Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)

Efficient Lifting for Online Probabilistic Inference

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 graphical models, but constructing the lifted network can itself be quite costly. In online applications (e.g., video segmentation) repeatedly constructing the lifted network for each new inference can be extremely wasteful, because the evidence typically changes little from one inference to the next. The same is true in many other problems that require repeated inference, like utility maximization, MAP inference, interactive inference, parameter and structure learning, etc. In this paper, we propose an efficient algorithm for updating the structure of an existing lifted network with incremental changes to the evidence. This allows us to construct the lifted network once for the initial inference problem, and amortize the cost over the subsequent problems. Experiments on video segmentation and viral marketing problems show that the algorithm greatly reduces the cost of inference without affecting the quality of the solutions.

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. 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. 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). However, constructing the lifted model can itself be quite costly; sometimes more so than the actual probabilistic inference. This is particularly problematic in online applications, where the lifted network must be constructed for each new set of observations. This can be extremely wasteful, since the evidence typically changes little from one set of observations to the next. The same is true in many other problems that require repeated inference, such as utility maximization, MAP inference (finding the most probable state of a subset of the variables while summing out others), interactive inference, parameter and structure learning, etc.

Another problem with standard lifted inference algorithms is they often yield a model 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 network construction (e.g., de Salvo Braz et al. 2009), which groups together variables that behave similarly, but are not completely identical. The *lifted belief propagation* (LBP) algorithm by Singla and Domingos can also be straightforwardly approximated, by terminating LNC early (Singla, Nath, and Domingos 2010). Approximate lifting can yield a much smaller model, and therefore a greater speedup.

In this paper, we propose ΔLNC, an efficient algorithm for updating the structure of an existing lifted network with incremental changes to the evidence. This allows us to construct the lifted network once for the initial inference problem, and amortize the cost over the subsequent problems. ΔLNC can also be used to efficiently update an approximate lifted network. Experiments on video segmentation and viral marketing problems show that approximate ΔLNC provides great speedups over both ground belief propagation and standard LNC, without significantly affecting the quality of the solutions.

Background

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 prod-

Copyright © 2010, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

uct of factors (Pearl 1988): $P(\mathbf{X}=\mathbf{x}) = \frac{1}{Z}\prod_k \phi_k(\mathbf{x}_k)$, where each factor ϕ_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:

$$\mu_{xf}(x) = \prod_{h \in nb(x)\setminus\{f\}} \mu_{hx}(x)$$

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

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

where $nb(f)$ are the arguments of f, and the sum is over all of these except x. The (unnormalized) marginal of variable x is given by: $\prod_{h\in nb(x)} \mu_{hx}(x)$. 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.

A related task is to infer the most probable values of all variables. The *max-product* algorithm does this by replacing summation with maximization in the factor-to-variable messages.

Lifted Inference

Markov logic (Domingos and Lowd 2009) 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 (*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 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 predicate. 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 ("*children*"). 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.

Pseudo-code for this algorithm is given in 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. (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:

$$\mu_{XF}(x) = \mu_{FX}(x)^{n(F,X)-1} \prod_{H \in nb(X)\setminus F} \mu_{HX}(x)^{n(H,X)}$$

The marginal becomes $\prod_{H\in nb(X)} \mu_{HX}(x)^{n(H,X)}$.

