Constraint Propagation for Efficient Inference in Markov Logic

Tivadar Papai[1] papai@cs.rochester.edu
Parag Singla[2] parag@cs.utexas.edu

Henry Kautz[1] kautz@cs.rochester.edu

[1] University of Rochester, Rochester NY 14627, USA
[2] University of Texas, Austin TX 78701, USA

Abstract. Many real world problems can be modeled using a combination of hard and soft constraints. Markov Logic is a highly expressive language which represents the underlying constraints by attaching real-valued weights to formulas in first order logic. The weight of a formula represents the strength of the corresponding constraint. Hard constraints are represented as formulas with infinite weight. The theory is compiled into a ground Markov network over which probabilistic inference can be done. For many problems, hard constraints pose a significant challenge to the probabilistic inference engine. However, solving the hard constraints (partially or fully) before hand outside of the probabilistic engine can hugely simplify the ground Markov network and speed probabilistic inference. In this work, we propose a generalized arc consistency algorithm that prunes the domains of predicates by propagating hard constraints. Our algorithm effectively performs unit propagation at a lifted level, avoiding the need to explicitly ground the hard constraints during the pre-processing phase, yielding a potentially exponential savings in space and time. Our approach results in much simplified domains, thereby, making the inference significantly more efficient both in terms of time and memory. Experimental evaluation over one artificial and two real-world datasets show the benefit of our approach.

1 Introduction

Combining the power of logic and probability has been a long standing goal of AI research. The last decade has seen a significant progress towards this goal, with the emergence of the research area called statistical relational learning (SRL). Many different representation languages have been proposed which combine subsets of full-first order logic with various probabilistic graphical representations [4]. One such powerful language is Markov Logic [2], which represents a joint probability distribution over worlds defined by relationships over entities by attaching weights to formulas in first order logic.

A Markov logic theory can be seen as a combination of hard and soft constraints. Hard constraints are modeled by formulas with infinite weight, and must be satisfied in any world with non-zero probability. The typical approach to inference in Markov logic involves grounding out the theory and jointly dealing with both hard and soft constraints. For many problems, hard constraints can pose a significant challenge to the underlying probabilistic inference engine, making it difficult for a sampler to move between different modes. Much work has gone into the development of probabilistic inference algorithms that are robust in the face of hard constraints (for example, MC-SAT [11], SampleSearch [5]), but the general problem of efficiently handling hard constraints is far from solved.

The key idea in this paper is that, intuitively, each hard constraint in the knowledge base reduces the set of possible worlds that have a non-zero probability. In particular, a set of hard constraints together can restrict the number of groundings of a predicate about which we are uncertain (i.e., the probability of an instance of the predicate holding is strictly between 0 and 1). We refer to this phenomenon as *domain pruning*. Domain pruning can significantly simplify the ground network over which probabilistic inference needs to be done, as the pruned groundings can be treated as *evidence* (fully observed). Therefore, we propose an approach to probabilistic inference which has two components: 1) Solve the hard constraints (fully or partially) to identify the pruned domains 2) Use a standard probabilistic inference engine with pruned domains input as evidence. Building on ideas in the area of constraint satisfaction, we propose a novel generalized arc consistency algorithm for propagating the hard constraints. Since our algorithm deals only with hard constraints to prune the domains, it is guaranteed to produce the same solution as the standard techniques. Our algorithm can be seen as a form of lifted unit propagation. We show that our approach can use exponentially less space and time than performing unit propagation on the grounded theory. Experiments on three different datasets clearly show the advantage our approach.

The organization of this paper is as follows. We first present some background on Markov logic and constraint propagation. This is followed by the details of the generalized arc consistency algorithm. We present our results on two real and one artificial datasets. Next, we discuss some of the related work in this area. We conclude with the directions for future work.

2 Background

2.1 Markov Logic

First-order probabilistic languages combine graphical models with elements of first-order logic, by defining template features that apply to whole classes of objects at once. One such powerful language is *Markov logic* [2]. A *Markov logic network (MLN)* is a set of weighted first-order formulas. The weight of a formula represents the strength of the constraint. *Soft* constraints are formulas with finite weight, while *hard* constraints have infinite weight. A theory consists of a combination of hard and soft constraints. Together with a set of constants representing the objects of interest, it defines a Markov network with one node per ground atom and one feature per ground formula. The weight of a feature is the weight of the first-order formula that originated it. More formally,

Definition 1. *[2] A Markov logic network (MLN) L is a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, \ldots, c_{|C|}\}$, it defines a Markov network $M_{L,C}$ as follows:*

1. *$M_{L,C}$ contains one binary node for each possible grounding of each predicate (ground atom) appearing in L. The value of the node is 1 if the ground predicate is true, and 0 otherwise.*
2. *$M_{L,C}$ contains one feature for each possible grounding of each formula F_i (ground formula) in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i associated with F_i in L.*

