# Approximate Lifting Techniques for Belief Propagation

**Parag Singla**
Department of Computer Science and Engineering
Indian Institute of Technology Delhi
Hauz Khas, New Delhi, 110016, INDIA.
*parags@cse.iitd.ac.in*

**Aniruddh Nath** and **Pedro Domingos**
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350, U.S.A.
{*nath, pedrod*}*@cs.washington.edu*

**Abstract**

Many AI applications need to explicitly represent relational structure as well as handle uncertainty. First order probabilistic models combine the power of logic and probability to deal with such domains. A naive approach to inference in these models is to propositionalize the whole theory and carry out the inference on the ground network. Lifted inference techniques (such as lifted belief propagation; Singla and Domingos 2008) provide a more scalable approach to inference by combining together groups of objects which behave identically. In many cases, constructing the lifted network can itself be quite costly. In addition, the exact lifted network is often very close in size to the fully propositionalized model. To overcome these problems, we present approximate lifted inference, which groups together similar but distinguishable objects and treats them as if they were identical. Early stopping terminates the execution of the lifted network construction at an early stage resulting in a coarser network. Noise-tolerant hypercubes allow for marginal errors in the representation of the lifted network itself. Both of our algorithms can significantly speed up the process of lifted network construction as well as result in much smaller models. The coarseness of the approximation can be adjusted depending on the accuracy required, and we can bound the resulting error. Extensive evaluation on six domains demonstrates great efficiency gains with only minor (or no) loss in accuracy.

## 1 Introduction

Intelligent agents must be able to handle the complexity and uncertainty of the real world. First-order logic is useful for the first, and probabilistic graphical models for the second. Combining the two has been the focus of much recent research in the emerging field of *statistical relational learning* (Getoor and Taskar 2007). A variety of languages have been proposed that combine the desirable features of both these representations. The first inference algorithms for these languages worked by propositionalizing all atoms and clauses, and then running standard probabilistic inference algorithms on the ground model.

More recently, several algorithms have been proposed for *lifted* inference, which deals with groups of indistinguishable variables, rather than individual ground atoms. Approaches for identifying sets of indistinguishable variables include *lifted network construction* (LNC; Singla and Domingos, 2008), *shattering* (de Salvo Braz, Amir, and Roth 2005) and *finding lifting partitions* using graph automorphisms (Bui, Huynh, and Riedel 2013). Since the first-order representation is often much smaller than the fully ground model, lifted inference is potentially much more efficient than propositionalized inference. The first lifted probabilistic inference algorithm was *first-order variable elimination* (FOVE), proposed by Poole (2003), and extended by de Salvo Braz, Amir, and Roth (2005) and Milch et al. (2008). Singla and Domingos (2008) proposed the first lifted approximate inference algorithm, a first-order version of *loopy belief propagation* (BP; Yedidia, Freeman, and Weiss, 2003). Interest in lifted inference has grown rapidly in recent years (Kersting, Ahmadi, and Natarajan 2009; Kisyński and Poole 2009a; 2009b; Gordon, Hong, and Dudík 2009; Brafman and Engel 2009; Taghipour et al. 2009; Kersting, Ahmadi, and Natarajan 2009; Choi, Hill, and Amir 2010; Meert, Taghipour, and Blockeel 2010; Kersting et al. 2010; Gogate and Domingos 2011; Van den Broeck et al. 2011; Van den Broeck, Choi, and Darwiche 2012; Gogate, Jha, and Venugopal 2012; Venugopal and Gogate 2012; Bui, Huynh, and Riedel 2013).

The cost of inference for these algorithms depends on the size of the lifted network; they exploit the fact that the lifted network is potentially much smaller than the full ground network. However, lifted network construction can itself be quite expensive. The cost of LNC is highly sensitive to the representation of the lifted network. Another problem with standard lifted inference algorithms is that they often yield a lifted network very close in size to the fully propositionalized network. In these situations, the expensive process of lifted network construction yields little or no speedup. Both these problems can be averted through approximate lifted inference, which groups together variables that behave similarly, but are not completely identical. Approximate lifting can yield a much smaller model, and therefore a greater speedup. Approximate lifting can also reduce the cost of lifted network construction.

In this paper, we present two methods for approximate lifting. First, we introduce *early stopping*, which runs LNC only up to a fixed number of iterations, leading to the construction of a coarser network. Second, we propose a com-

pact *hypercube representation* for the lifted network. This forms the basis for approximate lifting by allowing for a threshold noise in hypercubes while constructing the lifted network. We describe a bound on the error for a given level of approximation. We perform an extensive evaluation of both exact and approximate lifted inference; the results show that our techniques greatly reduce the cost of inference without significantly affecting the quality of the solutions. Like Singla and Domingos (2008), we use Markov logic as the representation language, but our methods are applicable to many other first-order probabilistic representations.

## 2 Related Work

De Salvo Braz et al. (2009) proposed a form of approximate lifting that combines lifted belief propagation with box propagation (Mooij and Kappen 2008). To our knowledge, this algorithm has not been implemented and evaluated, and a detailed description has not yet been published. Sen, Deshpande, and Getoor (2009) proposed a lifting algorithm based on the notion of bisimulation, and an approximate variant of it using mini-buckets (Dechter and Rish 2003). Their technique is very effective on some domains, but it is not clear how scalable it is on more general domains, such as those considered in this paper (including Cora entity resolution task where it fails to provide a speed-up).

