LIFTING ITERATIVE JOIN GRAPH PROPAGATION

by

Srikanth Doss K.R.

APPROVED BY SUPERVISORY COMMITTEE:

Vibhav Gogate, Chair

Haim Schweitzer

Vincent Ng

This thesis is dedicated to my
family and friends. A special thanks
to my parents who supported
me throughout the process.

LIFTING ITERATIVE JOIN GRAPH PROPAGATION

by

SRIKANTH DOSS K.R., BE

THESIS

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER IN SCIENCE IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

August 2014

ACKNOWLEDGMENTS

The author thanks Vibhav Gogate and David Smith for their help and guidance. The author would also like to thank his family and friends for their constant love and support in this entire process.

June 2014

LIFTING ITERATIVE JOIN GRAPH PROPAGATION

Publication No. _____

Srikanth Doss K.R., MS
The University of Texas at Dallas, 2014

Supervising Professor: Vibhav Gogate

Many real-world application domains are both relational and uncertain. Statistical relational models (SRMs) enable us to compactly model these domains by combining the power of first-order (or relational) logic with probabilistic graphical models. A key problem in these models is inference, namely the means of answering queries posed over SRMs. In this thesis, we present a new lifted algorithm, namely an algorithm that treats indistinguishable random variables as one and infers over them in one shot, for scalable inference in SRMs. Our algorithm lifts the iterative join graph propagation (IJGP) algorithm, a state-of-the-art message-passing algorithm for probabilistic graphical models, to the first-order level by taking advantage of relational structure. Our approach is based on the observation that when the SRM is non-shared, IJGP can be elegantly lifted. We utilize this observation by converting the given SRM to a non-shared SRM. Our experimental evaluation on benchmark SRMs clearly shows that lifted IJGP is superior to IJGP not only in terms of time and space cost of inference, but also in terms of convergence.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Probabilistic graphical models, (PGMs) (Pearl, 1988; Darwiche, 2009; Koller and Friedman, 2009) such as Bayesian networks and Markov networks, are a popular framework for representing and reasoning about uncertainty. They exploit conditional independence in order to compactly represent a joint probability distribution over a large number of random variables. They have paved the way to a number of important applications in various domains such as bio-informatics, natural language processing, the World Wide Web and computer vision. One limitation of PGMs is that they are propositional in nature (based on propositional logic) and, as a result, are unable to compactly encode relational knowledge. For example, PGMs can elegantly represent statements such as "If Jack is a friend of Jill and Jill is a friend of James, then Jack and James are likely to be friends too." However, they cannot concisely represent more general knowledge such as "Friendship is transitive with high probability."

Recently, a number of frameworks have been developed that are able to represent PGMs for such multiple-object scenarios in a compact manner. These representations, which are often called statistical relational models (SRMs) (Getoor and Taskar, 2007) or first-order probabilistic models, achieve this by combining PGMs with first-order logic. For example, SRMs can represent the friendship relation using a predicate $\texttt{Friends}(x, y)$, where x and y are arguments that can be substituted with persons such as *Ana*, *Bob*, *David*, and *Rita*.

Among all the statistical relational representations proposed to date, Markov logic networks (MLNs) (Richardson and Domingos, 2006; Domingos and Lowd, 2009) are the most popular one because of their simplicity. MLNs can compactly represent both uncertain and relational knowledge. An MLN is specified by a set of weighted first-order formulas; each

weight takes a real value between $-\infty$ and $+\infty$, which represents the strength of the constraint. A positive (negative) weight represents that the worlds satisfying the formula are more (less) probable than the worlds that do not, all other things being equal. If a weight is zero, the probability of worlds satisfying the formula equals that of the worlds not satisfying the formula (with all other things being equal), making the formula redundant. When the weight equals ∞ (or $-\infty$), the formula (or its negation) represents a hard constraint and worlds not satisfying the formula (or its negation) have a probability of zero. Thus, MLNs generalize first-order logic.

As such, an MLN does not represent a joint probability distribution. It has a probabilistic meaning only when constants or objects are specified. Specifically, given a set of constants, a Markov logic network represents a log-linear model or a Markov network, which has one propositional feature for each grounding (obtained by substituting quantified variables with constants) of each first-order formula, with the weight of the feature being the weight of the corresponding first-order formula.

The key task in Markov logic networks is *inference* - the problem of answering a query posed over the MLN. In principle, we can perform inference over MLNs by first grounding or propositionalizing the MLN, which yields a Markov network, and then by using any probabilistic inference algorithm for PGMs over this Markov network. Examples of such algorithms include variable elimination (Dechter, 1999), loopy belief propagation and its generalizations (Murphy et al., 1999; Yedidia et al., 2005), Gibbs sampling (Geman and Geman, 1984) and importance sampling (Gogate and Dechter, 2005). However, these approaches are impractical because the PGMs obtained by grounding the MLN can be quite large, often having millions of variables and billions of features. Existing inference algorithms for PGMs are unable to handle problems of this scale.

An alternative approach, which has gained prominence since the work of Poole (Poole, 2003) is lifted inference. It is similar to resolution theorem proving (Robinson, 1965), where

inference is performed directly on the first-order representation, grounding only as necessary. Lifted probabilistic inference algorithms can also be understood as probabilistic inference algorithms that infer over groups of indistinguishable random variables together. These indistinguishable groups arise from symmetries in the first-order representation.

Several lifted algorithms have been proposed to date. Prominent exact algorithms include first-order variable elimination (Poole, 2003) and its extensions (de Salvo Braz, 2007), which lift the variable elimination algorithm, and probabilistic theorem proving (PTP) (Gogate and Domingos, 2011), which lifts the weighted model counting algorithm (Chavira and Darwiche, 2008). Notable approximate inference algorithms are lifted Belief propagation (Singla and Domingos, 2008), lifted importance sampling (Gogate et al., 2012), and lifted Gibbs sampling (Venugopal and Gogate, 2012), which lift Belief propagation, importance sampling and Gibbs sampling respectively. Although the general principle of exploiting symmetries is the same, the types of symmetries that can be exploited are different for different algorithms.

In this thesis, we will show how to lift the Iterative Join Graph Propagation (IJGP) algorithm (Mateescu et al., 2010) to the first-order level. IJGP is a state-of-the-art approximation scheme that has won numerous probabilistic inference competitions over the last few years (Elidan and Globerson, 2010, 2011). IJGP is a type of generalized Belief propagation algorithm (Yedidia et al., 2005) and can be understood as an approximate, iterative version of the junction tree propagation algorithm, an exact inference technique. The junction tree algorithm is a message-passing algorithm that operates on a graph structure called the junction tree, which is a tree of clusters of variables that satisfy the *running intersection property*. This property states that if two clusters contain a variable X, then all clusters on the (unique) path between the two clusters should also contain X. The complexity of junction tree propagation is exponential in the largest cluster size. IJGP operates on a graph structure called a join graph, which is a graph (not necessarily a tree) of clusters that satisfies the running intersection property. The complexity of IJGP can be controlled by using

a parameter called the i-bound, an integer which bounds the cluster size (the complexity is exponential in i). The accuracy typically increases with i and thus the i-bound helps us explore the tradeoff between complexity and accuracy. Also, unlike the junction tree propagation algorithm, in which the message-passing can always be made to converge in two steps, the message-passing in IJGP may require multiple iterations to converge, or it may not converge at all.

The core idea in lifting IJGP is to treat all ground nodes that are indistinguishable from each other as one. In particular, if a subset of ground nodes/clusters send and receive the same messages, we treat all nodes in the subset as indistinguishable from each other. We develop techniques that exploit symmetry in the first-order representation for identifying such subsets; we then pass messages for just one representative of each subset. This greatly reduces the time and space cost of inference. Similar to IJGP, we also introduce a parameter p that bounds the lifted cluster size, yielding novel complexity-accuracy trade-offs.

