Approximate Lifted Belief Propagation

Parag Singla
Department of Computer Science
University of Texas at Austin
Austin, TX 78712, U.S.A.
parag@cs.utexas.edu

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

Lifting can greatly reduce the cost of inference on first-order probabilistic models, but constructing the lifted network can itself be quite costly. In addition, the minimal lifted network is often very close in size to the fully propositionalized model; lifted inference yields little or no speedup in these situations. In this paper, we address both these problems. We propose a compact hypercube-based representation for the lifted network, which can greatly reduce the cost of lifted network construction. We also present two methods for approximate lifted network construction, which groups together similar but distinguishable objects and treats them as if they were identical. This can greatly reduce the size of the lifted network as well as the time required for lifted network construction, but potentially at some cost to accuracy. The coarseness of the approximation can be adjusted depending on the accuracy required, and we can bound the resulting error. Experiments on six domains show great efficiency gains with only minor loss in accuracy.

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 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. Algorithms for identifying sets of indistinguishable variables include *lifted network construction* (LNC; Singla and Domingos 2008) and *shattering* (de Salvo Braz, Amir, and Roth 2005). Since the first-order representation is often much smaller than the fully ground model, lifted inference is potentially much more efficient than propositionalized inference. Interest in lifted inference has grown rapidly in recent years (see Kersting, Ahmadi, and Natarajan (2009); Kisyński and Poole (2009a; 2009b); Gordon, Hong, and Dudík (2009); Brafman and Engel (2009); Taghipour et al. (2009) etc.). The first lifted probabilistic inference algorithm was *first-order variable elimination*, proposed by Poole (2003) and extended by de Salvo Braz, Amir, and Roth (2005). 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).

The cost of inference for these algorithms depends on the size of the lifted network; this is often 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. We present a compact *hypercube representation*, which can greatly reduce the cost of LNC.

Another problem with standard lifted inference algorithms is that they often yield a lifted network very close to the fully propositionalized network. In these situations, the expensive process of lifted network construction yields little or no speedup. This 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 propose two methods for approximate lifted belief propagation. We also provide a bound on the error for a given level of approximation. We perform an extensive evaluation of both exact and approximate lifted inference; results show that our techniques greatly reduce the cost of inference without significantly affecting the quality of the solutions.

Two other approaches for approximate lifted inference have recently been proposed. 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 failed to yield a speedup in the Cora entity resolution task, where our algorithms produce a sig-

nificant improvement. It is not clear how scalable their algorithms are on the other domains considered in this paper.

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.

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 to a factor 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 $M_{x,i}(a) = \prod_{h \in nb(x)} \mu_{hx,i}(a)$

Many message-passing schedules are possible; the most widely used one is *flooding*, where all nodes send messages at each step. 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.

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). Similarly, 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 ith 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 constructs a *lifted network* composed of *supernodes* and *superfeatures*, and and applies BP to it. 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. Initially, one supernode is created for each possible value of each predicate (for binary predicates, *true*, *false* and *unknown*). All groundings of each value are inserted into the corresponding supernode. This is the initial set of supernodes; in any given supernode, every atom would send the same message in the first iteration of BP.

The next stage of LNC involves creating a set of superfeatures corresponding to each clause. For a clause containing predicates $P_1, \ldots, P_k$, perform a join on each tuple of matching supernodes $(X_1, \ldots, X_k)$ (i.e. take the Cartesian product of the relations, and select the tuples whose corresponding arguments agree with each other). The result of each join is a new superfeature, consisting of ground clauses that would receive the same messages in the first BP step.

These superfeatures can be used to refine the supernodes, by projecting each feature down to the arguments it shares with each of its predicates. Atoms with the same projection counts are indistinguishable, since they would have received the same number of identical messages. These can be split off into new, finer supernodes. Thus, the network is refined by alternating between these two steps:

1. Refine superfeatures by performing joins on supernodes.
2. Refine supernodes by projecting superfeatures down to their predicates, and merging atoms with the same projection counts.

For a detailed description of this algorithm, see Singla and Domingos (2008).

The count $n(s, F)$ is the number of identical messages atom s would receive in message passing from the features in F. (As in Singla and Domingos (2008), we assume for simplicity that each predicate occurs at most once in each clause. This assumption can be easily relaxed by maintaining a separate count for each occurrence.)

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 from N to F becomes: $\mu_{XF,i+1}(x) = \mu_{FX,i}(x)^{n(X,F)-1} \prod_{H \in nb(X) \setminus \{F\}} \mu_{HX,i}(x)^{n(X,H)}$

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 (where each supernode and superfeature is represented by a single symbol), it can greatly affect the cost of constructing the lifted network. In general, finding the most compact representation for supernodes and superfeatures is intractable.

