Just Count the Satisfied Groundings: Scalable Local-Search and Sampling Based Inference in MLNs

Deepak Venugopal
Department of Computer Science
The University of Texas at Dallas
dxv021000@utdallas.edu

Somdeb Sarkhel
Department of Computer Science
The University of Texas at Dallas
somdeb.sarkhel@utdallas.edu

Vibhav Gogate
Department of Computer Science
The University of Texas at Dallas
vgogate@hlt.utdallas.edu

Abstract

The main computational bottleneck in various sampling based and local-search based inference algorithms for Markov logic networks (e.g., Gibbs sampling, MC-SAT, MaxWalksat, etc.) is computing the number of groundings of a first-order formula that are true given a truth assignment to all of its ground atoms. We reduce this problem to the problem of counting the number of solutions of a constraint satisfaction problem (CSP) and show that during their execution, both sampling based and local-search based algorithms repeatedly solve dynamic versions of this counting problem. Deriving from the vast amount of literature on CSPs and graphical models, we propose an exact junction-tree based algorithm for computing the number of solutions of the dynamic CSP, analyze its properties, and show how it can be used to improve the computational complexity of Gibbs sampling and MaxWalksat. Empirical tests on a variety of benchmarks clearly show that our new approach is several orders of magnitude more scalable than existing approaches.

Introduction

A large number of application domains (e.g. NLP, computer vision, robotics, etc.) require rich modeling languages that are capable of handling uncertainty. Markov logic networks (MLNs) (Domingos and Lowd 2009) combine logical and probabilistic representations (Getoor and Taskar 2007), and are therefore ideally suited for such applications. At a high level, MLNs use weighted first-order formulas to define a compact template for generating large Markov networks.

Naturally, one can perform inference in MLNs by performing inference over the (ground) Markov network. However, this approach is not scalable because in large real-world domains, the ground Markov network can have millions of variables and features. For example, consider a simple first-order formula $\forall x, \forall y, \forall z, \forall u\ \texttt{R}(x, y) \vee \texttt{S}(y, z) \vee \texttt{T}(z, u)$ and assume that each logical variable has 100 objects (constants) in its domain. Grounding this formula gives rise to 100 million ground formulas and performing any kind of inference (even approximate inference) on such a large Markov network is an extremely challenging task.

In this paper, we identify the main computational bottleneck in MLN inference, which specifically affects sampling-based and local-search based inference algorithms. This bottleneck is counting the true groundings of a first-order formula f given a world ω. We observed that existing MLN systems such as Alchemy (Kok et al. 2006) and Tuffy (Niu et al. 2011) solve this counting problem using the following naive generate-and-test approach: generate each possible grounding and test whether it is true in ω. This naive approach is the chief reason for their poor scalability and is particularly problematic because algorithms such as Gibbs sampling (Geman and Geman 1984; Venugopal and Gogate 2012) and MaxWalksat (Kautz, Selman, and Jiang 1997) perform the above operation in every iteration.

In this paper, we propose a novel, practical approach for solving the aforementioned counting problem. The key advantages of our approach are that it never grounds the full MLN and in most cases is orders of magnitude better than the "generate-and-test" approach. Specifically, we encode each formula (f) as a CSP ($\mathcal{C}$) such that the number of solutions to $\mathcal{C}$ can be directly used to count the satisfied groundings of f. The main advantage of this encoding is that we can now leverage several years of advances in CSPs and graphical model inference and use virtually any exact/approximate inference algorithm along with its associated guarantees to efficiently solve the counting problem.

We demonstrate the power of our approach by showing that it greatly improves the computational complexity of two popular approximate inference algorithms: Gibbs Sampling and MaxWalkSAT. We show that in both these algorithms, the main computational steps involve solving the encoded CSP, where the constraints change over time (dynamic CSP). To solve this dynamic CSP, we compile an exact junction tree for the CSP and account for the changing constraints by modifying the junction tree messages. We evaluated both algorithms on a wide variety of MLN benchmarks and compared them with algorithms implemented in two existing MLN systems, Alchemy and Tuffy. Our experiments clearly show that our new algorithms are several orders of magnitude more scalable than existing systems.

Background and Preliminaries

First-Order Logic

We assume a strict subset of first-order logic, called finite Herbrand logic. Thus, we assume that we have no function constants and finitely many object constants. A first-order knowledge base (KB) is a set of first-order formulas. A formula in first-order logic is made up of quantifiers ($\forall$ and $\exists$), logical variables, constants, predicates and logical connectives ($\vee$, $\wedge$, $\neg$, $\Rightarrow$, and $\Leftrightarrow$). We denote logical variables by lower case letters (e.g., x, y, z, etc.) and constants by strings that begin with an upper case letter (e.g., A, Ana, Bob, etc.). Constants model objects in the real-world domain. A predicate is a relation that takes a specific number of arguments (called its arity) as input and outputs either True (synonymous with 1) or False (synonymous with 0). A term is either a logical variable or a constant. We denote predicates by strings in typewriter font (e.g., `R`, `S`, `Smokes`, etc.) followed by a parenthesized list of terms.

A first-order formula is recursively defined as follows:(i) An atomic formula is a predicate; (ii) Negation of an atomic formula is a formula; (iii) If f and g are formulas then connecting them by binary connectives such as $\wedge$ and $\vee$ yields a formula; and (iv) If f is a formula and x is a logical variable then $\forall x f$ and $\exists x f$ are formulas.