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
      value v
  end for
end for
for all clauses involving predicates (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 new superfeature by joining (X_1, ..., X_k)
  end for
end for
repeat
  for all supernodes X of predicate P do
    for all superfeatures F connected to X 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
  for all superfeatures F involving (P_1, ..., P_k) do
    for all tuples of supernodes (X_1, ..., X_k),
      where X_i is a P_i supernode connected to F do
      Form a new superfeature by joining (X_1, ..., X_k)
    end for
  end for
until convergence
return Lifted network L, containing all current
  supernodes and superfeatures
```

Approximate Lifted Belief Propagation

The LNC algorithm described in the previous section 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; i.e., only refine the supernodes and superfactors $k-1$ times each (Singla, Nath, and Domingos 2010), thus creating k lifted networks $\mathbf{L}_1, \ldots, \mathbf{L}_k$ at increasing levels of refinement. The superfeature counts $n(X, F)$ in the last iteration are averaged over all atoms X.

Each resulting supernode contains nodes that send the same messages during the first k iterations of BP. By stopping LNC after $k-1$ 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 the 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 early can be bounded by calculating the additional error introduced at each BP step as a result of the approximation. The bound can be calculated using a message passing algorithm similar to BP (Ihler, Fisher, and Willsky 2005). Details of the approximation bound are described in Singla, Nath, and Domingos (2010).

Lifted Online Inference

Even with the above approximation, LNC can be expensive. This is particularly problematic in applications involving several queries on a single network, with different values for evidence variables. We refer to this setting as *online inference*. Various algorithms exist for online inference and related problems on graphical models (e.g., Acar et al. (2008), Delcher et al. (1996)), but to our knowledge, no algorithms exist for lifted online probabilistic inference.

In the online setting, it can be extremely wasteful to construct the lifted network from scratch for each new set of observations. Particularly if the changes are small, the structure of the minimal network may not change much from one set of observations to the next. To take advantage of this, we propose an algorithm called ΔLNC. ΔLNC efficiently updates an existing lifted network, given a set of changes to the values of evidence variables.

Given a set Δ of changed atoms, ΔLNC works by retracing the steps these nodes would have taken over the course of LNC. As the network is updated, we must also keep track of which other atoms were indirectly affected by the changes, and retrace their paths as well. At the start of ΔLNC, each changed node x is placed in the initial supernode corresponding to its new value ($true$, $false$, $unknown$), and removed from its previous starting supernode. The next step is to update the superfeatures connected to the starting supernodes. The tuples containing x must be removed from the superfeatures connected to x's original starting supernode. These tuples are then added to matching superfeatures connected to the newly assigned supernode, creating a new superfeature if no match is found.

At the next step, we must also consider the nodes that would have received messages from the factors that changed in the first iteration of BP. These nodes are added to Δ. Now, for each node $x \in \Delta$, we start by removing it from its level two supernode, and then locate a new supernode to insert it into. This is done by calculating the number of tuple projections to x from each of the superfeatures connected to its level one supernode. If the projection counts match any of the children of x's initial supernode, x is added to that child. If not, a new child supernode is created for x.

ΔLNC thus closely mirrors standard LNC, alternately refining the supernodes and the superfeatures. The only difference is that the finer supernodes and superfeatures do not need to be constructed from scratch; only the changed tuples need to be moved. Let $sn_k(x)$ be the supernode containing node x at level k. Pseudocode for ΔLNC is given in

Algorithm 2 ΔLNC(changed nodes Δ, lifted networks $(\mathbf{L}_1, \ldots, \mathbf{L}_k)$)

