Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

Lifted First-Order Belief Propagation

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

Abstract

Unifying first-order logic and probability is a long-standing goal of AI, and in recent years many representations combining aspects of the two have been proposed. However, inference in them is generally still at the level of propositional logic, creating all ground atoms and formulas and applying standard probabilistic inference methods to the resulting network. Ideally, inference should be lifted as in first-order logic, handling whole sets of indistinguishable objects together, in time independent of their cardinality. Poole (2003) and Braz et al. (2005, 2006) developed a lifted version of the variable elimination algorithm, but it is extremely complex, generally does not scale to realistic domains, and has only been applied to very small artificial problems. In this paper we propose the first lifted version of a scalable probabilistic inference algorithm, belief propagation (loopy or not). Our approach is based on first constructing a lifted network, where each node represents a set of ground atoms that all pass the same messages during belief propagation. We then run belief propagation on this network. We prove the correctness and optimality of our algorithm. Experiments show that it can greatly reduce the cost of inference.

Introduction

Representations used in AI fall into two broad categories: logical and probabilistic. Their strengths and weaknesses are complementary: first-order logic is best for handling complex relational structure, and probabilistic graphical models for handling uncertainty. AI problems generally contain both, and there have been many proposals to unify the two languages, most recently in the emerging field of statistical relational learning (Getoor & Taskar 2007). Unfortunately, at inference time these approaches typically become purely probabilistic, in the sense that they propositionalize all atoms and clauses and apply standard probabilistic inference algorithms. A key property of first-order logic is that it allows *lifted* inference, where queries are answered without materializing all the objects in the domain (e.g., resolution (Robinson 1965)). Lifted inference is potentially much more efficient than propositionalized inference, and extending it to probabilistic logical languages is a desirable goal.

The only approach to lifted probabilistic inference to date was developed by Poole (2003) and extended by Braz *et al.* (2005; 2006). (Limited lifted aspects are present in some earlier systems, like Pfeffer *et al.*'s (1999) SPOOK.) Poole and Braz *et al.* introduced a lifted version of variable elimination, the simplest algorithm for inference in probabilistic graphical models. Unfortunately, variable elimination has exponential cost in the treewidth of the graph, making it infeasible for most real-world applications. Scalable approximate algorithms for probabilistic inference fall into three main classes: loopy belief propagation (BP), Monte Carlo methods, and variational methods. In this paper we develop a lifted version of BP, building on the work of Jaimovich *et al.* (2007).

Jaimovich *et al.* pointed out that, if there is no evidence, BP in probabilistic logical models can be trivially lifted, because all groundings of the same atoms and clauses become indistinguishable. Our approach proceeds by identifying the subsets of atoms and clauses that remain indistinguishable even after evidence is taken into account. We then form a network with *supernodes* and *superfeatures* corresponding to these sets, and apply BP to it. This network can be vastly smaller than the full ground network, with the corresponding efficiency gains. We show that there is a unique minimal lifted network for every inference problem, and that our algorithm returns it.

Our method is applicable to essentially any probabilistic logical language, including approaches based on Bayesian networks and Markov networks. We will use Markov logic as a concrete example (Richardson & Domingos 2006). Our algorithm is also much simpler than the algorithms of Poole and Braz *et al.* We present the first experimental results for lifted probabilistic inference on real data. These, and systematic experiments on synthetic problems, show that lifted BP can greatly outperform the standard propositionalized version.

Background

Belief Propagation

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*

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Table 1: Example of a Markov logic network. Free variables are implicitly universally quantified.