We assume that each argument of each predicate is typed and can only be assigned to a fixed subset of constants. By extension, each logical variable in each formula is also typed. We further assume that all first-order formulas are disjunctive (clauses), have no free logical variables (namely, each logical variable is quantified), have only universally quantified logical variables (CNF), and have no constants. Note that all first-order formulas can be easily converted to this form. A ground atom is an atom that contains no logical variables. A possible world, denoted by ω, is a truth assignment to all possible ground atoms that can be formed from the constants and the predicates.

Discrete Graphical Models

Discrete graphical models or Markov networks consist of a set of variables $\mathbf{x} = \{x_1, \ldots, x_n\}$, each of which take values from a finite domain $D(x)$, and a set of positive functions $\Phi = \{\phi_1, \ldots, \phi_m\}$, each defined over a subset $S(\phi)$ of variables, called its scope. A Markov network represents the probability distribution $\Pr(\overline{\mathbf{x}}) = \frac{1}{Z}\prod_{i=1}^{m} \phi_i\left(\overline{\mathbf{x}}_{S(\phi_i)}\right)$, where $\overline{\mathbf{x}}$ is an assignment of values to all variables in $\mathbf{x}$, $\overline{\mathbf{x}}_{S(\phi_i)}$ is a projection of $\overline{\mathbf{x}}$ on $S(\phi_i)$, and Z is a normalization constant called the partition function which is given by $Z = \sum_{\overline{\mathbf{x}}}\prod_{i=1}^{m} \phi_i\left(\overline{\mathbf{x}}_{S(\phi_i)}\right)$[1].

Important inference queries over graphical models are computing the partition function, finding the marginal probability of a variable given evidence (an assignment to a subset of variables) and finding the most probable assignment to all variables given evidence (MAP inference).

When the range of all functions is $\{0, 1\}$, the graphical model represents a constraint network. A 0/1 function ϕ, can also be thought of as a constraint or a relation in which all assignments $\mathbf{y}$ such that $\phi(\mathbf{y}) = 1$ are allowed while the ones having $\phi(\mathbf{y}) = 0$ are not allowed. A constraint satisfaction problem (CSP) is to find an assignment of values to all variables such that all constraints are satisfied (namely a solution). Another important problem over constraint networks is computing the number of solutions to a CSP. This problem is equivalent to computing the partition function.

Markov Logic Networks

Markov logic networks (MLNs) combine Markov networks and first-order logic. Formally, an MLN is a set of pairs $(f_i, \mathtt{w}_i)$ where f_i is a formula in first-order logic and $\mathtt{w}_i$ is a real number. Given a set of constants, an MLN represents a ground Markov network, defined as follows. We have one binary random variable in the Markov network for each possible ground atom. We have one propositional feature for each possible grounding of each first-order formula. The weight associated with the feature is the weight attached to the corresponding formula. A propositional feature having weight $\mathtt{w}$ represents the following function: $\phi(\mathbf{y}) = \exp(\mathtt{w})$ if $\mathbf{y}$ evaluates the feature to true and $\phi(\mathbf{y}) = 1$ otherwise. The ground Markov network represents the following probability distribution:

$$\Pr(\omega) = \frac{1}{Z}\exp\left(\sum_i \mathtt{w}_i N_{f_i}(\omega)\right) = \frac{1}{Z}\prod_i \left[\phi_i(\omega_{S(\phi_i)})\right]^{N_{f_i}(\omega)}$$

where $N_{f_i}(\omega)$ is the number of groundings of f_i that evaluate to True given ω. The main computational bottleneck in several inference algorithms for MLNs, in particular Gibbs sampling and MaxWalksat, is computing $N_{f_i}(\omega)$. We call this the #`SATGround` problem.

#`SATGround` as a CSP

We demonstrate our proposed CSP encoding[2] on a simple MLN having just one clause: $f = \forall x, \forall y, \forall z\ \mathtt{R}(x, y) \vee \mathtt{S}(y, z)$. We will focus on counting the number of false groundings of f given a world ω because it is easier to compute. Moreover, we can easily compute the number of true groundings $N_f(\omega)$ from it; $N_f(\omega)$ is equal to the number of all possible groundings (which is simply a product of the domain sizes) minus the number of false groundings.

Let us assume that the domain of each logical variable in f is $\{A, B\}$. Each triple (x, y, z) where each x, y and z can take values from the domain $\{A, B\}$ uniquely identifies each grounding of the formula. Consider a world ω shown in Fig. 1(a). Let us associate two 0/1 functions $\phi_1(x, y)$ and $\phi_2(y, z)$ with $\mathtt{R}(x, y)$ and $\mathtt{S}(y, z)$ respectively. The 0/1 function has a value 1 iff the corresponding grounding is False in the world and 0 otherwise (see Fig. 1(b)).

Given this set up, notice that if we take a product of the two functions $\phi_1(x, y)$ and $\phi_2(y, z)$, then the resulting function $\phi_3(x, y, z)$ will have a 1 associated with an entry (x, y, z) iff both $\mathtt{R}(x, y)$ and $\mathtt{S}(y, z)$ are False. Since the ground formula (x, y, z) evaluates to False iff both $\mathtt{R}(x, y)$ and $\mathtt{S}(y, z)$ are False, by extension $\phi_3(x, y, z) = 1$ implies that the ground formula (x, y, z) is False. Therefore, we can count the number of groundings of f that evaluate to False, by simply counting the number of ones in $\phi_3(x, y, z)$, which is the same as counting the number of solutions to the CSP having two functions $\phi_1(x, y)$ and $\phi_2(y, z)$.

