Speeding Up Inference in Markov Logic Networks by Preprocessing to Reduce the Size of the Resulting Grounded Network

Jude Shavlik and Sriraam Natarajan
Department of Computer Sciences
University of Wisconsin, Madison USA

Abstract

Statistical-relational reasoning has received much attention due to its ability to robustly model complex relationships. A key challenge is *tractable inference*, especially in domains involving many objects, due to the combinatorics involved. One can accelerate inference by using approximation techniques, "lazy" algorithms, etc. We consider Markov Logic Networks (MLNs), which involve counting how often logical formulae are satisfied. We propose a preprocessing algorithm that can substantially reduce the effective size of MLNs by rapidly counting how often the evidence satisfies each formula, regardless of the truth values of the query literals. This is a general preprocessing method that loses no information and can be used for any MLN inference algorithm. We evaluate our algorithm empirically in three real-world domains, greatly reducing the work needed during subsequent inference. Such reduction might even allow exact inference to be performed when sampling methods would be otherwise necessary.

1 Introduction

Over the past decade, there has been an increasing interest in addressing challenging problems that involve rich relational and probabilistic dependencies in data. Statistical Relational Learning (SRL) [Getoor and Taskar, 2007], which combines the power of probability theory with the expressiveness of the first-order logic, has been actively explored for this reason.

Inference is one of the central problems in these SRL models. Most approaches ground (i.e., bind variables to constants) first-order statements and directly use these grounded sentences to produce graphical models, then perform inference in these models. However, the number of ground objects often can be prohibitively large and prevent tractable inference. Consequently, sampling techniques [Domingos and Lowd, 2009; Milch and Russell, 2006] and lifted-inference methods [Braz *et al.*, 2005; Singla and Domingos, 2008], which avoid explicitly grounding all the logical statements, have been explored. In addition to answering statistical queries, learning of the parameters and structures of these models involve repeated (statistical) inference in their inner loops, often in the presence of hidden variables. Hence accelerating inference is an important research challenge.

We consider Markov Logic Networks (MLNs)[Domingos and Lowd, 2009]. For inference, MLNs require counting the number of grounded statements that are true given the current world state. Such counting can become impractical in the presence of much data, since the number of groundings can grow exponentially as the number of objects in a domain increases. We present an algorithm that we call *Fast Reduction Of Grounded networks* (FROG) that rapidly counts those groundings that are true independent of any subsequent queries, and hence this information need not be recomputed repeatedly as queries are asked of the MLN. FROG also produces a reduced grounded Markov network that can be orders of magnitude smaller than it would be without FROG's preprocessing; in this reduced network only those counts that are dependent on subsequent queries need to be calculated.

Our method is essentially a pre-processing step that supports simpler and hence faster inference. For example, consider the statement, $\forall x, y, z\ p(x, y) \rightarrow q(y, z) \vee r(x, z)$. If each of these logical variables (x, y, z) ranges over one million people, then there are 10^{18} groundings! However, if we know that $p(x, y)$ is only true for 100 specific groundings (and false otherwise), in one fell swoop we know that $(10^{12} - 100) \times 10^6$ of these groundings are true, regardless of $q(x, y)$ and $r(x, z)$, and can treat that large set of groundings as a unit without needing to explicitly enumerate them all. Should we also know that $r(x, z)$ is false for only $10,000$ of the remaining 10^8 groundings of this relation, then we know that another $(10^8 - 10^4)$ groundings are true, and we only need to consider during inference those 10^4 of the original 10^{18} groundings whose truth values depend on the $q(x, y)$'s.

In the next section, we present background on MLNs and briefly define some terms we use. In the third section, we present our algorithm and another worked example. Next, we present the experimental results in three domains already in the literature. Finally, we conclude by discussing some areas of future research and additional related work.