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. A better option is to use a more compact resolution-like representation, as in Poole (2003) and de Salvo Braz, Amir, and Roth (2005; 2006).

Hypercube Representation

The choice of representation of the lifted network can greatly affect the cost of LNC, as well as the memory cost of representation. Even with the resolution-like representation, LNC can be prohibitively expensive. We address this by proposing a new representation for the lifted network.

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. For example, the tuple set $\{(\texttt{X1}, \texttt{Y1}), (\texttt{X1}, \texttt{Y2}), (\texttt{X1}, \texttt{Y3}), (\texttt{X2}, \texttt{Y3}), (\texttt{X2}, \texttt{Y4}), (\texttt{X2}, \texttt{Y5})\}$ can be represented as a union of two hypercubes: $[\{\texttt{X1}\} \times \{\texttt{Y1}, \texttt{Y2}, \texttt{Y3}\}]$ and $[\{\texttt{X2}\} \times \{\texttt{Y3}, \texttt{Y4}, \texttt{Y5}\}]$. Notice that this representation can be exponentially more compact than both the extensional and resolution-like representations.

A *minimal decomposition* of a set of tuples is a decomposition with the minimal number of hypercubes. 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).) We consider two approaches to constructing the hypercubes.

Bottom up. This approach starts with hypercubes for individual tuples and then repeatedly merges pairs of hypercubes that differ from each other only in one argument. The process stops when no such pair can be found. This process is not guaranteed to find a globally minimal decomposition; for instance, in the previous example, it might create the following hypercubes: $[\{\texttt{X1}\} \times \{\texttt{Y1}, \texttt{Y2}\}]$, $[\{\texttt{X1}, \texttt{X2}\} \times \{\texttt{Y3}\}]$, $[\{\texttt{X2}\} \times \{\texttt{Y4}, \texttt{Y5}\}]$. However, the decomposition is locally minimal, in the sense that there are no further merges that can be made.

Top-down. First, a bounding hypercube is constructed, i.e., one which includes all the tuples in the set. For the previous example, the bounding hypercube would be $[\{\texttt{X1}, \texttt{X2}\} \times \{\texttt{Y1}, \texttt{Y2}, \texttt{Y3}, \texttt{Y4}, \texttt{Y5}\}]$. 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 1). Other criteria such as information gain can also be used, as is commonly done in decision trees. The process is continued recursively until we obtain pure hypercubes, i.e., each hypercube either contains all valid tuples or none (in which case it is discarded). When the number of tuples is large, top-down hypercube construction can be faster than bottom-up merging, but it 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 run the bottom-up approach on these to obtain a minimal set.

