Finding Most Probable Worlds of Probabilistic Logic Programs

Samir Khuller, Vanina Martinez, Dana Nau,
Gerardo Simari, Amy Sliva, V.S. Subrahmanian**

University of Maryland College Park, College Park, MD 20742, USA
{samir,mvm,nau,gisimari,asliva,vs}@cs.umd.edu

Abstract. Probabilistic logic programs have primarily studied the problem of entailment of probabilistic atoms. However, there are some interesting applications where we are interested in finding a possible world that is most probable. Our first result shows that the problem of computing such "maximally probable worlds" (MPW) is intractable. We subsequently show that we can often greatly reduce the size of the linear program used in past work (by Ng and Subrahmanian) and yet solve the problem exactly. However, the intractability results still make computational efficiency quite impossible. We therefore also develop several heuristics to solve the MPW problem and report extensive experimental results on the accuracy and efficiency of such heuristics.

1 Introduction

Probabilistic logic programs (PLPs) [1] have been proposed as a paradigm for probabilistic logical reasoning with no independence assumptions. PLPs used a possible worlds model based on prior work by [2], [3], and [4] to induce a set of probability distributions on a space of possible worlds. Past work on PLPs [5, 1] focuses on the entailment problem of checking if a PLP entails that the probability of a given formula lies in a given probability interval.

However, we have recently been developing several applications for cultural adversarial reasoning [6] where PLPs and their variants are used to build a model of the behavior of certain socio-cultural-economic groups in different parts of the world.[1] Such PLPs contain rules that state things like "There is a 50 to 70% probability that group g will take action(s) a when condition C holds." In such applications, the problem of interest is that of finding the most probable action (or sets of actions) that the group being modeled might do. This corresponds precisely to the problem of finding a "most probable world" that is the focus of this paper.

In Section 2 of this paper, we recall the syntax and semantics of such programs [5, 1]. We state the *most probable world* (MPW) problem by immediately using the linear programming methods of [5, 1] - these methods are exponential because the linear

** Authors listed in alphabetical order; authors 2, 4, 5, and 6 were funded in part by grant N6133906C0149, ARO grant DAAD190310202, AFOSR grants FA95500610405 and FA95500510298, and NSF grant 0540216.

[1] Our group has thus far built models of the Afridi tribe in Pakistan, Hezbollah in the Middle East, and the various stakeholders in the Afghan drug economy.

programs are exponential in the number of ground atoms in the language. Then, in Section 4, we present the *Head Oriented Processing (HOP)* approach where a (usually) smaller linear program is introduced. We show that using HOP, we can often find a much faster solution to the MPW problem. We define a variant of HOP called SemiHOP that has slightly different computational properties, but is still guaranteed to find the most probable world. Thus, we have three exact algorithms to find the most probable world.

Subsequently, in Section 5, we develop a heuristic that can be applied in conjunction with the *Naive*, HOP, and SemiHOP algorithms. The basic idea is that rather than examining all worlds, only some fixed number of worlds is explored using a linear program that is reduced in size. Section 6 describes a prototype implementation of our APLP framework and includes a set of experiments to assess combinations of exact algorithm and the heuristic. We assess both the efficiency of our algorithms, as well as the accuracy of the solutions they produce.

2 Overview of Action Probabilistic Logic Programs

Action probabilistic logic programs (APLPs) are an immediate and obvious variant of the probabilistic logic programs introduced in [5, 1]. We assume the existence of a logical alphabet that consists of a finite set $\mathcal{L}_{cons}$ of constant symbols, a finite set $\mathcal{L}_{pred}$ of predicate symbols (each with an associated arity) and an infinite set $\mathcal{V}$ of variable symbols. Function symbols are not allowed in our language. Terms and atoms are defined in the usual way [7]. We assume that a subset $\mathcal{L}_{act}$ of $\mathcal{L}_{pred}$ are designated as *action symbols* - these are symbols that denote some action. Thus, an atom $p(t_1, \ldots, t_n)$, where $p \in \mathcal{L}_{act}$, is an *action atom*. Every (resp. action) atom is a (resp. action) wff. If F, G are (resp. action) wffs, then $(F \wedge G), (F \vee G)$ and $\neg G$ are all wffs (resp. action wffs).