[1]Often, we will abuse notation and use $\overline{\mathbf{x}}$ instead of $\overline{\mathbf{x}}_{S(\phi_i)}$.

[2]The #`SATGround` problem can also be reduced to the conjunctive query problem in relational databases (Vardi 1982)

$\mathtt{R}(A,A)$	1
$\mathtt{R}(A,B)$	0
$\mathtt{R}(B,A)$	0
$\mathtt{R}(B,B)$	1

$\mathtt{S}(A,A)$	0
$\mathtt{S}(A,B)$	1
$\mathtt{S}(B,A)$	0
$\mathtt{S}(B,B)$	1

(a) world ω

x	y	$\phi_1(x,y)$
A	A	0
A	B	1
B	A	1
B	B	0

y	z	$\phi_2(y,z)$
A	A	1
A	B	0
B	A	1
B	B	0

(b) Functions ϕ_1 and ϕ_2

x	y	z	$\phi_3(x,y,z)$
A	A	A	0
A	A	B	0
A	B	A	1
A	B	B	0

x	y	z	$\phi_3(x,y,z)$
B	A	A	1
B	A	B	0
B	B	A	0
B	B	B	0

(c) Function $\phi_3 = \phi_1 \times \phi_2$

Figure 1: (a) A possible world of an MLN having only one formula: $f = \forall x, \forall y, \forall z\ \mathtt{R}(x,y) \vee \mathtt{S}(y,z)$. The domain of each logical variable is $\{A,B\}$; (b) Functions ϕ_1 and ϕ_2 corresponding to $\mathtt{R}(x,y)$ and $\mathtt{S}(y,z)$ respectively; and (c) Function ϕ_3 which is equal to the product of the two functions given in (b). The number of 1s in ϕ_3 equals the number of groundings of f that evaluate to False.

Algorithm	**Space Complexity**	**Time complexity**
Pre-Ground	$O(\sum_{i=1}^{M} d^{\lvert\mathbf{V}_i\rvert})$	$O(\sum_{i=1}^{M} d^{\lvert\mathbf{V}_i\rvert})$
Lazy-Ground	$O(1)$	$O(\sum_{i=1}^{M} d^{\lvert\mathbf{V}_i\rvert})$
And-OR Tree	$O(M)$	$O(\sum_{i=1}^{M} d^{w_i^* log(\lvert\mathbf{V}_i\rvert)+1})$
Junction Tree	$O(\sum_{i=1}^{M} d^{w_i^*})$	$O(\sum_{i=1}^{M} d^{w_i^*+1})$

Table 1: $\#$`SATGround` complexities using various strategies. M is the number of formulas, $\mathbf{V}_i$ is the set of variables in the CSP encoded for the i^{th} formula, d is the domain-size of each variable and w_i^* is the treewidth of the CSP encoded for the i^{th} formula.

Next, we will formalize this intuition and precisely define how to encode the $\#$`SATGround` problem as a CSP solution counting problem.

Encoding*f-to-CSP*. Given a first-order clause f and a world ω, the corresponding CSP $\mathcal{C}$ has a variable for each (universally quantified) logical variable in f. The domain of each variable in $\mathcal{C}$ is the set of constants in the domain of the corresponding logical variable. For each atom $\mathtt{R}(x_1,\ldots,x_u)$ in f, we have a relation ϕ in $\mathcal{C}$ defined as follows:

$$\phi(\overline{\mathbf{x}}_u) = \begin{cases} \omega_{\mathtt{R}(X_1,\ldots,X_u)} & \text{if } \mathtt{R} \text{ is negated in } f \\ \neg\omega_{\mathtt{R}(X_1,\ldots,X_u)} & \text{Otherwise} \end{cases}$$

where $\overline{\mathbf{x}}_u = (x_1 = X_1, \ldots, x_u = X_u)$ denotes an assignment to the CSP variables and $\omega_{\mathtt{R}(X_1,\ldots,X_u)}$ is the projection of the world ω on the ground atom $\mathtt{R}(X_1,\ldots,X_u)$, namely the truth-value of the ground atom $\mathtt{R}(X_1,\ldots,X_u)$ in ω.

By generalizing the arguments presented for the example MLN formula given above, we can show that:

Theorem 1. *Let f be a first-order clause, $x_1,\ldots,x_u$ be the (universally quantified) logical variables in f, ω be a world and let $\#Sol(\mathcal{C})$ denote the number of solutions of the CSP $\mathcal{C}$ obtained from (f_i,ω) using the f-to-CSP encoding. Then, $N_f(\omega) = \prod_{j=1}^{u} |\Delta(x_j)| - \#Sol(\mathcal{C})$ where $\Delta(x_j)$ is the set of constants in the domain of x_j.*

Counting the Number of Solutions of the CSP

Since we have reduced the $\#$`SATGround` problem to the CSP solution counting problem, it is clear that we can use any CSP/graphical model inference algorithm and leverage its advances and guarantees to efficiently compute the former. Table 1 shows the complexity bounds for various strategies and algorithms for solving the $\#$`SATGround` problem.