```
for all nodes x ∈ Δ do
  Remove tuples containing x from sn_1(x) and
    superfeatures connected to it
  Move x to X_new ∈ L_1 of appropriate value v
  for all features t containing x do
    if there exists superfeature F connected to X_new
      from the same clause as t, such that each atom in
      t is in a supernode connected to F then
      Insert t into F
    else
      Create new superfeature F, and insert t
    end if
  end for
end for
for all 2 ≤ i ≤ k do
  Insert ⋃_{x∈Δ} nb(nb(x)) into Δ
  for all nodes x ∈ Δ do
    Remove tuples containing x from superfeatures
      connected to sn_i(x)
    if sn_{i−1}(x) has child X in L_i with projection counts
      n(X, F) = n(x, F) for all F ∈ nb(X) then
      Move x from the previous sn_i(x) to X
    else
      Move x to a newly created child of sn_{i−1}(x)
    end if
    for all features t containing x do
      if there exists superfeature F connected to the new
        sn_i(x) from the same clause as t, such that each
        atom in t is in a supernode connected to F then
        Insert t into F
      else
        Create new superfeature F, and insert t
      end if
    end for
  end for
end for
return updated (L_1, ..., L_k)
```

Algorithm 2. (Like Algorithm 1, this pseudocode assumes for clarity that each predicate occurs at most once in each clause.) See Figure 1 for an example.

Note that the lifted network produced by running k iterations of ΔLNC is identical to that produced by running standard LNC. The error bounds from Singla, Nath, and Domingos (2010) apply to approximate ΔLNC as well. ΔLNC can also be run to convergence to update an exact lifted network.

Experiments

We evaluated the performance of ΔLNC on two problems: video segmentation and viral marketing. The experiments were run on 2.33 GHz processors with 2 GB of RAM.

Video Segmentation

BP is commonly used to solve a variety of vision problems on still images, but it can be very expensive to run BP on videos (e.g., Wang and Cohen (2005)). ΔLNC can be used to make BP on videos much more scalable. We illustrate this by running ΔLNC on a simple binary video segmentation problem. Starting from a complete segmentation of the first frame into background and foreground, we wish to segment the remaining frames. For each frame, pixels whose colors change by less than some threshold keep their segmentation labels from the previous frame as biases for their new labels. We then infer new segmentation labels for all pixels.

Table 1: Results of video stream experiments.

Shuttle takeoff	BP	LBP-4
Initial number of nodes	76800	2992
Initial number of factors	306080	13564
Initial LNC time (s)	-	200.46
Average ΔLNC time (s)	-	6.95
Average BP time (s)	394.46	30.91
Total time (s)	**3900.92**	**558.3**
Cricket	BP	LBP-4
Initial number of nodes	76800	4520
Initial number of factors	306080	19965
Initial LNC time (s)	-	870.69
Average ΔLNC time (s)	-	18.29
Average BP time (s)	398.55	78.99
Total time (s)	**3976.40**	**1619.07**

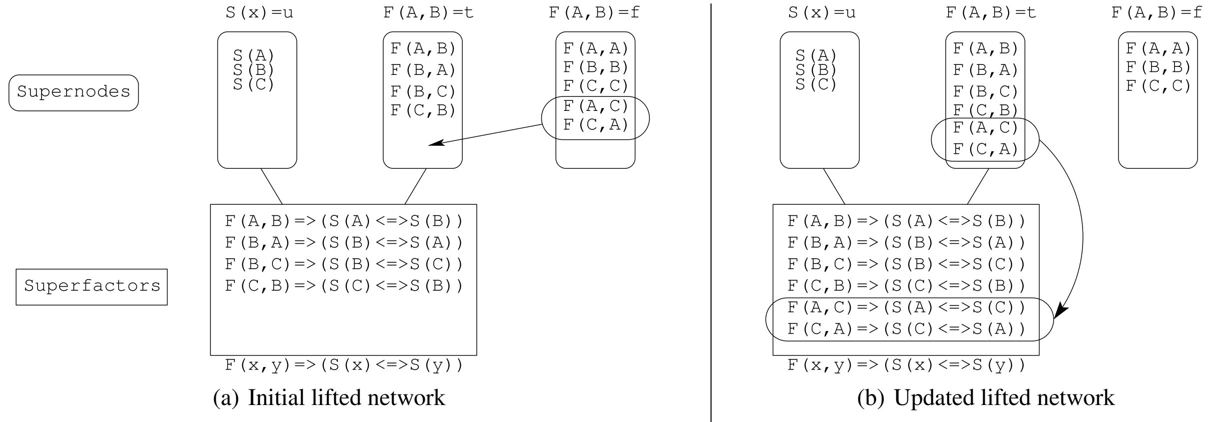
The model is defined by an MLN with predicates $\texttt{KnownLabel(x,l)}$, where $\texttt{l} \in \{\texttt{Background}, \texttt{Foreground}, \texttt{Unknown}\}$; $\texttt{PredictedLabel(x,l)}$; $\texttt{ColorDifference(x,y,d)}$. The formulas are:

$$\texttt{KnownLabel(x,l)} \Rightarrow \texttt{PredictedLabel(x,l)} \tag{1}$$

$$\texttt{ColorDifference(x,y,d)} \Rightarrow (\texttt{PredictedLabel(y,l)} \Leftrightarrow \texttt{PredictedLabel(x,l)}) \tag{2}$$

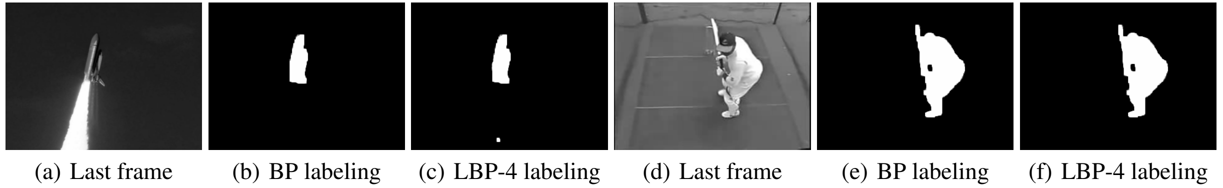
We also introduced deterministic constraints that $\texttt{KnownLabel(x,l)}$ and $\texttt{PredictedLabel(x,l)}$ can only be $true$ for a single $\texttt{l}$, and $\texttt{PredictedLabel(x,Unknown)}$ must be $false$. In our experiments, $\texttt{ColorDifference}$ was calculated as the RGB Euclidean distance, and discretized into six levels. $\texttt{KnownLabel(x,Unknown)}$ was set to $true$ if the difference from one frame to the next was over 30.0. The weight of Formula 1 was 1.0, and the weight of Formula 2 varied inversely with d, up to 2.0 for the smallest difference level.

We ran ΔLNC with four refinement levels (LBP-4), and compared it to ground belief propagation (BP). 1000 steps of max-product BP were performed, to infer the globally most probable labels. We predicted labels for 10 frames of two videos: a space shuttle takeoff (NASA 2010), and a cricket demonstration (Fox 2010). The first frames were segmented by hand. The results are in Table 1. Figure 2 shows the segmentations produced in the last frames. In both experiments, ΔLNC provided dramatic speedups over full ground BP, and produced a similar (but not identical) segmentation. On average, ΔLNC was about 28X faster than the initial LNC on the shuttle video, and 48X faster on the cricket video.