For many problems, a set of ground atoms are known to be *true* or *false* before hand. These are known as *evidence atoms.* The ground atoms whose value is not known at the inference time are called *query atoms.* The ground Markov network $M_{L,C}$ defines the probability of an assignment y to the query atoms Y, given an assignment x to the evidence atoms X, as

$$P(Y = y|X = x) = \frac{1}{Z_x} \exp\left(\sum_k w_k f_k(y, x)\right) \tag{1}$$

where the summation is taken over all the ground formulas. w_k is the weight of the kth ground formula, $f_k = 1$ if the kth ground formula is true, and $f_k = 0$ otherwise, and Z_x is the normalization constant. For any state to have a non-zero probability, all the hard constraints have to be satisfied, in which case the corresponding weight term (infinite) can be factored out from the denominator as well as the numerator. The evidence atoms can be input to the inference procedure in the form of a set called *evidence database.* The value of evidence atoms is set by fixing the assignment of the corresponding nodes in the network to the respective truth value. A large evidence results in effectively pruning the network, as the corresponding nodes assignments can be fixed and removed from the network. Marginal inference corresponds to the problem of finding the probability of true assignment to each of the query nodes in the network. Inference can be done using standard inference techniques such as Gibbs sampling or belief propagation. More efficient techniques which exploit the nature of the formula (hard or soft) [11] or the structure of the network [16] have also been proposed. None of these techniques is able to exploit the fact that the set of hard constraints in the knowledge base can, in many instances, be solved very efficiently, thereby significantly pruning the domains of predicates and shrinking the number of ground formulas.

Any first-order knowledge base can be equivalently converted into a clausal form by a series of mechanical steps [12]. We deal with finite first order logic, and all the function evaluations are assumed to be known in advance [2]. Hence, any existential can be equivalently replaced by a disjunction of corresponding ground literals. Therefore, without loss of generality, we will deal explicitly with clauses in the following formulation.

2.2 Constraint Satisfaction and Local Consistency

A *Constraint Satisfaction Problem* (CSP) refers to a set of variables $X = \{X_1, \ldots, X_n\}$, their domains $R = \{R_1, \ldots, R_n\}$ and a set of constraints $C = \{C_1, \ldots, C_k\}$ over the variables in X. Every constraint $C_i \in C$ is a relation over some non-empty set of variables in X, and specifies the set of values the variables appearing in it can take. A solution to a CSP is a set of assignments S to the variables in X, such that every member $s \in S$ satisfies all the constraints in C. In many instances, finding such a set is computationally challenging. However, for many problems, for each variable X_i, we can efficiently eliminate a subset of the values which are not part of any solution to the CSP. Let $V_i \subseteq R_i$ be the set of allowed values of X_i after eliminating such a subset of values. A variable is *generalized arc consistent* (or hyper-arc consistent) with a constraint, if for every value in its allowed set of values, there is an assignment to the remaining variables which satisfies the constraint. Consistency with respect to a set of constraints is defined in a similar manner. Generalized arc consistency only ensures *local consistency* i.e. it does not directly enforce any constraints among the variables which do not share a constraint. One way to ensure generalized arc consistency is to initialize the set of allowed values, V_i, to the respective domain R_i, and then iteratively eliminate those values which are not generalized arc consistent with the set of constraints. The algorithm continues until none of the V_i sets changes. This simple iterative procedure can lead to significant reduction in domain size for many problems. There are other forms of local consistency which could be enforced. A partial assignment to a set S of variables is said to be consistent if it does not violate any constraints which involve variables only from S. i-consistency requires that every consistent assignment of $i - 1$ variables can be extended by a value of any other variable not violating any of the constraints, and strong i-consistency ensures k-consistency for every $1 \leq k \leq i$. For a thorough introduction to CSPs and local consistency, see [1].

3 Constraint Propagation in Markov Logic

A set of first order clauses impose a set of constraints on the truth assignment to the ground atoms which participate in the respective ground clauses. Generalized arc consistency ensures that allowed truth assignments (true/false) to any ground atom have a corresponding assignment for all the other atoms in the clause such that the clause is satisfied. An evidence database fixes the assignment of the ground atoms in the database to true or false. Given a set of first order clauses, and an evidence database, our goal then, is to devise an algorithm so that the ground atoms in the domain are generalized arc consistent with the constraints imposed by the set of ground clauses. Because each atom can take only two possible assignments, any pruning on the domain of an atom essentially means that we can fix its assignment (if the whole domain is pruned then the constraints are inconsistent). Hence, ensuring generalized arc consistency on a set of hard clauses in a theory is a way to infer the additional truth assignments for some of the originally unknown ground atoms. These can then be set to evidence with the inferred truth value for any following probabilistic inference procedure. This leads to huge simplification in the network over which probabilistic inference needs to be performed.

The key idea for enforcing generalized arc consistency is to look at each ground clause in turn, and identify a ground atom whose assignment needs to be fixed in order to satisfy the clause, given current assignment to other atoms in the clause. This can be done iteratively, until no more truth assignments can be fixed. The main caveat with this approach is that it explicitly involves grounding out the whole theory, which is often prohibitively expensive. Next, we describe an algorithm which alleviates this problem.