English	First-Order Logic	Weight
Most people don't smoke.	$\neg\mathtt{Smokes}(\mathtt{x})$	1.4
Most people don't have cancer.	$\neg\mathtt{Cancer}(\mathtt{x})$	2.3
Most people aren't friends.	$\neg\mathtt{Friends}(\mathtt{x},\mathtt{y})$	4.6
Smoking causes cancer.	$\mathtt{Smokes}(\mathtt{x}) \Rightarrow \mathtt{Cancer}(\mathtt{x})$	1.5
Friends have similar smoking habits.	$\mathtt{Smokes}(\mathtt{x}) \wedge \mathtt{Friends}(\mathtt{x},\mathtt{y}) \Rightarrow \mathtt{Smokes}(\mathtt{y})$	1.1

field can have arbitrary factors. As long as $P(\mathbf{X}{=}\mathbf{x}) > 0$ for all x, the distribution can be equivalently represented as a *log-linear model*: $P(\mathbf{X}{=}\mathbf{x}) = \frac{1}{Z}\exp\left(\sum_i w_i g_i(\mathbf{x})\right)$, where the *features* $g_i(\mathbf{x})$ are arbitrary functions of (a subset of) the state.

Graphical models can be represented as *factor graphs* (Kschischang, Frey, & Loeliger 2001). A factor graph is a bipartite graph with a node for each variable and factor in the model. (For convenience, we will consider one factor $f_i(\mathbf{x}) = \exp(w_i g_i(\mathbf{x}))$ per feature $g_i(\mathbf{x})$, i.e., we will not aggregate features over the same variables into a single factor.) Variables and the factors they appear in are connected by undirected edges.

The main inference task in graphical models is to compute the conditional probability of some variables (the query) given the values of some others (the evidence), by summing out the remaining variables. This problem is #P-complete, but becomes tractable if the graph is a tree. In this case, the marginal probabilities of the query variables can be computed in polynomial time by *belief propagation*, which consists of passing messages from variable nodes to the corresponding factor nodes and vice-versa. The message from a variable x to a factor f is

$$\mu_{x\to f}(x) = \prod_{h\in nb(x)\setminus\{f\}} \mu_{h\to x}(x) \qquad (1)$$

where $nb(x)$ is the set of factors x appears in. The message from a factor to a variable is

$$\mu_{f\to x}(x) = \sum_{\sim\{x\}} \left(f(\mathbf{x}) \prod_{y\in nb(f)\setminus\{x\}} \mu_{y\to f}(y) \right) \qquad (2)$$

where $nb(f)$ are the arguments of f, and the sum is over all of these except x. The messages from leaf variables are initialized to 1, and a pass from the leaves to the root and back to the leaves suffices. The (unnormalized) marginal of each variable x is then given by $\prod_{h\in nb(x)} \mu_{h\to x}(x)$. Evidence is incorporated by setting $f(\mathbf{x}) = 0$ for states $\mathbf{x}$ that are incompatible with it. This algorithm can still be applied when the graph has loops, repeating the message-passing until convergence. Although this *loopy* belief propagation 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. Different schedules may be used for message-passing. Here we assume *flooding*, the most widely used and generally best-performing method, in which messages are passed from each variable to each corresponding factor and back at each step (after initializing all variable messages to 1).

Belief propagation can also be used for exact inference in arbitrary graphs, by combining nodes until a tree is obtained, but this suffers from the same combinatorial explosion as variable elimination.

Markov Logic

First-order probabilistic languages combine graphical models with elements of first-order logic, by defining template features that apply to whole classes of objects at once. A simple and powerful such language is *Markov logic* (Richardson & Domingos 2006). A *Markov logic network (MLN)* is a set of weighted first-order clauses.[1] Together with a set of constants representing objects in the domain of interest, it defines a Markov network with one node per ground atom and one feature per ground clause. The weight of a feature is the weight of the first-order clause that originated it. The probability of a state $\mathbf{x}$ in such a network is given by $P(\mathbf{x}) = \frac{1}{Z}\exp\left(\sum_i w_i g_i(\mathbf{x})\right) = \frac{1}{Z}\prod_i f_i(\mathbf{x})$, where w_i is the weight of the ith clause, $g_i = 1$ if the ith clause is true, and $g_i = 0$ otherwise. Table 1 shows an example of a simple MLN representing a standard social network model. In a domain with two objects Anna and Bob, ground atoms will include $\mathtt{Smokes}(\mathtt{Anna})$, $\mathtt{Cancer}(\mathtt{Bob})$, $\mathtt{Friends}(\mathtt{Anna},\mathtt{Bob})$, etc. States of the world where more smokers have cancer, and more pairs of friends have similar smoking habits, are more probable.