Kersting et al. (2010) proposed an algorithm called *informed* lifted belief propagation (iLBP) (Kersting et al. 2010), which interleaves the LNC steps and the BP steps and avoids creating the exact lifted network when BP converges in fewer iterations than LNC. iLBP can be further sped up by combining it with the approximation schemes and compact representations presented in this paper; this is a direction for future work.

Gogate and Domingos (2011) introduce a sampling version of their lifted weighted model counting approach which does approximate inference, but only over exact lifted models. Van den Broeck, Choi, and Darwiche (2012) present a technique for approximate lifting by first relaxing some of the constraints and then compensating for the relaxations. The set of relaxed constraints determines the quality of the approximation. Their approach provides a spectrum of solutions with lifted BP (fastest) on one extreme and lifted exact inference (most accurate) on the other. They report experiments over relatively small-sized domains, and it is not clear how well their approach would scale. On the other hand, our lifted approximations further relax the lifted network constructed by BP to make lifting scalable to much larger real world domains of practical interest.

Van den Broeck and Darwiche (2013) explore the possibility of lifted inference by approximating evidence using a low rank Boolean matrix factorization. The experimental evaluation is presented over a limited set of domains. Our noisy hypercube approximation can be seen as an instance of over-symmetric evidence approximation (Van den Broeck and Darwiche 2013) which we show works well in practice for a variety of real world domains.

Ahmadi et al. (2013) present an approach for breaking the network into tree-structured pieces over which (lifted) inference can be performed independently and results combined later. Our early stopping can be viewed as a form of piecewise decomposition up to depth $d$, although this decomposition may not in general result in a tree. Further, in our case, the decomposition is only used for lifting (inference still retains all the connections) unlike the piecewise inference of Ahmadi et al. (2013) where certain connections might be lost leading to larger deviations from the original network. Exploring the exact connection between the two approaches and making an empirical comparison is an item for future work.

## 3 Graphical Models

*Graphical models* compactly represent the joint distribution of a set of variables $\mathbf{X} = (X_1, X_2, \ldots, X_n) \in \mathcal{X}$ as a product of factors (Pearl 1988), $P(\mathbf{X}{=}\mathbf{x}) = \frac{1}{Z}\prod_k f_k(\mathbf{x}_k)$, where each factor $f_k$ is a non-negative function of a subset of the variables $\mathbf{x}_k$, and $Z$ is a normalization constant. Under appropriate restrictions, the model is a *Bayesian network* and $Z = 1$. A *Markov network* or *Markov random field* can have arbitrary factors. Graphical models can also be represented in *log-linear form*, $P(\mathbf{X}{=}\mathbf{x}) = \frac{1}{Z}\exp\left(\sum_i w_i g_i(\mathbf{x})\right)$, where the *features* $g_i(\mathbf{x})$ are arbitrary functions of the state. A *factor graph* (Kschischang, Frey, and Loeliger 2001) is a bipartite undirected graph with a node for each variable and factor in the model. Variables are connected to the factors they appear in.

A key inference task in graphical models is computing the marginal probabilities of some variables (the query) given the values of some others (the evidence). This problem is #P-complete, but can be solved approximately using loopy belief propagation, which works by passing messages from variable nodes to factor nodes and vice versa. The message from a variable $x$ to a factor $f$ at iteration $i+1$ is:

$$\mu_{xf,i+1}(a) = \prod_{h \in nb(x)\setminus\{f\}} \mu_{hx,i}(a)$$

where $nb(x)$ is the set of factors it appears in. (Evidence variables send 1 for the evidence value, and 0 for others.) Typically, $\mu_{fx,1} = 1$. The message from a factor to a variable is:

$$\mu_{fx,i}(a) = \sum_{\mathbf{x}_a} \left( f(\mathbf{x}_a) \prod_{y \in nb(f)\setminus\{x\}} \mu_{yf,i}(y_{\mathbf{x}_a}) \right)$$

where $nb(f)$ are the arguments of $f$; $\mathbf{x}_a$ is an assignment of values to the variables in $nb(f)$, with $x$ set to $a$; $y_{\mathbf{x}_a}$ is the value of $y$ in $\mathbf{x}_a$. The (unnormalized) marginal of variable $x$ is given by $M_{x,i}(a) = \prod_{h \in nb(x)} \mu_{hx,i}(a)$. In general, belief propagation is not guaranteed to converge, and it may converge to an incorrect result, but in practice it often approximates the true probabilities well.

## 4 Lifted Belief Propagation

Markov logic (Richardson and Domingos 2006) is a probabilistic extension of first-order logic. Formulas in first-order logic are constructed from logical connectives, predicates, constants, variables and functions. A *grounding* of a predicate (or *ground atom*) is a replacement of all its arguments

by constants (or, more generally, ground terms). A grounding of a formula is a replacement of all its variables by constants. A *possible world* is an assignment of truth values to all possible groundings of predicates.

A *Markov logic network (MLN)* is a set of weighted first-order formulas. Together with a set of constants, it defines a Markov network with one node per ground atom and one feature per ground formula. The weight of a feature is the weight of the first-order formula that originated it. The probability distribution over possible worlds $\mathbf{x}$ specified by the MLN and constants is thus $P(\mathbf{x}) = \frac{1}{Z} \exp(\sum_i w_i n_i(\mathbf{x}))$, where $w_i$ is the weight of the $i$th formula and $n_i(\mathbf{x})$ its number of true groundings in $\mathbf{x}$.

