Probabilistic Theorem Proving

By Vibhav Gogate and Pedro Domingos

Abstract

Many representation schemes combining first-order logic and probability have been proposed in recent years. Progress in unifying logical and probabilistic inference has been slower. Existing methods are mainly variants of lifted variable elimination and belief propagation, neither of which take logical structure into account. We propose the first method that has the full power of both graphical model inference and first-order theorem proving (in finite domains with Herbrand interpretations). We first define probabilistic theorem proving (PTP), their generalization, as the problem of computing the probability of a logical formula given the probabilities or weights of a set of formulas. We then show how PTP can be reduced to the problem of lifted weighted model counting, and develop an efficient algorithm for the latter. We prove the correctness of this algorithm, investigate its properties, and show how it generalizes previous approaches. Experiments show that it greatly outperforms lifted variable elimination when logical structure is present. Finally, we propose an algorithm for approximate PTP, and show that it is superior to lifted belief propagation.

1. INTRODUCTION

Unifying first-order logic and probability enables uncertain reasoning over domains with complex relational structure, and is a long-standing goal of AI. Proposals go back to at least Nilsson,[17] with substantial progress within the community that studies uncertainty in AI starting in the 1990s (e.g., Bacchus,[1] Halpern,[14] Wellman[25]), and added impetus from the new field of statistical relational learning starting in the 2000s.[11] Many well-developed representations now exist (e.g., DeRaedt,[7] and Domingos[10]), but the state of inference is less advanced. For the most part, inference is still carried out by converting models to propositional form (e.g., Bayesian networks) and then applying standard propositional algorithms. This typically incurs an exponential blowup in the time and space cost of inference, and forgoes one of the chief attractions of first-order logic: the ability to perform *lifted* inference, that is, reason over large domains in time independent of the number of objects they contain, using techniques like resolution theorem proving.[20]

In recent years, progress in lifted probabilistic inference has picked up. An algorithm for lifted variable elimination was proposed by Poole[18] and extended by de Salvo Braz[8] and others. Lifted belief propagation was introduced by Singla and Domingos.[24] These algorithms often yield impressive efficiency gains compared to propositionalization, but still fall well short of the capabilities of first-order theorem proving, because they ignore logical structure, treating potentials as black boxes. This paper proposes the first full-blown probabilistic theorem prover that is capable of exploiting both lifting and logical structure, and includes standard theorem proving and graphical model inference as special cases.

Our solution is obtained by reducing probabilistic theorem proving (PTP) to lifted weighted model counting. We first do the corresponding reduction for the propositional case, extending previous work by Sang et al.[22] and Chavira and Darwiche.[3] We then lift this approach to the first-order level, and refine it in several ways. We show that our algorithm can be exponentially more efficient than first-order variable elimination, and is never less efficient (up to constants). For domains where exact inference is not feasible, we propose a sampling-based approximate version of our algorithm. Finally, we report experiments in which PTP greatly outperforms first-order variable elimination and belief propagation, and discuss future research directions.

2. LOGIC AND THEOREM PROVING

We begin with a brief review of propositional logic, first-order logic and theorem proving.

The simplest formulas in propositional logic are *atoms*: individual symbols representing propositions that may be true or false in a given world. More complex formulas are recursively built up from atoms and the logical connectives ¬(negation), ∧ (conjunction), ∨ (disjunction), ⇒ (implication) and ⇔ (equivalence). For example, ¬A ∨ (B ∧ C) is true iff A is false or B and C are true. A *knowledge base (KB)* is a set of logical formulas. The fundamental problem in logic is determining *entailment*, and algorithms that do this are called *theorem provers*. A knowledge base K entails a query formula Q iff Q is true in all worlds in which all formulas in K are true, a *world* being an assignment of truth values to all atoms.

A world is a *model* of a KB iff the KB is true in it. Theorem provers typically first convert K and Q to *conjunctive normal form (CNF)*. A CNF formula is a conjunction of *clauses*, each of which is a disjunction of *literals*, each of which is an atom or its negation. For example, the CNF of ¬A ∨ (B ∧ C) is (¬A ∨ B) ∧ (¬A ∨ C). A *unit clause* consists of a single literal. Entailment can then be computed by adding ¬Q to K and determining whether the resulting KB K_Q is *satisfiable*, that is, whether there exists a world where all clauses in K_Q are true. If not, K_Q is unsatisfiable, and K entails Q. Algorithm 1 shows this basic theorem proving schema. CNF(K) converts K to CNF, and SAT(C) returns True if C is satisfiable and False otherwise.

Algorithm 1 TP(KB K, query Q)

$K_Q \leftarrow K \cup \{\neg Q\}$
return ¬SAT(CNF(K_Q))

The earliest theorem prover is the Davis–Putnam algorithm (henceforth called DP).[6] It makes use of the *resolution* rule: if a KB contains the clauses $A_1 \vee \ldots \vee A_n$ and $B_1 \vee \ldots \vee B_m$, where the A's and B's represent literals, and some literal A_i is the negation of some literal B_j, then the clause $A_1 \vee \ldots \vee A_{i-1} \vee A_{i+1} \vee \ldots \vee A_n \vee B_1 \vee \ldots \vee B_{j-1} \vee B_{j+1} \vee \ldots \vee B_m$ can be added to the KB. For each atom A in the KB, DP resolves every pair of clauses C_1, C_2 in the KB such that C_1 contains A and C_2 contains ¬A, and adds the result to the KB. It then deletes all clauses that contain a literal of A from the KB. If at some point the empty clause is derived, the KB is unsatisfiable, and the query formula (previously negated and added to the KB) is therefore proven to be entailed by the KB. DP is in fact just the variable elimination algorithm for the special case of 0-1 potentials.

Modern propositional theorem provers use the DPLL algorithm,[5] a variant of DP that replaces the elimination step with a *splitting* step: instead of eliminating all clauses containing the chosen atom A, resolve all clauses in the KB with A, simplify and recurse, and do the same with ¬A. If both recursions fail, the KB is unsatisfiable. DPLL has linear space complexity, compared to exponential for Davis–Putnam, and is the basis of the algorithms in this paper.

First-order logic inherits all the features of propositional logic, and in addition allows atoms to have internal structure. An atom is now a predicate symbol, representing a relation in the domain of interest, followed by a parenthesized list of variables and/or constants, representing objects. For example, Friends(Anna, x) is an atom. A *ground atom* has only constants as arguments. First-order logic has two additional connectives, $\forall$ (universal quantification) and $\exists$ (existential quantification). For example, $\forall x$ Friends(Anna, x) means that Anna is friends with everyone, and $\exists x$ Friends(Anna, x) means that Anna has at least one friend. In this paper, we 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).

