Introduction to Statistical Relational Learning

Adaptive Computation and Machine Learning

Thomas Dietterich, Editor
Christopher M. Bishop, David Heckerman, Michael I. Jordan, and Michael Kearns, Associate Editors

Bioinformatics: The Machine Learning Approach
Pierre Baldi and Søren Brunak, 1998

Reinforcement Learning: An Introduction
Richard S. Sutton and Andrew G. Barto, 1998

Graphical Models for Machine Learning and Digital Communication
Brendan J. Frey, 1998

Learning in Graphical Models
Michael I. Jordan, ed., 1998

Causation, Prediction, and Search, 2nd Edition
Peter Spirtes, Clark Glymour, and Richard Scheines, 2001

Principles of Data Mining
David Hand, Heikki Mannila, and Padhraic Smyth, 2001

Bioinformatics: The Machine Learning Approach, 2nd Edition
Pierre Baldi and Søren Brunak, 2001

Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond
Bernhard Schölkopf and Alexander J. Smola, 2001

Learning Kernel Classifiers: Theory and Algorithms
Ralf Herbrich, 2001

Introduction to Machine Learning
Ethem Alpaydin, 2004

Gaussian Processes for Machine Learning
Carl Edward Rasmussen and Christopher K. I. Williams, 2005

Semi-Supervised Learning
Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, eds. 2006

The Minimum Description Length Principle
Peter D. Grünwald, 2007

Introduction to Statistical Relational Learning
Lise Getoor and Ben Taskar, eds., 2007

Introduction to Statistical Relational Learning

edited by
Lise Getoor
Ben Taskar

The MIT Press
Cambridge, Massachusetts
London, England

©2007 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

Typeset by the authors using LaTeX 2ε
Printed and bound in the United States of America

Library of Congress Cataloging-in-Publication Data
Introduction to statistical relational learning / edited by Lise Getoor, Ben Taskar.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-262-07288-5 (hardcover : alk. paper)
1. Relational databases. 2. Machine learning–Statistical methods 3. Computer algorithms. I. Getoor, Lise. II. Taskar, Ben.
QA76.9.D3I68 2007 006.3'1–dc22 2007000951
10 9 8 7 6 5 4 3 2 1

Contents

Series Foreword **vii**

1 Lifted First-Order Probabilistic Inference **1**
Rodrigo de Salvo Braz, Eyal Amir, Dan Roth
1.1 Introduction 1
1.2 Language, Semantics and Inference problem 3
1.3 The First-Order Variable Elimination (FOVE) algorithm 5
1.4 An experiment 12
1.5 Auxiliary operations 13
1.6 Applicability of lifted inference 16
1.7 Future Directions 17
1.8 Conclusion 17

Contributors **21**

Index **27**

Series Foreword

The goal of building systems that can adapt to their environments and learn from their experience has attracted researchers from many fields, including computer science, engineering, mathematics, physics, neuroscience, and cognitive science. Out of this research has come a wide variety of learning techniques that have the potential to transform many scientific and industrial fields. Recently, several research communities have converged on a common set of issues surrounding supervised, unsupervised, and reinforcement learning problems. The MIT Press series on Adaptive Computation and Machine Learning seeks to unify the many diverse strands of machine learning research and to foster high quality research and innovative applications.

Thomas Dietterich

1 Lifted First-Order Probabilistic Inference

Rodrigo de Salvo Braz, Eyal Amir and Dan Roth

Most probabilistic inference algorithms are specified and processed on a propositional level, even though many domains are better represented by first-order specifications that compactly stand for a class of propositional instantiations. In the last fifteen years, many algorithms accepting first-order specifications have been proposed. However, these algorithms still perform inference on a mostly propositional model, generated by the instantiation of first-order constructs. When this is done, the rich and useful first-order structure is not explicit anymore. This first-order representation and structure allow us to perform *lifted* inference, that is, inference on the first-order representation directly, manipulating not only individuals but also groups of individuals. This has the potential of greatly speeding up inference. We precisely define the problem and present an algorithm that generalizes variable elimination and manipulates first-order representations in order to perform lifted inference.

1.1 Introduction

Probabilistic inference algorithms are widely employed in artificial intelligence. Among those, graphical models such as Bayesian and Markov networks (BNs and MNs respectively) (Pearl [1988]) are among the most popular. These models are specified by a set of conditional probabilities (for BNs) or factors, also called potential functions (for MNs). Both conditional probabilities and factors are defined over particular subsets of the available random variables, and map assignments of those random variables to positive real numbers (called *potentials* in MNs). For our purposes, it will be helpful to think of graphical models in general and simply consider conditional probabilities as a type of factor.

For example, in an application for document subject classification, one can specify a dependence between the random variables $subject_apple, word_mac$ (which

indicate that the subject of the document is "apple" and that the word "mac" is present in it) by defining a factor on their assignments. The higher the potential for a given assignment to these random variables, the more likely it will be in the joint distribution defined by the model.

A limitation of graphical models arises when the same dependence holds between different subsets of random variables. For example, we might declare the dependence above to hold also between $subject_microsoft, word_windows$. In traditional graphical models, we must use separate potential functions to do so, even though the dependence is the same. This brings redundancy to the model and possibly wasted computation. It is also an ad hoc mechanism since it does not cover other sets of random variables exhibiting the same dependence (in this case, some other company and product).

The root of this limitation is that graphical models are *propositional* (random variables can be seen as analogous to propositions in logic), that is, they do not allow quantifiers and parameterization of random variables by objects. A *first-order* or *relational* language, on the other hand, does allow for these elements. With such a language, we can specify a potential function that applies, for example, to *all* tuples of random variables obtained by instantiating X and Y in the tuple

$$subject(X), company(X), product(X, Y), word(Y). \tag{1.1}$$

This way we not only cover both cases presented before, but also unforeseen ones, with a single compact specification.

In the last fifteen years, many proposals for probabilistic inference algorithms accepting first-order specifications have been presented (Ngo and Haddawy [1995], Kersting and De Raedt [2000], Costa et al. [2003], Friedman et al. [1999], Poole [1993], Richardson and Domingos [2004], among many others), most of which based on the theoretic framework of Halpern [1990]. However, these solutions still perform inference at a mostly propositional level; they typically instantiate potential functions according to the objects relevant to the present query, thus obtaining a regular graphical model on propositional random variables, and then using a regular inference algorithm on this model. In domains with a large number of objects this may be both costly and essentially unnecessary. Suppose we have a medical application about the health of a large population, with a random variable per person indicating whether they are sick with a certain disease, and with a potential function representing the dependence between a person being sick and that person getting hospitalized. To answer the query "what is the probability that *someone* will be hospitalized?", an algorithm that depends on propositionalization will instantiate *a random variable per person*. However this is not necessary since one can calculate the same probability by reasoning about individuals on a general level, simply using the population size, in order to answer that query in a much shorter time. In fact, the latter calculation would not depend on the population size at all.

Naturally, it is possible to reformulate the problem so that it is solved in a

more efficient manner. However, this would require manual devising of a process *specific* to the model or query in question. It is desirable to have an algorithm that can receive a *general* first-order model and *automatically* answer queries like these without computational waste.