Inference in an MLN can be carried out by creating the corresponding ground Markov network and applying standard BP to it. A more efficient alternative is *lifted belief propagation*, which avoids grounding the network as much as possible (Singla and Domingos 2008). Lifted BP (LBP) constructs a *lifted network* composed of *supernodes* and *superfeatures*, and and applies BP to it. It is guaranteed to give the exact same results as running BP on the ground network. A supernode is a set of atoms that send and receive exactly the same messages throughout BP, and a superfeature is a set of ground clauses that send and receive the same messages. A supernode and a superfeature are connected iff some atom in the supernode occurs in some ground clause in the superfeature.

The lifted network is constructed by starting with an extremely coarse network, and then refining it essentially by simulating BP and keeping track of which nodes send the same messages (Algorithm 1). The count $n(s, F)$ is the number of identical messages atom $s$ would receive in message passing from the features in $F$. Belief propagation must be altered slightly to take these counts into account. Since all the atoms in a supernode have the same counts, we can set $n(X, F) = n(s, F)$, where $s$ is an arbitrary atom in $X$. The message $\mu_{FX}(a)$ from $F$ to $X$ remains the same as before: $\sum_{\mathbf{X}_a} \left( f(\mathbf{X}_a) \prod_{Y \in nb(F) \setminus \{X\}} \mu_{YF,i}(y_{\mathbf{X}_a}) \right)$. The message $\mu_{XF}(a)$ from $X$ to $F$ becomes $\mu_{FX,i}(a)^{n(X,F)-1} \prod_{H \in nb(X) \setminus \{F\}} \mu_{HX,i}(a)^{n(X,H)}$. The marginal becomes $\prod_{H \in nb(X)} \mu_{HX,i}(x)^{n(X,H)}$.

An important question remains: how to represent supernodes and superfeatures. Although this does not affect the space or time cost of inference on the lifted network, it can greatly affect the cost of constructing the lifted network. The choice of particular representation can also pave the way for an approximate construction of the lifted network (thereby saving computational cost). In general, finding the most compact representation for supernodes and superfeatures is an intractable problem.

Singla and Domingos (2008) considered two representations. The simplest option is to represent each supernode or superfeature extensionally as a set of tuples (i.e., a relation), in which case joins and projections reduce to standard database operations. However, in this case the cost of constructing the lifted network is similar to the cost of constructing the full ground network, and can easily become the bottleneck. Another option is to use a more compact representation using a constraint language, as done by Poole (2003) and de Salvo Braz et al. (2005; 2006). A simple such constraint language allows for equality/inequality constraints to be represented explicitly; we will refer to it as resolution-like representation. Taghipour et al. (2012) propose a framework for representing arbitrary constraints in lifted variable elimination. They propose a constraint language similar to our hypercube representation for lifted BP (Section 5).

**Algorithm 1** LNC(MLN $\mathbf{M}$, constants $\mathbf{C}$, evidence $\mathbf{E}$)

```
for all predicates P do
  for all truth values v do
    Form a supernode with all groundings of P
        with truth value v
  end for
end for
repeat
  for all clauses C involving P_1, ..., P_k do
    for all tuples of supernodes (X_1, ..., X_k),
    where X_i is a P_i supernode do
      Form a superfeature by joining X_1, ..., X_k
    end for
  end for
  for all supernodes X of predicate P do
    for all superfeatures F it is connected to do
      for all tuples s in X do
        n(s, F) ← num of F's tuples that project to s
      end for
    end for
    Form a new supernode from each set of tuples in X with
        same n(s, F) counts for all F
  end for
until convergence
return Lifted network L, containing all current supernodes
    and superfeatures
```

## 5 Approximate Lifting

### Early Stopping

The LNC algorithm described in section 4 is guaranteed to create the minimal lifted network (Singla and Domingos 2008). Running BP on this network is equivalent to BP on the fully propositionalized network, but potentially much faster. However, the minimal exact lifted network is often very close in size to the propositionalized network. Running LNC to convergence may also be prohibitively expensive. Both of these problems can be alleviated with approximate lifting. The simplest way to make LNC approximate is to terminate LNC after some fixed number of iterations $k$, thus creating $k$ lifted networks at increasing levels of fineness (Nath and Domingos 2010). No new supernodes are created in the final iteration. Instead, we simply average the superfeature counts $n(X, F)$ for supernode $X$ over all atoms $s$ in $X$.

Each resulting supernode contains nodes that send and receive the same messages during the first $k$ steps of BP. By stopping LNC after $k$ iterations, we are in effect pretending that nodes with identical behavior up to this point will continue to behave identically. This is a reasonable approxima-

tion, since for two nodes to be placed in the same supernode, all evidence and network topology within a distance of $k-1$ links must be identical. If the nodes would have been separated in exact LNC, this would have been a result of some difference at a distance greater than $k-1$. In BP, the effects of evidence typically die down within a short distance. Therefore, two nodes with similar neighborhoods would be expected to behave similarly for the duration of BP.