Inference in Markov logic can be carried out by creating the ground network and applying belief propagation to it, but this can be extremely inefficient because the size of the ground network is $O(d^c)$, where d is the number of objects in the domain and c is the highest clause arity. In the next section we introduce a better, lifted algorithm for inference. Although we focus on Markov logic for simplicity, the algorithm is easily generalized to other representations. Alternatively, they can be translated to Markov logic and the algorithm applied directly (Richardson & Domingos 2006).

Lifted Belief Propagation

We begin with some necessary definitions. These assume the existence of an MLN $\mathbf{M}$, set of constants $\mathbf{C}$, and evidence database $\mathbf{E}$ (set of ground literals). For simplicity, our definitions and explanation of the algorithm will assume that each predicate appears at most once in any given MLN clause. We will then describe how to handle multiple occurrences of a predicate in a clause.

Definition 1 A *supernode* is a set of groundings of a predicate that all send and receive the same messages at each step

[1] In this paper we assume function-free clauses and Herbrand interpretations.

of belief propagation, given $\mathbf{M}$, $\mathbf{C}$ and $\mathbf{E}$. The supernodes of a predicate form a partition of its groundings.

A *superfeature* is a set of groundings of a clause that all send and receive the same messages at each step of belief propagation, given $\mathbf{M}$, $\mathbf{C}$ and $\mathbf{E}$. The superfeatures of a clause form a partition of its groundings.

Definition 2 A *lifted network* is a factor graph composed of supernodes and superfeatures. The factor corresponding to a superfeature $g(\mathbf{x})$ is $\exp(wg(\mathbf{x}))$, where w is the weight of the corresponding first-order clause. A supernode and a superfeature have an edge between them iff some ground atom in the supernode appears in some ground clause in the superfeature. Each edge has a positive integer weight. A *minimal lifted network* is a lifted network with the smallest possible number of supernodes and superfeatures.

The first step of lifted BP is to construct the minimal lifted network. The size of this network is $O(nm)$, where n is the number of supernodes and m the number of superfeatures. In the best case, the lifted network has the same size as the MLN; in the worst case, as the ground Markov network.

The second and final step in lifted BP is to apply standard BP to the lifted network, with two changes:

1. The message from supernode x to superfeature f becomes $\mu_{f\to x}^{n(f,x)-1} \prod_{h\in nb(x)\setminus\{f\}} \mu_{h\to x}(x)^{n(h,x)}$, where $n(h,x)$ is the weight of the edge between h and x.
2. The (unnormalized) marginal of each supernode (and therefore of each ground atom in it) is given by $\prod_{h\in nb(x)} \mu_{h\to x}^{n(h,x)}(x)$.

The weight of an edge is the number of identical messages that would be sent from the ground clauses in the superfeature to each ground atom in the supernode if BP was carried out on the ground network. The $n(f,x)-1$ exponent reflects the fact that a variable's message to a factor excludes the factor's message to the variable.

The lifted network is constructed by (essentially) simulating BP and keeping track of which ground atoms and clauses send the same messages. Initially, the groundings of each predicate fall into three groups: known true, known false and unknown. (One or two of these may be empty.) Each such group constitutes an initial supernode. All groundings of a clause whose atoms have the same combination of truth values (true, false or unknown) now send the same messages to the ground atoms in them. In turn, all ground atoms that receive the same number of messages from the superfeatures they appear in send the same messages, and constitute a new supernode. As the effect of the evidence propagates through the network, finer and finer supernodes and superfeatures are created.

If a clause involves predicates $R_1, \ldots, R_k$, and $\mathbf{N} = (N_1, \ldots, N_k)$ is a corresponding tuple of supernodes, the groundings of the clause generated by $\mathbf{N}$ are found by joining $N_1, \ldots, N_k$ (i.e., by forming the Cartesian product of the relations $N_1, \ldots, N_k$, and selecting the tuples in which the corresponding arguments agree with each other, and with any corresponding constants in the first-order clause). Conversely, the groundings of predicate R_i connected to elements of a superfeature F are obtained by projecting F onto the arguments it shares with R_i. Lifted network construction thus proceeds by alternating between two steps:

1. Form superfeatures by doing joins of their supernodes.
2. Form supernodes by projecting superfeatures down to their predicates, and merging atoms with the same projection counts.