3.1 Generalized Arc Consistency Algorithm

For the notational convenience, we will explain our algorithm for the case of untyped predicates; extending it to the more general case is straightforward. Let KB be a knowledge base with a set of hard constraints. Let L denote a predicate (or its negation). Let each argument take the values from the set of constants T. Therefore, the domain of L, denoted as $R(L)$, is T^k. Further, let $D(L)$ be the subset of tuples $\mathbf{t} \in R(L)$, for which $L(t)$ can be true in some model, i.e. $\mathbf{t} \in D(L)$ if $L(\mathbf{t}) = true$ is possibly a part of some assignment satisfying the constraints in KB. Let $N(L)$ be the subset of tuples for which $L(t)$ is necessarily true in any given model, i.e. $\mathbf{t} \in N(L)$ if $L(\mathbf{t}) = true$ in every assignment satisfying the constraints in KB. Note that $N(L) = R(L) \setminus D(\neg L)$.

The goal of the generalized arc consistency algorithm is to find the maximal $N(L)$ for every predicate L, while propagating constraints through the hard clauses. The algorithm starts with an initial set $N(L)$ for every L, and iteratively increases the size of $N(L)$, using the hard constraints given in the knowledge base, until none of the $N(L)$ sets can be further extended. The starting points of the algorithm are the ground atoms supplied in the evidence database. The algorithm is most easily described in the case where each predicate in a clause contains the same set of variables. Consider, for example:

$$C = L_1(x) \vee \ldots \vee L_k(x) \tag{2}$$

where x is a vector of variables. For every $1 \leq i \leq k$: $N(L_i)$ can be updated as follows:

$$N(L_i) \leftarrow N(L_i) \bigcup \left[\bigcap_{i \neq j, 1 \leq j \leq k} N(\neg L_j) \right] \tag{3}$$

In words, for every tuple c in the domain of x, we can conclude that $L_i(c)$ is true in every possible world if every other $L_j(c)$ appearing in the clause is false in every possible world. To generalize the update rule for predicates with different sets of variables we employ the (database) *Join* and *Project* operations. We define *Join* for two sets of tuples each of which has a corresponding vector of variables associated with it. Let S_i be a set of tuples and X_i be the corresponding vector of variables ($i \in \{1, 2\}$). We overload the notation such that X_i also refers to the set of variables in the corresponding vector. For now, we assume that a variable cannot appear more than once in a vector of variables (we will relax this assumption later in the text). For a tuple $s \in S_i$ and a variable $x \in X_i$ let $s[x]$ denote the value of the variable x in the tuple s. Let $X = X_1 \bigcup X_2$ and $R(X)$ be the full domain formed by the Cartesian product of the individual domains of the variables in X in some ordering of the variables. The join of the sets of tuples S_i, given corresponding vector of variables X_i, is defined as follows:

$$Join\{\langle X_i, S_i\rangle\} = \langle X, \{c | c \in R(X) \wedge \forall i, \exists s \in S_i \; \forall x \in X_i : s[x] = c[x]\}\rangle \tag{4}$$

Join is commutative and associative. The projection of a set S of tuples associated with a variable vector X to the variable vector Y is defined as follows:

$$Project(Y, \langle S, X\rangle) = \{c | c \in R(Y) \wedge \exists s \in S \; \forall y \in (Y \cap X) : s[y] = c[y]\} \tag{5}$$

For more details on natural join and project operations, see [3]. Using the above definitions we can extend the update rule to the general case (where each predicate in a clause can contain an arbitrary subset of variables):

$$N(L_i) \leftarrow N(L_i) \bigcup [Project(X_i, Join_{j \neq i}\{\langle X_j, N(\neg L_j)\rangle\}] \tag{6}$$

The space and time complexity of Equation (6) is sensitive to the order in which we perform the *Join*s (they can be performed in any order since *Join* is both commutative and associative). The worst case complexity (both space and time) is exponential in the number of variables involved in the operation. A number of different heuristic criteria could be used to decide the join order; we selected the literal L first with the smallest $N(L)$ set. Additionally, while performing a series of *Join*s, if the intermediate result contains a set of variables X' such that an $x \in X'$ variable does not occur in the remaining X_j sets, i.e., x is guaranteed not to appear on any other side of a *Join* and x is also not a member of X_i, then, we can project this partial result to $X' \setminus \{x\}$. This re-ordering of join and project operations can substantially reduce the space and time complexity of the algorithm. Consider, e.g.,

$$H(x) \vee O_1(x, y_1) \vee O_2(x, y_2) \vee \ldots \vee O_k(x, y_k) \tag{7}$$

where H is a hidden predicate while $O_1, O_2, \ldots, O_k$ are all observed. Also, let $|R(H)| = N$ and for every $1 \leq i \leq n : |R(O_i)| = N^2$. For every $1 \leq i \leq k$ we can perform

$$Project(x, \langle (x, y_i), N(\neg O_i)\rangle) \tag{8}$$

and feed the results to the *Join*s instead of using $N(\neg O_i)$ in the *Join*s, because every y_i occurs exactly in one predicate. This way, the space and time complexity of the algorithm reduces to $O(kN^2)$ from $O(N^{k+1})$.