A first step in this direction was given by Poole [2003], which proposes a generalized version of the variable elimination algorithm Zhang and Poole [1994] that is *lifted*, that is, deals with groups of random variables at a first-order level. The algorithm receives a specification in which parameterized random variables stand for all of their instantiations and then eliminates them in a way that is equivalent to, but much cheaper than, eliminating all their instantiations at once. For the parameterized potential function (1.1), for example, one can eliminate $product(X, Y)$ in a single step that would be equivalent to eliminating all of its instantiations.

The algorithm in Poole [2003], however, applies only to certain types of models because it uses a single elimination operation that can only eliminate parameterized random variables containing all parameters present in the potential function (the method can eliminate $product(X, Y)$ from (1.1) but not $company(X)$ because the latter does not contain the parameter Y). As we will see later, Poole's algorithm uses the operation we call *inversion elimination*. In addition to inversion elimination, we have developed further operations (the main ones called *counting elimination* and *partial inversion*) that broaden the applicability of lifted inference to a greater extent (de Salvo Braz et al. [2005, 2006]). These operations are combined to form the first-order variable elimination (FOVE) algorithm presented in this chapter. The cases to which lifted inference applies can be roughly summarized as those containing dependencies where the set of parameters of each parameterized random variable are disjoint or, when this is not the case, where there is a set of parameters whose instantiations create independent solvable cases. We specify these conditions in more detail when explaining the operations, and further discuss applicability in section 1.6. When no lifted inference operation applies to a specific part of a model, FOVE can still apply standard propositional methods to that part, assuring completeness and limiting propositional inference to only some parts of the model.

1.2 Language, Semantics and Inference problem

Like Markov networks, first-order probabilistic models (FOPMs) are essentially defined by a set of factors. However, unlike them, these factors are defined over *parameterized* random variables, and for this reason we call them **parfactors** (following Poole [2003]).

Given a universe of objects over which the parameters range, we can generate regular propositional factors from a parfactor by replacing its parameters by particular objects. A parfactor is therefore a compact representation of a set of regular factors, and a FOPM is a compact representation of a Markov network composed by all instantiations of all of its parfactors.

Based on the correspondence to logic concepts, we call a parameterized random variable an **atom**, and a parameter a **logical variable** (as opposed to *random* variables). We also call the functors of atoms **predicates**. Even though we informally refer to atoms as "parameterized random variables", they are not, technically speaking, random variables, but stand for classes of them. A ground atom, however, denotes a random variable. Sometimes we call random variables **ground** to emphasize their correspondence to ground atoms.

Logical variables are typed, with each type being a finite set of objects. We denote the **domain**, or **type**, of a logical variable X by D_X and its cardinality by $|X|$. In our examples, unless noted, all logical variables have the same type. Each predicate p also has its domain, D_p, which is the set of values that each of the random variables with that predicate can take.

Formally, a **parfactor** g is a tuple (ϕ_g, A_g, C_g), where ϕ_g is a potential function defined over atoms A_g to be instantiated by all substitutions of its logical variables satisfying a constraint C_g. A **constraint** is a pair (F, V) where F is an equational formula on logical variables and V is the set of logical variables to be instantiated (some of them may not be in the formula). We sometimes denote a constraint by its formula F alone, when the set of logical variables V is clear from context. Tautological formulas are represented by $\top$. For example, the parfactor $(\phi, (p(X), q(X, Y)), (X \neq a, \{X, Y\}))$ applies ϕ to all instantiations of $(p(X), q(X, Y))$ by substitutions of X and Y satisfying $X \neq a$. We denote the set of substitutions satisfying C by $[C]$.

While we are neutral as to how the potential functions are actually specified, logical formulas seem to be a convenient choice. For example, a weighted formula $0.7 : \mathit{epidemic}(D) \Rightarrow \mathit{sick}(P, D)$ might represent a potential function $\phi(\mathit{epidemic}(D), \mathit{sick}(P, D))$ with potential 0.7 for assignments in which the formula is true. This allows us to specify FOPMs by sets of weighted logical formulas that are intuitive and simple to read, and is the approach taken by Markov logic networks (Richardson and Domingos [2004]).

The **projection** $C_{|L}$ of a constraint $C = (F, V)$ **onto a set of logical variables** L is a constraint equivalent to $(\exists L' F, L)$ for $L' = V \setminus L$. Intuitively, $C_{|L}$ describes the conditions posed by C on L alone, that is, the possible substitutions on L that are part of substitutions in $[C]$. For example, $(X \neq a \wedge X \neq Y \wedge Y \neq b, \{X, Y\})_{|\{X\}} = (X \neq a, \{X\})$. FOVE uses a constraint solver which is able to solve several constraint problems, such as determining the number of solutions of a constraint and its projection onto sets of logical variables.

In certain contexts we wish to describe the class of random variables instantiated from an atom with constraints on its logical variables (for example, the set of random variables instantiated from $p(X, Y)$, with $X \neq a$). We call such pairs of atoms and constraints **constrained atoms**, or **c-atoms**. The **c-atoms of a parfactor** is the set of c-atoms formed by its atoms and its constraint.

Let α be a parfactor, c-atom, constraint or a set of those. We define $RV(\alpha)$ to be the set of (ground) random variables specified by α, and $\alpha\theta$ denotes the result of applying a substitution θ to α. $[C_g]$ is also denoted by Θ_g.

A FOPM is specified by a set of parfactors G and the types of its logical variables. Its semantics is a joint distribution defined on $RV(G)$ by the Markov network formed by all the instantiations of parfactors. Thus it is proportional to the product of all instantiated parfactors:

$$P(RV(G)) \propto \prod_{g \in G} \prod_{\theta \in \Theta_g} g\theta.$$

For convenience, we denote $\prod_{\theta \in \Theta_g} g\theta$ by $\Phi(g)$, and $\prod_{g \in G} \Phi(g)$ by $\Phi(G)$. Therefore we can write the above as $P(RV(G)) \propto \Phi(G)$.

The most important inference task in graphical models is marginalization. For FOPMs, it takes the following form: given a set of ground random variables Q, calculate

$$P(Q) \propto \sum_{RV(G) \setminus Q} \Phi(G), \tag{1.2}$$

where the summation ranges over all assignments to $RV(G) \setminus Q$. Posterior probabilities can be calculated by representing evidence as additional parfactors on the evidence atoms.

The FOVE algorithm makes the simplifying assumption that the FOPM is **shattered** w.r.t the query Q. A set of c-atoms is **shattered** if the instantiations of any pair of its elements are either identical or disjoint. A parfactor, or set of parfactors, is shattered if the set of their c-atoms is shattered. A FOPM is shattered w.r.t. a query Q if the union of its c-atoms and those of the query is shattered. For example, we can have c-atoms $(p(X), X \neq a)$,$(p(Y), Y \neq a)$ and $p(a)$ in a model, but not $p(Y)$ and $p(a)$, because $RV(p(a)) \subset RV(p(Y))$ but $RV(p(a)) \neq RV(p(Y))$. When a FOPM and query are not shattered, we can replace them by equivalent shattered FOPM and query through the process of **shattering**, detailed in section 1.5.2.