Our experiments show that our new algorithm has much smaller time and space overhead than ground IJGP (which runs on a join graph constructed from the Markov network obtained by grounding the MLN). The gain gets larger as the number of objects (constants in FOL) in the domain increases. In fact, for large domain sizes, the join graph obtained from the ground Markov network is so large that current computers runs out of memory (we ran our experiments on a Linux system having 8 GB RAM). We also observe that the our new lifted algorithm converges faster than the ground algorithm.

The rest of the thesis is organized as follows. In chapter 2, we present our notation and related work. In chapter 3, we present our main contribution – the lifted IJGP algorithm. In chapter 4, we present experimental results. Finally, we conclude in chapter 5 with a brief summary and provide avenues for future work.

CHAPTER 2

BACKGROUND

In this chapter, we will present our notation and give a brief introduction on propositional logic, first-order logic, graphical models, Markov logic and belief propagation. For details, refer to (Darwiche, 2009; Genesereth and Nilsson, 1987; Koller and Friedman, 2009; Domingos and Lowd, 2009).

2.1 Propositional Logic

The language of propositional logic consists of atomic sentences called propositions or atoms, and logical connectives such as $\wedge$ (conjunction), $\vee$ (disjunction), $\neg$ (negation), $\Rightarrow$ (implication) and $\Leftrightarrow$ (equivalence). Each proposition takes values from the binary domain {False, True} (or {0, 1}). A propositional formula f is either an atom or any complex formula that can be constructed from atoms using logical connectives. For example, A, B and C are propositional atoms, and $f = A \vee \neg B \wedge C$ is a propositional formula. A knowledge base (KB) is a set of formulas. A world is a truth assignment to all atoms in the KB.

2.2 First Order Logic

First-order logic (FOL) generalizes propositional logic by allowing atoms to have internal structure; an atom in FOL is a predicate that represents relations between objects. A predicate consists of a predicate symbol, e.g., `Friends`, `Smokes`, etc., followed by a parenthesized list of arguments called terms. A term is a logical variable, denoted by lower case letters such as x, y, z, etc., or a constant, denoted by upper case letters such as X, Y, Z, etc. We assume that each logical variable, e.g. x, is typed and takes values over a finite set Δx .

The language of FOL also includes two quantifiers: $\forall$ (universal) and $\exists$ (existential) which express properties on the entire collection of objects. A formula in first order logic is a predicate (atom), or any complex sentence that can be constructed from atoms using logical connectives and quantifiers. For example, the formula $\forall x$ $\texttt{Smokes}(x) \Rightarrow \texttt{Asthma}(x)$ states that all persons who smoke have asthma. $\exists x$ $\texttt{Cancer}(x)$ states that there exists a person x who has cancer. A first-order KB is a set of first-order formulas.

Throughout the thesis, we use a subset of FOL which has no function symbols, equality constraints or existential quantifiers. We also assume that domains are finite (and therefore function-free) and that there is a one-to-one mapping between constants and objects in the domain (Herbrand interpretations). We make these assumptions in order to ensure a sound and complete algorithm since inference in FOL based languages is undecidable in general. We assume that each formula f is of the form $\forall x f$, where x is a set of variables in f and f is a conjunction or disjunction of literals; each literal being an atom or its negation. For brevity, we will drop $\exists$ from all the formulas. Given variables $x = \{x_1, ..., x_n\}$ and constants $X = \{X_1, ..., X_n\}$ where $X_i \in \Delta x_i,\ f[X/x]$ is obtained by substituting every occurrence of variable x_i in f with X_i . A ground formula is a formula obtained by substituting all of its variables with a constant. A ground KB is a KB containing all possible groundings of all of its formulas. For example, the grounding of a KB containing one formula, $Smokes(x) \Rightarrow Asthma(x)$ where $\Delta x = \{Ana, Bob\}$, is a KB containing two formulas: $Smokes(Ana) \Rightarrow Asthma(Ana)$ and $Smokes(Bob) \Rightarrow Asthma(Bob)$. A world in FOL is a truth assignment to all atoms in its grounding.

2.3 Markov Networks

Probabilistic graphical models such as Bayesian and Markov networks are a graph-based, factored representation of a joint probability distribution, defined over a large number of random variables. In a Bayesian network, the joint distribution is represented by a directed

acyclic graph (DAG). Every node in the DAG corresponds to a random variable that is associated with a conditional probability distribution of the variable given its parents in the DAG. In a Markov network, the joint distribution is compactly represented using an undirected graph, with potential functions defined on the cliques in the graph. The joint distribution represented by a Markov network is defined as the normalized product of the potential functions. In general, at inference time, there is no difference between Bayesian networks and Markov networks. In this thesis, the focus is on Markov networks.

Let $G = (V, E)$ be an undirected graph, where $V = \{1, \ldots, N\}$ is the set of vertices, and $E \subseteq \{(i, j)|i, j \in V; i \neq j\}$ is the set of edges. Let $X = \{X_i|i \in V\}$ denote the set of random variables associated with G and let $\Delta(X_i)$ denote the domain of X_i. Let $\Phi = \{\phi_1, \ldots, \phi_m\}$ be a set of real-valued functions, called potentials, where $\phi_i(X_j, \ldots, X_k) : \Delta(X_i) \times \ldots \times \Delta(X_k) \rightarrow \mathbb{R}_0^+$, and such that the vertex set $\{j, \ldots, k\}$ is a maximal clique in G. Then, the pair $M = \langle G, \Phi \rangle$ forms a Markov network. We denote the set of random variables associated with a potential ϕ_i by vars(ϕ_i).

A Markov network M, induces the following probability distribution:

$$P_M(x) = \frac{1}{Z} \prod_{1 \leq i \leq m} \phi_i(x_i)$$

where x is a full assignment to all variables in X, x_i is the projection of the assignment x on the variables in vars(ϕ_i), and Z is the normalization constant, also called the partition function. Z is given by:

$$Z = \sum_x \prod_{1 \leq i \leq m} \phi_i(x_i)$$

For example, let M be a Markov network defined over the set $X = \{A, B, C, D, E\}$ of variables. Let M contain the following potentials:

1. $\phi_1(A, B)$

2. $\phi_2(B, C, D)$

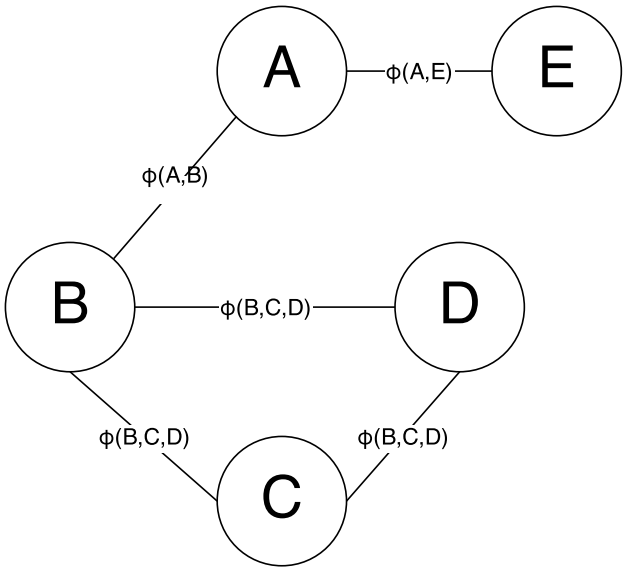

Figure 2.1. The graph associated with a Markov network having three potentials: $\phi(A, B)$, $\phi(B, C, D)$ and $\phi(A, E)$.

Table 2.1. Example potential value for $\phi(A, B)$ where A and B are random variables with domain size 2.

A	B	$\phi(A, B)$
0	0	4
0	1	10
1	0	20
1	1	30

3. $\phi_3(A, E)$

The graph associated with M is shown in Figure 2.1. An example of potential is shown in Table 2.1.