2 Background

SRL combines the strengths of logic (Inductive Logic Programming (ILP), relational databases, etc.) and probabilistic learning (Bayesian networks, Markov networks, etc.). An

ILP system learns a logic program given background knowledge as a set of first-order logic formulae and a set of examples expressed as *facts* represented in logic. In first-order logic, *terms* represent objects in the world and comprise *constants* (e.g., Mary), *variables* (x), and *functions* (fatherOf(John)). *Literals* are truth-valued and represent properties of objects and relations among objects, e.g. *married(John, Mary)*. Literals can be combined into compound sentences using connectives ($\rightarrow$, $\neg$, $\wedge$, etc.). It is common [Russell and Norvig, 2003], to convert sets of sentences into a canonical form, producing sets of *clauses*, and FROG assumes its input is a set of such clauses. A clause is a disjunction of literals, some of which may be negated, e.g., $p(x) \vee \neg\, q(y)$. A clause is *satisfied* if at least one of its literals has the correct truth value (i.e., if negated, a literal is false; otherwise it is true). A key aspect of clauses that our algorithm exploits is that *only one* of possibly many literals needs to have the correct truth value in order for a clause to be satisfied.

A *ground literal* is one with no variables as its arguments, and a grounded clause is one where all of its literals are grounded (as is standard with MLNs, we assume we have finite domains and that all functions are replaced by the constants to which they refer). Grounding refers to the process of assigning constants to variables. A ground literal can be *true*, *false* or *unknown* in a given knowledge base. *Evidence* literals are facts whose truth values are given or observed in the domain and *query* literals are those about which statistical questions are asked ("what is $Prob(siblings(John, Mary) \mid evidence)$?"). The commonly used *Closed World Assumption* (CWA) states that for certain literals, if they are not known to be true, assume they are false. CWA is not applied to query literals. One can mark additional literals, saying that CWA should not be applied to them; we call these *hidden* literals, because they are not explicitly stated when one makes queries about conditional probabilities ("hidden" literals will explicitly appear in an MLN knowledge base).

SRL models add probabilistic semantics to first-order logic. They often are graphical models whose arcs are either directed or as is the case in MLNs, undirected. An MLN is a set of weighted first-order clauses. Given a set of constants, an MLN defines a (potentially huge) Markov network whose nodes are the ground literals. Each ground clause becomes a clique in the Markov Network, and the weight associated with the clause is the so-called *clique potential*.

The key calculation in MLNs is the following, where $num(F, s)$, counts how many times an ungrounded logical formula F is true in a state s of the world, and where Z is a normalizing term that ranges over all possible world states:

$$Prob(World\ X) = (1/Z)e^{\left(\sum_i weight_i \times num(F_i, X)\right)} \quad (1)$$

The state-of-the-art algorithm for computing MLN probabilities uses a combination of Markov chain Monte Carlo and a SAT solver [Domingos and Lowd, 2009]. The structure of MLNs is typically learned via ILP, while the weights are usually learned via gradient descent. If there are hidden literals, statistical inference repeatedly has to be performed in the inner loop of weight learning.

A central issue with inference in MLNs is that while computing probabilities, one counts the total number of groundings. The combinatorics of doing so can be prohibitively large in many domains. For instance, if a literal has 5 variables, each of which can take about 100 values, the number of groundings for that literal alone is 100^5, and such sizes readily occur in real-world domains. Computing the groundings for all the clauses in MLN will thus be at least as hard. Hence, Poon *et al.* [2008] proposed a method by which they evaluate the clauses "lazily." Their key idea is to cache only the non-default values of the literals and clauses, and consider only this list for future statistical sampling of possible world states. We compare and contrast their method, as well as other related approaches, to ours later.

3 Fast MLN Grounding via FROG

As mentioned, a key issue with MLNs is the counting of the number of groundings of a clause. While exact inference is accurate but impractical in many domains, sampling methods sacrifice accuracy for efficiency. Our algorithm might reduce the size of a grounded MLN sufficiently to allow exact inference where otherwise it would be intractable, as well as reduce the size of the grounded network to which sampling need be applied, possibly increasing accuracy given a fixed computation-time budget.