1.3 The First-Order Variable Elimination (FOVE) algorithm

Computing (1.2) directly is intractable since it would take exponential time in the number of random variables in $RV(G) \setminus Q$. This is the case even for the propositional case, which is the reason why algorithms have been developed that take advantage of independences represented in the model in order to compute marginals more efficiently. One of these algorithms is variable elimination (VE) Zhang and Poole [1994]. First-order variable elimination (FOVE) is a first-order generalization of VE. While VE eliminates a random variable at a time, FOVE eliminates a c-atom, or set of c-atoms, at each step. By eliminating a c-atom, we implicitly eliminate all of its instantiations at the same time. Let E be a set of c-atoms to be eliminated from a FOPM with a set G of parfactors. Let $G_E, G_{\neg E} \subseteq G$ be the sets of parfactors

depending and not depending on E, respectively. Then

$$\sum_{RV(G)\setminus Q} \Phi(G) = \sum_{(RV(G)\setminus RV(E))\setminus Q} \Phi(G_{\neg E}) \sum_{RV(E)} \Phi(G_E).$$

We later show operations computing a parfactor g' such that $\sum_{RV(E)} \Phi(G_E) = \Phi(g')$. Once we have g', the right-hand side of the above is equal to

$$\sum_{(RV(G)\setminus RV(E))\setminus Q} \Phi(G_{\neg E})\Phi(g') = \sum_{(RV(G)\setminus RV(E))\setminus Q} \Phi(G_{\neg E} \cup \{g'\}) = \sum_{RV(G')\setminus Q} \Phi(G')$$

where $G' = G_{\neg E} \cup \{g'\}$. In other words, we have reduced the original marginalization to a smaller instance that does not include $RV(E)$. This is repeated until only Q is left.

A crucial difference between VE and FOVE is elimination ordering. VE eliminates random variables according to an ordering given a priori. In FOVE, eliminating certain c-atoms may require eliminating some other c-atoms first, so it may be the case that some c-atoms are not eliminable at all times (these conditions will be clarified later). Because parfactors and c-atoms are sometimes changed and reorganized during the algorithm, it is not a simple matter to choose an ordering in advance. Instead, the elimination ordering is dynamically determined.

Before we move on to explaining the operations for calculating $\sum_{RV(E)} \Phi(G_E) = \Phi(g')$, we mention that, in fact, they only calculate $\sum_{RV(E)} \Phi(g)$ for a single parfactor g. This is not a problem because the operation of **fusion**, covered in section 1.5.1, calculates g such that $\Phi(g) = \Phi(G)$ for any set of parfactors G.

1.3.1 Counting Elimination

We first show counting elimination on a specific example and later generalize it. Consider the summation

$$\sum_{RV(p(X))} \prod_{X,Y} \phi(p(X), p(Y)),$$

where p is a boolean predicate. (Note that the X used under the summation is not the same X used by the product. $RV(p(X))$ is shorthand for all assignments over the set $\{p(X) : X \in D_X\}$, so X is locally used. In fact, we could have written $RV(p(Y))$, or even $RV(p(Z))$, to the same effect. We choose to use X or Y to make the link with the atom in the parfactor more obvious.)

Counting elimination is based on the following insight: because a parfactor will typically only evaluate to a few different potentials, large groups of its instantiations will evaluate to the same potential. So the summation is rewritten

$$\sum_{RV(p(X))} \phi(0,0)^{|(0,0)|}\phi(0,1)^{|(0,1)|}\phi(1,0)^{|(1,0)|}\phi(1,1)^{|(1,1)|},$$

where $|(v_1, v_2)|$ indicates the number of possible choices for X and Y so that $p(X) =$

v_1 and $p(Y) = v_2$ given the current assignment to $RV(p(X))$. These partition sizes can be calculated by a combinatorial, or counting, argument. Assume we know $\vec{N}_p$, a vector of integers that indicates how many random variables in $RV(p(X))$ are currently assigned a particular value, that is, $\vec{N}_{p,i} = |\{r \in RV(p(X)) : r = i\}|$ for each $i \in D_p$. Naturally, $\sum_i \vec{N}_{p,i} = |RV(p(X))|$. Then there are $\vec{N}_{p,v_1}$ possible values for X (so that $p(X) = v_1$) and $\vec{N}_{p,v_2}$ distinct possible values for Y (so that $p(Y) = v_2$), so $|(v_1, v_2)| = \vec{N}_{p,v_1}\vec{N}_{p,v_2}$.

We take advantage of the fact that the values $|(v_1, v_2)|$ do not depend on the particular assignments to $RV(p(X))$, but only on $\vec{N}_p$. This allows us to iterate over the *groups* of assignments with the same $\vec{N}_p$ and do the calculation for the entire group. We also take into account the group size, which is provided by the binomial coefficient of $\vec{N}_p$, $\binom{|RV(p(X))|}{\vec{N}_{p,0}}$ (or, equivalently, $\binom{|RV(p(X))|}{\vec{N}_{p,1}}$). We then have

$$\sum_{\vec{N}_p} \binom{|RV(p(X))|}{\vec{N}_{p,0}} \prod_{(v_1,v_2)} \phi(v_1, v_2)^{\vec{N}_{p,v_1}\vec{N}_{p,v_2}}$$

which has a number of terms linear in $|RV(p(X))|$, as opposed to the previous exponential number.

Counting elimination is not a universal method. The counting argument presented above requires that there be little interaction between the logical variables of atoms. If a parfactor is on $p(X, Y), q(X, Z)$, for example, the counting argument does not work because the choices for (X, Z) depend on the particular X chosen for $p(X)$; we can no longer compute number of choices using counters alone but need to know the particular assignment to $RV(p(X))$. Generally, under counting elimination, choices for one atom cannot constrain the choices for another atom (there are exceptions to this rule, as for example *just-different* atoms, presented in de Salvo Braz et al. [2006]).

We now give the formal account of counting elimination, starting with some preliminary definitions.

First, we define the notion of **independent atoms given a constraint**. Intuitively, this happens when choosing a substitution for the logical variables of one atom does not change the possible choices of substitutions for the other atom. Let $\bar{X}_1$ and $\bar{X}_2$ be two sets of logical variables such that $\bar{X}_1 \cup \bar{X}_2 \subseteq V$. $\bar{X}_1$ is *independent from* $\bar{X}_2$ *given* C if, for any substitution $\theta_2 \in [C_{|\bar{X}_2}]$, $C_{|\bar{X}_1} \Leftrightarrow (C\theta_2)_{|\bar{X}_1}$. $\bar{X}_1$ and $\bar{X}_2$ are *independent given* C if $\bar{X}_1$ is independent from $\bar{X}_2$ given C and vice-versa. Two *atoms* $p_1(\bar{X}_1)$ and $p_2(\bar{X}_2)$ are *independent given* C if $\bar{X}_1$ and $\bar{X}_2$ are independent given C.

Finally, we define **multinomial counters**. Let a be a c-atom with domain D_a. Then the *multinomial counter of* a, $\vec{N}_a$, is a vector where $\vec{N}_{a,j}$ indicates how many instantiations of a are assigned the j-th value in D_a. The *multinomial coefficient* $\vec{N}_a! = \frac{(\vec{N}_{a,1}+\cdots+\vec{N}_{a,|D_a|})!}{\vec{N}_{a,1}!\ldots\vec{N}_{a,|D_a|}!}$ is a generalization of binomial coefficients and indicates how many assignments to $RV(a)$ exhibit the particular value distribution counted by $\vec{N}_a$.