As long as the domain is finite, first-order theorem proving can be carried out by *propositionalization*: creating atoms from all possible combinations of predicates and constants, and applying a propositional theorem prover. However, this is potentially very inefficient. A more sophisticated alternative is first-order resolution,[20] which proceeds by resolving pairs of clauses and adding the result to the KB until the empty clause is derived. Two first-order clauses can be resolved if they contain complementary literals that *unify*, that is, there is a *substitution* of variables by constants or other variables that makes the two literals identical up to the negation sign. Conversion to CNF is carried out as before, with the additional step of removing all existential quantifiers by a process called *skolemization*.

First-order logic allows knowledge to be expressed more concisely than propositional logic. For example, the rules of chess can be stated in a few pages in first-order logic, but require hundreds of thousands in propositional logic. Probabilistic logical languages extend this power to uncertain domains. The goal of this paper is to similarly extend the power of first-order theorem proving.

3. PROBLEM DEFINITION

Following Nilsson,[17] we define PTP as the problem of determining the probability of an arbitrary query formula Q given a set of logical formulas F_i and their probabilities $P(F_i)$. For the problem to be well defined, the probabilities must be consistent, and Nilsson[17] provides a method for verifying consistency. Probabilities estimated by maximum likelihood from an observed world are guaranteed to be consistent. In general, a set of formula probabilities does not specify a complete joint distribution over the atoms appearing in them, but one can be obtained by making the *maximum entropy* assumption: the distribution contains no information beyond that specified by the formula probabilities.[17] Finding the maximum entropy distribution given a set of formula probabilities is equivalent to learning a maximum-likelihood log-linear model whose features are the formulas; many algorithms for this purpose are available (iterative scaling, gradient descent, etc.).

We call a set of formulas and their probabilities together with the maximum entropy assumption a *probabilistic knowledge base (PKB)*. Equivalently, a PKB can be directly defined as a log-linear model with the formulas as features and the corresponding weights or potential values. Potentials are the most convenient form, since they allow determinism (0-1 probabilities) without recourse to infinity. If $\mathbf{x}$ is a world and $\Phi_i(\mathbf{x})$ is the potential corresponding to formula F_i, by convention (and without loss of generality) we let $\Phi_i(\mathbf{x}) = 1$ if F_i is true, and $\Phi_i(\mathbf{x}) = \phi_i \geq 0$ if the formula is false. *Hard* formulas have $\phi_i = 0$ and *soft* formulas have $\phi_i > 0$. In order to compactly subsume standard probabilistic models, we interpret a universally quantified formula as a set of features, one for each grounding of the formula, as in Markov logic.[10] A PKB $\{(F_i, \phi_i)\}$ thus represents the joint distribution

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_i \phi_i^{n_i(\mathbf{x})} \qquad (1)$$

where $n_i(\mathbf{x})$ is the number of false groundings of F_i in $\mathbf{x}$, and Z is a normalization constant (the *partition function*). We can now define PTP succinctly as follows:

Probabilistic theorem proving (PTP)
Input: Probabilistic KB K and query formula Q
Output: $P(Q|K)$

If all formulas are hard, a PKB reduces to a standard logical KB. Determining whether a KB K logically entails a query Q is equivalent to determining whether $P(Q|K) = 1$.[10] Graphical models can be easily converted into equivalent PKBs.[3] Conditioning on evidence is done by adding the corresponding hard ground atoms to the PKB, and the conditional marginal of an atom is computed by issuing the atom as the query. Thus PTP has both logical theorem proving and inference in graphical models as special cases.

In this paper, we solve PTP by reducing it to lifted weighted model counting. *Model counting* is the problem of determining the number of worlds that satisfy a KB. *Weighted* model counting can be defined as follows.[3] Assign a weight to each literal, and let the weight of a world be the product of the weights of the literals that are true in it. Then weighted

model counting is the problem of determining the sum of the weights of the worlds that satisfy a KB:

Weighted model counting (WMC)
Input: CNF C and set of literal weights W
Output: Sum of weights of worlds that satisfy C

Figure 1 depicts graphically the set of inference problems addressed by this paper. Generality increases in the direction of the arrows. We first propose an algorithm for propositional weighted model counting and then lift it to the first-order level. The resulting algorithm is applicable to all the problems in the diagram. (Weighted SAT/MPE inference requires replacing sums with maxes with an additional traceback or backtracking step, but we do not pursue it here and leave it for future work.)

4. PROPOSITIONAL CASE

This section generalizes the Bayesian network inference techniques in Darwiche[4] and Sang et al.[22] to arbitrary propositional PKBs, evidence, and query formulas. Although this is of value in its own right, its main purpose is to lay the groundwork for the first-order case.

The probability of a formula is the sum of the probabilities of the worlds that satisfy it. Thus to compute the probability of a formula Q given a PKB K it suffices to compute the partition function of K with and without Q added as a hard formula:

$$P(Q|K) = \frac{\sum_{\mathbf{x}} 1_Q(\mathbf{x}) \prod_i \Phi_i(\mathbf{x})}{Z(K)} = \frac{Z(K \cup \{(Q,0)\})}{Z(K)} \quad (2)$$

where $1_Q(\mathbf{x})$ is the indicator function (1 if Q is true in $\mathbf{x}$ and 0 otherwise).

In turn, the computation of partition functions can be reduced to weighted model counting using the procedure in Algorithm 2. This replaces each soft formula F_i in K by a corresponding hard formula $F_i \Leftrightarrow A_i$, where A_i is a new atom, and assigns to every $\neg A_i$ literal a weight of ϕ_i (the value of the potential Φ_i when F_i is false).

Figure 1. Inference problems addressed in this paper. TP_0 and TP_1 is propositional and first-order theorem proving respectively, PI is probabilistic inference (computing marginals), MPE is computing the most probable explanation, SAT is satisfiability, MC is model counting, W is weighted and L is lifted. A = B means A can be reduced to B.
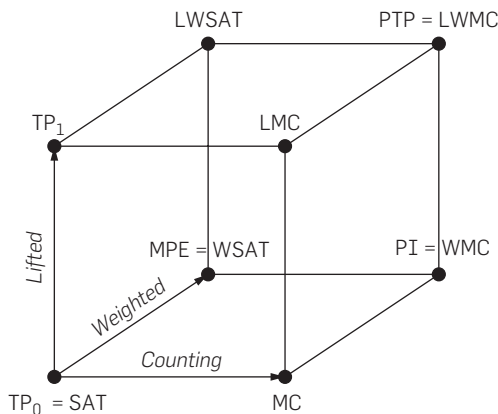
THEOREM 1. $Z(K) = WMC(WCNF(K))$.