Table 2: Lifted network construction.

```
function LNC($\mathbf{M}$, $\mathbf{C}$, $\mathbf{E}$)
  inputs: $\mathbf{M}$, a Markov logic network
          $\mathbf{C}$, a set of constants
          $\mathbf{E}$, a set of ground literals
  output: $\mathbf{L}$, a lifted network
for each predicate $P$
  for each truth value $t$ in {true, false, unknown}
    form a supernode containing all groundings of $P$
      with truth value $t$
repeat
  for each clause $C$ involving predicates $P_1, \ldots, P_k$
    for each tuple of supernodes $(N_1, \ldots, N_k)$,
      where $N_i$ is a $P_i$ supernode
      form a superfeature $F$ by joining $N_1, \ldots, N_k$
  for each predicate $P$
    for each superfeature $F$ it appears in
      $S(P, F) \leftarrow$ projection of the tuples in $F$ down to
        the variables in $P$
      for each tuple $s$ in $S(P, F)$
        $T(s, F) \leftarrow$ number of $F$'s tuples that were
          projected into $s$
    $S(P) \leftarrow \bigcup_F S(P, F)$
    form a new supernode from each set of tuples in $S(P)$
      with the same $T(s, F)$ counts for all $F$
until convergence
add all current supernodes and superfeatures to $\mathbf{L}$
for each supernode $N$ and superfeature $F$ in $\mathbf{L}$
  add to $\mathbf{L}$ an edge between $N$ and $F$ with weight $T(s, F)$
return $\mathbf{L}$
```

Pseudo-code for the algorithm is shown in Table 2. The projection counts at convergence are the weights associated with the corresponding edges.

To handle clauses with multiple occurrences of a predicate, we keep a tuple of edge weights, one for each occurrence of the predicate in the clause. A message is passed for each occurrence of the predicate, with the corresponding edge weight. Similarly, when projecting superfeatures into supernodes, a separate count is maintained for each occurrence, and only tuples with the same counts for all occurrences are merged.

Theorem 1 Given an MLN $\mathbf{M}$, set of constants $\mathbf{C}$ and set of ground literals $\mathbf{E}$, there exists a unique minimal lifted network $\mathbf{L}^*$, and algorithm LNC($\mathbf{M}$, $\mathbf{C}$, $\mathbf{E}$) returns it. Belief propagation applied to $\mathbf{L}^*$ produces the same results as belief propagation applied to the ground Markov network generated by $\mathbf{M}$ and $\mathbf{C}$.

Proof. We prove each part in turn.

The uniqueness of $\mathbf{L}^*$ is proved by contradiction. Suppose there are two minimal lifted networks $\mathbf{L}_1$ and $\mathbf{L}_2$. Then there exists a ground atom a that is in supernode N_1 in $\mathbf{L}_1$ and in supernode N_2 in $\mathbf{L}_2$, and $N_1 \neq N_2$; or similarly for some superfeature c. Then, by Definition 1, all nodes in N_1 send the same messages as a and so do all nodes in N_2, and therefore $N_1 = N_2$, resulting in a contradiction. A similar argument applies to c. Therefore there is a unique minimal lifted network $\mathbf{L}^*$.

We now show that LNC returns $\mathbf{L}^*$ in two subparts:

1. The network $\mathbf{L}_i$ obtained by LNC at any iteration i is no finer than $\mathbf{L}^*$ in the sense that, if two ground atoms are in different supernodes in $\mathbf{L}_i$, they are in different supernodes in $\mathbf{L}^*$, and similarly for ground clauses.
2. LNC converges in a finite number of iterations to a network $\mathbf{L}$ where all ground atoms (ground clauses) in a supernode (superfeature) receive the same messages during ground BP.