Definition 1. *If F is a wff (resp. action wff) and $\mu = [\alpha, \beta] \subseteq [0, 1]$, then $F : \mu$ is called a p-annotated (resp. ap-annotated—short for "action probabilistic" annotated) wff. μ is called the p-annotation (resp. ap-annotation) of F.*

Without loss of generality, throughout this paper we will assume that F is in conjunctive normal form (i.e. it is written as a conjunction of disjunctions).

Definition 2 (ap-rules). *If F is an action formula, $A_1, A_2, ..., A_m$ are action atoms, $B_1, \ldots, B_n$ are non-action atoms, and $\mu, \mu_1, ..., \mu_m$ are ap-annotations, then $F : \mu \leftarrow A_1 : \mu_1 \wedge A_2 : \mu_2 \wedge ... \wedge A_m : \mu_m \wedge B_1 \wedge \ldots B_m$ is called an ap-*rule*. If this rule is named c, then $Head(c)$ denotes $F : \mu$; $Body^{act}(c)$ denotes $A_1 : \mu_1 \wedge A_2 : \mu_2 \wedge ... \wedge A_m : \mu_m$ and $Body^{state}(c)$ denotes $B_1 \wedge \ldots B_n$.*

Intuitively, the above ap-rule says that an unnamed entity (e.g. a group g, a person p etc.) *will take action F with probability in the range μ if $B_1, \ldots, B_n$ are true in the current state (we will define this term shortly) and if the unnamed entity will take each action A_i with a probability in the interval μ_i for $1 \leq i \leq n$.*

Definition 3 (ap-program). *An* action probabilistic logic program *(ap-program for short) is a finite set of ap-rules.*

1. *kidnap:* $[0.35, 0.45]$ $\leftarrow$ *interOrganizationConflicts.*
2. *kidnap:* $[0.60, 0.68]$ $\leftarrow$ *notDemocratic* $\wedge$ *internalConflicts.*
3. *armed_attacks:* $[0.42, 0.53]$ $\leftarrow$ *typeLeadership(strongSingle)* $\wedge$ *orgPopularity(moderate).*
4. *armed_attacks:* $[0.93, 1.0]$ $\leftarrow$ *statusMilitaryWing(standing).*

Fig. 1. Four simple rules for modeling the behavior of a group in certain situations.

Figure 1 shows a small rule base consisting of some rules we have derived automatically about Hezbollah using behavioral data in [8]. The behavioral data in [8] has tracked over 200 terrorist groups for about 20 years from 1980 to 2004. For each year, values have been gathered for about 300 measurable variables for each group in the sample. These variables include tendency to commit assassinations and armed attacks, as well as background information about the type of leadership, whether the group is involved in cross border violence, etc. Our automatic derivation of these rules was based on a data mining algorithm we have separately developed [9]. We show 4 rules we have extracted for the group Hezbollah in Figure 1. For example, the third rule says that when Hezbollah has a strong, single leader and its popularity is moderate, its propensity to conduct armed attacks has been 42 to 53%. However, when it has had a standing military, its propensity to conduct armed attacks is 93 to 100%.

Definition 4 (world/state). *A* world *is any set of ground action atoms. A* state *is any finite set of ground non-action atoms.*

Note that both worlds and states are just ordinary Herbrand interpretations. As such, it is clear what it means for a state to satisfy $Body^{state}$.

Definition 5. *Let Π be an ap-program and s a state. The* reduction of Π w.r.t. s*, denoted by Π_s is $\{F : \mu \leftarrow Body^{act} \mid s$ satisfies $Body^{state}$ and $F : \mu \leftarrow Body^{act} \wedge Body^{state}$ is a ground instance of a rule in $\Pi\}$.*

Note that Π_s never has any non-action atoms in it.