Alchemy (Kok et al. 2006) uses the pre-ground strategy, namely it grounds all clauses that it is unable to lift. Thus, its worst case time and space complexity bounds are exponential in the maximum number of variables in the MLN formulas. The pre-ground strategy is useful when there is large amount of evidence. In presence of evidence, one can use unit propagation to remove all clauses that are either True or False. This reduces the space complexity as well as the time complexity of subsequent counting problems.

An alternative strategy is to do lazy grounding, which reduces to solving the CSP using the generate and test approach. In this approach, we count the solutions by generating each tuple and testing whether it is a solution or not. Although this approach has constant space complexity, its worst case time complexity is the same as the pre-ground approach. Moreover, unlike the pre-ground approach, this approach is unable to take advantage of unit propagation and the worst-case results apply to all subsequent operations.

A better, more powerful approach is to use advanced search and elimination techniques such as AND/OR search (Dechter and Mateescu 2007), recursive conditioning (Darwiche 2001), junction tree propagation (Lauritzen and Spiegelhalter 1988), as well as knowledge compilation techniques such as arithmetic circuits (Darwiche 2003) and AND/OR multi-valued decision diagrams (Mateescu, Dechter, and Marinescu 2008). In this paper, we focus on using the junction tree algorithm for computing the solution counts, noting that in future, one can use other advanced techniques mentioned above as well as approximate solution counting approaches such as WISH (Ermon et al. 2013), SampleSearch (Gogate and Dechter 2007) and generalized BP (Yedidia, Freeman, and Weiss 2005).

Junction Trees for Solution Counting

We now briefly review the junction tree algorithm used to compute the number of solutions $\#Sol(\mathcal{C})$ of $\mathcal{C}$.

Definition 1. *Given the CSP $\mathcal{C}$, a* **junction tree** *is a tree $\mathcal{T}(\mathbf{V},\mathbf{E})$ in which each vertex $V \in \mathbf{V}$ (also called a cluster) and edge $E \in \mathbf{E}$ are labeled with a subset of variables, denoted by $L(V)$ and $L(E)$ such that:(i) for every function ϕ defined in $\mathcal{C}$, there exists a vertex $L(V)$ such that $S(\phi) \subseteq L(V)$ and (ii) for every variable x in $\mathcal{C}$, the set of vertexes and edges in $\mathcal{T}$ that mention x form a connected sub-tree in $\mathcal{T}$ (called the running intersection property).*

Given a junction tree $\mathcal{T}$ of $\mathcal{C}$, we can compute the solution counts as well as the marginal probability of each CSP variable (the fraction of solutions that the variable participates in) by calibrating $\mathcal{T}$. We *calibrate* $\mathcal{T}$ by selecting a cluster as the root and performing sum-product message passing in two passes: from the leaves to the root (collect pass) and then from the root to the leaves (distribute pass). Formally, the message sent from cluster i to cluster j, denoted by $m_{i\rightarrow j}$, is given by

$$m_{i\rightarrow j}(\mathbf{y}) = \sum_{\overline{\mathbf{z}}} \prod_{\phi\in\Phi(V_i)} \phi(\overline{\mathbf{y}},\overline{\mathbf{z}}) \prod_{k\in N(i)\setminus\{j\}} m_{k\rightarrow i}(\overline{\mathbf{y}},\overline{\mathbf{z}}) \quad (1)$$

where $\Phi(V_i)$ is the set of functions assigned to vertex V_i, and $N(i)$ is the set of indexes of the neighbors of V_i in $\mathcal{T}$.

The number of solutions to $\mathcal{C}$ can be computed from any vertex V_k using the following equation:

$$\#Sol(\mathcal{C}) = \sum_{\overline{\mathbf{x}}} \prod_{\phi\in\Phi(V_k)} \phi(\overline{\mathbf{x}}) \prod_{j\in N(k)} m_{j\rightarrow k}(\overline{\mathbf{x}}) \quad (2)$$

The complexity of computing the solution counts using a junction tree is exponential in the maximum cluster size of the tree which equals treewidth plus 1. Next, we describe efficient implementations of two classical approximate inference algorithms, Gibbs sampling and MaxWalkSAT, using calibrated[3]

Application I: Gibbs Sampling

In Gibbs sampling, we start with a random world $\omega^{(0)}$. Then at each iteration $i > 0$, we compute the conditional distribution over a randomly chosen ground atom $\mathtt{R}$ given $\omega_{-\mathtt{R}}^{(i-1)}$ where $\omega_{-\mathtt{R}}^{(i-1)}$ is the projection of $\omega^{(i-1)}$ on all ground atoms of the MLN except $\mathtt{R}$. Then, we sample a new value for $\mathtt{R}$, denoted by $\overline{\mathtt{R}}$, from this conditional distribution and set $\omega^{(i)} = (\omega_{-\mathtt{R}}^{(i-1)},\overline{\mathtt{R}})$. Note that for brevity, we have abused notation and denoted the ground atom $\mathtt{R}(X_1,\ldots,X_r)$ as $\mathtt{R}$.

The main computational bottleneck in Gibbs sampling is computing the conditional distribution over the ground atom $\omega^{(i)}$. It is given by:

$$\Pr(\mathtt{R}=j|\omega_{-\mathtt{R}}^{(i)}) \propto \sum_{f_k\in F(\mathtt{R})} \mathtt{w}_k N_{f_k}(\omega_{-\mathtt{R}}^{(i)},\mathtt{R}=j) \quad (3)$$

where $j \in \{0,1\}$ and $F(\mathtt{R})$ is the set of first-order formulas in the MLN that contain $\mathtt{R}$ (the Markov Blanket of $\mathtt{R}$).