Algorithm 1 shows the pseudo-code for our generalized arc consistency algorithm. Line 3 initializes the $N(L_i)$ sets based on the evidence database. In line 8 of the algorithm we start iterating through all the hard constraints. In line 10 we update the $N(L_i)$ sets for every positive or negative literal using Equation (6).

Algorithm 1 Update Algorithm for Generalized Arc Consistency on Clauses

```
1: for all $C \in KB$ do
2:   for all $L_i$ literal $\in C$ do
3:     $N(L_i) = \{t | L_i(t) = true$; given the evidence database$\}$
4:   end for
5: end for
6: repeat
7:   $changed \leftarrow false$
8:   for all $C \in KB$ do
9:     for all $L_i$ literal $\in C$ do
10:      $\Delta \leftarrow [Project(X_i, Join_{j \neq i}\{\langle X_j, N(\neg L_j)\rangle\}]$
11:      if $\Delta \neq \emptyset$ then
12:        $changed \leftarrow changed \vee N(L_i) \neq N(L_i) \bigcup \Delta$
13:        $N(L_i) \leftarrow N(L_i) \bigcup \Delta$
14:      end if
15:    end for
16:  end for
17: until $\neg changed$
```

The algorithm keeps iterating over all the hard constraints until none of the $N(L_i)$ sets change. It is easy to prove the convergence as well as the correctness of our generalized arc consistency algorithm. First, for convergence, clearly, the algorithm stops if in any iteration, none of the clauses results in a change in the $N(L_i)$ sets. Alternatively stated, each iteration results in at least one of the $N(L_i)$ sets increasing in size. Further, size of each $N(L_i)$ is upper bounded by the size of the corresponding domain $R(L_i)$. Therefore, the algorithm terminates in finite steps. By correctness we mean that, if $N(L_i)$ is the set obtained for predicate L_i at the end of the algorithm, then, for each tuple $t_i \in N(L_i)$, every model contains $L(t_i)$ in it, i.e. in any satisfying solution to the hard constraints $L_i(t_i) = true$. Let us prove it by induction. Initially, each $N(L_i)$ is set using the evidence database. Hence, the claim is true in the beginning. Next, let the claim holds at the k^{th} update step (to any of the $N(L_i)$'s) during the execution of the algorithm. Considering $k+1^{th}$ update, if an atom t_i is added to the set $N(L_i)$, then, there must have been a ground clause, $L_1(t_1) \vee L_2(t_2) \cdots \vee L_i(t_k) \cdots \vee L_k(t_k)$, such that each of $L_j(t_j) = false, \forall j \neq i$. This follows from the generalized arc consistency update rule (Equation (6)) and the fact that the claim holds true at the k^{th} update step. Hence, $L(t_i)$ must be *true* as setting it otherwise would lead to violation of this clause. Further, since we assumed the claim to be true at step k, and any new additions to the set $N(L_i)$ satisfy the claim by above argument, the claim is true at step $k+1$. Hence, proved.

3.2 Extension to Other Cases

Existentials We extend the update rule for clauses to allow existentially quantified conjunctions besides regular literals. E.g., consider the formula:

$$P(x) \vee \exists y\, [Q(x,y) \wedge R(z,y)] \tag{9}$$

For all the instantiations of x when $\exists y\,[Q(x,y) \wedge R(z,y)]$ is necessarily false $P(x)$ must be true. Thus, all we need to do is to extend the definition of $N(L_i)$ to allow L_i to be the negation of an existentially quantified conjunction.

Let $F = \neg\exists Y\,[L_1(X_1) \wedge \ldots \wedge L_k(X_k)]$ where $Y \subseteq \bigcup_i X_i$. Let $X = \bigcup_i X_i \setminus Y$ and $R(X)$ be the full domain formed by the Cartesian product of the individual domains of the non-quantified variables in X in some ordering of the variables. Then:

$$N(F) \leftarrow R(X) \setminus \mathit{Project}(X, \mathit{Join}_{1\leq i\leq k}\{\langle X_i, R(L_i) \setminus N(\neg L_i)\rangle\}) \tag{10}$$

$N(F)$ has to be updated if $N(\neg L_i)$ changes for any of the L_i's appearing in F.

Constant Arguments If a predicate P in a clause has a constant argument c, we can do the following transformation of the clause to a new clause which provides an equivalent hard constraint without having constant arguments in the predicates:

$$P(x,c) \vee \ldots \mathit{is\ replaced\ by} P(x,y) \vee \neg E_c(y) \vee \ldots \tag{11}$$

Where $E_c(y)$ is a fully observed predicate and is true if and only if $y = c$.