Since a Markov network represents a joint probability distribution, it can be used to answer any query posed over the distribution. Probabilistic inference is the method of answering these queries. The two main inference problems over Markov networks are:

- **Marginal Inference**: computing the (marginal) probability distribution over a subset of variables given evidence, where evidence is an assignment of values to a subset of

variables in the Markov network. For the Markov network given in Figure 2.1, an example of marginal inference query is computing the marginal probability distribution over A given $E = True$.

- **Maximum-a-posterior (MAP) Inference**: computing an assignment of values to all variables that has the highest probability given evidence. For the Markov network in Figure 2.1, an example of MAP inference query is computing an assignment to all variables that has the maximum probability given $E = True$.

Most queries in real-world applications can be reduced to the two inference problems given above. This thesis focuses on the marginal inference problem. Specifically, we focus on computing the marginal distribution over a (single) variable (single variable marginals) given evidence.

2.4 Markov Logic

Markov logic (Richardson and Domingos, 2006; Domingos and Lowd, 2009) extends FOL by softening the hard constraints expressed by the formulas and is arguably the most popular modeling language for statistical relational learning (SRL) (Getoor and Taskar, 2007). A soft formula or a weighted formula is a pair (f, w) where f is a formula in FOL and w is a real-number. A Markov logic network(MLN) is denoted by $M = \langle F, P \rangle$, where $F = \{f_i, w_i\}$ is a set of weighted formulas, P is a set of predicates in the MLN. Given a set of constants that represent objects in the domain, a Markov logic network defines a Markov network or a log-linear model. The Markov network is obtained by grounding the weighted first-order knowledge base and represents the probability distribution in equation 2.1.

$$P_M(\omega) = \frac{1}{Z(M)} \exp\Big(\sum_i w_i N(f_i, \omega)\Big) \tag{2.1}$$

where ω is a world, $N(f_i, \omega)$ is the number of groundings of f_i that evaluate to True in the world ω and $Z(M)$ is the normalization constant or the partition function.

In order to ensure completeness and correctness of our proposed algorithm, we assume that the input MLN (to our algorithm) is in normal form (Jha et al., 2010; Milch et al., 2008). A normal MLN is an MLN that satisfies the following two properties:

1. There are no constants in any formula, and
2. If two distinct atoms with the same predicate symbol have variables x and y in the same position, then $\Delta x = \Delta y$.

Note that in a normal MLN, we assume that the variables in each atom are ordered and therefore we can identify each variable by its position in the order.

For example, consider the following MLN with just one formula: $\Big(R(x) \vee S(y),\ \ w\Big)$. Given two constants Ana and Bob, $\Delta(x) = \Delta(y) = \{Ana, Bob\}$, the MLN yields a ground Markov network having the following weighted propositional formulas:

- $R(Ana) \vee S(Ana) - w$
- $R(Ana) \vee S(Bob) - w$
- $R(Bob) \vee S(Ana) - w$
- $R(Bob) \vee S(Bob) - w$

Since all the ground formulas have the same weight, the corresponding potentials for all the groundings have the same form. One of them is shown in Table 2.2. The ground Markov network is shown in Figure 2.2.

2.5 Factor Graphs

Factor graphs (Kschischang et al., 2001) are special bipartite graphs that allow us to systematically study and understand the properties of sum-product algorithms such as loopy belief propagation (Murphy et al., 1999). A factor graph contains a corresponding vertex

Table 2.2. Example potential value for one grounding of the MLN $\Big(R(x) \vee S(y),\ w\Big)$.

R(Ana)	S(Bob)	$\phi\Big(R(Ana), S(Bob)\Big)$
0	0	e^0
0	1	e^w
1	0	e^w
1	1	e^w

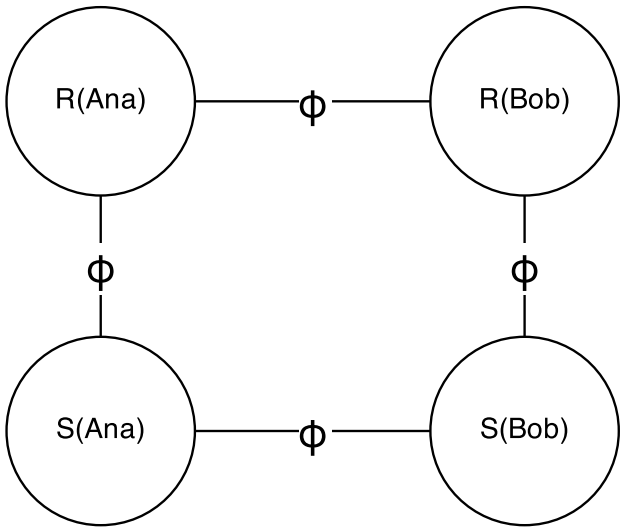

Figure 2.2. The graph structure of ground Markov network consisting of one formula $\Big(R(x) \vee S(y),\ w\Big)$.

for each (random) variable and each potential in the Markov network. A potential vertex is connected (via an undirected edge) to a variable vertex if the the potential contains the variable in its scope. Formally, given a Markov network $M = \langle G = (V, E), \Phi \rangle$, the corresponding factor graph, denoted by $F = (V_F, E_F)$, is an undirected bipartite graph with vertex set $V_F = \{V_i | i \in V\} \cup \{P_i | \phi_i \in \Phi\}$ and edge set $E_F = \{(V_i, P_j) | X_i \in \text{vars}(\phi_j)\}$, where X_i is the random variable corresponding to V_i.

Figure 2.3 shows the factor graph for a Markov network having the following three potentials:

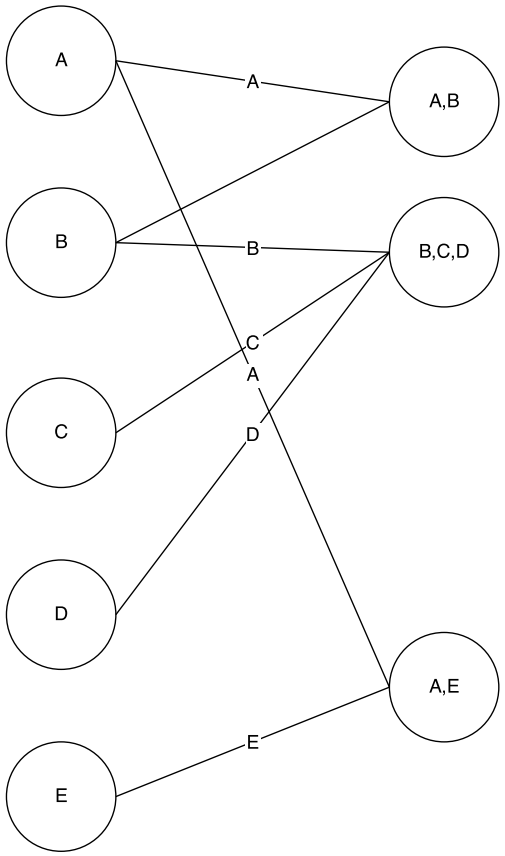

Figure 2.3. The factor graph of the model containing 3 potentials : $\phi(A, B)$, $\phi(B, C, D)$, $\phi(A, E)$.