Algorithm 1 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_{C_1}, ..., r_{C_k})
    C+ ← merge all C_i ∈ C with r_{C_i} > r_C
    C- ← merge all C_i ∈ C with r_{C_i} ≤ 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
```

The join and project operations are now defined in terms of the hypercubes.

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 remaining ones as is. For example, given $\texttt{P}(\texttt{X}, \texttt{Y})$ and $\texttt{Q}(\texttt{Y}, \texttt{Z})$ defined over the singleton hypercube sets $\{[\{\texttt{X1}, \texttt{X2}\} \times \{\texttt{Y1}, \texttt{Y2}\}]\}$ and $\{[\{\texttt{Y1}, \texttt{Y3}\} \times \{\texttt{Z1}, \texttt{Z2}\}]\}$, respectively, the joined hypercube set is $\{[\{\texttt{X1}, \texttt{X2}\} \times \{\texttt{Y1}\} \times \{\texttt{Z1}, \texttt{Z2}\}]\}$. Instead of joining each possible pair of hypercubes, we maintain an index 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. For example, let us say we have superfeatures $\texttt{P}(\texttt{X}, \texttt{Y}) \vee \texttt{Q}(\texttt{Y}, \texttt{Z})$ and $\texttt{P}(\texttt{X}, \texttt{Y}) \vee \texttt{R}(\texttt{Y}, \texttt{Z})$ defined over

the hypercube sets $\{[\{\texttt{X1}, \texttt{X2}\} \times \{\texttt{Y1}\} \times \{\texttt{Z1}, \texttt{Z2}\}]\}$ and $\{[\{\texttt{X1}, \texttt{X2}\} \times \{\texttt{Y1}, \texttt{Y2}\} \times \{\texttt{Z1}, \texttt{Z2}\}]\}$, respectively. Projecting on the predicate $\texttt{P}(\texttt{X}, \texttt{Y})$, the resulting hypercube sets are $\{[\{\texttt{X1}, \texttt{X2}\} \times \{\texttt{Y1}\}]\}$ and $\{[\{\texttt{X1}, \texttt{X2}\} \times \{\texttt{Y1}, \texttt{Y2}\}]\}$, 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. In the above example, the finer set of hypercubes to split into would be $\{[\{\texttt{X1}, \texttt{X2}\} \times \{\texttt{Y1}\}], [\{\texttt{X1}, \texttt{X2}\} \times \{\texttt{Y2}\}]\}$.

Approximate Lifting

Early stopping

The above LNC algorithm 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, in many problems, the minimal exact lifted network is 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. 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 iterations 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 approximation, 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 belief propagation, 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 bounding the additional error introduced at each BP step as a result of the approximation (Ihler, Fisher, and Willsky 2005). The bound can be calculated using a message passing algorithm similar to BP on the level $k+1$ lifted network. The details of the LBP bound calculation are described in Singla, Nath, and Domingos (2010).

Noise-tolerant hypercube formation

This approximation can be used when the hypercube representation is used for supernodes and superfeatures. 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 the previous section, we can derive an error bound by bounding the additional error at each step, but calculating this error bound requires running a message-passing algorithm similar to BP on the exact lifted network.

Experimental Results

We compared the performance of lifted BP (exact and approximate) with the ground version on five real domains and one artificial domain. We implemented lifted BP as an extension of the open-source Alchemy system (Kok et al. 2008). Unless otherwise stated, weights were trained using L-BFGS to optimize pseudo-likelihood (Besag 1975), or using voted perceptron learning (Singla and Domingos 2005). Diagnosing the convergence of BP is a difficult problem; we ran it for 1000 steps for all algorithms in all experiments. See Singla, Nath, and Domingos (2010) for more details of the experiments and datasets.

Datasets

Entity Resolution We used the version of McCallum's Cora database available on the Alchemy website (Kok et al. 2008), and the TF-IDF-based model described by Singla and Domingos (2005). The dataset contains 1295 citations to 132 different research papers. The inference task was to detect duplicate citations, authors, titles and venues.

Advising Relationships This task was to predict advising relationships between students and professors (as described in Richardson and Domingos (2006)), using the UW-CSE database and MLN publicly available from the Alchemy website (Kok et al. 2008), without clauses containing existential quantifiers. The database contains a total of 2678 groundings of predicates.

Protein Interactions We predicted protein interactions in the MIPS Comprehensive Yeast Genome Database (Mewes et al. 2002), using the model and dataset described by Davis and Domingos (2009). The dataset contains four disjoint subsets, each with about 450 proteins.

Hyperlink Analysis The goal of this task was to predict which web pages point to each other, given their topics. We used the relational version of the WebKB dataset (Craven and Slattery 2001), which consists of labeled web pages from the computer science departments of four universities. We used only those web pages for which we had page class information (person, student, faculty, department, research project, course). This contained 1208 web pages and 10063 web links. The MLN contained one rule for each pair of classes, stating states that if a page is of the first class and links to another page, then the linked page is likely to have the second class. An additional rule for stating reflexivity was also incorporated.

Image Denoising Image denoising is the problem of removing noise from an image where some of the pixel values have been corrupted. We used a simple binary image from Bishop (2006, Section 8.3.3), down-sampled to 400 by 400 pixels. We randomly introduced noise in each of the pixels with 10% probability (i.e. noisy pixels were flipped from being in the background to foreground and vice-versa). The MLN contained rules stating that the actual value of a pixel is likely to be same as the observed value (with a weight of 2.1), and that neighbors are likely to have the same pixel value (with a weight of 1.0).