Counters can be applied to *sets* of c-atoms with the same general meaning. The

set of multinomial counters for a set of c-atoms A is denoted $\vec{N}_A$, and the product $\prod_{a \in A} \vec{N}_a!$ of their multinomial coefficients is denoted $\vec{N}_A!$.

Theorem 1.1 Counting Elimination
Let g be a shattered parfactor and $E = \{E_1, \ldots, E_k\}$ be a subset of A_g such that $RV(E)$ is disjoint from $RV(A_g \setminus E)$, $A' = A_g \setminus E$ are all ground, and where each pair of atoms is independent given C_g. Then

$$\sum_{RV(E)} \prod_{\theta \in \Theta_g} \phi(A_g\theta) = \sum_{\vec{N}_E} \vec{N}_E! \prod_{v \in D_E} \phi(v, A')^{\prod_{i=1}^{k} \vec{N}_{E_i, v_i}}.$$

The theorem's proof reflects the argument given above. Counting elimination brings a significant computational advantage because iterating over assignments is exponential in $|RV(E)|$ while doing so over groups of assignments is only polynomial in it.

It is important to notice that E must contain all non ground c-atoms in g. Also, if all c-atoms in g are ground, E can be any subset of them and we will have a simple propositional summation, the same used in VE (counters over 1-random variable c-atoms reduce to ordinary assignments).

1.3.2 Inversion

Counting elimination requires a parfactor's atoms to be independent given its constraints. In particular, logical variables shared between atoms may render them dependent on each other. In some of these cases, the operation of *inversion* can be applied. In fact, even in cases in which counting elimination can be applied, it is advantageous to apply inversion first, if possible, for efficiency reasons.

Let us consider a couple of examples before we formalize inversion. Consider the following:

$$\sum_{RV(q(X,Y))} \prod_{XY} \phi(p(X), q(X,Y))$$
$$= \sum_{q(o_1,o_1)} \sum_{q(o_1,o_2)} \cdots \sum_{q(o_n,o_n)} \phi(p(o_1), q(o_1,o_1))\phi(p(o_1), q(o_1,o_2)) \ldots \phi(p(o_n), q(o_n,o_n))$$

(by observing that $\phi(p(o_i), q(o_i,o_j))$ depends on $q(o_i,o_j)$ only)

$$= \sum_{q(o_1,o_1)} \phi(p(o_1), q(o_1,o_1)) \cdots \sum_{q(o_n,o_n)} \phi(p(o_n), q(o_n,o_n))$$

(by observing that only $\phi(p(o_i), q(o_i,o_j))$ depends on $q(o_i,o_j)$)

$$= \Big(\sum_{q(o_1,o_1)} \phi(p(o_1), q(o_1,o_1)) \Big) \ldots \Big(\sum_{q(o_n,o_n)} \phi(p(o_n), q(o_n,o_n)) \Big)$$
$$= \prod_{XY} \sum_{q(X,Y)} \phi(p(X), q(X,Y))$$

(by observing that only the summation is the same for all $q(X,Y)$)

$$= \prod_{XY} \phi'(p(X)).$$

Inversion works by establishing a one-to-one correspondence between parfactor instantiations and summations. If the summation were on the instantiations of $p(X)$, such correspondence would not be possible because there would be less summations ($|X|$ of them) than parfactor instantiations ($|X| * |Y|$ of them).

Another condition for inversion is that the c-atom being inverted not have different instances in the same instance of the parfactor. For example, we cannot use inversion on $p(X,Y)$ for a parfactor on $p(X,Y), p(Y,X)$ because for any pair of objects o_i, o_j, neither of the parfactor instantiations $p(o_i,o_j)$ and $p(o_j,o_i)$ can be factored out of the innermost of $\sum_{p(o_i,o_j)}$ and $\sum_{p(o_j,o_i)}$. This breaks the one-to-one correspondence between summations and instantiated parfactors.

In the case above, the resulting inner summation was propositional. Inversion resulting in propositional summations were called *inversion elimination* in de Salvo Braz et al. [2005, 2006]. In the next example, the inner summation is one computed by counting elimination.

Suppose we want to calculate

$$\begin{aligned}
&\sum_{RV(p(X,Y))} \prod_{X,Y,Z} \phi(p(X,Y),p(X,Z)) \\
&= \sum_{RV(p(X,Y))} \prod_{X} \prod_{Y,Z} \phi(p(X,Y),p(X,Z)) \\
&= \sum_{RV(p(o_1,Y))} \cdots \sum_{RV(p(o_n,Y))} \prod_{Y,Z} \phi(p(o_1,Y),p(o_1,Z)) \cdots \prod_{Y,Z} \phi(p(o_n,Y),p(o_n,Z)) \\
&= \Big(\sum_{RV(p(o_1,Y))} \prod_{Y,Z} \phi(p(o_1,Y),p(o_1,Z)) \Big) \ldots \Big(\sum_{RV(p(o_n,Y))} \prod_{Y,Z} \phi(p(o_n,Y),p(o_n,Z)) \Big) \\
&= \prod_{X} \sum_{RV(p(X,Y))} \prod_{Y,Z} \phi(p(X,Y),p(X,Z))
\end{aligned}$$

Because X is now bound before the summation, it works as a constant (whose exact identity is irrelevant), and so it is not included in the counting argument. The counting argument now involves only Y and Z and is actually very similar to our original counting argument example. For that reason, the above is equal to $\prod_X \phi'()$, for ϕ' the result of counting elimination.

Inversions resulting in counting elimination problems only invert on a subset of the parfactor's logical variables. For this reason, they have been called *Partial inversions* in our previous work. Note however that, since propositional sums are a trivial case of counting elimination, both inversion operations can be unified into one. This is what we do in the formalization below.

1.3.2.1 Uniform Solution Counting Partition (USCP)

Before we present the theorem formalizing inversion, we touch a last issue. Consider the inversion of X resulting in the expression

$$\prod_{X} \sum_{RV(p(X,Y))} \prod_{Y \neq X, Z \neq X, Y \neq a, Z \neq a} \phi(p(X,Y), p(X,Z)).$$

The summation can be done by counting elimination since X is bound. However, it will depend on $|RV(p(X,Y))|$, but that depends on whether $X = a$ or not. One needs to split the expression according to cases $X = a$ and $X \neq a$:

$$\Big(\sum_{RV(p(a,Y))} \prod_{Y \neq a, Z \neq a} \phi(p(a,Y), p(a,Z)) \Big) \\ \times \Big(\prod_{X \neq a} \sum_{RV(p(X,Y))} \prod_{Y \neq X, Z \neq X, Y \neq a, Z \neq a} \phi(p(X,Y), p(X,Z)) \Big)$$

and then proceed as usual.

In general, one needs to consider the **uniform solution counting partition (USCP)** of the inverted logical variables with respect to an original constraint system. The USCP $U_L(C)$ of a set of logical variables L with respect to a constraint C is a set of constraints $\{C_1, \ldots, C_k\}$ such that $\{[C_i]\}_i$ forms a partition of $[C_{|L}]$ and

$$\forall i \cdot \forall \theta', \theta'' \in [C_i] \cdot |[C\theta']| = |[C\theta'']|,$$

that is, the number of solutions for the constraint conditioned on L is the same for each of the components C_i.