Repeated Arguments If a predicate P in a clause has a variable argument which appears more than once, the following transformation could handle this case:

$$P(x,x) \vee \ldots \mathit{is\ replaced\ by} P(x,x') \vee \neg E(x,x') \vee \ldots \tag{12}$$

Where x' is a variable not appearing in the original clause, and $E(x,x')$ is a fully observed predicate being true if and only if $x = x'$.

3.3 Relation to Unit Propagation

Running unit propagation on the ground hard clauses using the evidence would produce exactly the ground unit clauses which correspond to the N sets created by running the proposed generalized arc consistency algorithm. [3] Initially, the N sets are set according to the evidence, and to the unit clause hard constraints. [4] At this point the ground unit clauses available for unit propagation are exactly the ground unit clauses corresponding to the members of the N sets. Let the claim holds true after k updates to the N sets. Then, if unit propagation can derive a new ground unit clause so can the generalized arc consistency algorithm, because the new unit clause is the result of resolving a ground clause with ground unit clauses to each of which there is a corresponding member of N. This makes sure that the *Join*s and *Project*s in Equation (6) result in a

[3] This result holds in general when we do not perform any special pruning for existential quantifiers (Section 3.2). They are simply treated as disjunction of literals.

[4] Algorithm 1 initializes the N sets based only on evidence, but it is easy to see that both forms of initializations become equivalent after one step of running the original algorithm on unit clause hard constraints.

non-empty set containing a member corresponding to the newly derived ground unit clause. Also, when Equation (6) updates an $N(L_i)$ set based on the clause $C = L_1(X_1) \vee \ldots \vee L_n(X_n)$, it uses the values in $N(\neg L_1), \ldots, N(\neg L_j), \ldots N(\neg L_n)$ $(i \neq j)$. The ground unit clauses corresponding to these values are available to unit propagation, hence unit propagation can derive the ground unit clauses corresponding to the update of $N(L_i)$. Therefore, the claim holds true after $k+1$ updates to the N sets. Using the induction argument, the claim holds true for all values of k, and in particular, at the termination of the generalized arc consistency algorithm.

Although, the end result is the same, the generalized arc consistency algorithm can use significantly less space and time. Revisiting the example in Equation (7), there are $O(N^{k+1})$ ground clauses created, and hence, unit propagation would need $O(N^{k+1})$ space and time. However, as we pointed out earlier, generalized arc consistency algorithm requires only $O(kN^2)$ space and time.

3.4 Moving Beyond Generalized Arc Consistency

A natural question that may arise is why not use other forms of local consistency instead of generalized arc consistency (e.g. strong i-consistency). There is a trade-off between the strength of the consistency requirement and the time spent in processing the hard constraints. Stronger consistency requirements will typically result in better pruning but it comes at the cost of increased processing time. It is easy to see that if l is the maximum length of a clause in the Markov logic theory, then, strong i-consistency $(i \geq l)$ subsumes generalized arc consistency. Following example is illustrative in this regard. Consider the axioms:

1. $P(x) \wedge Q(x) \Rightarrow O(x)$
2. $S(x) \Rightarrow P(x)$
3. $S(x) \Rightarrow Q(x)$

where O is an observed predicate such that $R(O) = \{a, b, c, d\}$ and $D(O) = \{a, b, c\}$. Let $R = R(S) = R(P) = R(Q) = R(O)$. Together these imply that the domain of S is limited to $\{a, b, c\}$. But this cannot be inferred by generalized arc consistency on the CSP created from these axioms. Enforcing 3-consistency on the groundings of P, Q and O will ensure that both $P(d) = \mathit{true}$ and $Q(d) = \mathit{true}$ cannot hold at the same time. Moreover, enforcing 3-consistency on the groundings of P, Q and S ensures that for every $m \in R$ if at least one of $P(m)$ and $Q(m)$ is false then $S(m)$ must be false as well. Hence, we could try to enforce strong i-consistency on the CSP for some value of $i \geq 3$. But strong i-consistency requirements do not fall out naturally from the clausal structure imposed by the Markov logic theory. However, the same effect can be achieved by applying FOL resolution [12] to the axioms before creating the CSP. For instance, resolving 1 and 2 yields $\neg Q(x) \vee \neg S(x) \vee O(x)$. Resolving this with 3 yields $\neg S(x) \vee O(x)$. This new clause then does allow $D(S) = \{a, b, c\}$ to be inferred by generalized arc consistency.

Pre-processing a theory by resolving (hard) constraints can be done exhaustively or in a limited manner; for example, resolution could be performed in a breadth-first manner up to a fixed depth. Because a Markov logic theory contains no uninterpreted function symbols, even exhaustive resolution is guaranteed to terminate, although in the worst case an exponential number of resolvants would be created. We did some preliminary experiments with performing resolution to varying depths before applying generalized arc consistency, but little additional benefit was obtained on our test domains. Exploring this further is a direction for future work.