1. $\phi_1(A, B)$

2. $\phi_2(B, C, D)$

3. $\phi_3(A, E)$

2.6 Join Graphs

Join graphs are graphs over clusters of variables and are a generalization of factor graphs. Given a Markov Network $M = \langle G, \Phi \rangle$, the join graph JG is denoted by $JG = \langle J, \chi, \psi \rangle$,

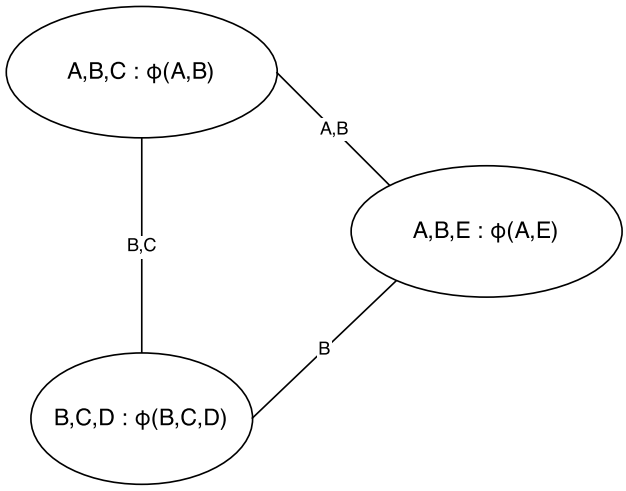

Figure 2.4. A valid join graph of the model containing 3 potentials : $\phi(A, B)$, $\phi(B, C, D)$, $\phi(A, E)$.

where J is a graph $J = (V_J, E_J)$, and χ and ψ are labelling functions which associate with each vertex $v \in V_j$ two sets, $\chi(v) \subseteq X$ and $\psi(v) \subseteq \Phi$ such that,

1. For each potential $\phi_i \in \Phi$, there is at least one vertex $v \in V_j$ such that $\phi_i \in \psi(v)$, and $\text{vars}(\phi_i) \subseteq \chi(v)$.

2. For each variable $X_i \in X$, the set $\{v \in V_j | X_i \in \chi(v)\}$ induces a connected sub-graph of J. The connectedness requirement is also called the *running intersection property.*

Figure 2.4 shows a valid join graph for the factor graph in Figure 2.2.

2.7 Loopy Belief Propagation

Loopy Belief Propagation (Pearl, 1988; Murphy et al., 1999) is a widely used algorithm for performing approximate inference in graphical models. Given a Markov network M, it computes approximations to the set of marginal distributions, $P_M = \{P(X_i)\}_{i \in V}$ by passing messages on the factor graph corresponding to M. At each iteration (of message-passing), each vertex passes a message to each of its neighbors. We denote the message from the

variable vertex V_i to the potential vertex P_j by $\mu_{V_i \to P_j}(x_i)$ and the message from P_j to V_i by $\mu_{P_j \to V_i}(x_i)$. Note that each message is a potential over a single variable. On each iteration it, the messages a node sends are determined by the messages received from its neighbors on the previous iteration $it-1$, as well as by the associated potential ϕ_j if the node is a potential node P_j. They are calculated by the following update equations:

$$\mu^{(it+1)}_{V_i \to P_j}(x_i) \propto \prod_{P_k \in N_F(V_i) \setminus P_j} \mu^{(it)}_{P_k \to V_i}(x_i) \tag{2.2}$$

$$\mu^{(it+1)}_{P_j \to V_i}(x_i) \propto \sum_{\text{vars}(\phi_j) \setminus X_i} \phi(x_i) \prod_{V_k \in N_F(P_j) \setminus V_i} \mu^{(it)}_{V_k \to P_j}(x_k) \tag{2.3}$$

$N_F(V_i)$ and $N_F(P_j)$ are the sets of neighbors of V_i and P_j in the factor graph F respectively.

In practice, the messages are normalized in order to prevent underflow or overflow (hence $\propto$ instead of $=$). The loopy belief propagation algorithm iterates until it reaches some iteration it such that $\forall i, j, x_i \in \Delta(X_i), |\mu^{(it-1)}_{V_i \to P_j}(x_i) - \mu^{(it)}_{V_i \to P_j}(x_i)| < \epsilon$ and similarly, $\forall i, j, x_i \in \Delta(X_i) |\mu^{(it-1)}_{P_j \to V_i}(x_i) - \mu^{(it)}_{P_j \to V_i}(x_i)| < \epsilon$, for some small positive constant $\epsilon > 0$.

At this point, the algorithm is said to have converged and we can calculate the approximations to the single variable marginals (often called *beliefs*) via the following equation:

$$b_i(x_i) = N_i \prod_{P_j \in N(V_I)} \mu^{(it)}_{P_j \to V_I}(x_i) \tag{2.4}$$

where N_i is a normalization constant, such that $\sum_{x_i \in \Delta(X_i)} b_i(x_i) = 1$.

2.8 Iterative Join Graph Propagation

Iterative Join-Graph Propagation (IJGP) (Mateescu et al., 2010) can be perceived as a generalization of the loopy belief propagation algorithm. It applies the same message passing as loopy belief propagation to join graphs rather than to factor graphs. The messages are no longer defined over a single variable, but are defined over the variables in the label attached to the edge between the two clusters. IJGP(i) takes a parameter called an i-bound as input, which bounds the number of variables in a cluster. Since the complexity of computing each

message is exponential in the cluster size, the time and space complexity of one full iteration of IJGP(i) is exponential in i times the number of clusters (which is typically linear in the number of variables and potentials). It has been observed that the accuracy of IJGP typically increases with i. Therefore, the parameter i helps us explore the trade-off between time and accuracy.

IJGP may or may not converge and it is known that when IJGP converges, its accuracy is typically good. The convergence of IJGP can be improved by making the join graph arc-minimal. Arc-minimal join graphs are join graphs from which no variable can be deleted from any label without violating the running intersection property. When the join graph is a tree, the algorithm always converges in a finite number of iterations. In fact, there exists a message passing schedule such that the algorithm converges in one iteration (two passes). Moreover, when the join graph is a tree, IJGP yields exact answers and is equivalent to the junction tree propagation algorithm (Lauritzen and Spiegelhalter, 1988), a popular exact inference algorithm.

2.9 Lifted Belief Propagation

Lifted Belief Propagation(LBP) (Singla and Domingos, 2008) is an approximate algorithm that extends the loopy belief propagation to MLNs. LBP does this by constructing supernodes and superfeatures, which are lifted representations of variable nodes and factor nodes respectively.

A supernode is a set of groundings of a predicate that send and receive the same messages at each step of LBP, given an MLN. The supernodes of a predicate form a partition of its groundings. A superfeature is a set of groundings of a clause that send and receive the same messages at each step of belief propagation, given an MLN. The superfeatures of a clause form a partition of its groundings.

A lifted network is a factor graph composed of supernodes and superfeatures. The factor corresponding to a superfeature g(x) is exp(wg(x)), where w is the weight of the corresponding first-order clause. A supernode and a superfeature have an edge between them if and only if some ground atom of the supernode appears in some ground clause of 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 messages sent in LBP are similar to those sent in standard BP, with the only following distinctions:

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) is given by $\prod_{h\in nb(x)} \mu_{h\to x}^{n(h,x)}(x)$.

This lifted network simulates propositional LBP. The messages in BP are over a single predicate. LBP has similar convergence properties as loopy BP. In cases where it converges it is often faster than loopy BP.

CHAPTER 3

LIFTED ITERATIVE JOIN GRAPH PROPAGATION

3.1 Introduction

IJGP works well on propositional models; it is a state-of-the-art algorithm that has achieved superior results on many of the standard benchmark propositional models, as demonstrated by its various UAI competition victories. However, propositional models lack the compactness of MLNs. We propose a lifted IJGP algorithm, which operates as much as possible on the first-order representation, grounding the MLN only as necessary. In earlier work, (Singla and Domingos, 2008) lifted the loopy belief propagation algorithm to the first-order level. However, their results were not directly applicable to IJGP and our goal in this thesis is to fill in this need. Figure 3.1 shows relationship between the various message passing algorithms.