1.3.2.2 Inversion Formalization

Theorem 1.2 Inversion
Let g be a shattered parfactor, L a set of logical variables and E a set of c-atoms such that $RV(E)$ and $RV(A_g \setminus E)$ are disjoint. If

1. $\forall e_k, e_l \in E \cdot \forall \theta_i, \theta_j \in [C_{g|L}] \cdot \theta_i \neq \theta_j \Rightarrow RV(e_k\theta_i) \cap RV(e_l\theta_j) = \emptyset$.
2. $\forall \theta_i, \theta_j \in [C_{g|L}] \cdot \theta_i \neq \theta_j \Rightarrow RV(E\theta_i) \cap RV(E\theta_j) = \emptyset$.

then

$$\sum_{RV(E)} \Phi(g) = \prod_{C \in U_L(C_g)} \Phi(g_C),$$

where g_C is the parfactor $(\phi_{g'}, A_{g'}, C \wedge C_{g'})$ and using g' defined by the recursive computation $g'\theta = \sum_{RV(E\theta)} \Phi(g\theta)$, for θ an arbitrary element of $[C]$ (by the definition of USCP, it does not matter which).

Proof Let $E = \{e_1, \ldots, e_n\}$, $[C_{g|L}] = \{\theta_1, \ldots, \theta_m\}$. Below, we decompose C_g into

the part w.r.t. L and the remaining logical variables:

$$\begin{aligned}
&\sum_{RV(E)} \Phi(g) = \sum_{RV(E)} \prod_{\theta \in \Theta_g} \phi(A_g\theta) \\
&= \sum_{RV(E)} \prod_{\theta \in [C_{g|L}]} \prod_{\theta' \in [C_g\theta]} \phi(A_g\theta\theta') \\
&= \sum_{RV(e_1)} \cdots \sum_{RV(e_n)} \prod_{\theta \in [C_{g|L}]} \prod_{\theta' \in [C_g\theta]} \phi(A_g\theta\theta') \\
&= \sum_{RV(e_1\theta_1)} \cdots \sum_{RV(e_n\theta_1)} \cdots \sum_{RV(e_1\theta_m)} \cdots \sum_{RV(e_n\theta_m)} \big(\prod_{\theta' \in [C_g\theta_1]} \phi(A_g\theta_1\theta') \big) \ldots \big(\prod_{\theta' \in [C_g\theta_m]} \phi(A_g\theta_m\theta') \big) \\
&= \Big(\sum_{RV(e_1\theta_1)} \cdots \sum_{RV(e_n\theta_1)} \prod_{\theta' \in [C_g\theta_1]} \phi(A_g\theta_1\theta') \Big) \ldots \Big(\sum_{RV(e_1\theta_m)} \cdots \sum_{RV(e_n\theta_m)} \prod_{\theta' \in [C_g\theta_m]} \phi(A_g\theta_m\theta') \Big) \\
&= \prod_{\theta \in [C_{g|L}]} \sum_{RV(e_1\theta)} \cdots \sum_{RV(e_n\theta)} \Big(\prod_{\theta' \in [C_g\theta]} \phi(A_g\theta\theta') \Big) \\
&= \prod_{\theta \in [C_{g|L}]} \sum_{RV(E\theta)} \Big(\prod_{\theta' \in [C_g\theta]} \phi(A_g\theta\theta') \Big) \\
&= \prod_{C \in U_L(C_g)} \prod_{\theta \in [C]} \sum_{RV(E\theta)} \Phi(g\theta) = \prod_{C \in U_L(C_g)} \prod_{\theta \in [C]} \Phi(g_C\theta) = \prod_{C \in U_L(C_g)} \Phi(g_C).
\end{aligned}$$

Note that condition 1 is used to ensure the summations on $\sum_{RV(e_1\theta_1)} \cdots \sum_{RV(e_n\theta_m)}$ are indeed distinct. Condition 2 ensures that the innermost products are on distinct sets of random variables and can therefore be factored out as shown. ■

1.3.3 The Algorithm

Figure 1.1 shows the main pseudocode for FOVE. The algorithm consists of successively choosing eliminations $(E, \{L_1, \ldots, L_k\})$, consisting of a collection of atoms E to eliminate after performing a series of inversions based on sets $L_1, \ldots, L_k$ of logical variables. A possible way of choosing eliminations is presented in figure 1.2. It is presented separately from the main algorithm for clarity, but because these two phases have many operations in common, actual implementations will typically integrate them more tightly.

There are potentially many ways to choose eliminations. The one we present starts by choosing an atom and checking if its inversion will produce a propositional summation, since this is the most efficient case. If not, we successively add atoms to E until G_E forms a parfactor where all atoms with logical variables are part of E (because counting elimination requires it). Then, for efficiency and to avoid shared logical variables between atoms, we try to determine as many inversions as possible, coded in the sequence $L_1, \ldots, L_k$, to be done before counting elimination (or explicit summation when counting cannot be done).

FUNCTION *FOVE*(G, Q)
G a set of parfactors, $Q \subseteq RV(G)$, G shattered against Q (section 1.5.2).
1. If $RV(G) = Q$, return G.
2. $(E, \{L_1, \ldots, L_k\}) \leftarrow$ *CHOOSE-ELIMINATION*(G, Q).
3. $g_E \leftarrow fs(G_E)$ (fusion, section 1.5.1).
4. $G' \leftarrow$ *ELIMINATE*(g_E, E).
5. Return *FOVE*$(G' \cup G_{\neg E}, Q)$.

FUNCTION *ELIMINATE*$(g, E, \{L_1, \ldots, L_k\})$
1. If $k = 0$ (no inversion)
   return *SUMMATION-WITHOUT-INVERSION*(g, E).
2. $E_1 \leftarrow \{e \in E : LV(e) \cap L_1 \neq \emptyset\}$ (get inverted atoms).
3. Return $\bigcup_{C_1 \in U_L(C_g)}$ *ELIMINATE-GIVEN-UNIFORMITY*$(g, E_1, C_1, \{L_2, \ldots, L_k\})$.

FUNCTION *ELIMINATE-GIVEN-UNIFORMITY*$(g, E_1, C_1, \{L_2, \ldots, L_k\})$
1. Choose $\theta_1 \in [C_1]$ (bind inverted logical variables arbitrarily).
2. $G' \leftarrow$ *ELIMINATE*$(g\theta_1, E_1\theta_1, \{L_2, \ldots, L_k\})$.
3. $G'' \leftarrow \bigcup_{g'\theta_1 \in G'} (\phi_{g'}, A_{g'}, C_1 \wedge C_{g'})$.
4. Return $\bigcup_{g'' \in G''}$ *SIMPLIFICATION*(g'') (simplification, section 1.5.3).

FUNCTION *SUMMATION-WITHOUT-INVERSION*(g, E)
1. If $E = \{E_1, \ldots, E_k\}$ atoms are independent given C_g and $A_g \setminus E$ is ground
   return $(\sum_{\vec{N}_E} \vec{N}_E! \prod_v \phi_g(v, A_g \setminus E)^{\prod_{i=1}^{k} \vec{N}_{E_i, v_i}}, A_g \setminus E, \top)$ (counting, section 1.3.1).
2. Return $(\sum_{RV(E)} \prod_{\theta \in \Theta_g} \phi_g(A_g\theta_g), A_g \setminus E, C_g)$ (propositional elimination).