4 Experiments

We experimented on two real and one artificial datasets to compare the time and memory performances of CPI (Constraint Propagation based Inference) and the standard approach to inference (i.e. no prior pruning of the predicate domains is done). We used the freely available Alchemy [9] system for all our experiments. For the standard approach to inference, we used the Alchemy implementation as is. For the constraint propagation, we implemented a separate program to prune the domains by propagating the constraints amongst hard clauses. The output of this program was passed as additional evidence to Alchemy for the CPI. For the probabilistic inference in both the approaches, exactly the same knowledge base was used (including all the soft and hard rules). Since exact marginal inference was not tractable, we used the MCMC based MC-SAT [11] algorithm implemented in Alchemy. It was run to collect 1000 samples (default in Alchemy) for both the approaches. All the experiments were run on a cluster of nodes with processor speed of 2.4 GHz. We do not report accuracy since both the approaches are guaranteed to give the same results at the point of convergence of MC-SAT (Section 4.3 discusses some of the issues relating to the convergence of MC-SAT). We first describe the datasets in detail followed by our results.

4.1 Datasets

Cora Entity resolution is the problem of determining which observations (e.g., records in a database) correspond to the same objects. We used the version of McCallum's Cora database available on the Alchemy website (Kok et al. 2007). The inference task was to de-duplicate citations, authors and venues (i.e., to determine which pairs of citations refer to the same underlying paper, and similarly for author fields and venue fields). We used the MLN (formulas and weights) used by Singla and Domingos [15] in their experiments. This contains first-order clauses stating regularities such as: if two fields have high TF-IDF similarity, they are (probably) the same; if two records are the same, their fields are the same, and vice-versa; etc. For each field, we added the hard rules for deciding that two fields are a non-match if their TF-IDF similarity was below a threshold. This effectively implements the canopies as described by McCallum [10], to eliminate obvious non-matches. We also added another set of rules deciding a pair of citations as non-match if any of the fields did not match. The final knowledge base contained 25 predicates and 52 formulas (6 hard and 46 soft). Maximum formula-arity was 4 and maximum predicate domain size was 71,000.

Capture the Flag (CTF) Our second dataset deals with the task of activity recognition. Sadilek and Kautz [13] collected this dataset by having subjects play the game of capture the flag on a University campus. The dataset contains the details of the GPS location of each player at each time step. The task is to determine all the captured events during the course of the game. The dataset contains information about 3 different games with 14 players (divided onto two teams), running for an average of 625 time steps. Each GPS location was uniquely snapped (model as hidden predicate) to one of the 6499 cells. We used the knowledge base hand-coded by Sadilek & Kautz (2010) stating hard facts such as "captured players stay at the same location" and soft rules such as "if two players from different teams are snapped to the same cell at a time step, then it is likely to result into a capture event". We added another hard rule stating if two agents are at same place, then they must be snapped to nearby cells. The original knowledge base involves some soft rules with real-valued features. Since current Alchemy implementation does not support them, we ignored these rules for our experiments. The final knowledge base contained 9 predicates and 17 formulas (15 hard and 2 soft). Maximum formula-arity was 4 and maximum predicate domain size was 29 million.

Library We also experimented with an artificially generated online library dataset. The goal of the system is to recommend books to each user that they might like to read. Each user can read books in one or more of the four languages that they can speak. A user needs to read a book in order to like it. The system can recommend a book to a user if they have not already read it. These are modeled as hard constraints. The system recommends a book to a user if the user shares the liking of another book with a user who likes this book as well. This is modeled as a soft constraint. *Read*, *available* and *speaks* are modeled as fully observed. *Likes* is partially observed. The task is to predict *recommends*. The final knowledge base contained 5 predicates and 4 formulas (3 hard and 1 soft). Maximum formula-arity was 4 and maximum predicate domain size was 0.5 million.

We generated a dataset containing 100 users. The number of books was varied from 500 to 5000, at intervals of 500. For each user (book), the set of languages spoken (available) was chosen using a Bernoulli trial for each of the 4 languages. The parameters of the Bernoulli trials were set to model that certain languages are more popular than others. The number of books read by each user followed a Gaussian distribution with $\mu = 30$ and $\sigma = 5$. The subset of books read by a user was assigned uniformly at random from the set of books available in the languages that user could speak. A user left feedback for a book he read with 0.3 probability and the feedback was *likes* with 0.7 and *not likes* with 0.3 probability.

4.2 Results

Tables 1 presents the results on the three datasets. For Library, the reported results are for 2500 books. Standard (Stand.) and CPI refer to the standard approach to inference, and the constraint propagation based inference, respectively.

We report the running time of the two algorithms as well as the memory requirement, measured in terms of number of ground clauses created. For both time and memory, results are split into two parts a) cost of constraint propagation b) cost of probabilistic inference. First cost is zero for the standard inference. For CPI, total time cost is the sum of two costs. As evident from the table, time cost of constraint propagation is negligible compared to the cost for probabilistic inference. On CTF, CPI is faster than standard inference by a factor of 3; on Cora, by a factor of 7. On library, the gain is an order of magnitude.

