Evaluating Inference Algorithms for the Prolog Factor Language

Tiago Gomes and **Vítor Santos Costa**
CRACS & INESC TEC, Faculty of Sciences, University of Porto
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
{tgomes,vsc}@fc.up.pt

Abstract

Over the last years there has been some interest in models that combine first-order logic and probabilistic graphical models to describe large scale domains, and in efficient ways to perform inference on these domains. Prolog Factor Language (PFL) is a extension of the Prolog language that allows a natural representation of this first-order probabilistic models (either directed or undirected). PFL is also capable of solving probabilistic queries on these models through the implementation of several inference techniques: variable elimination, belief propagation, lifted variable elimination and lifted belief propagation. We show how these models can be easily represented using PFL and then we perform a comparative study between the different inference algorithms in four artificial problems.

1 Introduction

Over the last years there has been some interest in models that combine first-order logic and probabilistic graphical models to describe large scale domains, and in efficient ways to perform inference and learning using these models (Getoor and Taskar 2007; Raedt et al. 2008). A significant number of languages and systems has been proposed and made available, such as Independent Choice Logic (Poole 1997), PRISM (Sato and Kameya 1997; Sato and Kameya 2008), Stochastic Logic Programs (ICL) (Muggleton 1996), Markov Logic Networks (MLNs) (Richardson and Domingos 2006), CLP($\mathcal{BN}$) (Santos Costa et al. 2003), ProbLog (Raedt, Kimmig, and Toivonen 2007; Kimmig et al. 2011), and LPADs (Riguzzi and Swift 2011), to only mention a few. These languages differ widely, both on the formalism they use to represent structured knowledge, on the graphical model they encode, and on how they encode it. Languages such as ICL, Prism, or ProbLog use the distribution semantics to encode probability distributions. MLNs encode relationships through first-order formulas, such that the strength of a true relationship serves as a parameter to a corresponding ground markov network. Last, languages such as CLP($\mathcal{BN}$) approach the problem in a more straightforward way, by using the flexibility of logic programming as a way to quickly encode graphical models.

Research in Probabilistic Logic Languages has made it very clear that it is crucial to design models that can support efficient inference. One of the most exciting developments toward this goal has been the notion of lifted inference (Poole 2003; de Salvo Braz, Amir, and Roth 2005). The idea is to take advantage of the regularities in structured models and perform a number of operations in a fell swoop. The idea was first proposed as an extension of variable elimination (Poole 2003; de Salvo Braz, Amir, and Roth 2005), and has since been applied to belief propagation (Singla and Domingos 2008; Kersting, Ahmadi, and Natarajan 2009) and in the context of theorem proving and model counting (Gogate and Domingos 2011; Van den Broeck et al. 2011).

Most work in probabilistic inference computes statistics from a sum of products representation, where each element is known as a *factor*. Lifted inference approaches the problem by generalizing factors through the notion of *parametric factor*, commonly called *parfactor*. Parfactors can be seen as templates, or classes, for the actual factors found in the inference process. Lifted inference is based on the idea of manipulating these parfactors, thus creating intermediate parfactors in the process, and in general delaying as much as possible the use of fully instantiated factors.

Parfactors are a compact way to encode distributions and can be seen as a natural way to express complex distributions. This was recognized in the Bayesian Logic Inference Engine (BLOG) (Milch et al. 2005), and more recently has been the basis for proposals such as Relational Continuous Models (Choi, Amir, and Hill 2010), and parametrized randvars (random variables) (Taghipour et al. 2012). In this same vein, we propose an extension of Prolog designed to support probabilistic reasoning with the parfactors, the *Prolog Factor Language (PFL)*.

The PFL aims at two goals. First, we would like to use the PFL to understand the general usefulness of lifted inference and how it plays with logical and probabilistic inference. Second, we would like to use the PFL as a tool for multi-relational learning. In this work, we focus on the first task. The work therefore introduces two contributions: a new language for probabilistic logical inference, and a comparative evaluation of a number of state-of-the-art inference techniques.

The paper is organized as follows. First, we present the main ideas of the PFL. Second, we discuss how the PFL is implemented. Third, we present a first experimental evaluation of the PFL and draw some conclusions.