The claim follows immediately from these two statements, since if $\mathbf{L}$ is no finer than $\mathbf{L}^*$ and no coarser, it must be $\mathbf{L}^*$.

For subpart 1, it is easy to see that if it is satisfied by the atoms at the ith iteration, then it is also satisfied by the clauses at the ith iteration. Now, we will prove subpart 1 by induction. Clearly, it is true at the start of the first iteration. Suppose that a supernode N splits into N_1 and N_2 at the ith iteration. Let $a_1 \in N_1$ and $a_2 \in N_2$. Then there must be a superfeature F in the ith iteration such that $T(a_1, F) \neq T(a_2, F)$. Since $\mathbf{L}_i$ is no finer than $\mathbf{L}^*$, there exist superfeatures F_j in $\mathbf{L}^*$ such that $F = \bigcup_j F_j$. Since $T(a_1, F) \neq T(a_2, F)$, $\exists j \; T(a_1, F_j) \neq T(a_2, F_j)$, and therefore a_1 and a_2 are in different supernodes in $\mathbf{L}^*$. Hence $\mathbf{L}_{i+1}$ is no finer than $\mathbf{L}^*$, and by induction this is true at every iteration.

We prove subpart 2 as follows. In the first iteration each supernode either remains unchanged or splits into finer supernodes, because each initial supernode is as large as possible. In any iteration, if each supernode remains unchanged or splits into finer supernodes, each superfeature also remains unchanged or splits into finer superfeatures, because splitting a supernode that is joined into a superfeature necessarily causes the superfeature to be split as well. Similarly, if each superfeature remains unchanged or splits into finer superfeatures, each supernode also remains unchanged or splits into finer supernodes, because (a) if two nodes are in different supernodes they must have different counts from at least one superfeature, and (b) if two nodes have different counts from a superfeature, they must have different counts from at least one of the finer superfeatures that it splits into, and therefore must be assigned to different supernodes.

Therefore, throughout the algorithm supernodes and superfeatures can only remain unchanged or split into finer ones. Because there is a maximum possible number of supernodes and superfeatures, this also implies that the algorithm converges in a finite number of iterations. Further, no splits occur iff all atoms in each supernode have the same counts as in the previous iteration, which implies they receive the same messages at every iteration, and so do all clauses in each corresponding superfeature.

The proof that BP applied to $\mathbf{L}$ gives the same results as BP applied to the ground network follows from Definitions 1 and 2, the previous parts of the theorem, modifications 1 and 2 to the BP algorithm, and the fact that the number of identical messages sent from the ground atoms in a superfeature to each ground atom in a supernode is the cardinality of the projection of the superfeature onto the supernode. □

Clauses involving evidence atoms can be simplified (false literals and clauses containing true literals can be deleted). As a result, duplicate clauses may appear, and the corresponding superfeatures can be merged. This will typically result in duplicate instances of tuples. Each tuple in the merged superfeature is assigned a weight $\sum_i m_i w_i$, where m_i is the number of duplicate tuples resulting from the ith superfeature and w_i is the corresponding weight. During the creation of supernodes, $T(s, F)$ is now the number of F tuples projecting into s multiplied by the corresponding weight. This can greatly reduce the size of the lifted network. When no evidence is present, our algorithm reduces to the one proposed by Jaimovich *et al.* (2007).

An important question remains: how to represent supernodes and superfeatures. Although this does not affect the space or time cost of belief propagation (where each supernode and superfeature is represented by a single symbol), it can greatly affect the cost of constructing the lifted network. 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 *intensional* representation, as done by Poole (2003) and Braz *et al.* (2005; 2006).[2]