Domain	**Time (in mins)**				**Ground Tuples (in 1000's)**			
	Const. Propagation		Prob. Inference		Const. Propagation		Prob. Inference	
	Stand.	CPI	Stand.	CPI	Stand.	CPI	Stand.	CPI
CTF	0	0.37	1536.6	528.0	0	585.5	2107.8	1308.7
Cora	0	0.07	181.1	26.2	0	153.6	488.2	81.4
Library	0	0.20	286.4	23.0	0	462.7	366.2	45.9

Table 1: Time and memory costs comparing the two inference approaches

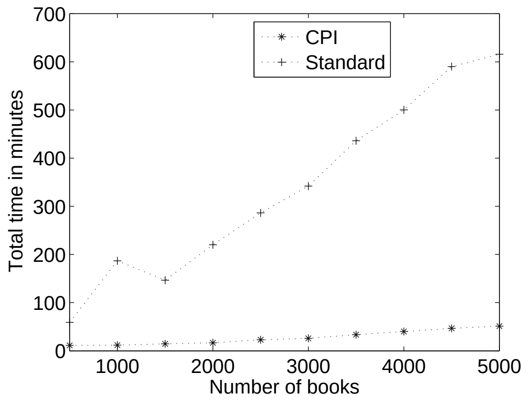

Fig. 1: Inference time for varying number of books

Since we run the two inference pieces sequentially (constraint propagation followed by probabilistic inference), memory cost for CPI is the maximum of the cost for the two parts. For CTF, the cost of probabilistic inference dominates. Memory cost for CPI for this dataset is about 60% of the standard inference. For Cora and Library, constraint propagation dominates the memory cost. This is due to the *join* operation in the generalized arc consistency algorithm, which can turn out to be quite expensive. On Cora, CPI saves memory by more than a factor of 4. On Library the gain is relatively less. Figure 1 shows the inference time results for the Library dataset as the number of books is varied from 500 to 5000. The time cost for CPI stays almost constant, whereas, it goes linearly up for standard inference. The number of ground clauses constructed (during actual probabilistic inference) follows a similar trend. This is attributed to the fact that as the number of books increases, the problem becomes sparser i.e. chances of two people having liked the same book and hence, one causing the recommendation to the other decreases with increasing number of books. Most recommended groundings need not be considered for inference and CPI can take advantage of this. Standard inference, not being able to prune the domains, scales linearly with increasing number of books. It should be noted that the Library dataset had to be carefully hand-engineered for the standard inference approach to run on it, whereas CPI did not have any problems with the intuitive formulation of the knowledge base. [5] Exploring this further is a direction for future work.

Results above demonstrate that hard constraints in Markov logic can be used to significantly reduce both the time and memory cost of inference. The advantage can be huge for the problems where domains are already very sparse. Generalized arc consistency is extremely fast relative to the probabilistic inference. Its memory requirements can be relatively high sometimes, but still it saves significant memory in many cases, in comparison to the standard approach.

4.3 Note About MC-SAT Convergence

Alchemy does not give a way to detect if the MC-SAT algorithm has converged. But we compared the differences in the marginals obtained by two approaches at the end of 1000 steps of MC-SAT (all our results are obtained by running MC-SAT for 1000 steps.). On the Cora dataset, 99% of the differences were within 0.01 threshold. For Library, this number was 0.05. For Capture the Flag, we noticed a much larger variation. This is due the fact that it is a much bigger domain, and many more samples are needed to converge to the right marginals. Nevertheless, it should be noted that any increase in the number of samples during probabilistic inference would lead to even larger gain for our approach. This is because we have a much simpler network, and collecting each sample takes lesser time compared to the standard approach. For the same reason, we also expect a faster convergence (in terms of the number of samples needed) for our approach. Exploring these convergence issues in detail is a direction for future work.

5 Related Work

There has been some related work which exploits the structure of the network to make inference in Markov logic more efficient, but none has separately analyzed the hard constraints to reduce the size of the predicate domains over which network is constructed. LazySAT [15] exploits the fact that for many problems, most ground atoms are false and most ground clauses are satisfied, hence, a local solver (such as MaxWalkSAT [7]), does not need to explicitly instantiate them. Lifted Belief Propagation (LBP) [16] performs inference over a lifted network by clustering the nodes that would pass the same BP message in the ground network. None of these approaches are able to explicitly eliminate the nodes which are categorically false (or true) by virtue of the hard constraints. This

[5] Results reported above for the Library dataset are for the hand-engineered case.