2 The Prolog Factor Language (PFL)

First, we briefly review parfactors. We define parfactor as a tuple of the form:

$$< A, C, \phi >$$

where A is a set of atoms (atomic formulas), C is a set of constraints, and the ϕ function defines a potential function on $\mathbb{R}_0^+$. Intuitively, the atoms in A describe a set of random variables, the constraints in C describe the possible instances for those random variables, and the ϕ describe the potential values. The constraints in C apply over a set of logical variables L in A.

As an example, a parfactor for the MLN language could be written as:

$$2.33 : \mathit{Smokes}(X) \land X \in \{\texttt{John}, \texttt{Mary}, \texttt{William}\}$$

in the example, $A = \{\mathit{Smokes}(X)\}$, and $L = \{X\}$. Each MLN factor requires a single parameter, in this case 2.33. The constraint $X \in \{\texttt{John}, \texttt{Mary}, \texttt{William}\}$ defines the possible instances of this parfactor, in the example the three factors:

$$\begin{aligned}2.33 &: \mathit{Smokes}(\texttt{John})\\ 2.33 &: \mathit{Smokes}(\texttt{Mary})\\ 2.33 &: \mathit{Smokes}(\texttt{William})\end{aligned}$$

Notice that all factors share the same potential values.

The main goal of the PFL is to enable one to use this compact representation as an extension of a logic program. The PFL inherits from previous work in CLP($\mathcal{BN}$), which in turn was motivated by prior work on probabilistic relational models (PRMs) (Getoor et al. 2001). A PRM uses a Bayesian network to represent the joint probability distribution over fields in a relational database. Then, this Bayesian network can be used to make inferences about missing values in the database. In Datalog, missing values are represented by Skolem constants; more generally, in logic programming missing values, or existentially-quantified variables, can be represented by terms built from Skolem functors. CLP($\mathcal{BN}$) represents the joint probability distribution as a function over terms constructed from the Skolem functors in a logic program. Thus, in CLP($\mathcal{BN}$), we see random variables as a special interpretation over a set V of function symbols, the skolem variables.

2.1 Introducing the PFL

The first insight of the PFL is that CLP($\mathcal{BN}$) skolem functions naturally map to atoms in parfactor formulae. That is, in a parfactor the formula $a(X) \land b(X)$ can be seen as identifying two different skolem functions $X \to a(X)$ and $X \to b(X)$. The second observation is that both the constraints and the potential values can be obtained from a logic program. Thus, an example of a parfactor described by PFL is simply:

```
bayes abi(K) ; abi_table ; [professor(K)].
```

This factor would define the random variable `abi(K)` (shorthand for ability), within the context of a directed network, having the constraint $K \in \{professor(K)\}$, and a distribution `abi_table/1`.

More precisely, the PFL syntax for a factor is

$$\mathit{Type}\ F\ ;\ \phi\ ;\ C.$$

Thus, a PFL factor has four components:

- *Type* refers the type of the network over which the parfactor is defined. It can be `bayes`, for directed networks, or `markov`, for undirected ones.
- F is a sequence of Prolog terms that define sets of random variables under the constraints in C. Each term can be seen as the signature of a skolem function whose arguments are given by the unbound logical variables. The set of all logical variables in F is named L.

  The example includes a single term, $abi(K)$; the only logical variable is K.

- The table ϕ is either a list of potential values or a call to a Prolog goal that will unify its last argument with a list of potential values.
- C is a list of Prolog goals that impose bindings on the logical variables in L. In other words, the successful substitutions for the goals in C are the valid values for the variables in L. In the example, the goals constrain K to match a professor.

By default all random variables are boolean. However, it may be useful to define a different domain for some random variables. To do so, one can use:

```
bayes abi(K)::[h,m,l] ;
      abi_table ;
      [professor(K)].
```

to indicate that `h,m,l` are the possible values for the random variables instantiated by `abi(K)`.

A more complex example is a parfactor for a student's grade in a course:

```
bayes grade(C,S)::[a,b,c,d], int(S), diff(C) ;
      grade_table ;
      [registration(_,C,S)].
```

In the example, the `registration/3` relation is part of the extensional school data-base.