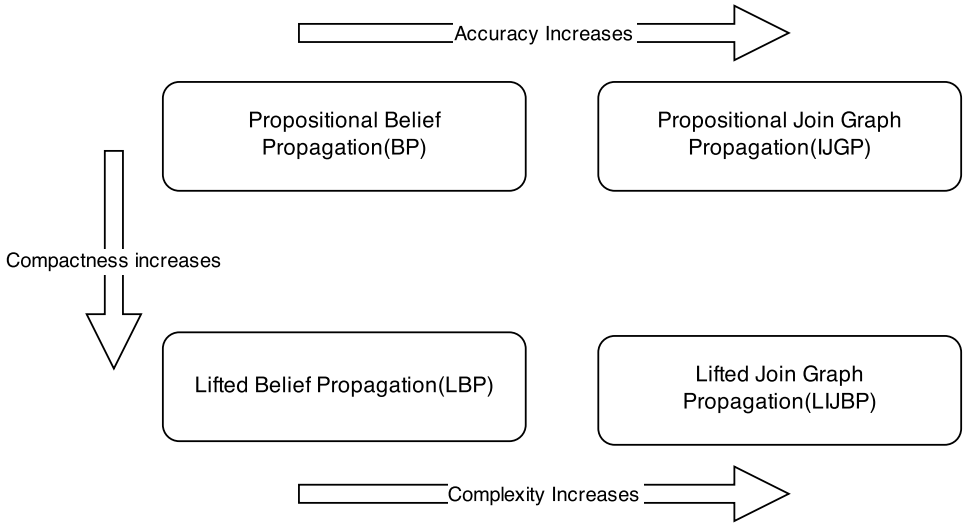

Figure 3.1. Relationship between the various message passing algorithms.

3.2 MLN Representation and Normal Forms

In our algorithm, we assume that the input MLN is a normal MLN(Jha et al., 2010; Milch et al., 2008). Formally, a *normal* MLN is an MLN that satisfies the following two properties:

1. There are no constants in any formula.
2. If two distinct atoms with the same predicate symbol have variables x and y in the same position then $\Delta_x = \Delta_y$.

For example, consider an MLN having two formulas: $(\texttt{Smokes}(x) \Rightarrow \texttt{Asthma}(x), w)$ and $(\texttt{Smokes}(Ana), \infty)$ (evidence). It is not in normal form. Its normal form has three formulas: $(\texttt{Smokes}(x') \Rightarrow \texttt{Asthma}(x'), w)$, $(\texttt{Smokes1}(y) \Rightarrow \texttt{Asthma1}(y), w)$ and $(\texttt{Smokes1}(y), \infty)$, where $\Delta'_x = \Delta_x \setminus \{Ana\}$ and $\Delta_y = \{Ana\}$.

Also, following (Sarkhel et al., 2014), we enforce an additional constraint that the clauses must not have any shared variables. This constraint helps us to ensure that we can process each cluster in polynomial time, irrespective of the domain size. Formally, we enforce the following constraint:

$$\forall f_i \in F\Big(vars_{f_i}(p_i) \cap vars_{f_i}(p_j) = \emptyset | i \neq j; p_i, p_j \in preds(f_i)\Big)$$

where $vars_{f_i}(p_i)$ is the variables of predicate p_i in the formula f_i, $preds(f_i)$ is the predicates of f_i, and F is the set of formulas in the MLN.

For example, the following clauses satisfy the above constraint.

- $R(x) \vee S(y)$
- $R(x) \vee S(y) \vee T(z)$

whereas the following do not.

- $R(x) \vee S(x)$ (x is shared)

- $R(y) \vee S(x, y)$ (y is shared)

Note that our approach can be applied to general MLNs by simply grounding all shared terms in the MLN. For example, consider an MLN having one formula $R(x) \vee S(x, y),\ where\ \Delta_x = \Delta_y = \{Ana, Bob, Cat\}$. We can convert this MLN to a non-shared MLN by grounding x. The new MLN will have the following three clauses:

- $A(x) \vee D(y),\ where\ \Delta_x = \{Ana\}, \Delta_y = \{Ana, Bob, Cat\}, R \equiv A, D(y) \equiv S(Ana, y)$
- $B(x) \vee E(y),\ where\ \Delta_x = \{Bob\}, \Delta_y = \{Ana, Bob, Cat\}, R \equiv B, E(y) \equiv S(Bob, y)$
- $C(x) \vee F(y),\ where\ \Delta_x = \{Cat\}, \Delta_y = \{Ana, Bob, Cat\}, R \equiv C, F(y) \equiv S(Cat, y)$

In the above MLN, A, B and C are equivalent to R(A), R(B) and R(C) respectively, and D, E and F are equivalent to $S(A, y)$, $S(B, y)$ and $S(C, y)$ respectively.

3.3 Lifted Join Graphs

Our lifted IJGP algorithm operates on a graph structure, which is called a lifted join graph. Lifted join graphs are similar to join graphs with a few differences. Nodes in lifted join graphs are defined over first-order atoms and thus correspond to multiple ground atoms. Edges in join graphs are labeled with propositional atoms, whereas the edges in lifted join graphs are labeled with first order atoms. Moreover, lifted join graphs can contain self loops; namely, a node can send a message to itself. Self-loops are described later in this chapter.

Formally, given an MLN M, we denote the lifted join graph LJG by $\langle J, \chi \rangle$, where J is a graph $J = (V, E)$, V is the vertex set, E is the edge set and χ is a set of labelling functions which associates with each vertex $v \in V$ with the set, $\chi(v) \subseteq P$, P is the set of all first-order atoms in the MLN, such that,

1. For each formula $f_i \in F$, there is atleast one vertex $v \in V$ such that $preds(f_i) \subseteq \chi(v)$.

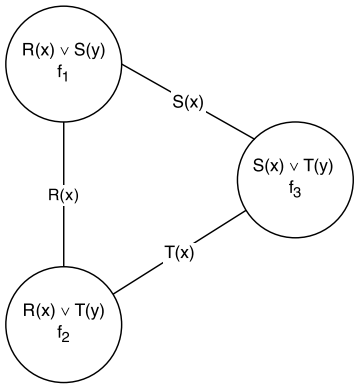

Figure 3.2. A valid lifted join graph for the MLN : $R(x) \vee S(y), R(x) \vee T(y), S(x) \vee T(y)$.

2. For each predicate $p_i \in P$, the set $\{v \in V | p_i \in \chi(v)\}$ induces a connected sub graph of J. The connectedness requirement is also called the running intersection property.

Figure 3.2 shows a valid lifted join graph for the following MLN:

- $f_1 : \Big(R(x) \vee S(y), w_1\Big)$
- $f_2 : \Big(R(x) \vee T(y), w_2\Big)$
- $f_3 : \Big(S(x) \vee T(y), w_3\Big)$

Each of the clusters are connected by the shared predicate and they send messages over the shared predicates.

3.4 Constructing the Lifted Join Graph

Given an MLN, the goal of the construction algorithm is to find a lifted join graph J such that:

- J is of minimal size.
- Each cluster $v_i \in J$ contains groundings that are indistinguishable w.r.t the model (and thus propagation can be performed in a lifted manner over those groundings).

- There exists a corresponding valid ground join graph upon which propagation is equivalent to the propagation performed on J.

Construction of a lifted join graph can be done with any method that works on the propositional case. The only condition is that the running intersection property must hold for all the predicates. In the propositional case, the clusters in the join graph are connected on the random variables whereas in the lifted case they join on the predicates. Here, we use an algorithm similar to the bucket elimination(Dechter, 1999), to create the lifted join graph. The algorithma assures that the number of unique predicates in the cluster is bound by an integer parameter p called the p-bound. p helps us trade-off time and space with accuracy. Algorithm 1 gives the method to construct the lifted join graph.