We can break down Equation 1's world state into two sets: S_A, the evidence literals, and S_B, the query and hidden literals. Let A be shorthand for the weighted count of the number of times formulae are satisfied by literals in set S_A in state X; ditto for S_B and B, except here B only counts those formula not already satisfied by a literal in S_A. Our algorithm exploits the fact that $e^{A+B} = e^A \times e^B$.

Assuming the evidence is unchanged, the "A" part, which counts the groundings due to the evidence literals, is constant across all world states, and hence the identical e^A appears in *all summands* in Z, i.e., $Z = e^A \times (e^{B_1} + e^{B_2} + ... + e^{B_n})$. The e^A in the numerator and denominator of Eq. 1 cancel and hence, without any loss of accuracy, we only need to compute the e^{B_i} terms in order to compute probabilities.

In a knowledge base, each literal is in one of the following cases: it *satisfies* the clause, it does *not satisfy* the clause, or its truth value is is *unknown*. Roughly speaking, our FROG algorithm stores the count of the variable groundings that satisfy grounded clauses (since these counts may be needed during learning), discards (from specific clauses) those grounded literals that are known to not satisfy, and keeps those ground literals that might satisfy grounded clauses not satisfied by the evidence. If there are many fewer potentially satisfiable combinations than the total number of possible groundings, FROG saves substantial work during inferencing.

3.1 The FROG Algorithm

Table 1 presents FROG (the next section presents a worked example, and readers may wish to visit that while reading this section). FROG processes each clause in an MLN independently. Any clauses that contain no query nor hidden literals can be removed from the MLN before FROG starts because their counts will be constant in all world states (see above).

Assume clause c has distinct variables $v_1, \ldots, v_N$. For example, $p(x, y, 1, x) \vee \neg q(x, z)$ has three distinct variables,

and literal p only two (even though its arity is four). As it progresses, FROG maintains the tuples $\langle v_1, \ldots, v_N \rangle$ that can still satisfy clause c; each such tuple represents a grounding of c. Initially the number of these is simply

$$\prod_{i=1}^{N} | \, constantsWithTypeOf(v_i) \, |$$

that is, the product of the number of known constants whose type matches each v_i's type. The initial tuples can be recorded implicitly, but as FROG calculates it may need to record explicit tuples that possibly satisfy clause c, and a key goal of FROG is to keep the number of explicit tuples relatively small (more details in a later subsection). FROG's main goal is, for each clause c, to compute a "reduced" form of c containing:

1. The number of times c is true, given the evidence (and any use of CWA), regardless of the truth value of query and hidden literals.
2. Any remaining literals in the reduced form of clause c.
3. Any remaining tuples that can satisfy the reduced clause; the grounded clauses that result from these will be addressed by some inference method after FROG is done.

Imagine FROG selects clause c's literal p (later we will discuss how FROG chooses the order for processing a clause's literals). When FROG uses the remaining tuples to ground literal p, these groundings fall into three disjoint categories:

1. Those whose truth value is known, possibly via CWA, and satisfy clause c. The number of these we call $\#sat_p$.
2. Those whose truth value is known and do not satisfy c. These are counted in $\#unsat_p$.
3. Those whose truth value is unknown; counted by $\#unk_p$.

What does FROG do with these statistics?

1. It increments the count of groundings that satisfy clause c by $\#sat_p$.
2. If $\#unk_p = 0$, FROG can drop literal p from the reduced version of clause c; otherwise it is kept and it will need to be dealt with during inference following FROG's processing.
3. If $\#unk_p + \#unsat_p = 0$, FROG need no longer consider this clause; all of its groundings have been accounted for and any unprocessed literals will not appear in the reduced version of c.
4. It updates its "sparse" representation of the tuples remaining for this clause.

If FROG has processed all the literals in a clause but retained no literals, then the clause is unsatisfied (i.e., false) for any remaining tuples, because no literal satisfied the clause when using these variable bindings. After reducing all the clauses provided to it, FROG merges identical remaining clauses, summing their weights.