PROOF. If a world violates any of the hard clauses in K or any of the $F_i \Leftrightarrow A_i$ equivalences, it does not satisfy C and is therefore not counted. The weight of each remaining world $\mathbf{x}$ is the product of the weights of the literals that are true in $\mathbf{x}$. By the $F_i \Leftrightarrow A_i$ equivalences and the weights assigned by WCNF(K), this is $\prod_i \Phi_i(\mathbf{x})$. (Recall that $\Phi_i(\mathbf{x}) = 1$ if F_i is true in $\mathbf{x}$ and $\Phi_i(\mathbf{x}) = \phi_i$ otherwise.) Thus $\mathbf{x}$'s weight is the unnormalized probability of $\mathbf{x}$ under K. Summing these yields the partition function $Z(K)$. $\square$

Equation 2 and Theorem 1 lead to Algorithm 3 for PTP. (Compare with Algorithm 1.) WMC(C, W) can be any weighted model counting algorithm.[3] Most model counters are variations of Relsat, itself an extension of DPLL.[2] Relsat splits on atoms until the CNF is decomposed into sub-CNFs sharing no atoms, and recurses on each sub-CNF. This decomposition is crucial to the efficiency of the algorithm. In this paper we use a weighted version of Relsat, shown in Algorithm 4. $\mathbf{A}(C)$ is the set of atoms that appear in C. $C|A$ denotes the CNF obtained by removing the literal A and its negation $\neg A$ from all clauses in which they appear in and setting to *Satisfied* all clauses in which A appears in. Notice that, unlike in DPLL, satisfied clauses cannot simply be deleted, because we need to keep track of which atoms they are over for model counting purposes. However, they can be ignored in the decomposition step, since they introduce no dependencies. The atom to split on in the splitting step can be chosen using various heuristics.[23]

Algorithm 2 WCNF(PKB K)

---

for all $(F_i, \phi_i) \in K$ s.t. $\phi_i > 0$ **do**
　　$K \leftarrow K \cup \{(F_i \Leftrightarrow A_i, 0)\} \setminus \{(F_i, \phi_i)\}$
$C \leftarrow \text{CNF}(K)$
for all $\neg A_i$ literals **do** $W_{\neg A_i} \leftarrow \phi_i$
for all other literals L **do** $W_L \leftarrow 1$
return (C, W)

Algorithm 3 PTP(PKB K, query Q)

$K_Q \leftarrow K \cup \{(Q, 0)\}$
return WMC(WCNF(K_Q))/WMC(WCNF(K))

Algorithm 4 WMC(CNF C, weights W)

// *Base case*
if all clauses in C are satisfied **then**
　　return $\prod_{A \in \mathbf{A}(C)} (W_A + W_{\neg A})$
if C has an empty unsatisfied clause **then return** 0
// *Decomposition step*
if C can be partitioned into CNFs $C_1, \ldots, C_k$ sharing no atoms **then**
　　return $\prod_{i=1}^{k} \text{WMC}(C_i, W)$
// *Splitting step*
Choose an atom A
return $W_A \text{WMC}(C|A; W) + W_{\neg A} \text{WMC}(C|\neg A; W)$

THEOREM 2. *Algorithm WMC(C,W) correctly computes the weighted model count of CNF C under literal weights W.*

PROOF SKETCH. If all clauses in C are satisfied, all assignments to the atoms in C satisfy it, and the corresponding total weight is $\prod_{A\in A(C)}(W_A + W_{\neg A})$. If C has an empty unsatisfied clause, it is unsatisfiable given the truth assignment so far, and the corresponding weighted count is 0. If two CNFs share no atoms, the WMC of their conjunction is the product of the WMCs of the individual CNFs. Splitting on an atom produces two disjoint sets of worlds, and the total WMC is therefore the sum of the WMCs of the two sets, weighted by the corresponding literal's weight. □

5. FIRST-ORDER CASE
We now lift PTP to the first-order level. We consider first the case of PKBs without existential quantifiers. Algorithms 2 and 3 remain essentially unchanged, except that formulas, literals and CNF conversion are now first-order. In particular, for Theorem 1 to remain true, each new atom A_i in Algorithm 2 must now consist of a new predicate symbol followed by a parenthesized list of the variables and constants in the corresponding formula F_i. The proof of the first-order version of the theorem then follows by propositionalization. Lifting Algorithm 4 is the focus of the rest of this section.

We begin with some necessary definitions. A *substitution constraint* is an expression of the form x = y or x ≠ y, where x is a variable and y is either a variable or a constant. (Much richer substitution constraint languages are possible, but we adopt the simplest one that allows PTP to subsume both standard function-free theorem proving and first-order variable elimination.) Two literals are *unifiable* under a set of substitution constraints S if there exists at least one ground literal consistent with S that is an instance of both, up to the negation sign. A (C, S) pair, where C is a first-order CNF whose variables have been standardized apart and S is a set of substitution constraints, represents the ground CNF obtained by replacing each clause in C with the conjunction of its groundings that are consistent with the constraints in S. For example, using upper case for constants and lower case for variables, and assuming that the PKB contains only two constants A and B, if $C = $ R(A, B) $\wedge$ (¬R(x, y) $\vee$ S(y, z)) and $S = \{$x = y, z ≠ A$\}$, (C, S) represents the ground CNF R(A, B) $\wedge$ (¬R(A, A) $\vee$ S(A, B)) $\wedge$ (¬R(B, B) $\vee$ S(B, B)). Clauses with equality substitution constraints can be abbreviated in the obvious way (e.g., T(x, y, z) with x = y and z = C can be abbreviated as T(x, x, C)).

We lift the base case, decomposition step, and splitting step of Algorithm 4 in turn. The result is shown in Algorithm 5. In addition to the first-order CNF C and weights on first-order literals W, LWMC takes as an argument an initially empty set of substitution constraints S which, similar to logical theorem proving, is extended along each branch of the inference as the algorithm progresses.