For the following MLN, Figure 3.3 shows the lifted join graph with a p-bound of 2 and Figure 3.4 shows the lifted join graph with a p-bound of 3:

- $f_1 : \Big(R(x) \vee S(y), w_1\Big)$
- $f_2 : \Big(R(x) \vee T(y), w_2\Big)$
- $f_3 : \Big(S(x) \vee T(y), w_3\Big)$

3.4.1 Self loops

Each node in the lifted join graph that the algorithm (see Algorithm 1) builds is a representation of multiple ground nodes. These ground nodes can either be completely independent or have internal structure. In the latter case, we need self loops. For example, consider the formula $R(x), \Delta(x) = \{Ana, Bob, Cat\}$. Here, all nodes in the completely grounded graph are independent of each other. Figure 3.5 shows the ground join graph corresponding to this formula. Figure 3.6 shows the corresponding lifted join graph. Figure 3.7 shows the join graph for the formula $R(x) \vee S(y)$, $\Delta(x) = \Delta(y) = \{Ana, Bob, Cat\}$. The ground join graph

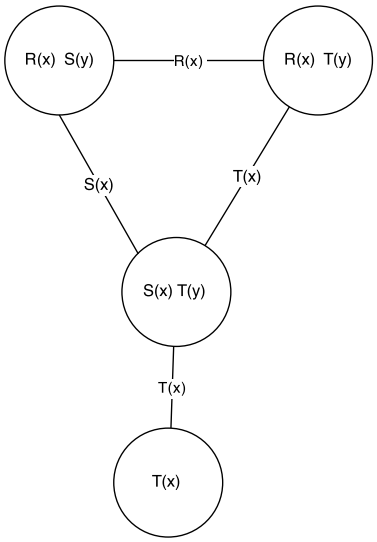

Figure 3.3. The lifted join graph with p-bound = 2.

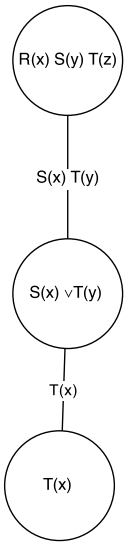

Figure 3.4. The lifted join graph with p-bound = 3.

input : An MLN M, an ordering of predicates $p_1..p_k$, Bound p on the number of unique predicates present in a cluster.
output: A lifted join graph J

Let X_i be the bucket corresponding to p_i;
Place all the $f \in F$ in appropriate bucket X_i.

for $i \leftarrow 1$ **to** k **do**
  if *(number_of_unique_predicates(X_i) $\leq p$)* **then**
    $f_{new} \Leftarrow union(X_i)$
    $\phi \Leftarrow \prod w \in X_i$
    Create a node V_{new} with f_{new} and ϕ.
    $V \Leftarrow V \cup V_{new}$
    else
    Create mini-clusters C such that $number\_of\_unique\_predicates \leq p$
    $V \Leftarrow V_{new} \cup C$
    Connect all the nodes in C with edge p_i
  end
  Eliminate p_i from X_i and place it in the correct bucket in the ordering.
end

Algorithm 1: Algorithm to build the lifted join graph

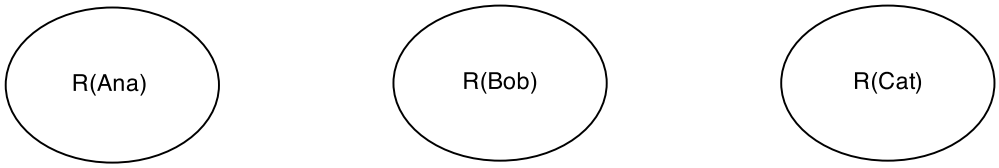

Figure 3.5. The ground join graph of the MLN containing one formula: R(x), $\Delta(x) = \{Ana, Bob, Cat\}$.

has nodes that send messages over the groundings of $R(x)$ and $S(y)$. Figure 3.8 shows the lifted join graph. Here, we need the self loops to represent the fact that the ground network has an internal structure as shown in Figure 3.7.

In the lifted join graph, we have to make sure that such messages in the ground join graph are accounted for. A node needs to have a self loop on a predicate p_i if none of the

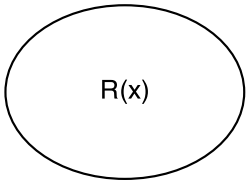

Figure 3.6. The lifted join graph of the MLN containing one formula: R(x), $\Delta(x) = \{Ana,$ $Bob, Cat\}$.

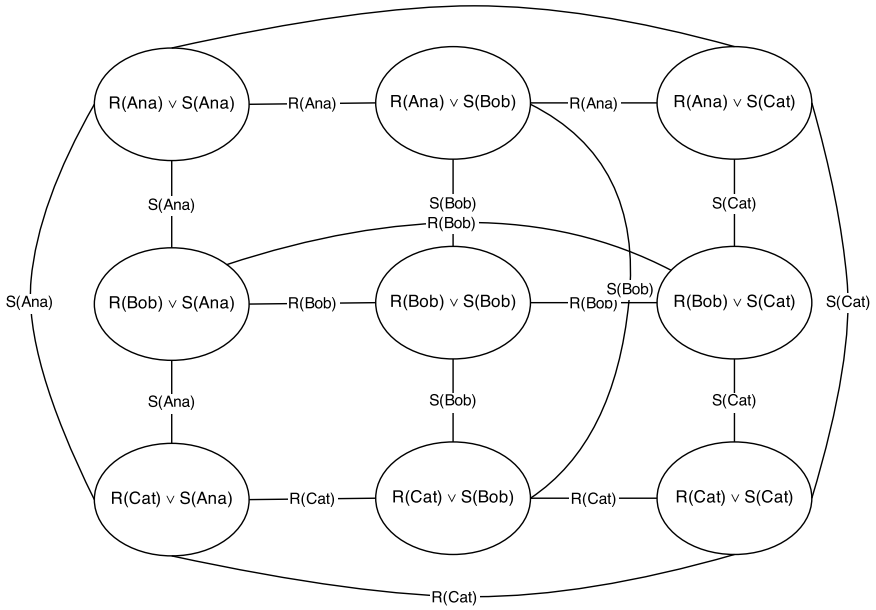

Figure 3.7. The ground join graph of the MLN containing one formula: $R(x) \vee S(y)$, $\Delta(x) = \Delta(y) = \{Ana, Bob, Cat\}$.

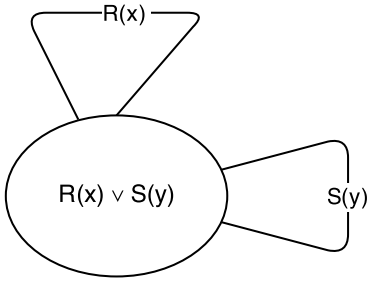

Figure 3.8. The lifted join graph of the MLN containing one formula: $R(x) \vee S(y)$, $\Delta(x) = \Delta(y) = \{Ana, Bob, Cat\}$.

edges of the node contains p_i. If there is an edge containing p_i then it makes sure that the messages are sent over the groundings of p_i.

Since we have as input to the algorithm an MLN which has the constraint *that the clauses have no shared variables*, a node $v \in V$ has a self loop on $p_i \in preds(v)$ iff $p_i \notin edges(v)$.

3.4.2 Processing the evidence

Handling evidence is the trickiest step of our proposed algorithm. The main idea in constructing a lifted join graph is to exploit symmetry; i.e. we wish to make sure that similar nodes are present in a single cluster. This is a necessary condition, since all the ground nodes within the cluster receive and send the same messages. Evidence breaks symmetry, and therefore the nodes split with each evidence to make sure that a cluster always contains indistinguishable ground nodes.

We process the evidence by processing the first-order atoms one by one. For each atom which has evidence, we split its nodes into True, False and Unknown groundings, similar to the method described in (Singla and Domingos, 2008), connect it appropriately to yield a new lifted join graph, repeat the steps until all atoms having evidence are processed. Algorithm 2 formally describes our method.

Figure 3.9 shows how the algorithm splits the join graph given in Figure 3.8 into two clusters. The evidence is on $R(x)$. $R(x)$ is present in two clusters and therefore we need to split both the nodes. In the new join graph, P contains all the True groundings of R, Q contains all the False groundings of R and R contains all the Unknown groundings of R.