Given a formula f_k and a world ω, let $\mathcal{C}_k$ denote the constraint network obtained from (f_k,ω) using the *f-to-CSP* encoding. Let $\mathcal{T}_k$ denote the junction tree obtained from $\mathcal{C}_k$. If we calibrate the junction tree $\mathcal{T}_k$, then we can easily compute $N_{f_k}(\omega)$ from it (see Eq.2). The main challenge is computing $N_{f_k}(\omega')$ where $\omega' = (\omega_{-\mathtt{R}},\neg\omega_{\mathtt{R}})$. We describe how to compute it next.

Consider the two CSPs $\mathcal{C}_k$ and $\mathcal{C}'_k$ obtained from (f_k,ω) and (f_k,ω') respectively using the f-to-CSP encoding. Since f_k defines the scope of the functions in the CSP, both CSPs have functions defined over the same scope. Moreover, since ω and ω' differ only in a truth assignment to one ground atom, the corresponding functions in $\mathcal{C}_k$ and $\mathcal{C}'_k$ differ only in at most one entry. Thus, we can efficiently construct a calibrated junction tree $\mathcal{T}'_k$ from $\mathcal{T}_k$ by appropriately propagating the changed entries. Specifically, assuming that all functions that have changed are present in a cluster V in $\mathcal{T}_k$, we designate V as the root and distribute messages away from it, stopping propagation beyond a cluster if the new message and the old message to the cluster are the same.

Application II: MaxWalkSAT

The MAP problem in Markov logic networks reduces to the problem of finding a possible world that maximizes sum of weights of satisfied clauses. Any weighted satisfiability solver can used to solve the MAP problem, however the most commonly used solver is MaxWalkSAT (Kautz, Selman, and Jiang 1997). The latter is a weighted variant of WalkSAT, a local-search algorithm for satisfiability testing (Selman, Kautz, and Cohen 1996).

MaxWalkSAT takes as input an MLN $\mathcal{M}$ and a probability p. The algorithm begins by randomly generating a possible world. Then, at each iteration, it selects a false ground clause uniformly at random and flips the value assigned to one of its atoms as follows. With probability p, the atom (literal) to be flipped is selected uniformly at random and with probability $(1-p)$, the atom which when flipped maximizes the number of satisfied ground clauses is selected (greedy hill-climbing step which selects the best atom).

The last two steps of flipping a random atom and the best atom can be accomplished by using the same approach described in the previous section. Namely, we maintain a calibrated junction tree for the current world ω and each formula f_k, updating it as necessary to determine the weight of the world after the flip. The most challenging step is selecting a false ground clause uniformly at random. Next we describe a procedure for accomplishing this.

Selecting a False Clause uniformly at random: We solve this problem using a two step procedure. In the first step, we select a first order formula f_i such that the probability of selecting f_i is proportional to the number of its false groundings. In the second step, we select a false ground clause of f_i uniformly at random.

To select a first-order formula, we first compute the number of false ground clauses for all first-order formulas (using the calibrated junction tree) and normalize them to yield a distribution over the formulas. We then sample a first-order formula from this distribution.

Let f_i be the sampled first-order formula. To select a false ground clause of f_i uniformly at random, we sample the calibrated junction tree of f_i using the junction tree solution sampling method described in (Dechter et al. 2002;

[3]Technically, Gibbs sampling can be implemented more efficiently using uncalibrated junction trees.

Gogate 2009). Sampling the junction tree yields a solution to the corresponding CSP $\mathcal{C}_i$. Based on our encoding, each solution of $\mathcal{C}_i$ corresponds to a false clause of f_i.

Selecting a False Clause in Presence of Evidence: In presence of evidence, the solution sampling method described above cannot be used because the sampled ground clause may be *trivially false*, namely, none of the atoms of the clause can be flipped because all of its atoms are evidence atoms. Thus, the solution sampling method must be modified so that it never selects a trivially false clause.

Before describing our method to solve this problem, we introduce some notation. Let $\mathcal{E}$, $\mathcal{E}_u$ and $\mathcal{E}_t$ denote the set of ground clauses, false ground clauses and trivially false ground clauses of f respectively. From the definition, it is obvious that $\mathcal{E}_t \subseteq \mathcal{E}_u \subseteq \mathcal{E}$ and any method that samples an element of the set $(\mathcal{E}_u \setminus \mathcal{E}_t)$ will not sample a trivially false clause. We can use rejection sampling to find such a clause but this will be quite slow, especially when there is a large amount of evidence.