The error induced by stopping after $k$ iterations can be bounded by calculating the additional error introduced at each BP step as a result of the approximation. To do this, we perform one additional iteration of LNC (for a total of $k+1$), and compute the difference between the messages calculated at the level $k$ and level $k+1$ at each BP step (after initializing the messages at level $k+1$ from those at level $k$). To see why this captures the new error in step $i$ of BP, consider a fully ground network initialized with the messages at level $k$ of the lifted network. Run one step of BP on the ground network, passing messages from variables to factors and vice versa. The resulting messages on the ground network are identical to those yielded by initializing the messages at level $k+1$ the same way, and running a single step of BP. In other words, computations on level $k+1$ tell us what the correct messages in the next step should be, assuming the current messages are correct. (In general, the current messages are actually incorrect, but the errors have already been accounted for.) This allows us to compute the error introduced in the current iteration, and incorporate it into the bound.

Let $sn_k(x)$ and $sf_k(f)$ respectively be the supernode containing $x$ and the superfactor containing $f$ at level $k$. We view the difference between the messages at level $k$ and level $k+1$ as additional multiplicative error introduced in each BP step in the variable-factor messages at level $k$:

$$\hat{\mu}_{xf,i}(x_s) = \tilde{\mu}_{xf,i}(x_s)\tilde{e}_{xf,i}(x_s)$$

where $\hat{\mu}_{xf,i}(x_s)$ is the message from $sn_k(x)$ to $sf_k(f)$ in BP step $i$; $\tilde{\mu}_{xf,i}(x_s)$ is the message from $sn_{k+1}(x)$ to $sf_{k+1}(f)$ in step $i$, after initializing $sn_{k+1}(x)$ from $sn_k(x)$ in step $i-1$; and $\tilde{e}_{xf,i}(x_s)$ is the multiplicative error introduced in the message from $x$ to $f$ at level $k$ in step $i$.

This allows us to bound the difference in marginal probabilities calculated by ground BP and lifted BP at level $k$.

**Theorem 1.** *If ground BP converges, then for node $x$, the probability estimated by ground BP at convergence ($p_x$) can be bounded as follows in terms of the probability estimated by level $k$ lifted BP ($\hat{p}_x$) after $n$ BP steps:*

$$p_x \geq \frac{1}{(\zeta_{x,n})^2[(1/\hat{p}_x)-1]+1} = lb(p_x)$$
$$p_x \leq \frac{1}{(1/\zeta_{x,n})^2[(1/\hat{p}_x)-1]+1} = ub(p_x)$$

*where*

$$\log \zeta_{x,n} = \sum_{f \in nb(x)} \log \nu_{fx,n}$$
$$\nu_{fx,1} = d(f)^2$$
$$\log \nu_{xf,i+1} = \sum_{h \in nb(x)\setminus\{f\}} \log \nu_{hx,i} + \log \delta_{xf,i+1}$$
$$\log \nu_{fx,i} = \log \frac{d(f)^2 \varepsilon_{fx,i}+1}{d(f)^2+\varepsilon_{fx,i}}$$
$$\log \varepsilon_{fx,i} = \sum_{y \in nb(f)\setminus\{x\}} \log \nu_{yf,i}$$
$$\delta_{xf,i} = \sup_{x_s,y_s} \sqrt{\frac{\hat{\mu}_{xf,i}(x_s)}{\tilde{\mu}_{xf,i}(x_s)} \frac{\tilde{\mu}_{xf,i}(y_s)}{\hat{\mu}_{xf,i}(y_s)}}$$
$$d(f) = \sup_{x,y} \sqrt{f(x)/f(y)}$$

*Proof.* $d(f)$ is the *dynamic range* of function $f$, as defined in Ihler, Fisher, and Willsky (2005). Since we can calculate $\hat{\mu}_{xf,i}(x_s)$ and $\tilde{\mu}_{xf,i}(x_s)$, the dynamic range of the error function can be calculated as follows:

$$d(\tilde{e}_{xf,i}) = \sup_{x_s,y_s} \sqrt{\frac{\hat{\mu}_{xf,i}(x_s)}{\tilde{\mu}_{xf,i}(x_s)} \frac{\tilde{\mu}_{xf,i}(y_s)}{\hat{\mu}_{xf,i}(y_s)}} = \delta_{xf,i}$$

Using the same logic as Theorem 15 of Ihler, Fisher, and Willsky (2005), for any fixed point beliefs $\{M_x\}$ found by ground BP, after $n \geq 1$ steps of BP at level $k$ resulting in beliefs $\{\hat{M}_{x,n}\}$[1]

$$\log d(M_x/\hat{M}_{x,n}) \leq \sum_{f \in nb(x)} \log \nu_{fx,n} = \log \zeta_{x,n}$$

It follows that $d(M_t/\hat{M}_{x,n}) \leq \zeta_{x,n}$, and therefore:

$$\frac{M_x(1)/\hat{M}_{x,n}(1)}{M_x(0)/\hat{M}_{x,n}(0)} \leq (\zeta_{x,n})^2$$

and

$$(1-\hat{p}_x)/\hat{p}_x \leq (\zeta_{x,n})^2(1-p_x)/p_x$$

where $p_x$ and $\hat{p}_x$ are obtained by normalizing $M_x$ and $\hat{M}_{x,n}$. The upper bound follows, and the lower bound can be obtained similarly. □

Intuitively, $\nu_{ut,i}$ can be thought of as a measure of the accumulated error in the message from $u$ to $t$ in BP step $i$. It can be computed iteratively using a message-passing algorithm similar to lifted BP on the level $k+1$ graph.

## Noise-Tolerant Hypercubes

Another approach to approximate lifting is to allow for marginal error in the representation of each supernode (superfeature) in the lifted network. We use a hypercube based representation for the lifted network. This forms the basis for approximate lifting by giving a framework for representing noise during the lifted network construction phase.