The next example shows an encoding for the competing workshops problem (Milch et al. 2008):

```
markov attends(P)::[t,f] , hot(W)::[t,f] ;
       [0.2, 0.8, 0.8, 0.8] ;
       [c(P,W)].

markov attends(P)::[t,f], series::[t,f] ;
       [0.501, 0.499, 0.499, 0.499] ;
       [c(P,_)].
```

One can observe that the model in this case is undirected. The encoding defines two parfactors: one connects workshop attendance with the workshop being hot, the other with the workshop being a series.

2.2 Querying and Generating Evidence for the PFL

In our approach, each random variable in PFL is implicitly defined by a predicate with the same name and an extra argument. More formally, the value V for a random variable or skolem function $R(A_1, \dots, A_n)$, is given by the predicate $R(A_1, \dots, A_n, V)$. Thus in order to query a professor's ability it is sufficient to ask:

```
?- abi(p0, V).
```

This approach follows in the lines of PRISM (Sato and Kameya 1997) and CLP($\mathcal{BN}$).

Evidence can be given as facts for the intentional predicates. Hence,

```
abi(p0,h).
```

can be used to indicate that we have evidence on `p0`'s (high) ability. Conditional evidence can also be given as part of a query, hence:

```
?- abi(p0,h), pop(p0,X).
```

would return the marginal probability distribution for professor `p0`'s popularity given that he had a high ability.

3 Inference in the PFL

One of our main motivations in designing the PFL is to research on the interplay between logical and probabilistic inference. Indeed, how to execute a PFL program very much depends on the probabilistic inference method used.

Solving the main inference tasks in first-order probabilistic models can be done by first grounding the network and then applying a ground solver, such as variable elimination or belief propagation. However, the cost of this operation will strongly depend on the domain size (number of objects). It may be much more efficient to solve this problem in a *lifted* way, that is, by exploiting the repeated structure of the model to speed up inference by potentially several orders of magnitude.

The PFL thus supports both lifted and grounded inference methods. In *fully grounded solving* the PFL implementation creates a ground network and then calls the solver to obtain marginals. In *lifted solving* the PFL implementation first finds out a graph of parfactors that are needed to address the current query, an then calls the lifted solver.

The fully grounded algorithm implements a transitive closure algorithm, and is as follows. First, given a query goal Q, it obtains the corresponding random variable $\{V\}$. It also collects the set of evidence variables E. The algorithm then maintains two sets: an open set of variables, initially $O \leftarrow E \cup \{V\}$, and an explored set of variables X, initially empty ($X \leftarrow \{\}$). It proceeds as follows:

1. It selects a variable V from the open-set (O) and removes it from O;
2. Adds V to X;

3. For all parfactors *defining* the variable V, it computes the constraints as a sequence of Prolog goals, and it adds the other random variables V' to O if $V' \notin X$;

4. It terminates when O empty.

The grounded algorithm is currently used by the following solvers: non-lifted variable elimination, non-lifted belief propagation, and counting belief propagation.

Lifted solving performs a first step of computing the transitive closure, but only on the graph of parfactors. No constraints are called, and no grounding is made. A second step then evaluates the domain defined by the constraints as a set of tuples, that is compactly described as a tree. The network of parfactors and respective tuples is next sent to a lifted solver. This implementation supports lifted variable elimination and lifted first-order belief propagation.

The probabilistic inference algorithms used in the PFL are discussed next.

4 Probabilistic Inference Algorithms

A typical inference task is to compute the marginal probabilities of a set of random variables given the observed values of others (evidence). Variable elimination (VE) and belief propagation (BP) are two popular ways to solve this problem.

Next we briefly describe these two algorithms as well as their lifted versions.

4.1 Variable Elimination

Variable elimination (Zhang and Poole 1996) is one of the simplest exact inference methods. As the name indicates, it works by successively eliminating the random variables that appears in the factors until only the query variable remains.