Key differences. The key differences between action probabilistic LPs (APLPs) and the programs of [5, 1] are that APLPs have a bipartite structure (action atoms and state atoms) and they allow arbitrary formulas (including ones with negation) in rule heads ([5, 1] do not). They can easily be extended to include variable annotations and annotation terms as in [5]. Likewise, as in [5], they can be easily extended to allow complex formulas rather than just atoms in rule bodies. Due to space restrictions, we do not do either of these in the paper. *However, the most important difference between our paper and [5, 1] is that this paper focuses on finding most probable worlds, while those papers focus on entailment, which is a fundamentally different problem.*

Throughout this paper, we will assume that there is a fixed state s. Hence, once we are given Π and s, Π_s is fixed. We can associate a fixpoint operator T_{Π_s} with Π, s which maps sets of ground ap-annotated wffs to sets of ground ap-annotated wffs as follows.

Definition 6. *Suppose X is a set of ground annotated action atoms. We first define an intermediate operator $U_{\Pi_s}(X)$ as follows. $U_{\Pi_s}(X) = \{F : \mu \mid F : \mu \leftarrow A_1 :$*

$\mu_1 \wedge \cdots \wedge A_m : \mu_m$ *is a ground instance of a rule in* Π_s *and for all* $1 \leq j \leq m$*, there is an* $A_j : \eta_j \in X$ *such that* $\eta_j \subseteq \mu_j$*}*.

Intuitively, $U_{\Pi_s}(X)$ contains the heads of all rules in Π_s whose bodies are deemed to be "true" if the action atoms in X are true. However, $U_{\Pi_s}(X)$ may not contain all ground action atoms. This could be because such atoms don't occur in the head of a rule - $U_{\Pi_s}(X)$ never contains any action wff that is not in a rule head.

In order to assign a probability interval to each ground action atom, we use the same procedure followed in [5]. We use $U_{\Pi_s}(X)$ to set up a linear program $CONS_U(\Pi, s, X)$ as follows. For each world w_i, let p_i be a variable denoting the probability of w_i being the "real world". As each w_i is just a Herbrand interpretation, the notion of satisfaction of an action formula F by a world w, denoted by $w \mapsto F$, is defined in the usual way.

1. If $F : [\ell, u] \in U_{\Pi_s}(X)$, then $\ell \leq \Sigma_{w_i \mapsto F}\, p_i \leq u$ is in $CONS_U(\Pi, s, X)$.
2. $\Sigma_{w_i} p_i = 1$ is in $CONS_U(\Pi, s, X)$.

We refer to these as constraints of type (1) and (2), respectively. To find the lower (resp. upper) probability of a ground action atom A, we merely minimize (resp. maximize) $\Sigma_{w_i \mapsto A} p_i$ subject to the above constraints. We also do the same w.r.t. each formula F that occurs in $U_{\Pi_s}(X)$ — this is because this minimization and maximization may sharpen the bounds of F. Let $\ell(F)$ and $u(F)$ denote the results of these minimizations and maximizations, respectively. Our operator $T_{\Pi_s}(X)$ is then defined as follows.

Definition 7. *Suppose* Π *is an* APLP*,* s *is a state, and* X *is a set of ground ap-wffs. Our operator* $T_{\Pi_s}(X)$ *is then defined to be* $\{F : [\ell(F), u(F)] \mid (\exists \mu)\ F : \mu \in U_{\Pi_s}(X)\} \cup \{A : [\ell(A), u(A)] \mid A$ *is a ground action atom* $\}$.

Thus, $T_{\Pi_s}(X)$ works in two phases. It first takes each formula $F : \mu$ that occurs in $U_{\Pi_s}(X)$ and finds $F : [\ell(F), u(F)]$ and puts this in the result. Once all such $F : [\ell(F), u(F)]$'s have been put in the result, it tries to infer the probability bounds of all ground action atoms A from these $F : [\ell(F), u(F)]$'s.