[1]Proved by Ihler, Fisher, and Willsky (2005) for pairwise Markov networks, but true for arbitrary graphs; see supplement (Singla, Nath, and Domingos 2014).

**Algorithm 2** FormHypercubes(Tuple set $\mathbf{T}$)

```
Form bounding hypercube C_bound from T
H ← {C_bound}
while H contains some impure hypercube C do
    Calculate r_C, the fraction of true tuples ∈ C
    for all variables v in C do
        Decompose C into C = (C_1, ..., C_k)
            by value of v (one hypercube per value)
        Calculate (r_C1, ..., r_Ck)
        C+ ← merge all C_i ∈ C with r_Ci > r_C
        C− ← merge all C_i ∈ C with r_Ci ≤ r_C
        Calculate r_C+
    end for
    Split C into (C+, C−) with highest r_C+
    Remove C from H; insert C+ and C−
end while
while H contains some mergeable pair (C_1, C_2) do
    Replace C_1 and C_2 with merged hypercube C_new
end while
```

**Hypercube Representation** A *hypercube* is a vector of sets of literals, $[S_1, S_2, \ldots, S_k]$; the corresponding set of tuples is the Cartesian product $S_1 \times S_2 \times \ldots \times S_k$. A supernode or superfeature can be represented by a union of disjoint hypercubes. Hypercube representation can be exponentially more compact than both the extensional and resolution-like representations. Hypercube (or similar) representation can also be used effectively in lifted exact inference algorithms such as FOVE (Taghipour et al. 2012). In general, there can be more than one minimal decomposition, and finding the best one is an NP-hard problem. (In two dimensions, it is equivalent to the minimum biclique partition problem, Amilhastre, Vilarem, and Janssen, 1998.) In the construction of hypercubes, first, a bounding hypercube is constructed, i.e., one which includes all the tuples in the set. This is a crude approximation to the set of tuples which need to be represented. The hypercube is then recursively sub-divided so as to split apart tuples from non-tuples, using a heuristic splitting criterion (see Algorithm 2). Other criteria such as information gain can also be used, as is commonly done in decision trees. The process is continued recursively until each hypercube either contains all valid tuples or none (in which case it is discarded). This process does not guarantee a minimal splitting, since hypercubes from two different branches could potentially be merged. Therefore, once the final set of hypercubes is obtained, we recursively merge the resulting hypercubes in a bottom-up manner to get a minimal set. (Running the bottom approach directly on individual set of tuples can be inefficient.)

*Hypercube Join:* When joining two supernodes, we join each possible hypercube pair (each element of the pair coming from the respective supernode). Joining a pair of hypercubes simply corresponds to taking an intersection of the common variables and keeping the remaining ones as is. Instead of joining each possible pair of hypercubes, an index can be maintained which tells which pairs will have a non-zero intersection.

*Hypercube Project:* The project operation now projects superfeature hypercubes onto the arguments the superfeature shares with each of its predicates. Each supernode hypercube maintains a separate set of counts. This presents a problem because different superfeatures may now project onto hypercubes which are neither disjoint nor identical. Therefore, the hypercubes resulting from the project operation have to be split into finer hypercubes such that each pair of resulting hypercubes is either identical with each other or disjoint. This can be done by choosing each intersecting (non-disjoint) pair of hypercubes in turn and splitting them into the intersection and the remaining components. The process continues until each pair of hypercubes is disjoint.

**Noise Tolerance** During the top-down construction of hypercubes, instead of refining the hypercubes until the end, we stop the process on the way allowing for at most a certain maximum amount of noise in each hypercube. The goal is to obtain the largest possible hypercubes while staying within the noise tolerance threshold. Different measures of noise tolerance can be used. One such measure which is simple to use and does well in practice is the number of tuples not sharing the majority truth value in the hypercube. Depending on the application, other measures can be used.

This scheme allows for a more compact representation of supernodes and superfeatures, potentially saving both memory and time. This approximation also allows the construction of larger supernodes, by virtue of grouping together certain ground predicates which were earlier in different supernodes. These gains come at the cost of some loss in accuracy due to the approximations introduced by noise tolerance. However, as we will see in the experiment section, the loss in accuracy is offset by the gains in both time and memory.

As in section 5, we can derive an error bound based on Ihler, Fisher, and Willsky (2005) by treating approximate messages as multiplicative error on the 'true' messages. Calculating this error bound requires running a message-passing algorithm similar to BP on the exact lifted network. (See supplement (Singla, Nath, and Domingos 2014) for more details.)

## 6 Experimental Results

We compared the performance of lifted BP (exact and approximate) with the ground version on five real domains and one artificial domain. (Results on two datasets are in the supplementary material (Singla, Nath, and Domingos 2014) due to lack of space.) We implemented lifted BP as an extension of the open-source Alchemy system (Kok et al. 2008). Exact lifted BP is guaranteed to give the same results as the ground version. We report accuracy comparisons for the approximate versions. We used the trained models available from previous research wherever applicable. In other cases, the models were trained using the learning algorithms available in Alchemy. In all these cases, exactly the same model was used for testing all the algorithms on any given dataset. Diagnosing the convergence of BP is a difficult problem; we ran it for 1000 steps for all algorithms in all experiments. BP did not always converge.