At each step, each variable X is eliminated by first collecting the factors where X appears, then calculating the product of these factors and finally summing out X. The product of two factors works like a join operation in relational algebra, where the set of random variables of the resulting factor is the union of the random variables of the two operands, and the product is performed such that the values of common random variables match with each other. Summing out a variable X is done through the summation of all probabilities where the values of X varies, while the values of the other variables remains fixed.

Unfortunately, the cost of variable elimination is exponential in relation to the *treewidth* of the graph. The treewidth is the size of the largest factor created during the operation of the algorithm using the best order in which the variables are eliminated. And it gets worse: finding the best elimination order is a NP-Hard problem. In our experiments, we choose to eliminate the variable whose product of all factors where it appears contains the smallest number of variables.

4.2 Belief Propagation

Belief propagation is an algorithm that follows a message passing approach to efficiently solve probabilistic queries. Here, we describe the implementation of the algorithm over factor graphs (Kschischang, Frey, and Loeliger 2001). The algorithm consists in iteratively exchanging local messages between variable and factor nodes of a factor graph, until convergence is achieved (the probabilities for each variable stabilize). These messages consists in vectors of probabilities that measures the influence that variables have among others.

Belief propagation is known to converge to the exact answers when the factor graph does not contains loops. Otherwise, there is no guarantee on the convergence of the algorithm, or that the results will be good approximations of the exact solutions. However, experimental results show that this *loopy* belief propagation often it converges to good approximations and can finish several times faster than other methods (Murphy, Weiss, and Jordan 1999).

Next, we describe how the messages are calculated and how to obtain the marginals for each variable. The message from a variable X to a factor f is defined by:

$$u_{X\to f}(x) = \prod_{g \in \text{neighbors}(X)\setminus\{f\}} u_{g\to X}(x) \ , \qquad (1)$$

that is, it consists in the product of the messages received from X's other neighbor factors. The message that a factor f sends to a variable X is defined by:

$$u_{f\to X}(x) = \sum_{y_1,\ldots,y_k} f(x, y_1, \ldots, y_k) \prod_{i=1}^{k} u_{Y_i\to f}(y_i) \ , \qquad (2)$$

where $Y_1, \ldots, Y_k$ are the other f's neighbor variables. In other words, it is the product of the messages received from the other f's neighbor variables, followed by summing out all variables except X.

Finally, we estimate the marginal probabilities for a variable X by computing the product of the messages received by all X's neighbor factors:

$$P(x) \propto \prod_{f \in \text{neighbors}(X)} u_{f\to X}(x) \ . \qquad (3)$$

Depending on the graphical model, we may also need to normalize the probabilities. That is, to scale them to sum to 1. In our implementation, initially all messages are initialized uniformly.

4.3 Lifted Variable Elimination

Lifted variable elimination exploits the symmetries present in first-order probabilistic models, so that it can apply the same principles behind variable elimination to solve a probabilistic query without grounding the model. However, instead of summing out a random variable at a time, it works out by summing out a whole group of interchangeable random variables.

Initially a shattering operation is applied in all of the parfactors. Shattering consists in splitting the parfactors until all atoms represent under the constraint either identical or disjoint sets of ground random variables. Intuitively, this is necessary to ensure that the same reasoning steps are applied to all ground factors represented by a parfactor. Sometimes it may also be necessary to further split the parfactors for some lifted operation be correct, that is, equal to a set of multiple ground operations.

Work on lifting variable elimination started with (Poole 2003) and was later extended by (de Salvo Braz, Amir, and Roth 2007). (Milch et al. 2008) increased the scope of lifted inference by introducing counting formulas, that can be seen as a compact way to represent a product of repeated ground factors. The current state of art on lifted variable elimination is GC-FOVE (Taghipour et al. 2012). GC-FOVE extends previous work by redefining the operations described in (Milch et al. 2008) to be correct for whatever constraint representation is being used. In fact, the constraints can be even represented as sets of tuples over the corresponding logical variables.

4.4 Lifted Belief Propagation

There are currently two main approaches for applying belief propagation at lifted level: lifted first-order belief propagation (Singla and Domingos 2008) and counting belief propagation (Kersting, Ahmadi, and Natarajan 2009).