5.1. Lifting the base case
The base case changes only by raising each first-order atom A's sum of weights to $n_A(S)$, the number of groundings of A compatible with the constraints in S. This is necessary and

---

Algorithm 5 LWMC(CNF C, substs. S, weights W)

// *Lifted base case*
if all clauses in C are satisfied **then**
    return $\prod_{A\in A(C)}(W_A + W_{\neg A})^{n_A(S)}$
if C has an empty unsatisfied clause **then return** 0
// *Lifted decomposition step*
if there exists a lifted decomposition $\{C_{1,1}, \ldots, C_{1,m_1}, \ldots,$
    $C_{k,1}, \ldots, C_{k,m_k}\}$ of C under S **then**
    return $\prod_{i=1}^{k}[\text{LWMC}(C_{i,1}, S, W)]^{m_i}$
// *Lifted splitting step*
Choose an atom A
Let $\{\Sigma_{A,S}^{(1)}, \ldots, \Sigma_{A,S}^{(l)}\}$ be a lifted split of A for C under S
return $\sum_{i=1}^{(l)} n_i W_A^{t_i} W_{\neg A}^{f_i}$ LWMC $(C|\sigma_j; S_j, W)$
    where n_i, t_i, f_i, σ_j and S_j are as in Proposition 3

---

sufficient since each atom A has $n_A(S)$ groundings, and all ground atoms are independent because the CNF is satisfied irrespective of their truth values. Note that $n_A(S)$ is the number of groundings of A consistent with S that can be formed using all the constants in the original CNF.

5.2. Lifting the decomposition step
Clearly, if C can be decomposed into two or more CNFs such that no two CNFs share any unifiable literals, a lifted decomposition of C is possible (i.e., a decomposition of C into first-order CNFs on which LWMC can be called recursively). But the symmetry of the first-order representation can be further exploited. For example, if the CNF C can be decomposed into k CNFs $C_1, \ldots, C_k$ sharing no unifiable literals and such that for all i, j, C_i is identical to C_j up to a renaming of the variables and constants, then LWMC(C) = [LWMC(C_1)]^k. We formalize these conditions below.

DEFINITION 1. *The set of first-order CNFs $\{C_{1,1}, \ldots, C_{1,m_1}, \ldots, C_{k,1}, \ldots, C_{k,m_k}\}$ is called a* lifted decomposition *of CNF C under substitution constraints S if, given S, it satisfies the following three properties: (i) $C = C_{1,1} \wedge \ldots \wedge C_{k,m_k}$; (ii) no two $C_{i,j}$'s share any unifiable literals; (iii) for all i, j, j', such that $j \neq j'$, $C_{i,j}$ is identical to $C_{i,j'}$.*[a]

PROPOSITION 1. *If $\{C_{1,1}, \ldots, C_{1,m_1} \ldots, C_{k,1}, \ldots, C_{k,m_k}\}$ is a lifted decomposition of C under S, then*

$$LWMC(C, S, W) = \prod_{i=1}^{k}[LWMC(C_{i,1}, S, W)]^{m_i}. \quad (3)$$

Rules for identifying lifted decompositions can be derived in a straightforward manner from the inversion argument in de Salvo Braz[8] and the power rule in Jha et al.[15] An example of such a rule is given in the definition and proposition below.

DEFINITION 2. *A set of variables $X = \{x_1, \ldots, x_m\}$ is called a* decomposer *of a CNF C if it satisfies the following three properties: (i) for each clause C_j in C, there is exactly one variable x_i in X such that x_i appears in all atoms in C_j; (ii) if $x_i \in X$ appears as an argument of predicate R (say at position k in an atom having predicate symbol R), then all variables in all clauses that*

---

[a] Throughout this paper, when we say that two clauses are identical, we mean that they are identical up to a renaming of constants and variables.

appear as the same argument (namely at position k) of R *are included in X; and (iii) no pair of variables* x_i, $x_j \in X$ *such that* $i \neq j$ *appear as different arguments of a predicate* R *in C.*

For example, $\{x_1, x_2\}$ is a decomposer of the CNF $(R(x_1) \vee S(x_1, x_3)) \wedge (R(x_2) \vee T(x_2, x_4))$ while the CNF $(R(x_1, x_2) \vee S(x_2, x_3)) \wedge (R(x_4, x_5) \vee T(x_4, x_6))$ has no decomposer. Given a decomposer $\{x_1, \ldots, x_m\}$ and a CNF C, it is easy to see that for every pair of substitutions of the form $S_X = \{x_1 = X, \ldots, x_m = X\}$ and $S_Y = \{x_1 = Y, \ldots, x_m = Y\}$, with $X \neq Y$, the CNFs C_X and C_Y obtained by applying S_X and S_Y to C do not share any unifiable literals. A decomposer thus yields a lifted decomposition. Given a CNF, a decomposer can be found in linear time.

When there are no substitution constraints on the variables in the decomposer, as in the example above, all CNFs formed by substituting the variables in the decomposer with the same constant are identical. Thus, $k = 1$ in Equation (3) and m_1 equals the number of constants (objects) in the PKB. However, when there are substitution constraints, the CNFs may not be identical. For example, given the CNF $(R(x_1) \vee S(x_1, x_3)) \wedge (R(x_2) \vee T(x_2, x_4))$ and substitution constraints $\{x_1 \neq A, x_2 \neq B\}$, the CNF formed by substituting $\{x_1 = A, x_2 = A\}$ is not identical to the CNF formed by substituting $\{x_1 = C, x_2 = C\}$. Specifically, the first CNF is $(R(A) \vee T(A, x_4))$ (since the clause $(R(x_1) \vee S(x_1, x_3))$ has no valid groundings for the substitution $x_1 = A$ given the constraint $x_1 \neq A$) while the second CNF is $(R(C) \vee S(C, x_3)) \wedge (R(C) \vee T(C, x_4))$.

A possible approach for solving this problem is illustrated below. For simplicity, assume that each variable x in the decomposer is involved in exactly one substitution constraint of the form $x \neq X$ (or $x = X$) where X is a constant. Consider all possible combinations (Cartesian product) of the constraints and their negation on the decomposer. Observe that for each clause in the CNF, the subset of constants O that satisfy all constraints in a given combination also satisfy the following property: for any two distinct constants X_i and X_j in O, the clause (possibly having no valid groundings) obtained by substituting the decomposer variable in it by X_i is identical to the one obtained by substituting the decomposer variable by X_j (up to a renaming of constants and variables). Thus, a simple approach to decompose the CNF into subsets of identical but disjoint CNFs is to partition the constants, with each part corresponding to a possible combination of the constraints and their negation.