Given two sets X_1, X_2 of ap-wffs, we say that $X_1 \leq X_2$ iff for each $F_1 : \mu_1 \in X_1$, there is an $F_1 : \mu_2 \in X_2$ such that $\mu_2 \subseteq \mu_1$. Intuitively, $X_1 \leq X_2$ may be read as "X_1 is less precise than X_2." The following straightforward variation of similar results in [5] shows that

Proposition 1. *1.* T_{Π_s} *is monotonic w.r.t. the* $\leq$ *ordering.*
2. T_{Π_s} *has a least fixpoint, denoted* $T^{\omega}_{\Pi_s}$.

3 Maximally Probable Worlds

We are now ready to introduce the problem of finding the most probable world. As explained through our Hezbollah example, we may be interested in knowing what actions Hezbollah might take in a given situation.

Definition 8 (lower/upper probability of a world). *Suppose* Π *is an ap-program and* s *is a state. The* lower probability*,* $\mathsf{low}(w_i)$ *of a world* w_i *is defined as:* $\mathsf{low}(w_i) =$ $\mathbf{minimize}\ p_i$ ***subject to*** $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$*. The* upper probability*,* $up(w_i)$ *of world* w_i *is defined as* $up(w_i) = \mathbf{maximize}\ p_i$ ***subject to*** $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$.

Thus, the low probability of a world w_i is the lowest probability that that world can have in any solution to the linear program. Similarly, the upper probability for the same world represents the highest probability that that world can have. It is important to note that for any world w, we cannot *exactly* determine a point probability for w. This observation is true even if all rules in Π have a point probability in the head because our framework does not make any simplifying assumptions (e.g. independence) about the probability that certain things will happen.

We now state two simple results that state that checking if the low (resp. up) probability of a world exceeds a given bound (called the BOUNDED-LOW and BOUNDED-UP problems respectively) is intractable. The hardness results, in both cases, are by reduction from the problem of checking consistency of a generalized probabilistic logic program. The problem is in the class EXPTIME.

Proposition 2 (BOUNDED LOW COMPLEXITY). *Given an ap-program Π, a state s, a world w, and a probability threshold p_{th}, deciding if $low(w) > p_{th}$ is NP-hard.*

Proposition 3 (BOUNDED UP COMPLEXITY). *Given an ap-program Π, a state s, a world w_i, and a probability threshold p_{th}, deciding if $up(w) < p_{th}$ is NP-hard.*

The MPW Problem. The *most probable world* problem (MPW for short) is the problem where, given an APLP Π and a state s as input, we are required to find a world w_i where $low(w_i)$ is maximal. [2]

A Naive Algorithm. A *naive* algorithm to find the most probable world would be:

1. Compute $T^{\omega}_{\Pi_s}$; $Best = NIL$; $Bestval = 0$;
2. For each world w_i,
   (a) Compute $low(w_i)$ by minimizing p_i subject to the set $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ of constraints.
   (b) If $low(w_i) > Bestval$ then set $Best = w_i$ and $Bestval = low(w_i)$;
3. If $Best = NIL$ then return any world whatsoever, else return $Best$.

The Naive algorithm does a brute force search after computing $T^{\omega}_{\Pi_s}$. It finds the low probability for each world and chooses the best one. Clearly, we can use it to solve the MPW-Up problem by replacing the minimization in Step 2(a) by a maximization.

There are two key problems with the naive algorithm. The first problem is that in Step (1), computing $T^{\omega}_{\Pi_s}$ is very difficult. When some syntactic restrictions are imposed, this problem can be solved without linear programming at all as in the case when Π is a probabilistic logic program (or p-program as defined in [1]) where all heads are atomic.

The second problem is that in Step 2(a), the number of (linear program) variables in $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ is exponential in the number of ground atoms. When this number is, say 20, this means that the linear program contains over a million variables. However, when the number is say 30 or 40 or more, this number is inordinately large. This paper focuses primarily on improving Step 2(a).