Table 1: The FROG Algorithm for Creating a Reduced MLN

Given: an MLN and the grounded evidence
Do:
for each clause c in the MLN
  $c.done = false$
  $c.reducedLiterals$ = {}
  $c.\#satisfied$ = 0
  $c.tuples$ = sparse representation of groundings of c
  while (c has unprocessed literals and $\neg c$.done)
    heuristically choose unprocessed literal p
    compute $\#sat_p$, $\#unsat_p$, and $\#unk_p$ over $c.tuples$
    add $\#sat_p$ to $c.\#satisfied$
    if $\#unk_p \neq 0$ add p to $c.reducedLiterals$
    if $\#unsat_p + \#unk_p = 0$ then $c.done = true$
    remove the tuples counted by $\#sat_p$ from $c.tuples$
    mark p as processed
  aggregate equivalent ground clauses

3.2 Worked Example and Discussion

As an example, consider the following:

$$GradStudent(x) \wedge Prof(y) \wedge Prof(z) \wedge TA(x,z) \\ \wedge SameGroup(y,z) \rightarrow AdvisedBy(x,y).$$

where x, y, z are of type *human*. Assume there are $10,000$ people in a school. The total number of groundings that we need to consider while counting is $|x| \times |y| \times |z| = 10^{12}$.

Evaluating this many combinations is challenging. Assume that *AdvisedBy* is the query literal (for all groundings), and that there are about $1,000$ professors and $2,000$ graduate students. Imagine that 500 pairs of professors are in the same group (if A and B are in the same group so are B and A, thus *SameGroup* is true $1,000$ times). Among the students $1,000$ of them served as a TA.

FROG proceeds as follows. First, it chooses an unary literal randomly, say *GradStudent(x)*, and computes the number of times it is true $(2,000)$. Applying the closed-world assumption (CWA), it considers the remaining $8,000$ grounding to be false. These $8,000 \times 10^4 \times 10^4$ groundings satisfy the clause, and FROG records this count. Only the remaining $2,000$ groundings for x are kept.

Next, FROG processes *Prof(y)*, increases the count of satisfied groundings, and reduces the number of groundings for y to $1,000$. The same happens for the second occurrence of *Prof* and only $1,000$ bindings for variable z remain. So just from these simple steps, FROG reduces this original set of 10^{12} groundings to 2×10^9.

Once all the literals containing only one unique variable have been reduced, FROG proceeds to literals with more than one unique variable. Let us say it now picks *SameGroup(y,z)* to reduce. Although there are 10^6 possible combinations (corresponding to the remaining y and z values), only $1,000$ of them are true; the remaining $(10^6 - 1,000)$ combinations satisfy the clause. Now the number of groundings left to consider is down to 2×10^6, the remaining $1,000$ *y:z* combinations times the $2,000$ remaining bindings for x.

Processing *TA(x,z)* occurs next, and after doing so at most $1,000$ bindings for x will remain, and the $1,000$ *y:z* combinations will likely also be reduced. The final result is that, out of 10^{12} groundings, FROG quickly reduces this to at most 10^6 groundings, without making any approximations.

This huge savings is obtained due to the sparsity of the domain, but such sparsity is common in real-world tasks. FROG also works well when (a) CWA is employed and (b) the clauses in the MLN contain many negated literals, since it is these literals that CWA allows FROG to use to satisfy clauses. Horn clauses, which are commonly learned by ILP algorithms, typically contain multiple negated literals, but FROG can handle cases when there is more than one unnegated literal, e.g., $p(x) \wedge q(y) \rightarrow r(x,y) \vee w(y,x)$, which in clausal form is $\neg p(x) \vee \neg q(y) \vee r(x,y) \vee w(y,x)$.

FROG becomes increasingly helpful the more distinct variables a clause involves, since it is this number that determines the number of groundings.

FROG can handle cases that are considered hard for other methods. For instance, the lazy-inference method [Poon *et al.*, 2008] has difficulties handling clauses that contain existential quantifiers and hence Poon *et al.* had to remove such clauses from their experiments. Because FROG precomputes counts, it becomes possible for us to handle existential quantifiers and we include the clauses with these quantifiers in our experiments.