For instance, in our example CNF, given the decomposer $X = \{x_1, x_2\}$, and the constraints $\{x \neq A, x \neq B\}$ where $x \in X$, we have the following four combinations of constraints and their negation: $(1) (x \neq A, x \neq B)$; $(2) (x \neq A, x = B)$; $(3) (x = A, x \neq B)$; and $(4) (x = A, x = B)$. Notice that the last combination is inconsistent (has no solution) and therefore we can ignore it. Assuming that there are five constants $\{A, B, C, D, E\}$ in the domain, the three consistent combinations given above yield the following partition of the constants: $\{\{C, D, E\}, \{A\}, \{B\}\}$. The three corresponding parts of the lifted decomposition of the CNF are (for readability, we do not standardize variables apart): $(1) (R(x_1) \vee S(x_1, x_3)) \wedge (R(x_2) \vee T(x_2, x_4)), \{x_1, x_2 \in \{C, D, E\}, x_1 \neq A, x_2 \neq B\}$; $(2) (R(x_1) \vee S(x_1, x_3)) \wedge (R(x_2) \vee T(x_2, x_4)), \{x_1, x_2 \in \{B\}, x_1 \neq A, x_2 \neq B\}$; and (3) $(R(x_1) \vee S(x_1, x_3)) \wedge (R(x_2) \vee T(x_2, x_4)), \{x_1, x_2 \in \{A\}, x_1 \neq A, x_2 \neq B\}$.

In general, the partitioning problem described above can be solved using constraint satisfaction techniques. In summary:

PROPOSITION 2. *Let X be a decomposer of a CNF C and let S be a set of substitution constraints over C. Let $\{\{X_{1,1}, \ldots, X_{1,m_1}\}, \ldots, \{X_{k,1}, \ldots, X_{k,m_k}\}\}$ be a partition of the constants in the domain and let $C' = \{C_{1,1}, \ldots, C_{1,m_1}, \ldots, C_{k,1}, \ldots, C_{k,m_k}\}$ be such that (i) for all i, j, j', $j \neq j'$, $C_{i,j}$ is identical to $C_{i,j'}$ given S, and (ii) C_{i,m_i} is a CNF formed by substituting each variable in X by X_{i,m_i}. Then C' is a lifted decomposition of C under S.*

5.3. Lifting the splitting step

Splitting on a non-ground atom means splitting on all groundings of it consistent with the current substitution constraints S. Naively, if the atom has c groundings consistent with S this will lead to a sum of 2^c recursive calls to LWMC, one for each possible truth assignment to the c ground atoms. However, in general these calls will have repeated structure and can be replaced by a much smaller number. The lifted splitting step exploits this.

We introduce some notation and definitions. Let $\sigma_{A,S}$ denote a truth assignment to the groundings of atom A that is consistent with the substitution constraints S, and let $\Sigma_{A,S}$ denote the set of all possible such assignments. Let $C|\sigma_{A,S}$ denote the CNF formed by removing A and $\neg A$ from all clauses that satisfy S, and setting to *Satisfied* all ground clauses that are satisfied because of $\sigma_{A,S}$. This can be done in a lifted manner by updating the substitution constraints associated with each clause. For instance, consider the clause $R(x) \vee S(x, y)$ and substitution constraint $\{x \neq A\}$, and suppose the domain is $\{A, B, C\}$ (i.e., these are all the constants appearing in the PKB). Given the assignment $R(B) = $ True, $R(C) = $ False and ignoring satisfied clauses, the clause becomes $S(x, y)$ and the constraint set becomes $\{x \neq A, x \neq B\}$. $R(x)$ is removed from the clause because all of its groundings are instantiated. The constraint $x \neq B$ is added because the assignment $R(B) = $ True satisfies all groundings in which $x = B$.

DEFINITION 3. *The partition $\{\Sigma_{A,S}^{(1)}, \ldots, \Sigma_{A,S}^{(l)}\}$ of $\Sigma_{A,S}$ is called a lifted split of atom A for CNF C under substitution constraints S if every part $\Sigma_{A,S}^{(i)}$ satisfies the following two properties: (i) all truth assignments in $\Sigma_{A,S}^{(i)}$ have the same number of true atoms; (ii) for all pairs $\sigma_j, \sigma_k \in \Sigma_{A,S}^{(i)}$, $C|\sigma_j$ is identical to $C|\sigma_k$.*

PROPOSITION 3. *If $\{\Sigma_{A,S}^{(1)}, \ldots, \Sigma_{A,S}^{(l)}\}$ is a lifted split of A for C under S, then*

$$LWMC(C, S, W) = \sum_{i=1}^{l} n_i W_A^{t_i} W_{\neg A}^{f_i} LWMC(C|\sigma; S_i, W)$$

where $n_i = |\Sigma_{A,S}^{(i)}|$, $\sigma \in \Sigma_{A,S}^{(i)}$, t_i, and f_i are the number of true and false atoms in σ respectively, and S_i is S augmented with the substitution constraints required to form $C|\sigma$.

Again, we can derive rules for identifying a lifted split by using the counting arguments in de Salvo Braz[8] and the generalized binomial rule in Jha et al.[15] We omit the details for lack of space. In the worst case, lifted splitting defaults to splitting on a ground atom. In most inference problems,

the PKB will contain many hard ground unit clauses (the evidence). Splitting on the corresponding ground atoms then reduces to a single recursive call to LWMC for each atom. In general, the atom to split on in Algorithm 5 should be chosen with the goal of yielding lifted decompositions in the recursive calls (e.g., using lifted versions of the propositional heuristics[23]).

Notice that the lifting schemes used for decomposition and splitting in Algorithm 5 by no means exhaust the space of possible probabilistic lifting rules. For example, Jha et al.[15] and Milch et al.[16] contain examples of other lifting rules. Searching for new probabilistic lifted inference rules, and positive and negative theoretical results about what can be lifted, looks like a fertile area for future research.

The theorem below follows from Theorem 2 and the arguments above.

THEOREM 3. *Algorithm LWMC(C, ∅, W) correctly computes the weighted model count of CNF C under literal weights W.*

5.4. Extensions and discussion

Although most probabilistic logical languages do not include existential quantification, handling it in PTP is desirable for the sake of logical completeness. This is complicated by the fact that skolemization is not sound for model counting (skolemization will not change satisfiability but can change the model count), and so cannot be applied. The result of conversion to CNF is now a conjunction of clauses with universally and/or existentially quantified variables (e.g., $[\forall x \exists y (R(x, y) \vee S(y))] \wedge [\exists u \forall v \forall w T(u, v, w)]$). Algorithm 5 now needs to be able to handle clauses of this form. If no universal quantifier appears nested inside an existential one, this is straightforward, since in this case an existentially quantified clause is just a compact representation of a longer one. For example, if the domain is {A, B, C}, the unit clause $\forall x \exists y R(x, y)$ represents the clause $\forall x (R(x, A) \vee R(x, B) \vee R(x, C))$. The decomposition and splitting steps in Algorithm 5 are both easily extended to handle such clauses without loss of lifting (and the base case does not change). However, if universals appear inside existentials, a first-order clause now corresponds to a disjunction of conjunctions of propositional clauses. For example, if the domain is {A, B}, $\exists x \forall y (R(x, y) \vee S(x, y))$ represents $(R(A, A) \vee S(A, A)) \wedge (R(A, B) \vee S(A, B)) \vee (R(B, A) \vee S(B, A)) \wedge (R(B, B) \vee S(B, B))$. Whether these cases can be handled without loss of lifting remains an open question.