A ground atom can be viewed as a first-order atom with all variables constrained to be equal to constants, and similarly for ground clauses. (For example, $\mathtt{R}(\mathtt{A},\mathtt{B})$ is $\mathtt{R}(\mathtt{x},\mathtt{y})$ with $\mathtt{x} = \mathtt{A}$ and $\mathtt{y} = \mathtt{B}$.) We represent supernodes by sets of (α, γ) pairs, where α is a first-order atom and γ is a set of constraints, and similarly for superfeatures. Constraints are of the form $x = y$ or $x \neq y$, where x is an argument of the atom and y is either a constant or another argument. For example, $(\mathtt{S}(\mathtt{v},\mathtt{w},\mathtt{x},\mathtt{y},\mathtt{z}), \{\mathtt{w} = \mathtt{x}, \mathtt{y} = \mathtt{A}, \mathtt{z} \neq \mathtt{B}, \mathtt{z} \neq \mathtt{C}\})$ compactly represents all groundings of $\mathtt{S}(\mathtt{v},\mathtt{w},\mathtt{x},\mathtt{y},\mathtt{z})$ compatible with the constraints. Notice that variables may be left unconstrained, and that infinite sets of atoms can be finitely represented in this way.

Let the *default value* of a predicate $\mathtt{R}$ be its most frequent value given the evidence (true, false or unknown). Let $\mathbf{S}_{\mathtt{R},i}$ be the set of constants that appear as the ith argument of $\mathtt{R}$ only in groundings with the default value. Supernodes not involving any members of $\mathbf{S}_{\mathtt{R},i}$ for any argument i are represented extensionally (i.e. with pairs (α, γ) where γ contains

[2] Superfeatures are related, but not identical, to the parfactors of Poole and Braz *et al.*. One important difference is that superfeatures correspond to factors in the original graph, while parfactors correspond to factors created during variable elimination. Superfeatures are thus exponentially more compact.

a constraint of the form $\mathtt{x} = \mathtt{A}$, where A is a constant, for each argument x). Initially, supernodes involving members of $\mathbf{S}_{\mathtt{R},i}$ are represented using (α, γ) pairs containing constraints of the form $\mathtt{x} \neq \mathtt{A}$ for each $\mathtt{A} \in \mathbf{C} \setminus \mathbf{S}_{\mathtt{R},i}$.[3] When two or more supernodes are joined to form a superfeature F, if the kth argument of F's clause is the $i(j)$th argument of its jth literal, $\mathbf{S}_k = \bigcap_j \mathbf{S}_{r(j),i}$, where $r(j)$ is the predicate symbol in the jth literal. F is now represented analogously to the supernodes, according to whether or not it involves elements of $\mathbf{S}_k$. If F is represented intensionally, each (α, γ) pair is divided into one pair for each possible combination of equality/inequality constraints among the clause's arguments, which are added to γ. When forming a supernode from superfeatures, the constraints in each (α, γ) pair in the supernode are the union of (a) the corresponding constraints in the superfeatures on the variables included in the supernode, and (b) the constraints induced by the excluded variables on the included ones. This process is analogous to the shattering process of Braz *et al.* (2005).

In general, finding the most compact representation for supernodes and superfeatures is an intractable problem. Investigating it further is a direction for future work.

Experiments

We compared the performance of lifted BP with the ground version on three domains. All the domains are loopy (i.e., the graphs have cycles), and the algorithms of Poole (2003) and Braz *et al.* (2005; 2006) run out of memory, rendering them inapplicable. We implemented lifted BP as an extension of the open-source Alchemy system (Kok *et al.* 2007). Since our algorithm is guaranteed to produce the same results as the ground version, we do not report solution quality. Diagnosing the convergence of BP is a difficult problem; we ran it for 1000 steps for both algorithms in all experiments. BP did not always converge. Either way, it was marginally less accurate than Gibbs sampling. The experiments were run on a cluster of nodes, each node having 3.46 GB of RAM and two processors running at 3 GHz.

Entity Resolution

Entity resolution is the problem of determining which observations (e.g., records in a database) correspond to the same objects. This problem is of crucial importance to many large scientific projects, businesses, and government agencies, and has received increasing attention in the AI community in recent years. We used the version of McCallum's Cora database available on the Alchemy website (Kok *et al.* 2007). The inference task was to de-duplicate citations, authors and venues (i.e., to determine which pairs of citations refer to the same underlying paper, and similarly for author fields and venue fields). We used the MLN (formulas and weights) used by Singla and Domingos (2005) in their experiments. This contains 46 first-order clauses stating regularities such as: if two fields have high TF-IDF similarity, they are (probably) the same; if two records are the same, their fields are the same, and vice-versa; etc.