Social Networks We also experimented with the example "Friends and Smokers" MLN from Richardson and Domingos (2006), on a 1000 person network. For a randomly chosen 10% of the network, 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`. See Singla, Nath, and Domingos (2010) for details of how the evidence was generated. We queried all unknown atoms.

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 certain datasets, not all algorithms were compared; these cases will be mentioned specifically in the results section.

We measure conditional log-likelihood (**CLL**) to compare the accuracy of exact and approximate lifted BP.

Results

Table 1 compares time, memory and CLL for the algorithms described above. The reported results are the average over the respective splits described in Singla, Nath, and Domingos (2010). 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 some loss in CLL.

On all datasets besides Denoise, the exact lifted network has a much smaller number of (super)features than the fully ground network; the hypercube representation and both approximations further reduce the number of features and the memory requirements in some domains. 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, 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. We present these results for two datasets, one for each of early stopping (Denoise) and noise tolerance (Yeast). On Denoise, we varied the number of iterations of LNC from 1 to 10 (10 represents the case of running LNC until convergence). Time increases monotonically with increasing number of iterations. Memory usage is almost constant until six iterations, after which it increases monotonically. The almost constant behavior is due to the fact that very few extra superfeatures are created in the first few iterations. CLL converges to the optimum after four iterations; at this stage, time and memory requirements are still very small compared to running LNC until the end.

On the Yeast domain, the noise tolerance was varied from zero to five (zero being the exact case). Time and memory decrease monotonically with increasing noise tolerance, as expected. Increasing noise tolerance does not seem to have any effect on CLL, which stays constant. Singla, Nath, and Domingos (2010) contains additional experimental results, including graphs of time, memory, CLL, and area under the precision-recall curve on Denoise and Yeast.

Conclusions

In this paper, we presented a more compact lifted network representation, and two algorithms for approximate lifted belief propagation. Experiments on a number of datasets show that these algorithms often greatly reduce the time and memory requirements, relative to both ground and exact lifted BP, with little loss in accuracy.

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.

Table 1: Experimental Results

	Algorithm	Time (s)	Mem (MB)	CLL
Cora	Ground	310.2	138	-0.531
	Extensional	63.8	137	-0.531
	Resolution	64.1	138	-0.531
	Hypercube	53.8	110	-0.531
	Early Stop	35.4	108	-0.531
	Noise-Tol.	37.8	102	-1.624
UW-CSE	Ground	503.7	101	-0.022
	Extensional	223.6	193	-0.022
	Resolution	222.9	193	-0.022
	Hypercube	252.1	180	-0.022
	Early Stop	104.6	80	-0.022
	Noise-Tol.	99.8	76	-0.024
Yeast	Ground	1777.9	426	-0.033
	Extensional	81.3	443	-0.033
	Resolution	84.0	443	-0.033
	Hypercube	208.7	354	-0.033
	Early Stop	209.3	354	-0.033
	Noise-Tol.	99.0	304	-0.033
Web-KB	Ground	1346.3	393	-0.036
	Extensional	27.8	285	-0.036
	Resolution	27.9	287	-0.036
	Hypercube	0.8	94	-0.036
	Early Stop	0.8	94	-0.036
	Noise-Tol.	0.8	94	-0.036
Denoise	Ground	4406.6	748	-0.011
	Extensional	4525.3	2202	-0.011
	Resolution	4631.9	2247	-0.011
	Early Stop	34.0	440	-0.064
FS	Ground	5942.2	1873	
	Extensional	163.7	2074	
	Resolution	45.3	1423	
	Hypercube	52.1	1127	
	Early Stop	51.7	1127	
	Noise-Tol.	3.1	1123	

Acknowledgments

This research was partly funded by ARO grant W911NF-08-1-0242, AFRL contract FA8750-09-C-0181, DARPA contracts FA8750-05-2-0283, FA8750-07-D-0185, HR0011-06-C-0025, HR0011-07-C-0060 and NBCH-D030010, NSF grants IIS-0534881 and IIS-0803481, and ONR grant N00014-08-1-0670. 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 ARO, DARPA, NSF, ONR, or the United States Government.

References

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

Besag, J. 1975. Statistical analysis of non-lattice data. *Statistician* 24.

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

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

Craven, M., and Slattery, S. 2001. Relational learning with statistical predicate invention: Better models for hypertext. *Mach. Learn.* 43.

Davis, J., and Domingos, P. 2009. Deep transfer via second-order markov logic. In *Proc. ICML-09*.

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

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

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

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

Ihler, A. T.; Fisher, J. W.; and Willsky, A. S. 2005. Loopy belief propagation: Convergence and effects of message errors. *J. Mach. Learn. Res.* 6.

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

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

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

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

Mewes, H. W.; Frishman, D.; Güldener, U.; Mannhaupt, G.; Mayer, K.; Mokreis, M.; Morgenstern, B.; Münsterkötter, M.; Rudd, S.; and Weil, B. 2002. MIPS: a database for genomes and protein sequences. *Nucleic Acids Res.* 30.

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

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

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

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Mach. Learn.* 62.

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

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

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

Singla, P.; Nath, A.; and Domingos, P. 2010. Approximate lifted belief propagation. Technical report, Univ. of Washington.

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

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.