Counting belief propagation is defined over factor graphs and works out by grouping in clusters the variables and factors that are indistinguishable in terms of messages sent and received, creating through process a *compressed* factor graph. To achieve this, counting belief propagation simulates the use of belief propagation through a color based scheme, where colors are transmitted instead of real probabilities and identical colors identify identical messages.

While counting belief propagation requires grounding the model to a factor graph and only then performing the compression phase, lifted first-order belief propagation identifies the random variables which send and receive the same messages directly from the parfactors. This can be significantly more efficient.

In both approaches, the messages sent by belief propagation need to be adapted to consider the fact that an edge in the compressed/lifted network can represent multiples edges in the uncompressed/ground network.

5 Experimental Evaluation

We compare the performance of variable elimination, belief propagation and their lifted versions in four standard benchmark problems described using PFL: *workshop attributes* (Milch et al. 2008), *competing workshops* (Milch et al. 2008), *city* (Poole 2003) and *social domain* (Jha et al. 2010).

Our implementation of lifted variable elimination corresponds to GC-FOVE, with the difference that we use a simple tree to represent the constraints instead of a constraint tree (Taghipour et al. 2012). We have also implemented two forms of lifted belief propagation, namely one based on counting belief propagation (CBP) and one based on lifted first-order belief propagation (LFOBP). All algorithms were written in `C++` and are available with the development version of the YAP Prolog system (Santos Costa, Damas, and Rocha 2012).

The test environment was a machine with 2 Intel® Xeon® X5650@2.67GHz processors and 99 Gigabytes of main memory, running a Linux kernel 2.6.34.9-69.fc13.x86_64. The times for solving each query presented here are the minimum times of a series of multiple runs.

Figures 1(a) and 1(b) displays the running times to solve a query on *series* for both workshop attributes and competing workshops problems with an increased number of individuals (Milch et al. 2008). GC-FOVE and LFOBP are the methods that performs better, followed by CBP. For CBP, the size of the compressed factor graph remains constant as we increase the number of individuals. Hence its cost is mostly dominated by the time used to compress the factor graph. Variable elimination performs reasonably well even in the presence of large domains, as the treewidth remains constant for all number of individuals. Although it converges in few iterations, belief propagation is clearly the slowest method.