Link Prediction

Link prediction is an important problem with many applications: social network analysis, law enforcement, bibliometrics, identifying metabolic networks in cells, etc. We experimented on the link prediction task of Richardson and Domingos (2006), using the UW-CSE database and MLN publicly available from the Alchemy website (Kok *et al.* 2007). The database contains a total of 2678 groundings of predicates like: $\mathtt{Student(person)}$, $\mathtt{Professor(person)}$, $\mathtt{AdvisedBy(person1, person2)}$, $\mathtt{TaughtBy(course, person, quarter)}$, $\mathtt{Publication (paper, person)}$ etc. The MLN includes 94 formulas stating regularities like: each student has at most one advisor; if a student is an author of a paper, so is her advisor; etc. The task is to predict who is whose advisor, i.e., the $\mathtt{AdvisedBy(x, y)}$ predicate, from information about paper authorships, classes taught, etc. The database is divided into five areas (AI, graphics, etc.); we trained weights on the smallest using Alchemy's default discriminative learning algorithm, ran inference on all five, and averaged the results.

Social Networks

We also experimented with the example "Friends & Smokers" MLN in Table 1. The goal here was to examine how the relative performance of lifted BP and ground BP varies with the number of objects in the domain and the fraction of objects we have evidence about. We varied the number of people from 250 to 2500 in increments of 250, and the fraction of known people KF from 0 to 1. A KF of r means that we know for a randomly chosen r fraction of all people (a) whether they smoke or not and (b) who 10 of their friends are (other friendship relations are still assumed to be unknown). $\mathtt{Cancer(x)}$ is unknown for all x. The people with known information were randomly chosen. The whole domain was divided into a set of friendship clusters of size 50 each. For each known person, we randomly chose each friend with equal probability of being inside or outside their friendship cluster. All unknown atoms were queried.

Results

Results on all domains are summarized in Table 3. The Friends & Smokers results are for 1000 people and KF = 0.1; the Cora results are for 500 records. All results for Cora and Friends & Smokers are averages over five random splits. [4] LNC with intensional representation is comparable in time and memory with the extensional version on Cora and UW-CSE, but much more efficient on Friends & Smokers. All the results shown are for the intensional representation. LNC is slower than grounding the full network, but BP is much faster on the lifted network, resulting in better times in all domains (by two orders of magnitude on Friends & Smokers). The number of (super) features created is much smaller for lifted BP than for ground BP (by four orders of magnitude on Friends & Smokers). Memory (not reported here) is comparable on Cora and UW-CSE, and much lower for LNC on Friends & Smokers. Figure 1 shows how network size varies with the number of people in the Friends

[3] In practice, variables are typed, and $\mathbf{C}$ is replaced by the domain of the argument; and the set of constraints is only stored once, and pointed to as needed.

[4] For Cora, we made sure that each actual cluster was either completely inside or outside each split.

Table 3: Time and memory cost of ground and lifted BP.

Domain	Time (in seconds)						No. of (Super) Features	
	Construction		BP		Total			
	Ground	Lifted	Ground	Lifted	Ground	Lifted	Ground	Lifted
Cora	263.1	1173.3	12368.4	3997.7	12631.6	5171.1	2078629	295468
UW-CSE	6.9	22.1	1015.8	602.5	1022.8	624.7	217665	86459
Friends & Smokers	38.8	89.7	10702.2	4.4	10741.0	94.2	1900905	58