3.5 Message Passing

Given an MLN M, lifted iterative join graph propagation is an iterative algorithm that computes approximations to the set of marginals for all groundings in the clusters. It does so by passing messages on the lifted join graph J corresponding to M. In every iteration,

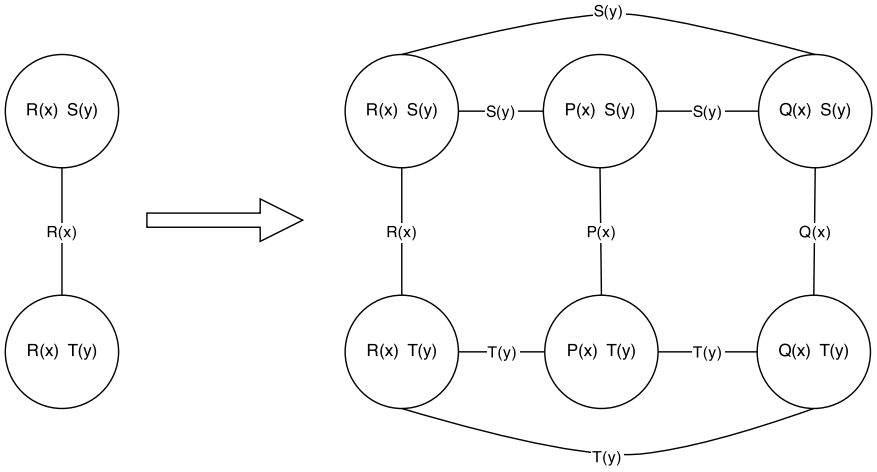

Figure 3.9. The evidence is on $R(x)$. $P(x)$, $Q(x)$ and $R(x)$ are the True, False and Unknown groundings of $R(x)$ respectively.

input : A Lifted Join graph J, The MLN M, Predicate evidences set $S|S_i = \{S_i^U, S_i^T, S_i^F\}$ which corresponds to the Unknown, True and False groundings of predicate p_i
output: Evidence instantiated join graph J

for *each* $S_i \in S$ **do**
  $L_i \leftarrow$ List of nodes containing p_i
  $n \leftarrow$ number of nodes containing p_i
  for *each* $j \leftarrow 1$ *to* n **do**
    Split the groundings of the in node $L_i[j]$ into 3 nodes $L_i[j]^U$, $L_i[j]^T$, $L_i[j]^F$ containing the groundings $S_i^U,\ S_i^T,\ S_i^F$
    Join the newly added nodes $L_i[j]^U$, $L_i[j]^T$ and $L_i[j]^F$ with the edge having predicates $\{preds(L_i[j])$ - $p_i\}$
    Add edges between the $L_i[j]^U$ to nodes $L_i[k]^U | k < j$
    Add edges between the $L_i[j]^T$ to nodes $L_i[k]^T | k < j$
    Add edges between the $L_i[j]^F$ to nodes $L_i[k]^F | k < j$
  end
end

Algorithm 2: Algorithm to split the lifted join graph

each node passes a message to each of its neighbors. We denote the message from node V_i to a node V_j by $\mu_{V_i \to V_j}(P)$, where P is the set of shared predicates between V_i and V_j. On each iteration it, the messages a node sends are determined by the messages it receives from its neighbors from the previous iteration $it-1$, as well as by the associated potential ϕ_i for the node V_i. They are calculated by the following update equations:

$$\mu^{(it+1)}_{V_i \to V_j}(P) \propto \Big(\sum_{preds(\phi_i) \setminus P} \; \prod_{V_k \in N_J(V_i) \setminus V_j} \mu^{(it)}_{V_k \to V_i}(preds(V_k)) \Big)^n \tag{3.1}$$

$N_F(V_i)$ and $N_F(V_j)$ are the sets of neighbors of V_i and V_j in the lifted join graph J respectively. n is the exponentiation of the message. This is the critical part of the algorithm where the compactness is achieved. Instead of sending n messages, we accommodate for it by exponentiating the message to n. The process of computing n is explained in detail in the next section.

In practice, the messages are normalized in order to prevent underflow or overflow (hence $\propto$ instead of $=$). The propagation algorithm iterates until it reaches some iteration it such that $\forall i, j, V_i \in V | \mu^{(it-1)}_{V_i \to V_j}(P) - \mu^{(it)}_{V_i \to V_j}(P)| < \epsilon$, for some small positive constant $\epsilon > 0$.

At this point, the algorithm is said to have converged and one can calculate the approximations to the single predicate marginals.

3.5.1 Exponentiation

When we send a message from V_i to V_j, we need to account for the fact that there are multiple ground nodes within each cluster. Specifically, we have to exponentiate the message sent from node V_i to V_j by n, computed using the following expression:

$$n = \prod_{p_k \in preds(V_i) \setminus preds(V_j)} N(p_k)$$

where $N(p_k)$ is number of grounding of predicate p_k in V_i.

Figure 3.10 and Figure 3.11 show the intuitive idea behind exponentiation and how it saves time and space.

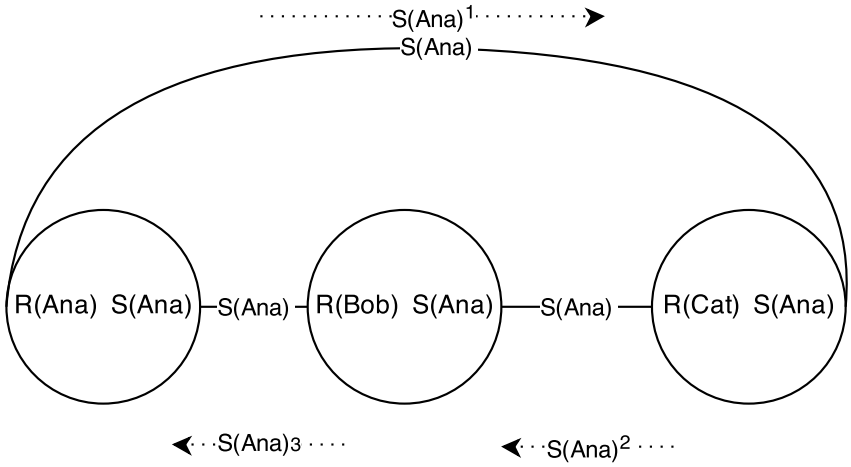

Figure 3.10. The ground join graph of the model $S(x) \vee R(y)$ where $\Delta(x) = \{Ana, Bob, Cat\}$ and $\Delta(y) = \{Ana\}$.

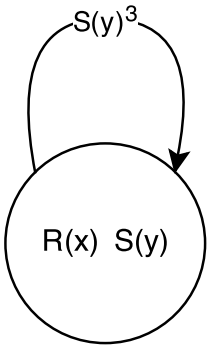

Figure 3.11. The lifted join graph of the model $S(x) \vee R(y)$ where $\Delta(x) = \{Ana, Bob, Cat\}$ and $\Delta(y) = \{Ana\}$.

As we can see in Figure 3.10, all the ground nodes are the same and all of them send and receive the same number of messages. Moreover, all the nodes join on $S(Ana)$ and they all have same potentials in the node. So instead of sending the same message n times, we can exponentiate it by n as shown in Figure 3.11.

CHAPTER 4

EXPERIMENTS

4.1 Setup

We evaluated our lifted IJGP algorithm on the following three benchmark MLNs. These MLNs are available on the Alchemy website (Kok et al., 2008), an open source software for inference and learning in Markov logic networks.

1. Student MLN (having the following four formulas):

   Teaches(teacher,course) $\wedge$ Takes(student,course) $\rightarrow$ JobOffers(student,company)

   Teaches(teacher,course)

   Takes(student,course)

   JobOffers(student,company)

2. WebKB MLN, consisting of three predicates and six formulas.

3. Social Network Friends and Smokers MLN, consisting of three predicates and five formulas.

The experiments were done by varying the domain sizes with the following parameters: p-bound = 3 and $\epsilon = 10^{-5}$. We recorded the runtime for the construction of the lifted join graph and the run time for convergence. Evidence was randomly generated for 10% of the groundings for each predicate.