FROG does not help on clauses that only contain query and hidden literals, since no reduction is possible without evidence (assuming the clauses are not tautological). Fortunately, FROG can quickly complete its processing of such clauses - all it needs to do is check if any evidence literals are present, and if not it is done. Whichever inference process follows FROG's processing needs to deal with such clauses.

FROG also provides little gain when clauses only contain unnegated literals, assuming the evidence contains few cases where such literals are true. For instance, consider the clause $p(x,y) \vee q(x,y,z)$. If there is a total of 10^{12} combinations of variables x, y, and z, and $q(x,y,z)$ is a query literal, then knowing $p(x,y)$ is only true $1,000$ times is of little help. The reduced clause still involves $(10^{12} - 10^3)$ groundings. Here again the complexity of inferencing needs to be addressed by some other method, such as lazy inference.

Lazy inference [Poon *et al.*, 2008] shares many of the goals of FROG. An initialization step in lazy inference is to collect all those clauses not satisfied by the evidence [Domingos and Lowd, 2009]. FROG provides an efficient way to perform this calculation. However, note that since FROG explicitly produces a reduced MLN, it is possible to judge whether or not lazy inference is even needed. It might be the case that the MLN has been so reduced that exact inference is possible. In addition, by making explicit the reduced MLN, any MLN inference algorithm can be subsequently employed, including those for which lazy versions have not been implemented. Note that FROG and methods such as sampling and lazy inference are not *competitors*. Rather, such approaches can be synergistically combined, since they complement one another.

As an informal runtime comparison, we also ran Alchemy (http://alchemy.cs.washington.edu/) using Poon *et al.*'s lazy evaluation [Poon *et al.*, 2008] on the CORA dataset (see next section). Alchemy, written in C/C++ spent 1 hour and 34 minutes for ground-network construction, while the Java-based FROG took 30 minutes to construct the ground network (we ran both on the same workstation). The reduced network, which has 10^6 groundings, is small enough to fit in main memory, eliminating the need for doing lazy inference.

3.3 Additional Algorithmic Details

The key challenge in FROG is maintaining a sparse representation of the remaining tuples in each clause, while reducing the number of these tuples to the minimal required to appear in the reduced MLN (i.e., the set of reduced clauses upon which subsequent inference is to take place).

Sometimes it is easy to maintain a sparse representation. Recall that initially we only need to explicitly keep track of the constants of each variable's type. Say we have the clause $\neg p(x) \vee \neg q(y,z) \vee r(x,y,z,x)$, where each of the three variables involves 10^6 constants and $r(x,y,z,x)$ is true for half of its groundings. Initially we have 10^{18} groundings, but need no storage, because the storage of constants of each type is shared across all clauses. Should FROG choose to reduce $p(x)$ and find that only 100 constants do not satisfy $\neg p(x)$, it would record that $(10^6 - 100) \times 10^{12}$ groundings are true and store 100 constants to sparsely represent the remaining 10^{14} groundings.

Imagine that FROG next chooses to process $q(y,z)$. Here it would have to explicitly maintain those *y:z* pairs that do not satisfy $\neg q(y,z)$; assume there are 1000. Now, FROG is storing the 100 constants for x and the 1000 *y:z* pairs. Finally it needs to process $r(x,y,z,x)$, and at this point FROG needs to produce the cross product of 10^5 tuples from the 100 stored values for x and the 1000 *y:z* pairs. Notice that if FROG had instead first processed $r(x,y,z,x)$ it would have needed to store the $\frac{1}{2} \times 10^{18}$ groundings for which this literal is $false$.

