91	1,573,208
wff.3.100.150	54	0.19	0.26	0.22	17
wff.4.100.500	78	1.73	2.04	1.89	33
wff.3.150.525	126	6.36	6.76	6.56	284

Table 1. Total messages sent (millions) in SAT experiments and number of average flips needed by Walksat.

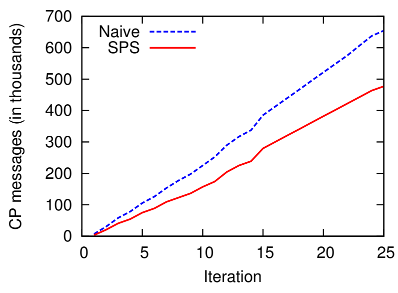
experiments on two AI tasks in which sequential clamping is essential, namely BP guided decimation for satisfiability problems and sampling in MLNs [13]. Both tasks essentially follow the decimation strategy shown in Alg. 1.

Lifted Satisfiability: We compared the performance of lifted message passing approaches with the corresponding ground versions on a CNF benchmark from [4]. We use decimation as described above to measure the efficiency of the algorithms. To assess performance, we report the number of messages sent. For the typical message sizes, e.g., for binary random variables with low degree, computing color messages is essentially as expensive as computing the actual messages. Therefore, we report both color and (modified) BP messages, treating individual message updates as atomic unit time operations. We used the "flooding" message protocol for (L)BP and (L)WP where messages are passed from each variable to all corresponding factors and back at each step. The convergence threshold was 10^{-8} for (L)BP, all messages were initialized to one (zero for (L)WP). As mentioned above, it is usually necessary to iteratively apply the SPS-lifting. The number of required iterations, however, can be high if long paths occur in the network. Therefore, we use the SPS-lifting only once but then continue with standard CP to determine the new lifting. This can still save several passes of color passing.

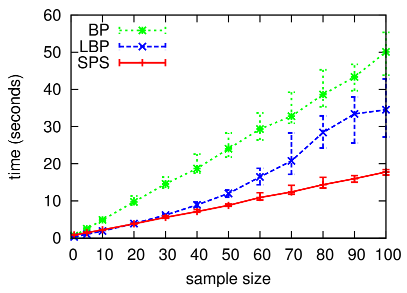
We evaluated (lifted) WP+BP decimation on different CNFs, ranging from problems with about 450 up to 78,510 edges. The CNFs contain structured problems as well as random instances. The statistics of the runs are shown in Tab. 1. As one can see, naive lifting already yields significant improvement. When applying the SPS-lifting we can do even better by saving additional messages in the compression phases. The savings in messages are visible in running times as well. Looking at the experiment in Fig. 4 and comparing ground decimation with its lifted counterpart, we only send 33% of the total ground messages. When using the SPS-lifting, we can save up to an additional 10% messages in the compression. In the decimation we always clamp the most magnetized (largest difference between negated and unnegated marginals) variable. We also applied the lifted message passing algorithms to random CNFs (last three rows in Tab.1). As expected, no lifting was possible because random instances do usually not



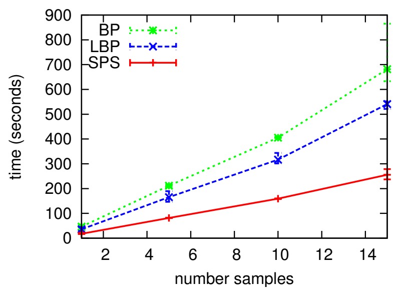
(a) WP+BP vs. LWP+LBP decimation on `ls8-normalized`



(b) naive vs. SPS-lifting for `ls8-normalized`



(c) (L)BP-guided sampling for varying sample size



(d) (L)BP-guided sampling for varying number of samples

Fig. 4. Experimental results for complex sequential inference tasks.

contain symmetries. In our experiments we were able to find satisfying solutions for all problems.

Although we are not aiming at presenting a state-of-the-art SAT solver, we solved all problems using Walksat [14] as well and we report results in Tab. 1 measured in variable flips. Even though Walksat requires fewer flips than we send messages, one can see that our lifted decimation strategy still scales well. In Fig.5 **(right)** we have compared the computational effort on increasing problem sizes for Walksat and our lifted decimation. The results indicate that our approach can handle large problem instances without employing complex heuristics and code optimization but exploiting symmetries in the problems.

Lifted Sampling: We investigated BP, LBP and SPS-LBP for sampling a joint configuration over a set of of variables sequentially, i.e. to a sequence of one-variable samples conditioned on a subset. Thus, to get a sample for $X = X_1, \ldots, X_k$, we first compute $P(X_1)$, then $P(X_2|X_1)$, …, $P(X_k|X_1, \ldots X_{k-1})$ as shown in Alg.1. For the "Friends-and-Smokers" dynamic MLN with 10 people over 10 time steps [6]. Fig. 4 summarizes the results.