Table 4.1. Without Evidence : N1 is the number of nodes in the ground join graph, N2 is the number of nodes in the lifted join graph. T1 is the time to construct the join graph(seconds). T2 is the time taken for the algorithm to converge(seconds).

MLN	Domain	N1	N2	T1	T2
Student	3	81	9	0.05	0.11
Student	10	10000	100	0.1	0.158
Student	20	160000	400	0.987	0.5
Student	30	810000	900	3.119	1.360
WebKB	3	138	39	0.124	0.679
WebKB	10	11310	1410	4.467	11.640
WebKB	20	169220	11834	409.891	213.192
Friends and Smokers	3	54	19	0.088	0.082
Friends and Smokers	10	460	240	0.43	0.181
Friends and Smokers	20	1720	999	2.394	0.57

4.2 Results

Tables 4.1 and Table 4.2 show our results for each of the three MLNs, with evidence and without evidence respectively. We can see that in general, the number of nodes that the lifted IJGP algorithm has to process is significantly smaller than the number of nodes that the ground IJGP algorithm has to process. Moreover, the ratio between the two decreases significantly with the domain size. Although evidence does increase the number of nodes in the lifted join graph, this number is still substantially smaller than the number of nodes in the ground join graph. In summary, our experiments clearly demonstrate the power of our new lifted algorithm; it is substantially superior to the ground algorithm, often by several orders of magnitude.

Table 4.2. With Evidence : N1 is the number of nodes in the ground join graph, N2 is the number of nodes in the lifted join graph. T1 is the time to construct the join graph(seconds). T2 is the time taken for the algorithm to converge(seconds).

MLN	Domain	N1	N2	T1	T2
Student	3	81	9	0.206	0.173
Student	10	10000	1500	1.034	11.976
Student	20	160000	8720	10.87	11.523
WebKB	3	138	39	0.148	0.593
WebKB	10	11310	4410	4.467	11.640
WebKB	20	169220	37750	602.92	31.488
Friends and Smokers	3	54	19	0.056	0.073
Friends and Smokers	10	460	320	0.373	0.191
Friends and Smokers	20	1720	1363	3.394	0.91

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this thesis, we have proposed a new lifted algorithm called lifted iterative join graph propagation (LIJGP) for large scale inference in Markov logic networks (MLNs). LIJGP lifts the IJGP algorithm (Mateescu et al., 2010), a state-of-the-art generalized belief propagation technique for inference in probabilistic graphical models. The key idea in our approach is to first convert the given MLN to a non-shared MLN, namely an MLN in which each formula has no shared terms. The latter can be achieved by grounding out all shared terms in each formula. The main advantage of non-shared MLNs is that each node in the corresponding lifted join graph can be processed in polynomial time and space.

Our experiments on benchmark MLNs shows that our lifted algorithm is superior to IJGP in terms of time and space requirements. Specifically, our results show that the number of nodes in the lifted join graph is several orders of magnitude smaller than the completely grounded join graph. The number of nodes is a direct measure of the memory footprint required by the algorithm. We also observe that, as the domain size increases (namely, the number of variables in the model), the rate of increase in the number of nodes is much smaller for lifted IJGP than IJGP. IJGP also converges faster on the lifted join graph than the on the completely grounded graph, since each (exponentiated) message represents multiple messages in the grounded join graph.

Future work consists of relaxing the non-shared MLN requirement, which is a sufficient but not a necessary requirement; developing structured message passing algorithms (Gogate and Domingos, 2013) that use structured representations rather than the tabular representations used in our algorithm; developing advanced evidence handling strategies that have

smaller time and space complexity requirements than those of the method proposed in the thesis; etc.

REFERENCES

Chavira, M. and A. Darwiche (2008). On probabilistic inference by weighted model counting. *Artificial Intelligence 172*(6-7), 772–799.

Darwiche, A. (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

de Salvo Braz, R. (2007). *Lifted First-Order Probabilistic Inference*. Ph. D. thesis, University of Illinois, Urbana-Champaign, IL.

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence 113*, 41–85.

Domingos, P. and D. Lowd (2009). *Markov Logic: An Interface Layer for Artificial Intelligence*. San Rafael, CA: Morgan & Claypool.

Elidan, G. and A. Globerson (2010). The 2010 UAI Approximate Inference Challenge. Available online at: http://www.cs.huji.ac.il/project/UAI10/index.php.

Elidan, G. and A. Globerson (2011). The Probabilistic Inference Challenge (PIC 2011). Available online at: http://www.cs.huji.ac.il/project/PASCAL.

Geman, S. and D. Geman (1984). Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence 6*, 721–741.

Genesereth, M. R. and N. J. Nilsson (1987). *Logical Foundations of Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann.

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

Gogate, V. and R. Dechter (2005). Approximate Inference Algorithms for Hybrid Bayesian Networks with Discrete Constraints. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pp. 209–216. AUAI Press.

Gogate, V. and P. Domingos (2011). Probabilistic Theorem Proving. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pp. 256–265. AUAI Press.

Gogate, V. and P. Domingos (2013). Structured Message Passing. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*.

Gogate, V., A. Jha, and D. Venugopal (2012). Advances in Lifted Importance Sampling. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pp. 1910–1916.

Jha, A., V. Gogate, A. Meliou, and D. Suciu (2010). Lifted Inference from the Other Side: The tractable Features. In *Proceedings of the 24th Annual Conference on Neural Information Processing Systems (NIPS)*, pp. 973–981.

Kok, S., M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, J. Wang, and P. Domingos (2008). The Alchemy System for Statistical Relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA. http://alchemy.cs.washington.edu.

Koller, D. and N. Friedman (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Kschischang, F. R., B. J. Frey, and H.-A. Loeliger (2001). Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory 47*, 498–519.

Lauritzen, S. L. and D. J. Spiegelhalter (1988). Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. *Journal of the Royal Statistical Society. Series B (Methodological) 50*(2), 157–224.

Mateescu, R., K. Kask, V. Gogate, and R. Dechter (2010). Iterative Join Graph Propagation algorithms. *Journal of Artificial Intelligence Research 37*, 279–328.

Milch, B., L. S. Zettlemoyer, K. Kersting, M. Haimes, and L. P. Kaelbling (2008). Lifted Probabilistic Inference with Counting Formulas. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pp. 1062–1068.

Murphy, K. P., Y. Weiss, and M. I. Jordan (1999). Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp. 467–475.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA: Morgan Kaufmann.

Poole, D. (2003). First-Order Probabilistic Inference. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, pp. 985–991. Morgan Kaufmann.

Richardson, M. and P. Domingos (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.

Sarkhel, S., V. Venugopal, P. Singla, and V. Gogate (2014). Lifted MAP Inference for Markov Logic Networks. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*.

Singla, P. and P. Domingos (2008). Lifted First-Order Belief Propagation. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, Chicago, IL, pp. 1094–1099. AAAI Press.

Venugopal, D. and V. Gogate (2012). On Lifting the Gibbs Sampling Algorithm. In *Second Workshop on Statistical Relational AI*.

Yedidia, J. S., W. T. Freeman, and Y. Weiss (2005). Constructing free-energy approximations and generalized Belief propagation algorithms. *IEEE Transactions on Information Theory 51*(7), 2282–2312.

VITA

Srikanth Doss K.R was born in Tamil Nadu, India. After completing schoolwork in DAV Higher Secondary School in Chennai, Srikanth enrolled in Anna University, Coimbatore for his Bachelors in Computer Science. He graduated from Coimbatore Institute of Technology in 2010 and joined as Software Developer for United Online at Hyderabad. In August 2012, he entered the graduate school of The University of Texas at Dallas.