Table 1: Experimental results. Memory is in MB; features and tuples are in thousands.

| | Algorithm | Time (in seconds) | | | Memory | | | Accuracy | |
|---|---|---|---|---|---|---|---|---|---|
| | | Construct | BP | Total | Memory | Features | Tuples | CLL | AUC |
| Cora | Ground | 10.9 | 299.3 | 310.2 | 138 | 110.3 | 110.3 | **-0.531** | **0.935** |
| | Extensional | 14.8 | 49.0 | 63.8 | 137 | 15.4 | 110.3 | **-0.531** | **0.935** |
| | Resolution | 15.0 | 49.1 | 64.1 | 138 | 15.4 | 110.3 | **-0.531** | **0.935** |
| | Hypercube | 7.4 | 46.4 | 53.8 | 110 | 15.4 | 38.3 | **-0.531** | **0.935** |
| | Early Stop | **5.0** | **30.3** | **35.4** | 108 | 14.4 | 38.3 | **-0.531** | **0.935** |
| | Noise-Tol. | 5.4 | 32.3 | 37.8 | **102** | **13.4** | **20.0** | -1.624 | 0.914 |
| WebKB | Ground | 13.3 | 1333.0 | 1346.3 | 393 | 997.8 | 997.8 | **-0.036** | **0.016** |
| | Extensional | 26.9 | 0.8 | 27.8 | 285 | **0.9** | 997.8 | **-0.036** | **0.016** |
| | Resolution | 27.1 | 0.8 | 27.9 | 287 | **0.9** | 997.8 | **-0.036** | **0.016** |
| | Hypercube | **0.1** | **0.6** | **0.8** | **94** | **0.9** | **1.0** | **-0.036** | **0.016** |
| | Early Stop | **0.1** | **0.6** | **0.8** | **94** | **0.9** | **1.0** | **-0.036** | **0.016** |
| | Noise-Tol. | **0.1** | **0.6** | **0.8** | **94** | **0.9** | **1.0** | **-0.036** | **0.016** |
| Denoise | Ground | **16.7** | 4389.8 | 4406.6 | 748 | 1269.3 | **1269.3** | **-0.011** | **0.997** |
| | Extensional | 211.6 | 4313.6 | 4525.3 | 2202 | 1269.3 | **1269.3** | **-0.011** | **0.997** |
| | Resolution | 342.1 | 4289.7 | 4631.9 | 2247 | 1269.3 | **1269.3** | **-0.011** | **0.997** |
| | Early Stop | 32.7 | **1.2** | **34.0** | **440** | **0.6** | **1269.3** | -0.064 | 0.987 |
| Smokers | Ground | 22.8 | 5919.4 | 5942.2 | 1873 | 1093.40 | 1093.4 | | |
| | Extensional | 163.6 | **0.1** | 163.7 | 2074 | 0.06 | 1093.4 | | |
| | Resolution | 45.2 | **0.1** | 45.3 | 1423 | 0.06 | 823.3 | | |
| | Hypercube | 52.0 | **0.1** | 52.1 | 1127 | 0.06 | 21.0 | | |
| | Early Stop | 51.6 | **0.1** | 51.7 | 1127 | 0.06 | 21.0 | | |
| | Noise-Tol. | **3.0** | **0.1** | **3.1** | **1123** | **0.04** | **4.6** | | |

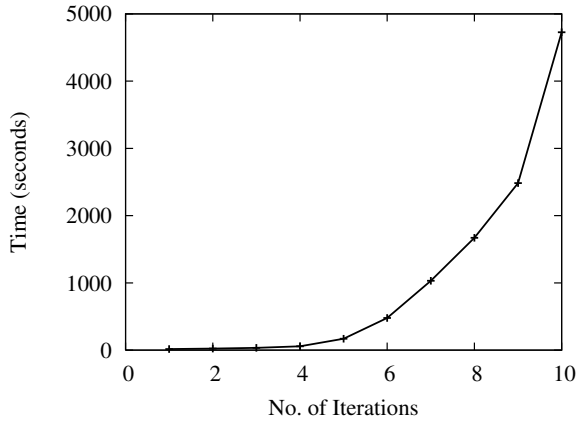


Figure 1: Time vs. number of iterations for early stopping on Denoise.

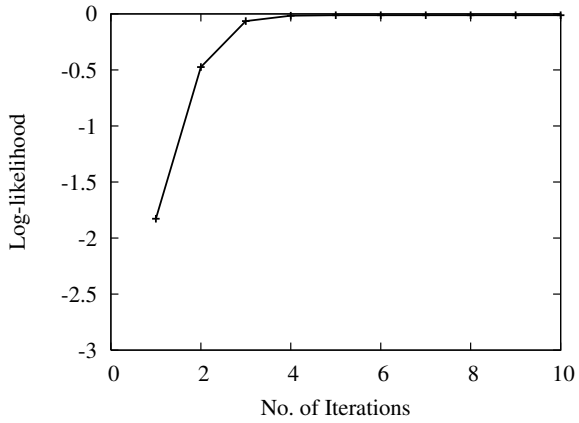


Figure 2: Log-likelihood vs. number of iterations for early stopping on Denoise.

## Datasets

**Entity Resolution** on McCallum's Cora database, consisting of 1295 citations to 132 different research papers. The inference task was to detect duplicate citations, authors, titles and venues. A TF-IDF-based model was used, as described by Singla and Domingos (2005). The data was divided into 5 splits for cross validation. Voted perceptron (with $\eta = 10^{-5}$ and a Gaussian prior) was used for training the model. (We used the parameters learned on one of the splits.) Canopies were used to eliminate obvious non-matches (McCallum, Nigam, and Ungar 2000).