Notation:
- $LV(\alpha)$: logical variables in object α.
- $g\theta$: parfactor $(\phi_g, A_g\theta, C_g\theta)$.
- $U_L(C_g)$: USCP of L with respect to C_g (section 1.3.2.1).
- $C_{|L}$: constraints projected to a set of logical variables L.
- G_E: subset of parfactors G which depend on $RV(E)$.
- $G_{\neg E}$: subset of parfactors G which do not depend on $RV(E)$.
- $\top$: tautology constraint.

Figure 1.1 The FOVE algorithm.

1.4 An experiment

We use the implementation available at `http://l2r.cs.uiuc.edu/~cogcomp` to compare average run times between lifted and propositional inference (which produce the exact same results) for two different models while increasing the number of objects in the domain. The first one, (I) in figure 1.3, answers the query $P(p)$ from a parfactor on $(p, q(X))$ and uses inversion elimination only. The inference in (II) answers query $P(r)$ from a parfactor on $(p(X), p(Y), r)$ and uses counting elimination only. In both cases propositional inference starts taking very long before any noticeable variation in lifted inference run times.

FUNCTION *CHOOSE-ELIMINATION*(G, Q)
1. Choose e from $A_G \setminus Q$.
2. $g \leftarrow fs(G_e)$ (fusion, section 1.5.1).
3. If $LV(e) = LV(g)$ and $\forall e' \in A_g\ RV(e') \neq RV(e)$
   return $(\{e\}, LV(e))$ (inversion eliminable).
4. $E \leftarrow \{e\}$.
5. While $E \neq$ non-ground atoms of G_E
   $E \leftarrow E \cup$ non-ground atoms of G_E.
6. Return (E, *GET-SEQUENCE-OF-INVERSIONS*($fs(G_E)$)).

FUNCTION *GET-SEQUENCE-OF-INVERSIONS*(g)
1. If there is no L_1 set of invertible logical variables in g (inversion, section 1.3.2)
   return $\emptyset$.
2. Choose $\theta_1 \in [C_{g|L_1}]$.
3. $\{L2, \ldots, L_k\} \leftarrow$ *GET-SEQUENCE-OF-INVERSIONS*($g\theta_1$).
4. Return $\{L_1, L_2, \ldots, L_k\}$.

Figure 1.2 One possible way of choosing an elimination.

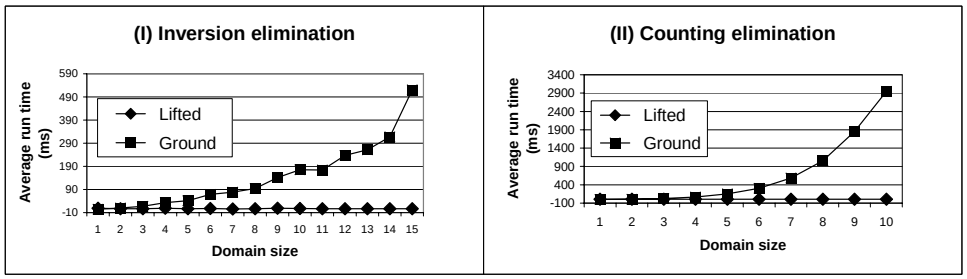

Figure 1.3 (I) Average run time for answering query $P(p)$ from a parfactor on $(p, q(X))$, using inversion elimination, with domain size $|X|$ being gradually increased. (II) Average run time for answering query $P(r)$ from a parfactor on $(p(X), p(Y), r)$, using counting elimination, with domain size $|X| = |Y|$ being gradually increased.

1.5 Auxiliary operations

1.5.1 Fusion

We have assumed in section 1.3 that we have operations to calculate $\sum_{RV(E)} \Phi(G_E)$, but elimination operations calculate $\sum_{RV(E)} \Phi(g)$, for g a *single* parfactor. *Fusion* bridges this gap by computing, for any set of parfactors G, a single parfactor $fs(G)$ such that $\Phi(G) = \Phi(fs(G))$.

Fusion works by replacing the constraints of all parfactor in the set by a single, common constraint which is the conjunction of them all. This guarantees that all parfactors get instantiated by the same set of substitutions on a single set of logical

variables, which allows their products (in the expression for $\Phi(G)$) to be unified under a single product. Note that not all parfactors contain all the logical variables, and will be instantiated to the same ground factor by distinct substitutions (those agreeing on the logical variables present in the parfactor, but disagreeing on some of the others). In other words, some of the parfactors will have their number of instantiatiations increased by this unification. For this reason, we also exponentiate the potential function to the inverse of how many times the number of instantiations was increased, keeping the final result the same as before.

This is illustrated in the example below:

$$\begin{aligned}\big(\prod_{D,P}\phi_1(e(D),s(D,P))\big)\big(\prod_{D'}\phi_2(e(D'))\big) &= \prod_{D,P,D'}\phi_1(e(D),s(D,P))\phi_2^{\frac{1}{|D,P|}}(e(D')) \\ &= \prod_{D,P,D'}\phi_3(e(D),s(D,P),e(D')).\end{aligned}$$

(note that logical variables in different parfactors must be standardized apart.) Formally, we have the fusion theorem below.

Theorem 1.3 Fusion
Let G be a set of parfactors. Define $C_G = \bigwedge_{g\in G} C_g$, $\Theta_G = [C_G]$ and $A_G = \bigcup_{g\in G} A_g$. Let $fs(G)$ be the parfactor $(\prod_{g\in G}\phi_g^{|\Theta_g|/|\Theta_G|}, A_G, C_G)$. Then $\Phi(G) = \Phi(fs(G))$.

Proof

$$\begin{aligned}\Phi(G) &= \prod_{g\in G}\prod_{\theta\in\Theta_g}\phi_g(A_g\theta) = \prod_{g\in G}\prod_{\theta\in\Theta_G}\phi_g(A_g\theta)^{|\Theta_g|/|\Theta_G|} \\ &= \prod_{\theta\in\Theta_G}\prod_{g\in G}\phi_g(A_g\theta)^{|\Theta_g|/|\Theta_G|} = \prod_{\theta\in\Theta_G}\phi_{fs(G)}(A_G\theta) = \Phi(fs(G))\end{aligned}$$

■

While the above is correct, it is rather unnatural to have $e(D)$ and $e(D')$ be distinct atoms. If a set of logical variables has the same possible substitutions, like D and D' here, we can do something better:

$$\begin{aligned}\Big(\prod_{D,P}\phi_1(e(D),s(D,P))\Big)\Big(\prod_{D'}\phi_2(e(D'))\Big) &= \Big(\prod_{D}\prod_{P}\phi_1(e(D),s(D,P))\Big)\Big(\prod_{D'}\phi_2(e(D'))\Big) \\ &= \prod_{D''}\Big(\big(\prod_{P}\phi_1(e(D''),s(D'',P))\big)\big(\phi_2(e(D''))\big)\Big) \\ &= \prod_{D''}\Big(\prod_{P}\phi_1(e(D''),s(D'',P))\phi_2(e(D''))^{\frac{1}{|P|}}\Big) \\ &= \prod_{D'',P}\phi_1(e(D''),s(D'',P))\phi_2^{\frac{1}{|P|}}(e(D'')) \\ &= \prod_{D'',P}\phi_3(e(D''),s(D'',P)).\end{aligned}$$

Formally, this process is similar to inversion with respect to D''. However, it does require the additional previous step of unifying distinct logical variables (but with identical sets of possible substitutions) into a single one first (in the example, D and D' are replaced by D''). For lack of space we omit the details of this improvement.