Several optimizations of the basic LWMC procedure in Algorithm 5 can be readily ported from the algorithms PTP generalizes. These optimizations can tremendously improve the performance of LWMC.

Unit Propagation When LWMC splits on atom A, the clauses in the current CNF are resolved with the unit clauses A and $\neg A$. This results in deleting false atoms, which may produce new unit clauses. The idea in unit propagation is to in turn resolve all clauses in the new CNF with all the new unit clauses, and continue to do this until no further unit resolutions are possible. This often produces a much smaller CNF, and is a key component of DPLL that can also be used in LWMC. Other techniques used

in propositional inference that can be ported to LWMC include pure literals, clause learning, clause indexing, and random restarts.[2, 3, 23]

Caching/Memoization In a typical inference, LWMC will be called many times on the same subproblems. The solutions of these can be cached when they are computed, and reused when they are encountered again. (Notice that a cache hit only requires identity up to renaming of variables and constants.) This can greatly reduce the time complexity of LWMC, but at the cost of increased space complexity. If the results of all calls to LWMC are cached (full caching), in the worst case LWMC will use exponential space. In practice, we can limit the cache size to the available memory and heuristically prune elements from it when it is full. Thus, by changing the cache size, LWMC can explore various time/space tradeoffs. Caching in LWMC corresponds to both caching in model counting[23] and recursive conditioning[4] and to memoization of common subproofs in theorem proving.

Knowledge-Based Model Construction KBMC first uses logical inference to select the subset of the PKB that is relevant to the query, and then propositionalizes the result and performs standard probabilistic inference on it.[25] A similar effect can be obtained in PTP by noticing that in Equation 2 factors that are common to $Z(K \cup \{(Q, 0)\})$ and $Z(K)$ cancel out and do not need to be computed. Thus we can modify Algorithm 3 as follows: (i) simplify the PKB by unit propagation starting from evidence atoms, etc.; (ii) drop from the PKB all formulas that have no path of unifiable literals to the query; (iii) pass to LWMC only the remaining formulas, with an initial S containing the substitutions required for the unifications along the connecting path(s).

We now state two theorems (proofs are provided in the extended version of the paper) which compare the efficiency of PTP and first-order variable elimination (FOVE).[8, 18]

THEOREM 4. *PTP can be exponentially more efficient than FOVE.*

THEOREM 5. *LWMC with full caching has the same worst-case time and space complexity as FOVE.*

De Salvo Braz's FOVE[8] and lifted BP[24] completely shatter the PKB in advance. This may be wasteful because many of those splits may not be necessary. In contrast, LWMC splits only as needed.

PTP yields new algorithms for several of the inference problems in Figure 1. For example, ignoring weights and replacing products by conjunctions and sums by disjunctions in Algorithm 5 yields a lifted version of DPLL for first-order theorem proving.

Of the standard methods for inference in graphical models, propositional PTP is most similar to recursive conditioning[4] and AND/OR search[9] with context-sensitive decomposition and caching, but applies to arbitrary PKBs, not just Bayesian networks. Also, PTP effectively performs formula-based inference[13] when it splits on one of the auxiliary atoms introduced by Algorithm 2.

PTP realizes some of the benefits of lazy inference for relational models[10] by keeping in lifted form what lazy inference would leave as default.

6. APPROXIMATE INFERENCE

LWMC lends itself readily to Monte Carlo approximation, by replacing the sum in the splitting step with a random choice of one of its terms, calling the algorithm many times, and averaging the results. This yields the first lifted sampling algorithm.

We first apply this importance sampling approach[21] to WMC, yielding the MC-WMC algorithm. The two algorithms differ only in the last line. Let $Q(A|C, W)$ denote the *importance* or *proposal* distribution over A given the current CNF C and literal weights W. Then we return $\frac{W_A}{Q(A|C,W)}$ MC-WMC$(C|A; W)$ with probability $Q(A|C, W)$, or $\frac{W_{\neg A}}{Q(\neg A|C,W)}$ MC-WMC$(C|\neg A; W)$ otherwise. By importance sampling theory[21] and by the law of total expectation, it is easy to show that:

THEOREM 6. *If $Q(A|C, W)$ satisfies $WMC(C|A; W) > 0 \Rightarrow Q(A|C, W) > 0$ for all atoms A and its true and false assignments, then the expected value of the quantity output by MC-WMC(C, W) equals $WMC(C, W)$. In other words, MC-WMC(C, W) yields an unbiased estimate of $WMC(C, W)$.*

An estimate of WMC(C, W) is obtained by running MC-WMC(C, W) multiple times and averaging the results. By linearity of expectation, the running average is also unbiased. It is well known that the accuracy of the estimate is inversely proportional to its variance.[21] The variance can be reduced by either running MC-WMC more times or by choosing Q that is as close as possible to the posterior distribution P (or both). Thus, for MC-WMC to be effective in practice, at each point, given the current CNF C, we should select $Q(A|C, W)$ that is as close as possible to the marginal probability distribution of A w.r.t. C and W.

In presence of hard formulas, MC-WMC suffers from the rejection problem[12]: it may return a zero. We can solve this problem by either backtracking when a sample is rejected or by generating samples from the backtrack-free distribution.[12]

Next, we present a lifted version of MC-WMC, which is obtained by replacing the (last line of the) lifted splitting step in LWMC by the following lifted sampling step:

---

return $\frac{n_i W_A^{t_i} W_{\neg A}^{f_i}}{Q(\Sigma_{A,S}^{(i)})}$ MC-LWMC$(C|\sigma, S_i, W)$

where $\Sigma_{A,S}^{(i)}$ is sampled from Q, and n_i, t_i, f_i, σ, and S_i are as in Proposition 3.

In the lifted sampling step, we construct a distribution Q over the lifted split and sample an element $\Sigma_{A,S}^{(i)}$ from it. Then we weigh the sampled element w.r.t. Q and call the algorithm recursively on the CNF conditioned on $\sigma \in \Sigma_{A,S}^{(i)}$.