[2] A similar **MPW-Up Problem** can also be defined. The *most probable world-up* problem (MPW-Up) is given an APLP Π and a state s as input, and tries to find a world w_i where $up(w_i)$ is maximal. Due to space constraints, we only address the MPW problem.

4 HOP: Head-Oriented Processing

We can do better than the naive algorithm. Given a world w, state s, and an ap-program Π, let $Sat(w) = \{F \mid c$ is a ground instance of a rule in Π_s and $Head(c) = F : \mu$ and $w \mapsto F\}$. Intuitively, $Sat(w)$ is the set of heads of rules in Π_s (without probability annotations) whose bodies are satisfied by w.

Definition 9. *Suppose Π is an APLP, s is a state, and w_1, w_2 are two worlds. We say that w_1 and w_2 are equivalent, denoted $w_1 \sim w_2$, iff $Sat(w_1) = Sat(w_2)$.*

In other words, we say that two worlds are considered equivalent iff the two worlds satisfy the formulas in the heads of exactly the same rules in Π_s. It is easy to see that $\sim$ is an equivalence relation. We use $[w_i]$ to denote the $\sim$-equivalence class to which a world w_i belongs. The intuition for the HOP algorithm is given in Example 1.

Example 1. Consider the set $\mathsf{CONS}_U(\Pi, s, T^\omega_{\Pi_s})$ of constraints. For example, consider a situation where $\mathsf{CONS}_U(\Pi, s, T^\omega_{\Pi_s})$ contains just the three constraints below:

$$0.7 \leq p_2 + p_3 + p_5 + p_6 + p_7 + p_8 \leq 1 \tag{1}$$

$$0.2 \leq p_5 + p_7 + p_8 \leq 0.6 \tag{2}$$

$$p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 = 1 \tag{3}$$

In this case, each time one of the variables p_5, p_7, or p_8 occur in a constraint, the other two also occur. Thus, we can replace these by one variable (let's call it y for now). In other words, suppose $y = p_5 + p_7 + p_8$. Thus, the above constraints can be replaced by the simpler set

$$0.7 \leq p_2 + p_3 + p_6 + y \leq 1$$

$$0.2 \leq y \leq 0.6$$

$$p_1 + p_2 + p_3 + p_4 + p_6 + y = 1$$

The process in the above example leads to a reduction in the size of $\mathsf{CONS}_U(\Pi, s, T^\omega_{\Pi_s})$. Moreover, suppose we minimize y subject to the above constraints. In this case, the minimal value is 0.2. As $y = p_5 + p_7 + p_8$, it is immediately obvious that the low probability of any of the p_i's is 0. Note that we can also group p_2, p_3, and p_6 together in the same manner.

We build on top of this intuition. The key insight here is that for any $\sim$-equivalence class $[w_i]$, the entire summation $\Sigma_{w_j \in [w_i]} p_j$ either appears *in its entirety* in each constraint of type (1) in $\mathsf{CONS}_U(\Pi, s, T^\omega_{\Pi_s})$ or does not appear at all (i.e. none of the p_j variables associated with worlds w_j in $[w_i]$ appear in any constraint of type (1) in $\mathsf{CONS}_U(\Pi, s, T^\omega_{\Pi_s})$). This is what the next result states.

Proposition 4. *Suppose Π is an ap-program, s is a state, and $[w_i]$ is a $\sim$-equivalence class. Then for each constraint of the form*

$$\ell \leq \Sigma_{w_r \mapsto F}\, p_r \leq u \tag{4}$$

in $\mathsf{CONS}_U(\Pi, s, T^\omega_{\Pi_s})$, either every variable in the summation $\Sigma_{w_j \in [w_i]} p_j$ appears in the summation in (4) above or no variable in the summation $\Sigma_{w_j \in [w_i]} p_j$ appears in the summation in (4).