In our first experiment, we randomly chose 1, 5, 10, 20, 30, …, 100 "cancer" nodes over all time steps, and sampled from the joint distribution. As one can see, LBP already provides significant improvement compared to BP, however, as the sample size increases, the speed-up is lower. The more evidence we have

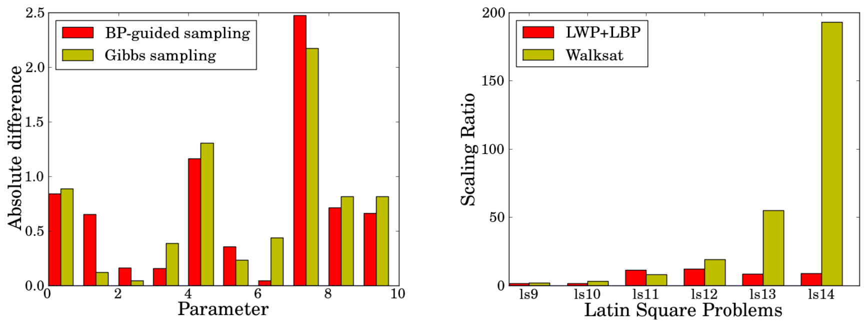


Fig. 5. left: Absolute difference of the learned parameter from the parameters of the original distribution the samples were drawn. **right:** Growth of computational costs on increasing problem sizes measured relative to the smallest problem.

in the network, the less lifting is possible. SPS-LBP has the additional gain in runtime as we do not need to perform the lifting in each step from scratch.

In our second experiment we fixed the sample size to 100, i.e. we sampled from the joint distribution of all $cancer(X, t)$ for all persons X in the domain and all time steps t. We drew 1, 5, 10 and 15 samples and the timings are averaged over 5 runs. Here, we see that LBP is only slightly advantageous compared to BP, as the sample size is 100, especially in the later iterations we have lots of evidence and long chains to propagate the evidence. Repeatedly running CP almost cancels the benefits. SPS-LBP on the other hand, shows significant speed-ups.

To evaluate the quality of the samples, we drew 100 samples of the joint distribution of all variables using the BP-guided approach and Gibbs sampling respectively. We learned the parameters of the model maximizing the conditional marginal log-likelihood (CMLL) using scaled conjugate gradient (SCG). Fig. 5**(left)** shows the absolute difference of the learnt weights from the model the datacases were drawn. As one can see parameter learning with BP-guided samples performs as good as with samples drawn by Gibbs sampling. The root-mean-square error (RMSE) for the bp parameters was 0.31 and for Gibbs parameters 0.3.

6 Conclusion

In this paper, we proposed the first decimation framework guided by lifted message-passing algorithms. To avoid the lifting from scratch in each iteration of a naive realization, we employed an efficient sequential clamping approach and gave a novel characterization of the main information required in terms of shorted-paths in a given network. The experimental results on two novel tasks for lifted inference, namely Boolean satisfiability and sampling from Markov logic networks validate the correctness of the proposed lifted decimation framework and demonstrate that instantiations can actually be faster than just using lifting.

Indeed, much remains to be done. Since lifting SAT solvers itself and the exploitation of efficient sequential lifting for it is a major advance, the most interesting avenue for future work is the tight integration of lifted SAT and lifted probabilistic inference. In many real-world applications, the problem formulation does not fall neatly into one of them. The problem may have a component that can be well-modeled as a SAT problem. Our work suggests to partition a problem into corresponding subnetworks, run the corresponding type of lifted message passing algorithm on each subnetwork, and to combine the information from the different sub-networks. Another interesting avenue is to explore lifted inference for (stochastic) planning.

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

References

1. U. Acar, A. Ihler, R. Mettu, and O. Sumer. Adaptive inference on general graphical models. In *Proc. of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI-08)*, Corvallis, Oregon, 2008. AUAI Press.
2. A. Braunstein, M. Mézard, and R. Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures and Algorithms*, 27(2):201–226, 2005.
3. A. L. Delcher, A. J. Grove, S. Kasif, and J. Pearl. Logarithmic-time updates and queries in probabilistic networks. *JAIR*, 4:37–59, 1996.
4. C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman. From sampling to model counting. In *20th IJCAI*, pages 2293–2299, Hyderabad, India, January 2007.
5. A.T. Ihler, J.W. Fisher III, and A.S. Willsky. Loopy belief propagation: Convergence and effects of message errors. *JMLR*, 6:905–936, 2005.
6. K. Kersting, B. Ahmadi, and S. Natarajan. Counting belief propagation. In *Proc. of the 25th Conf. on Uncertainty in AI (UAI–09)*, Montreal, Canada, 2009.
7. F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47, 2001.
8. M. Mezard and A. Montanari. *Information, Physics, and Computation.* Oxford University Press, Inc., New York, NY, USA, 2009.
9. B. Milch, L. Zettlemoyer, K. Kersting, M. Haimes, and L. Pack Kaelbling. Lifted Probabilistic Inference with Counting Formulas. In *Proc. of the 23rd AAAI Conf. on Artificial Intelligence (AAAI-08)*, July 13–17 2008.
10. A. Montanari, F. Ricci-Tersenghi, and G. Semerjian. Solving constraint satisfaction problems through belief propagation-guided decimation. In *Proc. of the 45th Allerton Conference on Communications, Control and Computing*, 2007.
11. A. Nath and P. Domingos. Efficient lifting for online probabilistic inference. In *Proceedings of the Twenty-Fourth AAAI Conference on AI (AAAI-10)*, 2010.
12. J. Pearl. *Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 2. edition, 1991.
13. M. Richardson and P. Domingos. Markov Logic Networks. *MLJ*, 62:107–136, 2006.
14. B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–532, 1995.
15. P. Sen, A. Deshpande, and L. Getoor. Bisimulation-based approximate lifted inference. In *Proc. of the 25th Conf. on Uncertainty in AI (UAI–09)*, 2009.
16. P. Singla and P. Domingos. Lifted First-Order Belief Propagation. In *Proc. of the 23rd AAAI Conf. on AI (AAAI-08)*, pages 1094–1099, 2008.