**Hyperlink Prediction** on the WebKB dataset (Craven and Slattery 2001), consisting of labeled web pages from the computer science departments of four universities. The goal was to predict which web pages point to each other, given their topics. We used only those web pages for which we had page class information, leaving 1,208 web pages with 10,063 web links. Each page is marked with some subset of the following categories: person, student, faculty, department, research project, and course. We used an MLN with

rules capturing homophily for each pair of classes, and a rule capturing reflexivity. The model was trained by optimizing pseudo-likelihood using L-BFGS, using default parameter settings in Alchemy (with no prior on weights).

**Image Denoising** using the binary image in Bishop (2006) (Section 8.3.3), scaled to 400×400 pixels. We introduced noise by randomly flipping 10% of the pixels. The MLN consists of four rules. The first pair of rules stated that the actual value of a pixel is likely to be same as the observed value. The second pair of rules stated that neighbors are likely to have the same pixel value. As suggested in Bishop (2006), the first two rules were given a weight of 2.1 and the last two were given a weight of 1.0.

**Social Network** link and label prediction, on the "Friends and Smokers" MLN from Richardson and Domingos (2006). The MLN has rules stating that smoking causes cancer, and friends have similar smoking habits. We used a network with 1000 people. The domain was divided into a set of friendship clusters of size 50 each. For a randomly chosen 10% of the people, we know (a) whether they smoke or not, and (b) who 10 of their friends are (other friendship relations are still assumed to be unknown). `Cancer(x)` is unknown for all `x`. For each known person, we randomly chose each friend with equal probability of being inside or outside their friendship cluster. All the unknown atoms were queried.

### Algorithms and Metrics

We report results for several algorithms: **ground** (plain, ground version of BP), **extensional**, **resolution**, **hypercube** (three representations for lifted BP), **early stopping** and **noise-tolerant** (two approximate versions of lifted BP). For early stopping, three iterations of lifted network construction were used, with the hypercube representation (unless otherwise mentioned). For noise-tolerant hypercube construction, we allowed at most one tuple to have a truth value differing from the majority value. For accuracy, we report the conditional log-likelihood (**CLL**) and the area under the precision-recall curve (**AUC**).

### Results

For each dataset, we present a table of results comparing time, memory and accuracy for the various algorithms described above (see Table 1). For all the datasets, the reported results are the average over the respective splits described in section 6. Separate results are not reported for the hypercube representation on Denoise, since the resulting hypercubes are degenerate, and equivalent to the extensional representation. For FS, we do not report accuracy results, as we did not have the ground truth.

In all cases, LNC with extensional or resolution-like representations is slower than grounding the full network. The hypercube representation allows LNC to overtake standard network construction on WebKB and Cora. Introducing hypercube noise tolerance makes LNC faster on FS as well, at some cost to accuracy.

Running BP on the lifted network is much faster than ground BP on all domains except Denoise. Here running LNC to the end does not give any benefit and degenerates into the ground network. Hence, BP running times are almost the same. However, the early stopping approximation is highly beneficial in this domain, resulting in order-of-magnitude speedups with negligible loss in AUC.

On all datasets besides Denoise, the exact lifted network has a much smaller number of (super)features than the fully ground network; approximations lead to further gains resulting in reduced memory requirements and number of (super)features over all domains (including Denoise). On Denoise, early stopping reduces the number of features by more than three orders of magnitude.

Accuracies for the ground and exact lifted versions of BP are the same. Interestingly, early stopping has little or no effect on accuracy. Introducing noise tolerance does result in some loss of accuracy (AUC and/or CLL), but it yields large improvements in time and memory.

We also analyzed how time, memory and accuracy change with varying criteria for early stopping and noise tolerance. On Denoise, we varied the number of iterations of LNC from 1 to 10 (LNC converged at 10 iterations). Time increases monotonically with increasing number of iterations. Memory usage is almost constant until six iterations, after which it increases monotonically. CLL and AUC both converge to the optimum after four iterations; at this stage, time and memory requirements are still very small compared to running LNC until the end. Figures 1 and 2 plot the change in time and CLL, respectively, as we vary the number of stopping iterations for Denoise. The figures for variations in memory and AUC can be can be found in the supplementary material (Singla, Nath, and Domingos 2014). On the Cora domain, the noise tolerance was varied from zero to five. Time and memory decrease monotonically with increasing noise tolerance. Increasing noise tolerance decreases CLL and AUC, as expected. See the supplement for the figures.

## 7 Conclusion

In this paper, we presented two algorithms for approximate lifted belief propagation and provided error bounds for them. Experiments on a number of datasets show the benefit of approximate lifting over exact lifting. Directions for future work include generalizing our approach to other kinds of inference, exploring other forms of approximation, and applying these algorithms to more domains.

## 8 Acknowledgments

This research was funded by ARO grant W911NF-08-1-0242, ONR grants N00014-13-1-0720, N00014-12-1-0312, and N-00014-05-1-0313, DARPA contracts NBCH-D030010/02-000225, FA8750-07-D-0185, and HR0011-07-C-0060, DARPA grant FA8750-05-2-0283, and NSF grant IIS-0534881. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DARPA, NSF, ONR, ARO or the United States Government.

## References


Ahmadi, B.; Kersting, K.; Mladenov, M.; and Natarajan, S. 2013. Exploiting symmetries for scaling loopy belief propagation and relational training. *Machine Learning* 92(1).