A more clever approach is to use two constraint networks. Formally, given a formula f_k and the constraint network $\mathcal{C}_k$ obtained from it we define another constraint network (referred to as evidence network) $\mathcal{C}_k^\epsilon$ as follows. $\mathcal{C}_k^\epsilon$ is defined over same set of variables as $\mathcal{C}_k$. Corresponding to each function $\phi(\overline{\mathbf{x}}_u)$ in $\mathcal{C}_k$, we define a function $\phi^\epsilon(\overline{\mathbf{x}}_u)$ in $\mathcal{C}_k^\epsilon$ as follows: let $\mathtt{R}(x_1, \ldots, x_u)$ denote the atom corresponding to $\phi(\overline{\mathbf{x}}_u)$ in $\mathcal{C}_k$

$$\phi^\epsilon(\overline{\mathbf{x}}_u) = \begin{cases} \phi(\overline{\mathbf{x}}_u) & \text{if } \mathtt{R}(x_1, \ldots, x_u) \text{ is an evidence atom} \\ 0 & \text{Otherwise} \end{cases}$$

where $\overline{\mathbf{x}}_u = (x_1 = X_1, \ldots, x_u = X_u)$. From construction of $\mathcal{C}_k^\epsilon$ it follows that each solution of $\mathcal{C}_k^\epsilon$ is a trivially false clause and vice versa.

Thus, we have a way of computing the number of non-trivial false groundings of a clause. As outlined earlier for the non-evidence case, this number can be used to select a first-order clause f_k (by sampling each formula with probability proportional to the number of its non-trivial false groundings). To select a non-trivial false ground clause of f_k, we use the following procedure. We maintain two calibrated junction trees $\mathcal{T}_k^\epsilon$ and $\mathcal{T}_k$ corresponding to the two constraint networks $\mathcal{C}_k^\epsilon$ and $\mathcal{C}_k$ respectively. Note that the junction tree $\mathcal{T}_k^\epsilon$ needs to be calibrated only once because the evidence does not change. We then create a new junction tree $\mathcal{T}_k'$ by *subtracting* $\mathcal{T}_k$ from $\mathcal{T}_k^\epsilon$. $\mathcal{T}_k'$ represents a uniform distribution over the elements of the set $(\mathcal{E}_u \setminus \mathcal{E}_t)$ and therefore we draw a sample, uniformly at random from it to yield a non-trivial false ground clause of f_k. For details of this subtraction operation and the sampling procedure, we refer the reader to the extended version of this paper (Venugopal and Sarkhel and Gogate 2014).

Extensions

Existential Quantifiers

In this subsection, we describe how our approach, specifically the *f-to-CSP encoding*, can be extended to handle existential quantifiers. We consider two cases.

Case 1: If no universal quantifier is nested inside an existential one then each first-order formula is just a compact representation of a disjunction over the groundings of the existentially quantified variables. For example, if the domain is $\{A, B\}$, then the first-order formula $\forall x, \forall z \; \exists y \; \mathtt{R}(x, y) \vee \mathtt{S}(z, y)$ represents the clause $\forall x, \forall z \; \mathtt{R}(x, A) \vee \mathtt{R}(x, B) \vee \mathtt{S}(z, A) \vee \mathtt{S}(z, B)$. Given a world ω, we can encode this clause as a CSP having two variables x and z, and four unary constraints $\phi_1(x)$, $\phi_2(x)$, $\phi_3(z)$ and $\phi_4(z)$ corresponding to the truth assignments to $\mathtt{R}(x, A)$, $\mathtt{R}(x, B)$, $\mathtt{S}(z, A)$ and $\mathtt{S}(z, B)$ respectively in the world ω. In general, the variables in the CSP encoding are the universally quantified variables in the clause and we have a constraint over the universally quantified variables for each atom and each possible value assignment to the existentially quantified variables. Thus, an existentially quantified variable increases the number of constraints but does not factor in determining the complexity of the junction tree computations.

Case 2: If a universal quantifier is nested within an existential one, then the first-order clause corresponds to a disjunction of conjunctions of propositional clauses. This case is much harder than the previous case and whether our approach can be extended to such cases is an open problem.

Lifted Inference

Our approach can be easily incorporated within lifted inference algorithms (cf. (Poole 2003; de Salvo Braz 2007; Gogate and Domingos 2011; Van den Broeck et al. 2011; Venugopal and Gogate 2012; Bui, Huynh, and Riedel 2013)). For example, consider the lifted MaxWalkSAT algorithm described in (Sarkhel et al. 2014). Sarkhel et al.'s algorithm works as follows. It first converts given normal MLN to a non-shared (normal) MLN by grounding all the shared terms in each formula. Then, it reduces the domain-size of all non-shared terms to 1 and computes the MAP value over this new MLN using MaxWalkSAT. Sarkhel et al's approach works because the MAP value for each atom in a non-shared MLN lies at the extreme: ground atoms in a non-shared MLN are either all true or all false in the MAP tuple. To combine Sarkhel et al.'s algorithm with our approach, all we have to do is set domain sizes of all non-shared logical variables to 1. Effectively, this removes the variable from all the junction tree computations, possibly decreasing its treewidth. Moreover, the proposed combination is more powerful because Sarkhel et al pre-ground the MLN, which can be exponentially worse than our approach (see Table 1).