Example 2. Here is a toy example of this situation. Suppose Π_s consists of the two very simple rules:

$$(a \vee b \vee c \vee d) : [0.1, 0.5] \leftarrow .$$
$$(a \wedge e) : [0.2, 0.5] \leftarrow .$$

Assuming our language contains only the predicate symbols a, b, c, d, e, there are 32 possible worlds. However, what the preceding proposition tells us is that we can group the worlds into four categories. Those that satisfy both the above head formulas (ignoring the probabilities), those that satisfy the first but not the second head formula, those that satisfy the second but not the first head formula, and those that satisfy neither. This is shown graphically in Figure 2, in which p_i is the variable in the linear program corresponding to world w_i. For simplicity, we numbered the worlds according to the binary representation of the set of atoms. For instance, world $\{a, c, d, e\}$ is represented in binary as 10111, and is thus w_{23}. Note that only three variables appear in the new linear constraints; this is because it is not possible to satisfy $\neg(a \vee b \vee c \vee d \vee e)$ and $(a \wedge e)$ at once.

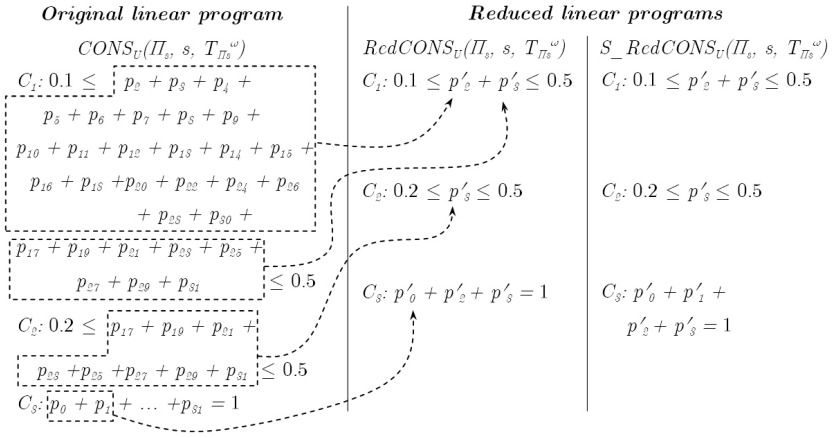

Fig. 2. Reducing $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ by grouping variables. The new LPs are called $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ and $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$, as presented in Definition 10 and 12.

Effectively, what we have done is to modify the number of variables in the linear program from $2^{card(\mathcal{L}_{act})}$ to $2^{card(\Pi_s)}$ - a saving that can be significant in some cases (though not always!). The number of constraints in the linear program stays the same. Formally speaking, we define a *reduced set of constraints* as follows.

Definition 10 ($\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$)**.** *For each equivalence class* $[w_i]$, $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ *uses a variable* p'_i *to denote the summation of the probability of each of the worlds in* $[w_i]$. *For each ap-wff* $F : [\ell, u]$ *in* $T^{\omega}_{\Pi_s}$, $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ *contains the constraint:*

$$\ell \leq \Sigma_{[w_i] \mapsto F} p'_i \leq u.$$

Here, $[w_i] \mapsto F$ *means that some world in* $[w_i]$ *satisfies* F*. In addition,* $\mathsf{RedCONS}_U(\Pi, s, T^\omega_{\Pi_s})$ *contains the constraint*

$$\Sigma_{[w_i]} p'_i = 1.$$

When reasoning about $\mathsf{RedCONS}_U(\Pi, s, T^\omega_{\Pi_s})$, we can do even better than mentioned above. The result below states that in order to find the most probable world, we only need to look at the equivalence classes that are of cardinality 1.