Figure 1(c) shows the running times to solve a query

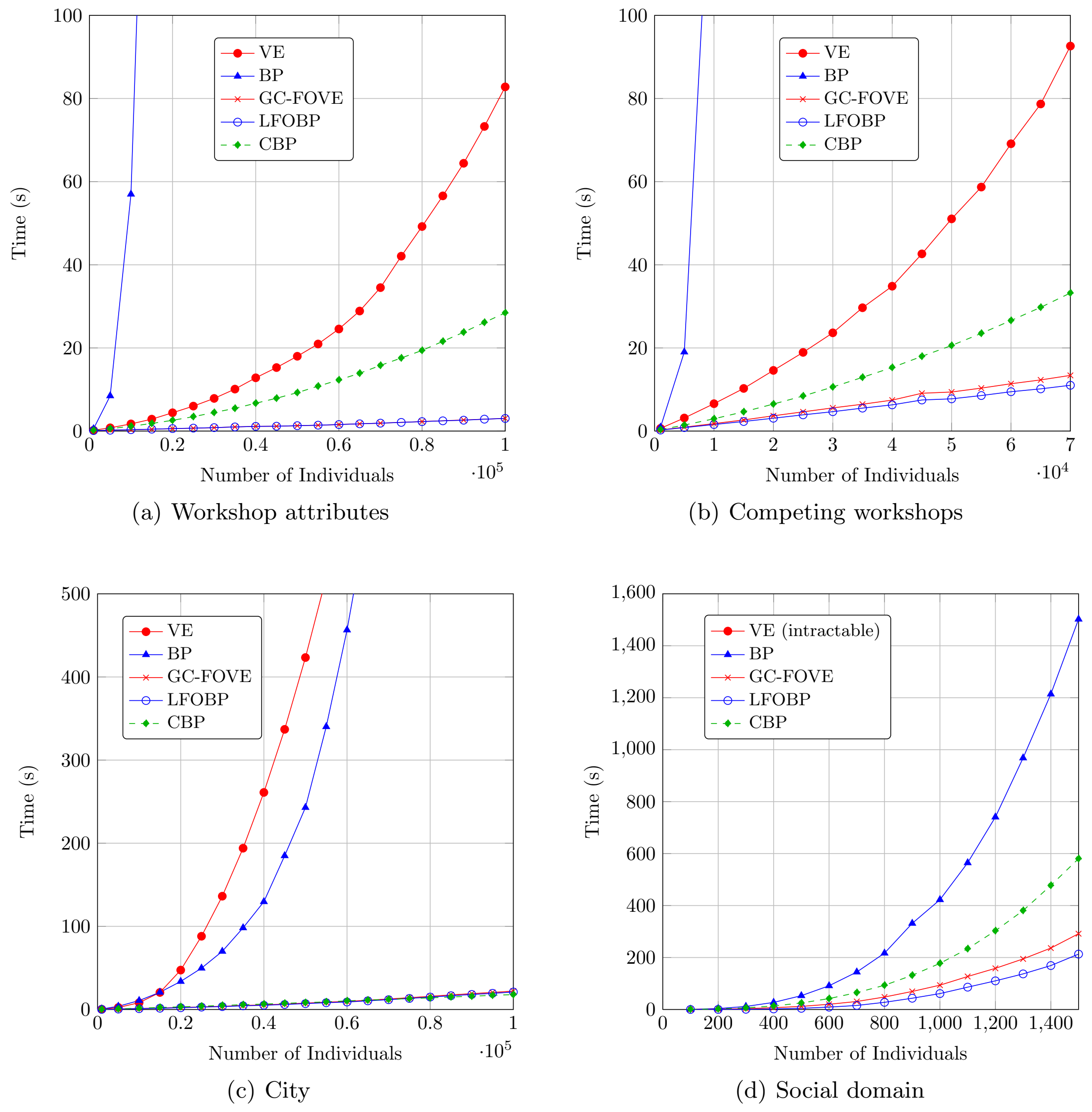

Figure 1: Performance on workshops attributes, competing workshops, city and social domain problems with an increased number of individuals.

on *guilty* for some individual in the city problem, given the descriptions of the others (Poole 2003). Here GC-FOVE, LFOBP and CBP have close performances, as well as their non-lifted versions.

More interesting is the social domain problem (Jha et al. 2010) where non-lifted exact inference is intractable, since the treewidth of the graph increases exponentially in relation to the number of individuals. Figure 1(d) displays the performances of belief propagation, GC-FOVE, LFOBP and CBP for a query on *friends*. We found that CG-FOVE is capable of exactly solving this problem in a very efficient way, even thought it has to do some operations whose cost depend on the number of individuals, namely counting conversion (Taghipour et al. 2012).

Now, we want to evaluate the effect caused by evidence on the value of some random variables for all the different inference algorithms. For this experiment we took the social domain network, fixed the number of individuals at 500 and varied the percentage of evidence on random variables produced by *smokes* (observed random variables as well their observed values were choose randomly). The results are show in Figure 2. While the run times remain stable for the three versions of belief propagation with all percentages of evidence, GC-FOVE behaves different. The first set of evidence causes an increase on the run time as some parfactors will be split. However adding more evidence will not cause further splitting and instead the run times will decrease gradually, since the absorption of the evidence will cause some operations used by GC-

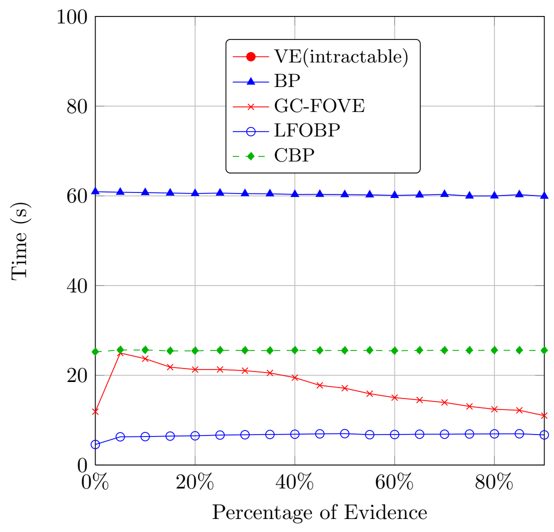

Figure 2: Performance on social domain problem varying the percentage of evidence.