Experiments

Setup

We used 10 benchmark MLNs with varied structures and random evidence ($< 25\%$) to evaluate the performance of our Gibbs sampling and MaxWalkSAT algorithms. We compared our system with two state-of-the-art MLN systems: Alchemy and Tuffy. Alchemy implements both Gibbs sampling as well as MaxWalkSAT whereas Tuffy only implements MaxWalkSAT. Of the 10 benchmarks, 5 were

MLN	#Groundings	C-Time	SRate
student-100	1.0003e+08	0	11397
student-500	6.25008e+10	0	496.72
student-1000	1e+12	0	117.901
relation-100	1.03e+06	0	7047.97
relation-500	1.2575e+08	1	274.901
relation-1000	1.003e+09	10	68.6392
longchain-100	1e+14	0	3235.09
longchain-500	7.8125e+18	0	126.147
longchain-1000	1e+21	1	31.836
transitive1-100	1.01e+06	0	88739.7
transitive1-500	1.2525e+08	0	24568.4
transitive1-1000	1.001e+09	0	8879.61
transitive2-100	1.01e+06	0	73.4589
transitive2-500	1.2525e+08	1	0.590163
webkb	1.01011e+10	1	183.836
seg	1.26038e+09	0	32.6557
er	1.59375e+09	21	5.83606
protein	5.00312e+08	1	116.672
coref	5.27568e+08	2	180.967

Figure 2: Results on benchmarks for Gibbs sampling using our approach. **SRate** is the sampling rate (#samples/second) and **C-Time** is the compilation time in seconds. Note that Alchemy timed out (2 hours) or ran out of memory on all the instances and therefore its results are not shown.

MLN	#Groundings	Ours	Alchemy	Tuffy
student-100	1.0003e+08	0;31629	-	-
student-500	6.25008e+10	0;252.5	-	-
student-1000	1e+12	0;72	-	-
relation-100	1.03e+06	0;2455.5	95;1000	75;300
relation-500	1.2575e+08	1;142.8	-	-
relation-1000	1.003e+09	13;36	-	-
longchain-100	1e+14	0;928.3	-	-
longchain-500	7.8125e+18	0;50	-	-
longchain-1000	1e+21	1;12.3	-	-
transitive1-100	1.01e+06	0;32082	75;350	65;100
transitive1-500	1.2525e+08	0;1032	-	-
transitive1-1000	1.001e+09	0;284.2	-	-
transitive2-100	1.01e+06	0;30	75;200	65;150
transitive2-500	1.2525e+08	1;0.22	-	-
webkb	1.01011e+10	1;48.5	-	-
seg	1.26038e+09	0;8.6	-	-
er	1.59375e+09	26;1.2	-	-
protein	5.00312e+08	1;6.8	-	-
coref	5.27568e+08	3;6.3	-	-

Figure 3: Results on benchmarks for MaxWalkSAT. For each system, we show **C-Time**;**FRate**, where **C-Time** is the compilation time (in seconds) for our system or the grounding time in Alchemy/Tuffy. **FRate** is the flip rate (#Flips/second). "-" denotes that the system ran out of time or memory.

from Alchemy: webkb, entity resolution (er), segmentation (seg), protein interaction (protein) and coreference resolution (coref). Our 5 synthetic benchmarks are as follows: (i) student: $\neg\texttt{Student}(x,\ p)\ \vee\ \neg\texttt{Publish}(x,\ z)$ $\vee\texttt{Cited}(z, u)$ (ii) relation: $\neg\texttt{Friends}(x, y) \vee \neg\texttt{Related}\ (y,$ $z)\ \vee\texttt{Likes}\ (z, x)$ (iii) longchain: $\neg \texttt{R}_1(x_1,x_2) \vee \neg \texttt{R}_2(x_2,x_3)$ $\ldots\ \texttt{R}_6(x_6,x_7)$ (iv) transitive1: $\neg\texttt{Likes}(x,\ y) \vee \neg\texttt{Likes}(y,$ $z)\ \vee\texttt{Likes}(y,\ x)$ (v) transitive2: $\neg\texttt{Friends}\ (x,\ y)\ \vee$ $\neg\texttt{Friends}(y, z) \vee \texttt{Friends}(z, x)$.

Each synthetic benchmark was designed to illustrate the influence of MLN structure on scalability. Specifically, though similar-looking, student has a smaller treewidth (for its encoded CSP) compared to relation. Longchain illustrates a long formula with small treewidth. Finally, though transitive1 and transitive2 appear similar, it turns out that in each step of Gibbs/MaxWalkSAT, very few messages of the junction tree underlying transitive1 need to be updated while for transitive2, nearly all messages need to be updated.

Figures 2 and 3 show our results for Gibbs sampling and MaxWalkSAT respectively. For our evaluation, we compute two metrics: the compilation time (**C-Time**) in seconds and the sampling/flip rate (**SRate/FRate**). **C-time** is the time taken to initialize our junction trees. **SRate** is the number of samples generated in a second for Gibbs sampling and **FRate** is the number of flips per second in MaxWalkSAT.

Results for Gibbs Sampling

Both **C-time** as well as **SRate** depends upon the structure as well as the #groundings in the MLN. We can see from Figure 2 that **C-time** was quite negligible for almost all the MLNs (at most 21 seconds). **SRate** depends upon the efficiency of updating the junction tree messages during Gibbs sampling. For example, in transitive1, we could generate 88,000 samples/second because each update operation is very efficient, while for transitive2 which has the same #groundings, we could generate only 73 samples/second. Similarly, the MLNs, student-100 and relation-500 have approximately the same #groundings, however, their **SRate**s are vastly different due to the treewidth of their encoded CSPs. Student has treewidth 1 whereas relation has treewidth 2, therefore, while we could collect more than 11,000 samples in a second for student-100, we could only collect about 275 samples for relation-500. On the other hand, for the same treewidth, #groundings affects **SRate**. For example, longchain-1000 is almost 10 million times larger than longchain-100. Therefore, **FRate** on longchain-1000 is just 10% of the **FRate** on longchain-1000. Note that Figure 2 does not include results for Alchemy because it timed out (2 hours) or ran out of memory on all the instances.