Hence it is important that FROG do a reasonable job of choosing literals to process without needing to use large amounts of space. It heuristically does this as follows. First it processes all literals with at most one distinct variable, such as $p(x)$, $q(y,y)$, or even $r()$; such literals only lead to shortened lists of size-one (or size-zero) tuples. After this, FROG scores each unprocessed literal p by $(\#unsat_p + \#unk_p)$ / $(\#sat_p + \#unsat_p + \#unk_p)$ and chooses the one with the lowest score, since this quantity represents the fraction of tuples that remaining after the literal is processed. However, unless there are no other candidates, FROG defers chooses literals whose number of remaining groundings is too large (e.g., 10^6); note that on later rounds the number of remaining groundings for such literals may have greatly shrunk.

FROG's literal selection is a greedy algorithm and hence is not guaranteed to produce the smallest possible representation of the remaining tuples throughout FROG's processing of a clause. Should an intermediate representation get too large, our implementation abandons the greedy algorithm and multiple times attempts to randomly choose the order to process literals (since clauses are unlikely to have more than a handful of literals with more than one distinct variable, it would also be possible to try all $k!$ orderings, stopping each try once the intermediate storage needs become too large). It should

also be noted that FROG provides the same reduced MLN regardless of the order it processes literals, assuming it does not run out of space during intermediate calculations.

Finally, one should note that as FROG operates, the same reduced clause can appear many times. For example, assume FROG is given $p(x, y) \rightarrow q(y)$, where p is in the evidence and q is the query. Imagine that p is only true for: $p(1, 2)$, $p(2, 2)$, $p(3, 2)$, and $p(1, 3)$. After processing p, FROG will have these grounded clauses: $\{q(2)\}$, $\{q(2)\}$, $\{q(2)\}$, and $\{q(3)\}$. FROG groups such duplicates into one grounded clause, and assigns such groups the sum of the weight of each grounded clause. In this example, instead of needing to keep four grounded clauses, FROG needs to only have two in the reduced MLN it produces.

Theorem: Given a MLN M, a set of evidence literals E, a set of query literals Q, and a set of hidden literals H, the algorithm FROG(M, E, Q, H) produces a reduced MLN MLN_R. Exact inference on MLN_R only involves literals in Q and H and always produces the same results as exact inference on the original MLN M

Proof Sketch: The first part of the theorem follows from the definition of the reduced network , which involves only the query and hidden literals. The second part of the theorem can be proved by considering the explanation provided in Section 2 where the counts for M can be split into two parts: one containing the counts due to the evidence literal and the other the counts due the the query and hidden literals . Since the evidence-literal counts is constant for all the groundings, they can be cancelled out, yielding only the query and hidden literal counts. This corresponds to computing the probabilities in MLN_R. Hence exact inference in MLN_R yields same result as exact inference in M.

4 Experiments

We evaluate our algorithm on three well-known, real-world datasets: Citeseer [Lawrence *et al.*, 1999], CORA [Bilenko and Mooney, 2003], and UW-CSE [Poon and Domingos, 2007]. The two citation domains involve information extraction, while the UW-CSE domain involves link prediction. The Citeseer and Cora datasets contain information about $1,593$ and $1,285$ citations, respectively and involve thousands of logical-inference rules ($8,334$ and $2,087$) and many facts ($102,475$ and $168,462$). Both of these datasets involve the same two query literals: whether or not two citations refer to the same paper ($SameBib$) and whether a particular paper is in a particular field ($InField$).

The other testbed we use is the UW-CSE domain, where the goal is to predict the $AdvisedBy$ relationship between a professor and a student. This database consists of 278 faculty members and students, 94 clauses and $2,008$ facts. We only used the 27 rules that contain $AdvisedBy$ since the others have no impact when all other literals are in the evidence.

We compare our FROG algorithm against the total number of grounded clauses. In all three domains, we vary the total number of objects in the domain, following methodology previously used [Poon *et al.*, 2008].