Notice that in the lifted sampling algorithm, A is a first-order atom and the distribution $Q(\Sigma_{A,S}^{(i)})$ is defined in a lifted manner. Moreover, since each $\Sigma_{A,S}^{(i)}$ represents a subset of truth assignments to the groundings of A, given a ground assignment $\sigma \in \Sigma_{A,S}^{(i)}$, the probability of sampling σ is $Q_G(\sigma) = Q(\Sigma_{A,S}^{(i)})/n_i$. Therefore, ignoring the decomposition step, MC-LWMC is equivalent to MC-WMC that uses Q_G to sample a truth assignment to the groundings of A. In the decomposition step, given a set of identical and disjoint CNFs, we simply sample just one of the CNFs and raise our estimate to the appropriate count. The correctness of this step follows from the fact that given a set $\{R_1, \ldots, R_k\}$ of k independent and identical random variables and m random samples $(r_{1,1}, \ldots, r_{1,m})$ of R_1, the expected value of the product of the random variables equals $E[R_1]^k$ and $(\frac{1}{m}\Sigma_{i=1}^m r_{1,i})^k$ is an asymptotically unbiased estimate of $E[R_1]^k$. Therefore, the following theorem immediately follows from Theorem 6.

THEOREM 7. *If $Q(\Sigma_{A,S}^{(i)})$ satisfies $WMC(C|\sigma; S_i, W) > 0 \Rightarrow Q(\Sigma_{A,S}^{(i)}) > 0$ for all elements $\Sigma_{A,S}^{(i)}$ of the lifted split of A for C under S, then MC-LWMC(C, S, W) yields an asymptotically unbiased estimate of $WMC(C, W)$.*

Because of the lifted decomposition and sampling steps, the time and space complexity of MC-LWMC is much smaller than that of MC-WMC. As a result, given a time bound the estimate returned by MC-LWMC will be based on a much larger sample size than the one returned by MC-WMC. Since variance goes down (and the accuracy goes up) as we increase the sample size, MC-LWMC has smaller variance (and is potentially more accurate) than MC-WMC.

7. EXPERIMENTS
7.1. Exact inference

We implemented PTP in C++ and ran all our experiments on a Linux machine with a 2.33 GHz Intel Xeon processor and 2GB of RAM. We used a constraint solver based on forward checking to implement the substitution constraints. We used the following heuristics for splitting. At any point, we prefer an atom which yields the smallest number of recursive calls to LWMC (i.e., an atom that yields maximum lifting). We break ties by selecting an atom that appears in the largest number of ground clauses; this number can be computed using the constraint solver. If it is the same for two or more atoms, we break ties randomly.

We compare the performance of PTP and FOVE[8] on a link prediction PKB (additional experimental results on randomly generated PKBs are presented in the full version of the paper). Link prediction is the problem of determining whether a link exists between two nodes in a network and is an important problem in many domains such as social network analysis and Web mining. We experimented with a simple link prediction PKB consisting of two clauses: GoodProf(x) ∧ GoodStudent(y) ∧ Advises(x, y) ⇒ FutureProf(y) and Coauthor(x, y) ⇒ Advises(x, y). The PKB has two types of objects: professors (x) and students (y). Given data on a subset of papers and "goodness" of professors and students, the task is to be predict who advises whom and who is likely to be a professor in the future.

We evaluated the performance of FOVE and PTP along two dimensions: (i) the number of objects and (ii) the amount of evidence. We varied the number of objects from 10 to 1000 and the number of evidence atoms from 10% to 80%.

Figure 2a shows the impact of increasing the number of evidence atoms on the performance of the two algorithms on a link prediction PKB with 100 objects. FOVE runs out of memory (typically after around 20 minutes of runtime) after the percentage of evidence atoms rises above 40%. PTP solves all the problems and is also much faster than FOVE (notice the log-scale on the y-axis). Figure 2b shows the impact of increasing the number of objects on a link prediction PKB
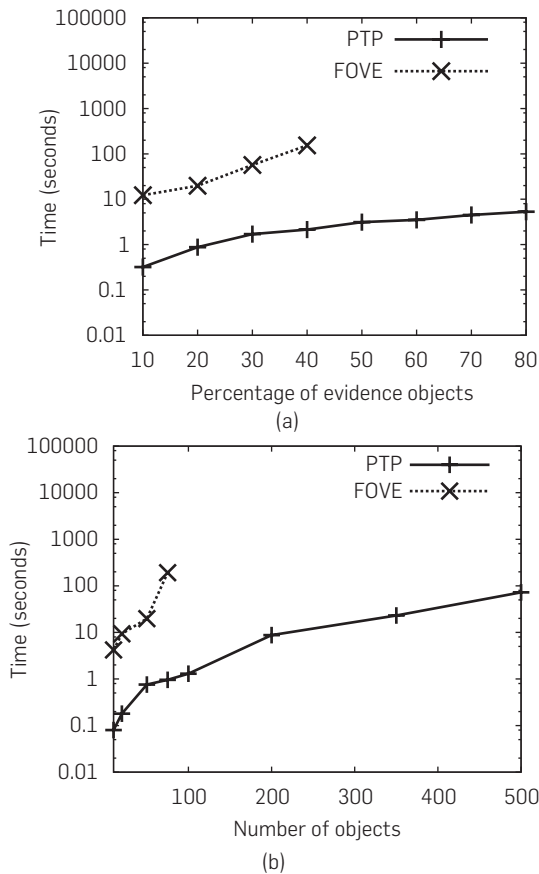
with 20% of the atoms set as observed. We see that FOVE is unable to solve any problems after the number of objects is increased beyond 100 because it runs out of memory. PTP, on the other hand, solves all problems in less than 100s.

7.2. Approximate inference

In this subsection, we compare the performance of MC-LWMC, MC-WMC, lifted belief propagation,[24] and MC-SAT[19] on two domains: entity resolution (Cora) and collective classification. The Cora dataset contains 1295 citations to 132 different research papers. The inference task here is to detect duplicate citations, authors, titles, and venues. The collective classification dataset consists of about 3000 query atoms.

Since computing the exact posterior marginals is infeasible in these domains, we used the following evaluation method. We partitioned the data into two equalsized sets: evidence set and test set. We then computed the probability of each ground atom in the test set given all atoms in the evidence set using the four inference algorithms. We measure the error using negative log-likelihood of the data according to the inference algorithms (the negative log-likelihood is a sampling approximation of the K–L divergence to the data-generating distribution, shifted by its entropy).