Results for MaxWalkSAT

Figure 3 shows our results for MaxWalkSAT. **C-Time** is very similar to the Gibbs sampler. Again, **FRate** depends upon the MLN structure and the number of groundings because both affect the efficiency of the junction tree operations. For instance, since student has lower treewidth than relation, the **FRate** for student is much higher. Transitive2 has the lowest **FRate** because each update involves recomputing the junction tree messages from scratch. When these updates are efficient as in transitive1, **FRate** is several orders of magnitude higher. Both Alchemy and Tuffy did not work on most of the benchmarks except on three of the smallest-sized ones; our approach was slightly worse than Tuffy and Alchemy only on transitive2-100.

In summary, our results clearly demonstrate the superior scalability of our approach over the pre-grounding approaches used by Tuffy and Alchemy.

Conclusion

For large MLNs a sub-step, which is typically a bottleneck, in several inference algorithms is "counting the true groundings of a first-order formula in a possible world." We proposed to solve this counting problem by encoding it as a CSP solution counting problem, thereby allowing us to exploit numerous advances, guarantees, principles and techniques in the active research area of CSP and probabilistic graphical models. Further, we developed a junction tree based exact CSP solution counting algorithm and applied it to two widely used MLN inference techniques, Gibbs sampling and MaxWalkSAT, both of which require an answer to a dynamic version of the counting problem at each iteration. Our experiments with these algorithms on a wide variety of MLN benchmarks with large domain-sizes clearly showed that our approach was orders of magnitude more scalable than existing state-of-the-art inference systems.

Acknowledgments This work was supported by the DARPA Probabilistic Programming for Advanced Machine Learning Program under AFRL prime contract number FA8750-14-C-0005. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views or official policies, either expressed or implied, of DARPA, AFRL or the US government.

References

Bui, H.; Huynh, T.; and Riedel, S. 2013. Automorphism groups of graphical models and lifted variational inference. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence*, 132–141. AUAI Press.

Darwiche, A. 2001. Recursive conditioning. *Artificial Intelligence* 126:5–41.

Darwiche, A. 2003. A Differential Approach to Inference in Bayesian Networks. *Journal of the ACM* 50:280–305.

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

Dechter, R., and Mateescu, R. 2007. AND/OR Search Spaces for Graphical Models. *Artificial Intelligence* 171(2-3):73–106.

Dechter, R.; Kask, K.; Bin, E.; and Emek, R. 2002. Generating random solutions for constraint satisfaction problems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 15–21.

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

Ermon, S.; Gomes, P.G.; Sabharwal, A.; and Selman, B. 2013. *Taming the Curse of Dimensionality: Discrete Integration by Hashing and Optimization*. In *Proceedings of the 30th International Conference on Machine Learning*, 334–342.

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

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

Gogate, V., and Dechter, R. 2007. Approximate Counting by Sampling the Backtrack-free Search Space. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, 198–203. AAAI Press.

Gogate, V., and Domingos, P. 2011. Probabilistic Theorem Proving. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, 256–265. AUAI Press.

Gogate, V. 2009. *Sampling Algorithms for Probabilistic Graphical Models with Determinism*. Ph.D. Dissertation, University of California, Irvine.

Kautz, H.; Selman, B.; and Jiang, Y. 1997. A General Stochastic Approach to Solving Problems with Hard and Soft Constraints. In Gu, D.; Du, J.; and Pardalos, P., eds., *The Satisfiability Problem: Theory and Applications*. New York, NY: American Mathematical Society. 573–586.

Kok, S.; Sumner, M.; Richardson, M.; Singla, P.; Poon, H.; and Domingos, P. 2006. The Alchemy System for Statistical Relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA. http://alchemy.cs.washington.edu.

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

Mateescu, R.; Dechter, R.; and Marinescu, R. 2008. AND/OR Multi-Valued Decision Diagrams (AOMDDs) for Graphical Models. *Journal of Artificial Intelligence Research* 33:465–519.

Niu, F.; Ré, C.; Doan, A.; and Shavlik, J. W. 2011. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *PVLDB* 4(6):373–384.

Poole, D. 2003. First-Order Probabilistic Inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 985–991. Acapulco, Mexico: Morgan Kaufmann.

Sarkhel, S.; Venugopal, D.; Singla, P.; and Gogate, V. 2014. Lifted MAP inference for Markov Logic Networks. *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS-14).*

Selman, B.; Kautz, H.; and Cohen, B. 1996. Local Search Strategies for Satisfiability Testing. In Johnson, D. S., and Trick, M. A., eds., *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. Washington, DC: American Mathematical Society. 521–532.

Van den Broeck, G.; Taghipour, N.; Meert, W.; Davis, J.; and De Raedt, L. 2011. Lifted Probabilistic Inference by First-Order Knowledge Compilation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2178–2185.

Vardi, Moshe Y. 1982. The Complexity of Relational Query Languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, 137–146.

Venugopal, D., and Gogate, V. 2012. On lifting the gibbs sampling algorithm. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS)*, 1664–1672.

Venugopal, D.; Sarkhel, S.; and Gogate, V. 2014. Just Count the Satisfied Groundings: Scalable Local-Search and Sampling Based Inference in MLNs. Technical Report, Computer Science Department, University of Texas at Dallas.

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