1.5.2 Shattering

In section 1.3 we mentioned the need for *shattering*, which we now discuss in more detail. This need arises from c-atoms representing overlapping, but not identical, classes of random variables. Consider the following marginalization over parfactors g_1 and g_2 with potential functions ϕ_1 and ϕ_2 respectively:

$$\sum_{RV(p(X,Y))} \Big(\prod_{X,Y} \phi_1(p(X,Y),q) \Big) \prod_{Y} \phi_2(p(a,Y))$$

If we pick $E = p(a,Y)$, $G_E = \{g_1, g_2\}$. However, only some instantiations of g_1 depend on $p(a,Y)$ (the ones with $X = a$). Moreover, the operations we later talk about require any pair of c-atoms in G_E to represent either identical classes of random variables, or those classes to be disjoint. This is violated by $RV(p(a,Y))$ being a subset of $RV(p(X,Y))$. Picking $E = p(X,Y)$ also violates this requirement.

The solution is to **split** parfactor g_1 into two different parfactors. The union of instantiations of $(\phi_1, (p(X,Y),q), X \neq a)$ and $(\phi_1, (p(a,Y),q), \top)$ is identical to the set of instantiations of g_1, so the summation can be simply rewritten as

$$\begin{aligned}
&\sum_{RV(p(X,Y))} \Big(\prod_{X,Y:X\neq a} \phi_1(p(X,Y),q) \Big) \Big(\prod_{Y} \phi_1(p(a,Y),q) \Big) \prod_{Y} \phi_2(p(a,Y)) \\
&= \sum_{RV(p(X,Y):X\neq a)} \Big(\prod_{X,Y:X\neq a} \phi_1(p(X,Y),q) \Big) \sum_{p(a,Y)} \Big(\prod_{Y} \phi_1(p(a,Y),q) \Big) \prod_{Y} \phi_2(p(a,Y))
\end{aligned}$$

Now $E = p(a,Y)$ satisfies the operations' requirements. Picking $E = p(X,Y), X \neq a$ would work equally well.

Splitting parfactors is done by pairwise comparisons of atoms of the same predicate. We split parfactors $g_1 = (\phi_1, A_1, C_1)$ and $g_2 = (\phi_2, A_2, C_2)$ around atoms $a_1 \in A_1$ and $a_2 \in A_2$ by replacing them by parfactors $(\phi_1, A_1, C_1 \wedge a_1 = a_2)$, $(\phi_1, A_1, C_1 \wedge a_1 \neq a_2)$, $(\phi_2, A_2, C_2 \wedge a_1 = a_2)$ and $(\phi_2, A_2, C_2 \wedge a_1 \neq a_2)$, after standardizing apart their logical variables. In fact, we only need to keep those whose constraint is satisfiable. (This is why g_2 does not need to be broken in the example above – that would only produce itself and another parfactor with zero instantiations.) In particular, if $RV(a_1) = RV(a_2)$, we end up obtaining the original parfactors.

The uniformity requirement is met by **shattering** the FOPM in advance, that is, by successively splitting the parfactors of each pair of c-atoms, including the query atoms, until no overlapping non-identically grounded pair remains. The query atoms need to be involved in shattering because if a c-atom includes query and non-query

random variables, it needs to be split so that the non-query ones can be eliminated.

As pointed out by Poole [2003], splitting parfactors resembles the role of unification in first-order resolution, which determines the conditions for two atoms to match. In probabilistic inference, however, we are interested not only in the overlapping of atoms but also in the *residual* parfactors that originate from the matching. The reason for this difference is that the *number* of instantiations of a parfactor matters for the final joint distribution. In regular resolution, the original clauses are kept because their redundancy with the clauses resulting from resolution makes no difference, while here we need to discount them and replace the originals with the non-matching cases.

1.5.3 Irrelevant Logical Variable Simplification

Inversion often produces parfactors with constraints with logical variables not present in its atoms. The first inversion example produces the expression below. We can simplify it by observing that the actual value of Y is irrelevant inside the product. Only the number $|Y|$ of possible values for Y will make a difference. Therefore we can write

$$\prod_{XY} \phi'(p(X)) = \prod_{X} \phi'(p(X))^{|Y|} = \prod_{X} \phi''(p(X)).$$

1.6 Applicability of lifted inference

As explained in the previous sections, the lifted operations of FOVE are not always applicable, each of them requiring certain conditions to be satisfied in advance. Therefore a natural question is to what kinds of FOPMs we can apply FOVE in an exclusively lifted manner.

It is not clear at this point whether it is possible to tell in advance if a FOPM can be solved with lifted operations alone. The main reason for this is that lifted operations will be applied to parfactors resulting from previous operations, so we do not know them in advance. It may be that two parfactors satisfying the lifted operations conditions fuse to form one which does not. (This is similar to the fact that an elimination ordering is not computed in advance but only as the algorithm proceeds.)

As a summarization, the conditions for applying lifted operations to eliminate a set $RV(E)$ from a parfactor g are the following: for counting elimination, the atoms in g must be independent given its C_g; for inversion on $L \subseteq LV(g)$,

1. $\forall e_k, e_l \in E \cdot \forall \theta_i, \theta_j \in [C_{g|L}] \cdot \theta_i \neq \theta_j \Rightarrow RV(e_k\theta_i) \cap RV(e_l\theta_j) = \emptyset$.
2. $\forall \theta_i, \theta_j \in [C_{g|L}] \cdot \theta_i \neq \theta_j \Rightarrow RV(E\theta_i) \cap RV(E\theta_j) = \emptyset$.

When lifted operations do not apply, FOVE uses non-lifted operations to calculate $\sum_{RV(E)} G_E$. These non-lifted methods could be propositionalization, sampling etc,

but with the advantage of being restricted to a subset of the model only.

1.7 Future Directions

There are several possible directions for further development of FOVE. One of the main ones is the incorporation of function symbols, both random (the color of an object, for example) and interpreted (summation over integers), which will greatly increase its expressivity and applicability.

In applications involving evidence over many objects (for example, the facts about all the words in an English document), shattering may take a long time because all parfactors have to be checked against it. The large number of objects involved may create the need for numerous parfactor splittings. This is unfortunate because often only some objects are truly relevant to the query. For example, analyzing only some words and phrases in a document will often be enough to determine its subject. Therefore a variant of FOVE that does only the necessary shattering, guided by the inference process, is of great interest.

Finally, lifted FOVE operations do not cover all possible cases and explicit summation may be required at times, so increasing their coverage is an important direction.

1.8 Conclusion

Intuitive descriptions of models very often include first-order elements. When these models are probabilistic, the dominant approach has been that of grounding the model to a propositional one and solving it with a regular propositional algorithm. This strategy loses the explicit representation of the model's first-order structure, which can be used to great computational advantage, and which is computationally hard to retrieve from the grounded model.