The results on UW-CSE appear in Figure 1. In order to understand them, we will first discuss the results on the full

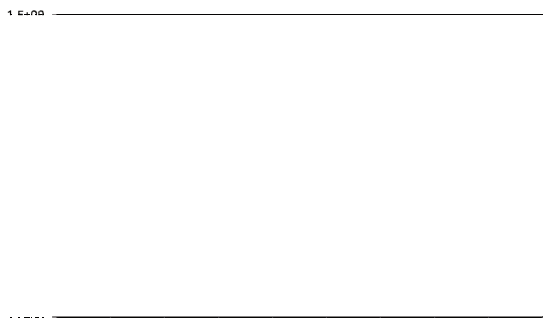

Figure 1: Results in the UW-CSE Domain

dataset (far right of the curves). The total number of possible groundings on the full data set is 1.1×10^9, while that of FROG's reduced MLN is 2.2×10^7, a 50-fold reduction. However, these numbers are skewed by the presence of one challenging clause, $AdvisedBy(s, p) \wedge AdvisedBy(s, q) \rightarrow SamePerson(p, q)$. Since the single evidence literal in this clause is unnegated, FROG can do little to reduce its number of groundings, as discussed earlier. Discarding this clause, the number of possible groundings is only reduced by 2.2×10^7 and remains 1.1×10^9, but the reduced MLN FROG produces only contains 6.6×10^5 clauses, a 1,700-fold reduction. FROG, which is written in Java, takes about 4.2 seconds on the full dataset for reduction on a 3 GHz Windows XP machine. The four combinations, with and without FROG and with (labeled *MLNHARD*) and without this hard clause, are the four curves in Figure 1. Note that the two fully grounded networks are same irrespective of whether or not the the hard clause is present.

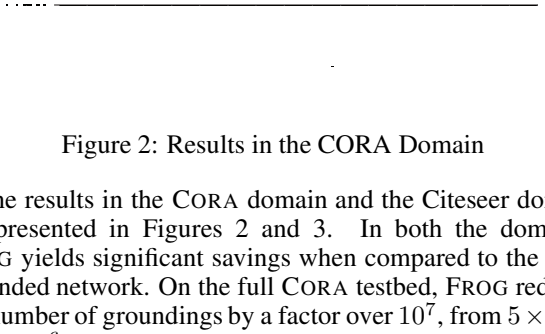

Figure 2: Results in the CORA Domain

The results in the CORA domain and the Citeseer domain are presented in Figures 2 and 3. In both the domains, FROG yields significant savings when compared to the fully grounded network. On the full CORA testbed, FROG reduces the number of groundings by a factor over 10^7, from 5×10^{13} to 2×10^6. Similarly, in the full Citeseer task, FROG reduces the network size by a factor of 5×10^6, from 1.7×10^{13} to 3.0×10^6. In both the domains, FROG yields a million times

fewer groundings. FROG spends $1,350$ seconds to process CORA and $5,150$ seconds on Citeseer, which is less than 700 milliseconds per clause.

When using the full Citeseer dataset, there are 1.7×10^{13} possible groundings. However, FROG only needs to consider 2.0×10^6 of these groundings (about 1 in 10 million) in order to determine that $3.0x10^6$ groundings remain. For Cora these numbers are 4.8×10^{13} possible, 2.3×10^6 considered, and 2.1×10^6 remaining, while for UW-CSE they are 1.1×10^9 possible, 6.6×10^5 considered, and 2.2×10^7 remaining (without the one challenging clause mentioned above, only 6.6×10^5 groundings remain). These numbers show that FROG can greatly reduce the size of a grounded network while only considering a small fraction of the complete set of groundings, as FROG exploits the fact that to satisfy a clause, one needs to find only *one* literal with the proper truth value.