(a) Initial lifted network (b) Updated lifted network

Figure 1: The first iteration of ΔLNC on an MLN with binary predicates $S(x)$ and $F(x)$, and formula $F(x,y) => (S(x) <=> S(y))$. Trivially false superfeatures are omitted. In the above example, all $S(x)$ atoms are $unknown$, and two $F(x,y)$ atoms are changing from $false$ to $true$.



(a) Last frame (b) BP labeling (c) LBP-4 labeling (d) Last frame (e) BP labeling (f) LBP-4 labeling

Figure 2: Output labelings for the final frames of both videos.

Viral Marketing

Viral marketing is based on the premise that members of a social network influence each other's purchasing decisions. The goal is then to select the best set of people to market to, such that the overall profit is maximized by propagation of influence through the network. Originally formalized by Domingos and Richardson (2001), this problem has since received much attention, including both empirical and theoretical results.

A standard dataset in this area is the Epinions "web of trust" (Richardson and Domingos 2002). Epinions.com is a knowledge-sharing website that allows users to post and read reviews of products. The web of trust is formed by allowing users to maintain a list of peers whose opinions they trust. We used this network, containing 75,888 users and over 500,000 directed edges, in our experiments. With over 75,000 action nodes, this is a very large decision problem, and no general-purpose utility maximization algorithms have previously been applied to it (only domain-specific implementations).

We modeled this problem as a Markov logic decision network (MLDN; Nath and Domingos (2009)) using the state predicates $\texttt{Buys(x)}$ and $\texttt{Trusts}(\texttt{x}_1,\texttt{x}_2)$, and the action predicate $\texttt{MarketTo(x)}$. The utility function is represented by the unit clause $\texttt{Buys(x)}$ (with positive utility, representing profits from sales) and $\texttt{MarketTo(x)}$ (with negative utility, representing the cost of marketing). The topology of the social network is specified by an evidence database of $\texttt{Trusts}(\texttt{x}_1,\texttt{x}_2)$ atoms. The core of the model consists of two formulas:

$$\texttt{Buys}(\texttt{x}_1) \wedge \texttt{Trusts}(\texttt{x}_2,\texttt{x}_1) \Rightarrow \texttt{Buys}(\texttt{x}_2) \quad (1)$$

$$\texttt{MarketTo(x)} \Rightarrow \texttt{Buys(x)} \quad (2)$$

The model also includes the unit clause $\texttt{Buys(x)}$ with a negative weight, representing the fact that most users do not buy most products. The weight of Formula 1 represents how strongly $\texttt{x}_1$ influences $\texttt{x}_2$, and the weight of Formula 2 represents how strongly users are influenced by marketing.

Maximizing utility in an MLDN involves finding an assignment of values to groundings of the action predicates (in this case, $\texttt{MarketTo(x)}$) that maximizes the expected utility, which is given by: $E[U(\mathbf{x}|\mathbf{a},\mathbf{e})] = \sum_i u_i \sum_{x \in G_i} P(x|\mathbf{a},\mathbf{e})$. Here, u_i is the utility of formula i, G_i is the set of its groundings, and $P(x|\mathbf{a},\mathbf{e})$ is the marginal probability of grounding x. Finding the optimal choice of actions is an intractable problem, since the search space is very large. We used a greedy search strategy, changing the value of one action at a time and recalculating the marginals using belief propagation. If the change does not prove to be beneficial, it is reversed.

On a lifted network, greedy search involves finding the optimal number of atoms to flip within each $\texttt{MarketTo(x)}$ supernode. In an exact lifted network, all atoms within each supernode would be identical, and the choice of which specific atoms we flip is not relevant. This is not precisely true for approximate lifted networks, but it is a useful approximation to make, since it reduces the number of flips that need to be considered. To find the greedy optimum within

Table 2: Results of viral marketing experiment.

	BP	LBP-2	LBP-1
Initial number of nodes	75888	37380	891
Initial number of factors	508960	484895	201387
Initial LNC time (s)	-	478.59	336.60
Average ΔLNC time (s)	-	0.82	0.68