We presented FOVE, a first-order generalization of the popular VE propositional inference algorithm. Like VE, FOVE successively eliminates random variables from the model by summing them out while taking advantage of independences for efficiency. Unlike VE, FOVE directly manipulates first-order representations, eliminating c-atoms that stand for potentially large sets of random variables at once. This can in some cases exponentially (in the domain size) speed inference up.

There are important directions in which FOVE needs to be extended, such as incorporating function symbols, avoiding unnecessary shattering, and extending operations for as of yet uncovered cases. However FOVE is already applicable, and especially useful, in domains with large objects about which we have identical knowledge. More than that, it is a general framework to be expanded and help close the gap between logic and probabilistic reasoning.

Acknowledgments

This work was partly supported by Cycorp in relation to the Cyc technology, the Advanced Research and Development Activity (ARDA)'s Advanced Question Answering for Intelligence (AQUAINT) program, NSF grant ITR-IIS- 0085980, and a Defense Advanced Research Projects Agency (DARPA) grant HR0011-05-1-0040.

References

V. S. Costa, D. Page, M. Qazi, and J. Cussens. CLP(BN): Constraint logic programming for probabilistic knowledge. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 517–524, 2003.

R. de Salvo Braz, E. Amir, and D. Roth. Lifted first-order probabilistic inference. In *Proceedings of IJCAI-05, 19th International Joint Conference on Artificial Intelligence*, 2005.

R. de Salvo Braz, E. Amir, and D. Roth. Mpe and partial inversion in lifted probabilistic variable elimination. In *Proceedings of the 21sth National Conference on Artificial Intelligence (AAAI)*, 2006.

N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309, 1999.

J. Y. Halpern. An analysis of first-order logics of probability. In *Proceedings of IJCAI-89, 11th International Joint Conference on Artificial Intelligence*, pages 1375–1381, Detroit, US, 1990.

K. Kersting and L. De Raedt. Bayesian logic programs. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 138–155, 2000.

L. Ngo and P. Haddawy. Probabilistic logic programming and Bayesian networks. In *Asian Computing Science Conference*, pages 286–300, 1995.

J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, San Mateo (Calif.), 1988.

D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

D. Poole. First-order probabilistic inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 985–991, 2003.

M. Richardson and P. Domingos. Markov logic networks. Technical report, Department of Computer Science, University of Washington, 2004.

N. L. Zhang and D. Poole. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Biennial Canadian Artificial Intelligence Conference*,

1994.

Contributors

Pieter Abbeel
Computer Science Department
Stanford University
abbeel@cs.stanford.edu

Eyal Amir
Department of Computer Science
University of Illinois at Urbana-Champaign
eyal@cs.uiuc.edu

Rodrigo de Salvo Braz
Department of Computer Science
University of Illinois at Urbana-Champaign
braz@uiuc.edu

Razvan C. Bunescu
Department of Computer Sciences
University of Texas at Austin
razvan@cs.utexas.edu

Elizabeth Burnside
Department of Radiology
Department of Biostatistics and Medical Informatics
University of Wisconsin - Madison
es.burnside@hosp.wisc.edu

Vítor Santos Costa
COPPE/Sistemas
Universidade Federal do Rio de Janeiro, Brazil
vitor@cos.ufrj.br

James Cussens
Department of Computer Science &
York Centre for Complex Systems Analysis
University of York, UK

jc@cs.york.ac.uk

Jesse Davis
Department of Computer Science
University of Wisconsin - Madison
jdavis@cs.wisc.edu

Luc De Raedt
Institute for Computer Science, Machine Learning Lab
Albert-Ludwigs-Universität Freiburg, Germany
deraedt@informatik.uni-freiburg.de

Pedro Domingos
Department of Computer Science and Engineering
University of Washington
pedrod@cs.washington.edu

Inês Dutra
COPPE/Sistemas
Universidade Federal do Rio de Janeiro, Brazil
ines@cos.ufrj.br

Sašo Džeroski
Department of Knowledge Technologies
Jožef Stefan Institute, Slovenija
Saso.Dzeroski@ijs.si

Nir Friedman
School of Computer Science and Engineering
Hebrew University, Israel
nir@cs.huji.ac.il

Lise Getoor
Computer Science Department
University of Maryland
getoor@cs.umd.edu

David Heckerman
Microsoft Research Redmond
heckerma@microsoft.com

David Jensen
Computer Science Department
University of Massachusetts, Amherst
jensen@cs.umass.edu

Kristian Kersting

Institute for Computer Science, Machine Learning Lab
Albert-Ludwigs-Universität Freiburg, Germany
kersting@informatik.uni-freiburg.de

Daphne Koller
Computer Science Department
Stanford University
koller@cs.stanford.edu

Andrey Kolobov
Computer Science Division
University of California at Berkeley
karayaone@rambler.ru

Andrew McCallum
Department of Computer Science
University of Massachusetts, Amherst
mccallum@cs.umass.edu

Chris Meek
Microsoft Research Redmond
meek@microsoft.com

Bhaskara Marthi
Computer Science Division
University of California at Berkeley
bhaskara@cs.berkeley.edu

Brian Milch
Computer Science Division
University of California at Berkeley
milch@cs.berkeley.edu

Raymond J. Mooney
Department of Computer Sciences
University of Texas at Austin
mooney@cs.utexas.edu

Stephen Muggleton
Department of Computing,
Imperial College London, UK
shm@doc.ic.ac.uk

Jennifer Neville
Computer Science Department
University of Massachusetts, Amherst

jneville@cs.umass.edu

Daniel L. Ong
Computer Science Division
University of California at Berkeley
dlong@ocf.berkeley.edu

David Page
Department of Biostatistics and Medical Informatics
University of Wisconsin - Madison
page@biostat.wisc.edu

Niels Pahlavi
Department of Computing,
Imperial College London, UK

Avi Pfeffer
Division of Engineering and Applied Sciences
Harvard University
avi@eecs.harvard.edu

Alexandrin Popescul
Department of Computer and Information Science
University of Pennsylvania
popescul@cis.upenn.edu

Raghu Ramakrishnan
Department of Computer Science
University of Wisconsin - Madison
raghu@cs.wisc.edu

Matthew Richardson
Microsoft Research Redmond
mattri@microsoft.com

Dan Roth
Department of Computer Science
University of Illinois at Urbana-Champaign
danr@uiuc.edu

Stuart Russell
Computer Science Division
University of California at Berkeley
russell@cs.berkeley.edu

Jude Shavlik
Department of Computer Science

University of Wisconsin - Madison
shavlik@cs.wisc.edu

David Sontag
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
dsontag@csail.mit.edu

Charles Sutton
Department of Computer Science
University of Massachusetts, USA
casutton@cs.umass.edu

Ben Taskar
Department of Computer and Information Science
University of Pennsylvania
taskar@cis.upenn.edu

Lyle H. Ungar
Department of Computer and Information Science
University of Pennsylvania
ungar@cis.upenn.edu

Ming-Fai Wong
Computer Science Department
Stanford University
mingfai.wong@cs.stanford.edu

Wen-tau Yih
Machine Learning and Applied Statistics Group
Microsoft Research Redmond
scottyih@microsoft.com

Index

Please find the index at `http://srl.cs.umd.edu/index.htm`