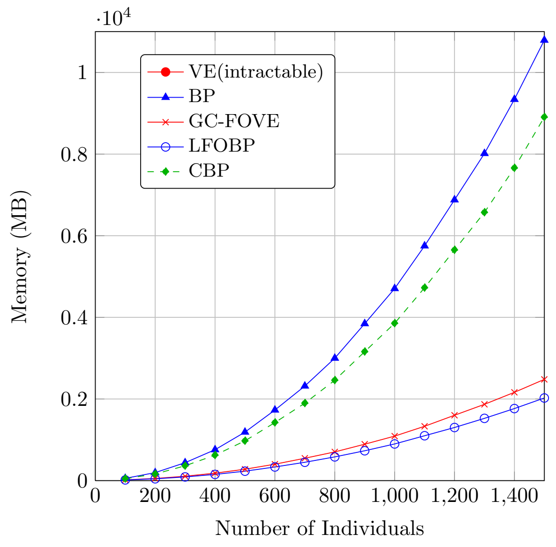

Figure 3: Memory usage on social domain problem with an increased number of individuals.

FOVE to become slightly cheaper.

Finally, we also found interesting in comparing the memory usage between ground and lifted solvers. Figure 3 shows the maximum memory used during the execution of each inference method for the social domain problem. As we expected, lifted solving requires less memory than ground solving. The memory usage of CBP is close to the memory usage of belief propagation because it needs to ground the model first.

6 Conclusions and Future Work

We introduce the PFL, an extension of Prolog to manipulate complex distributions represented as products of factors. The PFL has been implemented as an extension to an existing programming language, and is publicly available. As an initial experiment, we use the PFL to represent several lifted inference benchmarks, and compare a number of inference techniques. Our results illustrates the efficiency gains obtained from lifted solving these problems.

We see the PFL as a tool in experimenting with ways to combine probabilistic and logical inference, and plan to continue experimenting with different approaches, such as the ones based on knowledge compilation (Van den Broeck et al. 2011). We also plan to look in more detail to aggregates. Last, and not least, our ultimate focus is to be able to learn PFL programs.

Acknowledgments

This work is funded (or part-funded) by the ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project HORUS (PTDC/EIA/100897/2008) and project LEAP (PTDC/EIA-CCO/112158/2009).

References

Choi, Jaesik, Eyal Amir, and David J. Hill. 2010. "Lifted Inference for Relational Continuous Models." Edited by Peter Grünwald and Peter Spirtes, *UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8-11, 2010.* AUAI Press, 126–134.

de Salvo Braz, Rodrigo, Eyal Amir, and Dan Roth. 2005. "Lifted First-Order Probabilistic Inference." Edited by Leslie Pack Kaelbling and Alessandro Saffiotti, *IJCAI.* Professional Book Center, 1319–1325.

———. 2007. "Lifted first-order probabilistic inference." Chapter 15 of *Introduction to Statistical Relational Learning*, edited by Lise Getoor and Ben Taskar, 433–451. MIT Press.

Getoor, L., N. Friedman, D. Koller, and A. Pfeffer. 2001. "Learning Probabilistic Relational Models." Chapter 13 of *Relational Data Mining*, 307–335. Springer.

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

Gogate, Vibhav, and Pedro Domingos. 2011. "Probabilistic Theorem Proving." Edited by Fabio Gagliardi Cozman and Avi Pfeffer, *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011.* AUAI Press, 256–265.

Jha, Abhay Kumar, Vibhav Gogate, Alexandra Meliou, and Dan Suciu. 2010. "Lifted Inference Seen from the Other Side : The Tractable Features." Edited by John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta, *NIPS.* Curran Associates, Inc., 973–981.

Kersting, K., B. Ahmadi, and S. Natarajan. 2009. "Counting belief propagation." *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence.* AUAI Press, 277–284.