Average BP time (s)	20.79	20.09	13.41
Total flips	4038	4003	5863
Best utility found	102110	106965	137927
Improvement in utility	**668**	**5523**	**36485**

a supernode, we simply need to start with all its atoms set to $true$ or all its atoms set to $false$, and flip one atom at a time until we make a non-beneficial change (which we then reverse). The decision of whether to start with all atoms set to $true$ or $false$ is also made greedily.

We compared three algorithms: fully ground belief propagation (BP), lifted BP with LNC terminated after one refinement (LBP-1), and lifted BP with two refinements (LBP-2). Formula 1 had a weight of 0.6, and Formula 2 had a weight of 0.8. `Buys(x)` had a weight of -2 and a utility of 20. `MarketTo(x)` had a utility of -1. BP was run for 25 iterations. The starting action choice was to market to no one. The entire MEU inference was given 24 hours to complete, and all algorithms reported the highest utility found within that time. The final solution quality was evaluated with BP on the ground network, with 100 iterations.

The results of the experiment are shown in Table 2. LBP-1 was the only one of the three algorithms to converge to a local optimum within the time limit. ΔLNC achieved approximately a 500X speedup over the initial LNC. ΔLNC also significantly lowers the BP running time, due to the smaller factor graph. LBP yields a much higher utility improvement than ground BP in a comparable number of flips, since each non-beneficial flip on a lifted network tells us that we need not consider flipping other atoms in the same supernode. LBP-1 is competitive in performance with the most directly comparable algorithm of Richardson and Domingos (2002).

Conclusions and Future Work

In this paper, we proposed ΔLNC, an algorithm for efficiently updating an exact or approximate lifted network, given changes to the evidence. ΔLNC works by retracing the path the changed atoms would have taken during LNC, modifying the network as necessary. Experiments on video segmentation and viral marketing problems show that ΔLNC provides very large speedups over standard LNC, making it applicable to a wider range of problems.

There are several directions for future work. ΔLNC can be applied to a variety of problems, including learning and MAP inference. On video problems, it can be made even more scalable by combining it with efficient BP algorithms designed specifically for vision (e.g., Felzenszwalb and Huttenlocher 2006). Using BP for more complicated problems will also require more sophisticated models of inter-frame dependencies (e.g., Isard and Blake 1998). The ability to efficiently update a lifted network may also allow us to lift other probabilistic inference algorithms besides BP.

Acknowledgements

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

Acar, U. A.; Ihler, A. T.; Mettu, R. R.; and Šumer, O. 2008. Adaptive inference on general graphical models. In *Proc. UAI-08*.

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

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

Delcher, A. L.; Grove, A. J.; Kasif, S.; and Pearl, J. 1996. Logarithmic-time updates and queries in probabilistic networks. *J. Artif. Intel. Res.* 4.

Domingos, P., and Lowd, D. 2009. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan Kaufmann.

Domingos, P., and Richardson, M. 2001. Mining the network value of customers. In *Proc. KDD-01*.

Felzenszwalb, P. F., and Huttenlocher, D. P. 2006. Efficient belief propagation for early vision. *Int. J. Comput. Vision* 70.

Fox, E. 2010. This is batting. Retr. Jan 20 from commons.wikimedia.org/wiki/File:This_is_batting.OGG.

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

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.

Isard, M., and Blake, A. 1998. Condensation – conditional density propagation for visual tracking. *Int. J. Comput. Vision* 29.

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

NASA. 2010. Atlantis launches with supplies, equipment for station. Retr. Jan 20 from www.nasa.gov/multimedia/hd/.

Nath, A., and Domingos, P. 2009. A language for relational decision theory. In *Proc. SRL-09*.

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. 2002. Mining knowledge-sharing sites for viral marketing. In *Proc. KDD-02*.

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. To appear.

Wang, J., and Cohen, M. F. 2005. An iterative optimization approach for unified image segmentation and matting. In *Proc. ICCV-05*.