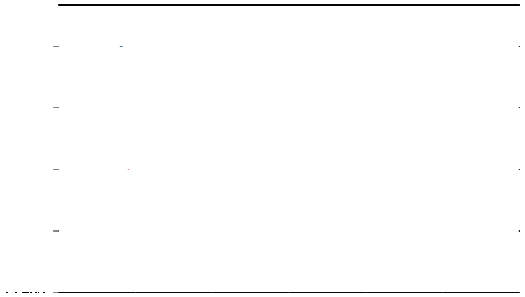

Figure 3: Results in the Citeseer Domain

5 Related Work and Conclusion

We have presented an algorithm, FROG, for preprocessing the groundings in MLNs and demonstrated success on three real-world datasets from the literature. FROG creates grounded MLNs that are equivalent to the original MLNs, but often orders of magnitude smaller. In order to create such reduced networks, FROG need only consider a small fraction of all the possible groundings. The key aspect it exploits is that the grounded clausal form of logical sentences only require one literal to have the proper truth value, and whenever the evidence contains such literals, FROG can count large blocks of groundings in one fell swoop. Earlier we discussed lazy inference and lifted inference, both of which are closely related and complementary to FROG. Before concluding, we discuss additional related work.

Recently Milch et al. [2008] explored the idea of counting formulas to accelerate inference in relational models. They extend an earlier method of lifted inference [Braz *et al.*, 2005] by extending the notion of parameterized factors (*parfactors*) with counting formulas. Another way to accelerate inference is by converting to arithmetic circuits [Chavira *et al.*, 2004]. Sen *et al.* [2008] propose another "clustered" inference approach that exploits the shared correlation in probabilistic databases by creating a new graph, which is then compressed. Our method has similarities to lifted inference in that it clusters the groundings together. FROG groups together in one, potentially huge, group all those groundings where the evidence satisfies a given clause, regardless of the state of the query and hidden variables.

One future area of research is to investigate the combination of lifted inference with our preprocessing algorithm. Lifted-inference methods might find additional clusters in FROG's reduced networks. A reduced MLN produced by FROG can answer a larger number of subsequent queries without any additional calculation, but another future research direction is to extend FROG to the case where evidence is allowed to change. Ideally FROG would perform an incremental amount of additional work when only a small portion of the evidence changes. Another possible research problem is to develop more efficient heuristics for choosing which literal FROG should process next; the order literals FROG reduces literals greatly impacts the number of groundings it considered. Finally, FROG does not yet exploit during its reduction phase commonality across clauses, many of which are likely to share multiple literals.

Acknowledgements The authors gratefully acknowledge support of the Defense Advanced Research Projects Agency under DARPA grants FA8650-06-C-7606 and HR0011-07-C-0060. Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied of the US government or of DARPA.

References

[Bilenko and Mooney, 2003] M. Bilenko and R. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, 2003.

[Braz *et al.*, 2005] R. De Salvo Braz, E. Amir, and D. Roth. Lifted first-order probabilistic inference. In *IJCAI*, 2005.

[Chavira *et al.*, 2004] M. Chavira, A. Darwiche, and M. Jaeger. Compiling Relational Bayesian networks for exact inference. *International Journal of Approximate Reasoning*, 42:49–56, 2004.

[Domingos and Lowd, 2009] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for AI*. Morgan and Claypool, 2009.

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

[Lawrence *et al.*, 1999] S. Lawrence, C. Giles, and K. Bollacker. Autonomous citation matching. In *ICAA*, 1999.

[Milch and Russell, 2006] B. Milch and S. Russell. General-purpose MCMC inference over relational structures. In *UAI*, 2006.

[Milch *et al.*, 2008] B. Milch, L. Zettlemoyer, K. Kersting, M. Haimes, and L. Kaelbling. Lifted probabilistic inference with counting formulas. In *AAAI*, 2008.

[Poon and Domingos, 2007] H. Poon and P. Domingos. Joint inference in information extraction. In *AAAI*, 2007.

[Poon *et al.*, 2008] H. Poon, P. Domingos, and M.Sumner. A general method for reducing the complexity of relational inference and its application to MCMC. In *AAAI*, 2008.

[Russell and Norvig, 2003] S. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach*. Prentice Hall, 2003.

[Sen *et al.*, 2008] P. Sen, A. Deshpande, and L. Getoor. Exploiting shared correlations in probabilistic databases. In *VLDB*, 2008.

[Singla and Domingos, 2008] P. Singla and P. Domingos. Lifted first-order belief propagation. In *AAAI*, 2008.