may lead to sub-optimal inference, for instance, flipping a false (inferred) node in LazySAT, or, putting two false nodes in the different clusters for the case of LBP. Our approach is orthogonal to the benefits obtained by above algorithms, and thus can be used in conjunction with them. Jha et al. [6] recently proposed a lifted inference algorithm which uses techniques from logic and database literature. Their algorithm handles only the case for exact inference and that, too, for a small class of very simple MLNs.
Shavlik and Natarajan [14] present an approach to pre-process MLN theory to reduce the size of the ground Markov network. Their pre-processing effectively implements a fast index based algorithm to eliminate trivially satisfied (or unsatisfied) clauses. Each clause is processed independently. They do not allow information to be transferred from one clause to another, which is a key aspect of our approach. Alchemy already implements their pre-processing step, and hence, any benefits obtained by our approach are in addition to theirs.
Kisyński and Poole [8] analyze the use of different algorithms for constraint satisfaction in lifted inference. Their analysis is in the context of FOVE (first-order variable elimination) where factors are eliminated in some order. It's not directly applicable to approximate inference setting. Whether their lifted solver can be used in place of generalized arc consistency in our framework is a direction for future work.
Our work can be seen in the light of constraints specified using SQL queries in Relational Markov Networks (RMNs) [17]. Our approach is more general than theirs because constraints do not have to be repeated for each clause. Further, unlike their approach, we propagate information from one constraint to another, thereby potentially leading to even smaller predicate domains, over which to construct the network.

6 Conclusion and Future Work

We proposed a generalized arc consistency algorithm to effectively propagate the hard constraints in a Markov logic theory. We are able to do this at a lifted level, without ever explicitly grounding out the whole theory. Our algorithm significantly prunes the predicate domains, thereby, resulting in much simpler networks and allowing for significant efficiency gains during probabilistic inference. Directions for future work include experimenting with a wider variety of domains, trying out other forms of consistency requirements, symbolic manipulation of the theory to propagate the constraints more effectively, and combining our approach with lifted and lazy inference.

7 Acknowledgements

We are thankful to Ray Mooney for helpful discussions, and to Adam Sadilek for sharing the CTF dataset. This research was partly funded by ARO grant W911NF-08-1-0242 and NSF award 1012017. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO, NSF or the United States Government.

References

1. Dechter, R.: Constraint Processing. Morgan Kaufmann (2003)
2. Domingos, P., Lowd, D.: Markov Logic: An Interface Layer for Artificial Intelligence. Morgan & Claypool, San Rafael, CA (2009)
3. Garcia-Molina, H., Ullman, J.D., Widom, J.D.: Database Systems: The Complete Book (2nd Ed.). Prentice Hall (2008)
4. Getoor, L., Taskar, B. (eds.): Introduction to Statistical Relational Learning. MIT Press, Cambridge, MA (2007)
5. Gogate, V., Dechter, R.: SampleSearch: Importance sampling in presence of determinism. Artificial Intelligence 175, 694–729 (2011)
6. Jha, A., Gogate, V., Meliou, A., Suciu, D.: Lifted inference seen from the other side: The tractable features. In: Advances in Neural Information Processing Systems 23 (NIPS 2010). pp. 973–981 (2010)
7. Kautz, H., Selman, B., Jiang, Y.: A general stochastic approach to solving problems with hard and soft constraints. In: Gu, D., Du, J., Pardalos, P. (eds.) The Satisfiability Problem: Theory and Applications, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 35, pp. 573–586. American Mathematical Society, New York, NY (1997)
8. Kisyński, J., Poole, D.: Constraint processing in lifted probabilistic inference. In: Proceedings of 25th Conference on Uncertainty in Artificial Intelligence (UAI-2009). pp. 292–302 (2009)
9. Kok, S., Sumner, M., Richardson, M., Singla, P., Poon, H., Lowd, D., Wang, J., Nath, A., Domingos, P.: The Alchemy system for statistical relational AI. Tech. rep., Department of Computer Science and Engineering, University of Washington (2010), alchemy.cs.washington.edu
10. McCallum, A.: Efficiently inducing features of conditional random fields. In: Proceedings of 19th Conference on Uncertainty in Artificial Intelligence (UAI-2003). pp. 403–410. Acapulco, Mexico (August 2003)
11. Poon, H., Domingos, P.: Sound and efficient inference with probabilistic and deterministic dependencies. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06). Boston, MA (2006)
12. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Upper Saddle River, NJ, 2 edn. (2003)
13. Sadilek, A., Kautz, H.: Recognizing mutli-agent activities from GPS data. In: Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-10) (2010)
14. Shavlik, J., Natarajan, S.: Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network. In: Proceedings of the Twenty First International Joint Conference on Artificial Intelligence (IJCAI-2009). pp. 1951–1956. Hyederabad, India (2009)
15. Singla, P., Domingos, P.: Discriminative training of Markov logic networks. In: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05). pp. 868–873. Pittsburgh, PA (2005)
16. Singla, P., Domingos, P.: Lifted first-order belief propagation. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08). pp. 1094–1099. Chicago, IL (2008)
17. Taskar, B., Abbeel, P., Koller, D.: Discriminative probabilistic models for relational data. In: Proceedings of 18th Conference on Uncertainty in Artificial Intelligence (UAI-2002). pp. 485–492. Edmonton, Canada (2002)