The results, averaged over 10 runs, are shown in Figure 3a and b. The figures show how the log-likelihood of the data varies with time for the four inference algorithms used. We see that MC-LWMC has the lowest negative log-likelihood of all algorithms by a large margin. It significantly dominates MC-WMC in about 2 min of runtime and is substantially superior to both lifted BP and MC-SAT (notice the log scale). This shows the advantages of approximate PTP over lifted BP and ground inference.

8. CONCLUSION

Probabilistic theorem proving (PTP) combines theorem proving and probabilistic inference. This paper proposed an algorithm for PTP based on reducing it to lifted weighted model counting, and showed both theoretically and empirically that it has significant advantages compared to previous lifted probabilistic inference algorithms. An implementation of PTP is available in the Alchemy 2.0 system (available at https://code.google.com/p/alchemy-2/).

Directions for future research include: extension of PTP to infinite, non-Herbrand first-order logic; new lifted inference rules; theoretical analysis of liftability; porting to PTP more speedup techniques from logical and probabilistic inference; lifted splitting heuristics; better handling of existentials; variational PTP algorithms; better importance distributions; approximate lifting; answering multiple queries simultaneously; applications; etc.

Figure 2. (a) Impact of increasing the amount of evidence on the time complexity of FOVE and PTP in the link prediction domain. The number of objects in the domain is 100. (b) Impact of increasing the number of objects on the time complexity of FOVE and PTP in the link prediction domain, with 20% of the atoms set as evidence.

(a)

(b)

Figure 3. Negative log-likelihood of the data as a function of time for lifted BP, MC-SAT, MC-WMC, and MC-LWMC on (a) the entity resolution (Cora) and (b) the collective classification domains.
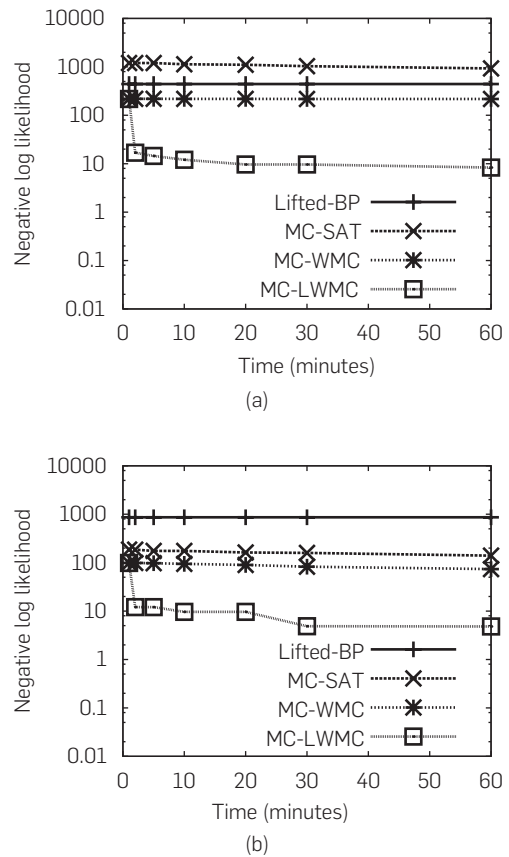
(a)

(b)

References

1. Bacchus, F. *Representing and Reasoning with Probabilistic Knowledge*. MIT Press, Cambridge, MA, 1990.
2. Bayardo, R.J., Jr., Pehoushek, J.D. Counting models using connected components. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence* (2000), 157–162.
3. Chavira, M., Darwiche, A. On probabilistic inference by weighted model counting. *Artif. Intell. 172*, 6–7 (2008), 772–799.
4. Darwiche, A. Recursive conditioning. *Artif. Intell. 126*, 1–2 (February 2001), 5–41.
5. Davis, M., Logemann, G., Loveland, D. A machine program for theorem proving. *Commun. ACM 5* (1962) 394–397.
6. Davis, M., Putnam, H. A computing procedure for quantification theory. *J. Assoc. Comput. Mach. 7*, 3 (1960), 201–215.
7. De Raedt, L., Kimmig, A., Toivonen, H. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (2007), 2462–2467.
8. de Salvo Braz, R. Lifted first-order probabilistic inference. PhD thesis, University of Illinois, Urbana-Champaign, IL (2007).
9. Dechter, R., Mateescu, R. AND/OR search spaces for graphical models. *Artif. Intell. 171*, 2–3 (2007), 73–106.
10. Domingos, P., Lowd, D. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool, San Rafael, CA, 2009.
11. Getoor, L., Taskar, B., eds. *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA, 2007.
12. Gogate, V., Dechter, R. SampleSearch: Importance sampling in presence of determinism. *Artif. Intell. 175*, 2 (2011), 694–729.
13. Gogate, V., Domingos, P. Formula-based probabilistic inference. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence* (2010), 210–219.
14. Halpern, J. An analysis of first-order logics of probability. *Artif. Intell. 46* (1990), 311–350.
15. Jha, A., Gogate, V., Meliou, A., Suciu, D. Lifted inference from the other side: The tractable features. In *Proceedings of the Twenty-Fourth Annual Conference on Neural Information Processing Systems* (2010), 973–981.
16. Milch, B., Zettlemoyer, L.S., Kersting, K., Haimes, M., Kaelbling, L.P. Lifted probabilistic inference with counting formulas. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence* (2008), 1062–1068.
17. Nilsson, N. Probabilistic logic. *Artif. Intell. 28* (1986) 71–87.
18. Poole, D. First-Order probabilistic inference. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (2003), 985–991.
19. Poon, H., Domingos, P. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence* (2006), 458–463.
20. Robinson, J.A. A machine-oriented logic based on the resolution principle. *J. ACM 12* (1965) 23–41.
21. Rubinstein, R.Y. *Simulation and the Monte Carlo Method*. John Wiley & Sons Inc. Hoboken, NJ, 1981.
22. Sang, T., Beame, P., Kautz, H. Solving Bayesian networks by weighted model counting. In *Proceedings of the Twentieth National Conference on Artificial Intelligence* (2005), 475–482.
23. Sang, T., Beame, P., Kautz, H. Heuristics for fast exact model counting. In *Eighth International Conference on Theory and Applications of Satisfiability Testing* (2005), 226–240.
24. Singla, P., Domingos, P. Lifted first-order belief propagation. In *Lifted first-order belief propagation. In Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence* (2008), 1094–1099.
25. Wellman, M., Breese, J.S., Goldman, R.P. From knowledge bases to decision models. *Knowledge Eng. Rev. 7* (1992), 35–53.

Vibhav Gogate (vgogate@hlt.utdallas.edu), The University of Texas at Dallas, Richardson, TX.

Pedro Domingos (pedrod@cs.washington.edu), University of Washington Seattle, WA.