Amilhastre, J.; Vilarem, M. C.; and Janssen, P. 1998. Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs. *Discrete applied mathematics* 86.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Brafman, R., and Engel, Y. 2009. Lifted optimization for relational preference rules. In *SRL-09*.

Bui, H.; Huynh, T.; and Riedel, S. 2013. Automorphism groups of graphical models and lifted variational inference. In *Proc. of UAI-13*.

Choi, J.; Hill, D.; and Amir, E. 2010. Lifted inference for relational continuous models. In *Proc. of UAI-10*.

Craven, M., and Slattery, S. 2001. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*.

de Salvo Braz, R.; Amir, E.; and Roth, D. 2005. Lifted first-order probabilistic inference. In *Proc. of IJCAI-05*.

de Salvo Braz, R.; Amir, E.; and Roth, D. 2006. MPE and partial inversion in lifted probabilistic variable elimination. In *Proc. of AAAI-06*.

de Salvo Braz, R.; Natarajan, S.; Bui, H.; Shavlik, J.; and Russell, S. 2009. Anytime lifted belief propagation. In *SRL-09*.

Dechter, R., and Rish, I. 2003. Mini-buckets: A general scheme for bounded inference. *J. ACM* 50(2).

Getoor, L., and Taskar, B., eds. 2007. *Introduction to Statistical Relational Learning*. MIT Press.

Gogate, V., and Domingos, P. 2011. Probabilisitic theorem proving. In *Proc. of UAI-11*.

Gogate, V.; Jha, A.; and Venugopal, D. 2012. Advances in lifted importance sampling. In *Proc. of AAAI-12*.

Gordon, G.; Hong, S. A.; and Dudík, M. 2009. First-order mixed integer linear programming. In *Proc. of UAI-09*.

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

Kersting, K.; Ahmadi, B.; and Natarajan, S. 2009. Counting belief propagation. In *Proc. of UAI-09*.

Kersting, K.; Massaoudi, Y. E.; Ahmadi, B.; and Hadiji, F. 2010. Informed lifting for message–passing. In *Proc. of AAAI-10*. AAAI Press.

Kisyński, J., and Poole, D. 2009a. Constraint processing in lifted probabilistic inference. In *Proc. of UAI-09*.

Kisyński, J., and Poole, D. 2009b. Lifted aggregation in directed first-order probabilistic models. In *Proc. of IJCAI-09*.

Kok, S.; Sumner, M.; Richardson, M.; Singla, P.; Poon, H.; Lowd, D.; Wang, J.; and Domingos, P. 2008. The Alchemy system for statistical relational AI. Technical report, University of Washington. http://alchemy.cs.washington.edu.

Kschischang, F. R.; Frey, B. J.; and Loeliger, H.-A. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory* 47.

McCallum, A.; Nigam, K.; and Ungar, L. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proc. of KDD-00*.

Meert, W.; Taghipour, N.; and Blockeel, H. 2010. First-order Bayes-ball. In *Proc. of ECML-10*.

Milch, B.; Zettlemoyer, L. S.; Kersting, K.; Haimes, M.; and Kaebling, L. P. 2008. Lifted probabilistic inference with counting formulas. In *Proc. of AAAI-08*.

Mooij, J. M., and Kappen, H. J. 2008. Bounds on marginal probability distributions. In *Proc. of NIPS-08*.

Nath, A., and Domingos, P. 2010. Efficient lifting for online probabilistic inference. In *Proc. of AAAI-10*.

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

Poole, D. 2003. First-order probabilistic inference. In *Proc. of IJCAI-03*.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62.

Sen, P.; Deshpande, A.; and Getoor, L. 2009. Bisimulation-based approximate lifted inference. In *Proc. of UAI-09*.

Singla, P., and Domingos, P. 2005. Discriminative Training of Markov Logic Networks. In *Proc. of AAAI-05*.

Singla, P., and Domingos, P. 2008. Lifted first-order belief propagation. In *Proc. of AAAI-08*.

Singla, P.; Nath, A.; and Domingos, P. 2014. Approximate lifting techniques for belief propagation: Supplementary material. http://www.cse.iitd.ac.in/˜parags/papers/albp-sup-aaai14.pdf.

Taghipour, N.; Meert, W.; Struyf, J.; and Blockeel, H. 2009. First-order Bayes-ball for CP-logic. In *SRL-09*.

Taghipour, N.; Fierens, D.; Davis, J.; and Blockeel, H. 2012. Lifted variable elimination with aribtrary constraints. In *Proc. of AISTATS-12*.

Van den Broeck, G., and Darwiche, A. 2013. On the complexity and approximation of binary evidence in lifted inference. In *Proc. of NIPS-13*.

Van den Broeck, G.; Taghipour, N.; Meert, W.; Davis, J.; and Raedt, L. D. 2011. Lifted probabilistic inference by first-order knowledge compilation. In *Proc. of IJCAI-11*.

Van den Broeck, G.; Choi, A.; and Darwiche, A. 2012. Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *Proc. of UAI-12*.

Venugopal, D., and Gogate, V. 2012. On lifting the Gibbs sampling algorithm. In *Proc. of NIPS-12*.

Yedidia, J. S.; Freeman, W. T.; and Weiss, Y. 2003. Understanding belief propagation and its generalizations. In *Exploring Artificial Intelligence in the New Millenium*. Science and Technology Books.