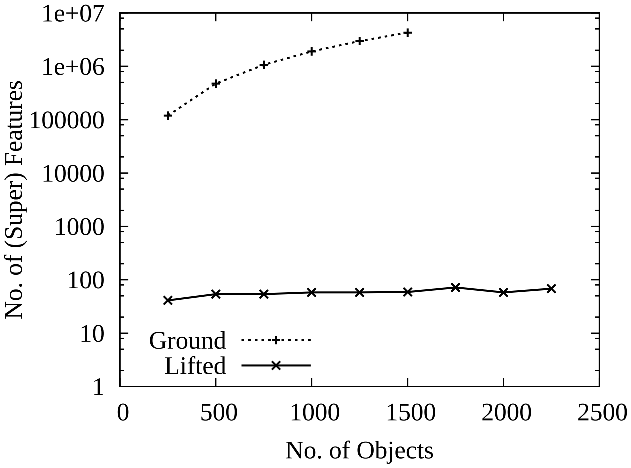

Figure 1: Growth of network size on Friends & Smokers domain.

& Smokers domain, for $KF = 0.1$. The lifted network is always much smaller, and the difference increases markedly with the number of objects (note the logarithmic scale on the Y axis). The ground version ran out of memory for more than 1500 people. We also varied KF while keeping the number of people constant at 1000 (results not shown due to lack of space). The lifted network is always smaller than the ground one by at least four orders of magnitude, rising to six for extreme values of KF.

Conclusion and Future Work

We presented the first scalable algorithm for lifted probabilistic inference, and the first application of these to real-world domains. Our algorithm constructs a network of supernodes and superfeatures, corresponding to sets of nodes and features that are indistiguishable given the evidence, and applies belief propagation to this network. Our experiments illustrate the efficiency gains obtainable by this method.

Directions for future work include: clustering atoms to further compress the representation of supernodes; merging nodes that pass approximately the same messages; generalizing our approach to other inference methods (e.g., MCMC) and tasks (e.g., MPE); fully unifying lifted BP and resolution; applying lifted BP to infinite domains (Singla & Domingos 2007); extending lifted BP to subsume lazy inference (Singla & Domingos 2006) and knowledge-based model construction (Wellman *et al.* 1992); using lifted BP in learning; applying it to other domains; etc.

Acknowledgments

This research was funded by DARPA contracts NBCH-D030010/02-000225, FA8750-07-D-0185, and HR0011-07-C-0060, DARPA grant FA8750-05-2-0283, NSF grant IIS-0534881, and ONR grant N-00014-05-1-0313. 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, or the United States Government.

References

de S. Braz, R.; Amir, E.; and Roth, D. 2005. Lifted first-order probabilistic inference. In *Proc. IJCAI-05*, 1319–1324.

de S. Braz, R.; Amir, E.; and Roth, D. 2006. MPE and partial inversion in lifted probabilistic variable elimination. In *Proc. AAAI-06*, 1123–1130.

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

Jaimovich, A.; Meshi, O.; and Friedman, N. 2007. Template based inference in symmetric relational Markov random fields. In *Proc. UAI-07*, 191–199.

Kok, S.; Sumner, M.; Richardson, M.; Singla, P.; Poon, H.; Lowd, D; and Domingos, P. 2007. The Alchemy system for statistical relational AI. Tech. Rept., Dept. Comp. Sci.& Eng., Univ. Washington, Seattle, WA. 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* 47:498–519.

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

Pfeffer, A.; Koller, D.; Milch, B.; and Takusagawa, K. T. 1999. Spook: A system for probabilistic object-oriented knowledge representation. In *Proc. UAI-99*, 541–550.

Poole, D. 2003. First-order probabilistic inference. In *Proc. IJCAI-03*, 985–991.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62:107–136.

Robinson, J. A. 1965. A machine-oriented logic based on the resolution principle. *Journal of the ACM* 12:23–41.

Singla, P., and Domingos, P. 2005. Discriminative training of Markov logic networks. In *Proc. AAAI-05*, 868–873.

Singla, P., and Domingos, P. 2006. Memory-efficient inference in relational domains. In *Proc. AAAI-06*, 488–493.

Singla, P., and Domingos, P. 2007. Markov logic in infinite domains. In *Proc. UAI-07*, 368–375.

Wellman, M.; Breese, J. S.; and Goldman, R. P. 1992. From knowledge bases to decision models. *Knowledge Engineering Review* 7.