Kimmig, Angelika, Bart Demoen, Luc De Raedt, Vítor Santos Costa, and Ricardo Rocha. 2011. "On the implementation of the probabilistic logic programming language ProbLog." *TPLP* 11 (2-3): 235–262.

Kschischang, F.R., B.J. Frey, and H.A. Loeliger. 2001. "Factor graphs and the sum-product algorithm." *Information Theory, IEEE Transactions on* 47 (2): 498–519.

Milch, B., L.S. Zettlemoyer, K. Kersting, M. Haimes, and L.P. Kaelbling. 2008. "Lifted probabilistic inference with counting formulas." *Proc. 23rd AAAI*, pp. 1062–1068.

Milch, Brian, Bhaskara Marthi, Stuart J. Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. 2005. "BLOG: Probabilistic Models with Unknown Objects." Edited by Leslie Pack Kaelbling and Alessandro Saffiotti, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005.* Professional Book Center, 1352–1359.

Muggleton, Stephen. 1996. "Stochastic Logic Programs." In *Advances in Inductive Logic Programming*, edited by Luc De Raedt, Volume 32 of *Frontiers in Artificial Intelligence and Applications*, 254–264. Amsterdam: IOS Press.

Murphy, K.P., Y. Weiss, and M.I. Jordan. 1999. "Loopy belief propagation for approximate inference: An empirical study." *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence.* Morgan Kaufmann Publishers Inc., 467–475.

Poole, David. 1997. "The Independent Choice Logic for Modelling Multiple Agents Under Uncertainty." *Artif. Intell.* 94 (1-2): 7–56.

———. 2003. "First-order probabilistic inference." Edited by Georg Gottlob and Toby Walsh, *IJCAI.* Morgan Kaufmann, 985–991.

Raedt, Luc De, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, eds. 2008. *Probabilistic Inductive Logic Programming - Theory and Applications.* Volume 4911 of *Lecture Notes in Computer Science.* Springer.

Raedt, Luc De, Angelika Kimmig, and Hannu Toivonen. 2007. "ProbLog: A Probabilistic Prolog and Its Application in Link Discovery." Edited by Manuela M. Veloso, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007.* 2462–2467.

Richardson, Matthew, and Pedro Domingos. 2006. "Markov logic networks." *Machine Learning* 62 (1-2): 107–136.

Riguzzi, Fabrizio, and Terrance Swift. 2011. "The PITA system: Tabling and answer subsumption for reasoning under uncertainty." *TPLP* 11 (4-5): 433–449.

Santos Costa, Vítor, Luís Damas, and Ricardo Rocha. 2012. "The YAP Prolog system." *Theory and Practice of Logic Programming* 12 (Special Issue 1-2): 5–34.

Santos Costa, Vítor, David Page, Maleeha Qazi, and James Cussens. 2003. "CLP(*BN*)*: Constraint Logic Programming for Probabilistic Knowledge.*" Edited by Christopher Meek and Uffe Kjærulff, *UAI '03, Proceedings of the 19th Conference in Uncertainty in Artificial Intelligence, Acapulco, Mexico, August 7-10 2003.* Morgan Kaufmann, 517–524.

Sato, Taisuke, and Yoshitaka Kameya. 1997. "PRISM: A Language for Symbolic-Statistical Modeling." *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes.* Morgan Kaufmann, 1330–1339.

———. 2008. "New Advances in Logic-Based Probabilistic Modeling by PRISM." In Raedt et al. 2008, 118–155.

Singla, P., and P. Domingos. 2008. "Lifted first-order belief propagation." *Proceedings of the 23rd national conference on Artificial intelligence*, Volume 2. 1094–1099.

Taghipour, N., D. Fierens, J. Davis, and H. Blockeel. 2012. "Lifted variable elimination with arbitrary constraints." *Proceedings of the fifteenth international conference on artificial intelligence and statistics.*

Van den Broeck, Guy, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. 2011. "Lifted Probabilistic Inference by First-Order Knowledge Compilation." Edited by Toby Walsh, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011.* IJCAI/AAAI, 2178–2185.

Zhang, N. L., and D. Poole. 1996. "Exploiting causal independence in bayesian network inference." *Journal of Artificial Intelligence Research, Vol 5, (1996), 301-328*, December.