Theorem 1. *Suppose* Π *is an ap-program,* s *is a state, and* w_i *is a world. If* $card([w_i]) > 1$*, then* $low(w_i) = 0$*.*

Going back to Example 1, we can conclude that $low(w_5) = low(w_7) = low(w_8) = 0$. As a consequence of this result, we can suggest the Head Oriented Processing (HOP) algorithm which works as follows. Before presenting HOP, we present some simple notation. Let $FixedWff(\Pi, s) = \{F | F : \mu \in U_{\Pi_s}(T^\omega_{\Pi_s})\}$. Given a set $X \subseteq FixedWff(\Pi, s)$, we define $Formula(X, \Pi, s)$ to be

$$\bigwedge_{G \in X} G \wedge \bigwedge_{G' \in FixedWff(\Pi,s) - X} \neg G'.$$

Here, $Formula(X, \Pi, s)$ is the formula which says that X consists of all and only those formulas in $FixedWff(\Pi, s)$ that are true. Given two sets $X_1, X_2 \subseteq FixedWff(\Pi, s)$, we say that $X_1 \approx X_2$ iff $Formula(X_1, \Pi, s)$ and $Formula(X_2, \Pi, s)$ are logically equivalent

HOP Algorithm.

1. Compute $T^\omega_{\Pi_s}$. $bestval = 0$; $best = NIL$.
2. Let $[X_1], \ldots, [X_n]$ be the $\sim$-equivalence classes defined above for Π, s.
3. For each equivalence class $[X_i]$ do:
   (a) If there is exactly one interpretation that satisfies $Formula(X_i, \Pi, s)$ then:
      i. **Minimize** p'_i **subject to** $\mathsf{RedCONS}_U(\Pi, s, T^\omega_{\Pi_s})$ where $[w_i]$ is the set of worlds satisfying exactly those heads in X_i. Let Val be the value returned.
      ii. If $Val > best$, then $\{best = w_i; bestval = Val\}$.
4. If $bestval = 0$ then return any world whatsoever otherwise return $best$.

Theorem 2 (correctness of HOP). *Algorithm HOP is correct, i.e. it is guaranteed to return a world whose low probability is greater than or equal to that of any other world.*

Step $3(a)$ of the HOP algorithm is known as the UNIQUE-SAT problem—it can be easily implemented via a SAT solver as follows.

1. If $\bigwedge_{F \in X} F \wedge \bigwedge_{G \in \bar{X}} \neg G$ is satisfiable (using a SAT solver that finds a satisfying world w) then
   (a) If $\bigwedge_{F \in X} F \wedge \bigwedge_{G \in \bar{X}} \neg G \wedge (\bigvee_{a \in w} \neg a \vee \bigvee_{a' \in \bar{w}} a')$ is satisfiable (using a SAT solver) then return "two or more" (two or more satisfying worlds exist) else return "exactly one"
2. else return "none."

The following example shows how the HOP algorithm would work on the program from Example 2.

Example 3. Consider the program from Example 2, and suppose $X = \{(a \vee b \vee c \vee d \vee e), (a \wedge e)\}$. In Step $(3a)$, the algorithm will find that $\{a, d, e\}$ is a model of $(a \vee b \vee c \vee d \vee e) \wedge (a \wedge e)$; afterwards, it will find $\{a, c, e\}$ to be a model of $(a \vee b \vee c \vee d \vee e) \wedge (a \wedge e) \wedge ((\neg a \vee \neg d \vee \neg e) \vee (b \vee c))$. Thus, X has more than one model and the algorithm will not consider any of the worlds in the equivalence class induced by X as a possible solution, which avoids solving the linear program for those worlds.

The complexity of HOP is also exponential. However, HOP can sometimes be preferable to the Naive algorithm. The number of variables in $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ is $2^{card(T^{\omega}_{\Pi_s})}$, which is much smaller than the number of variables in $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ when the number of ground rules whose bodies are satisfied by state s is smaller than the number of ground atoms. The checks required to find all the equivalence classes $[X_i]$ take time proportional to $2^{2*card(T^{\omega}_{\Pi_s})}$. Lastly, HOP avoids solving the reduced linear program for all the non-singleton equivalence classes (for instance, in Example 3, the algorithm avoids solving the LP three times). This last saving, however, comes at the price of solving SAT twice for each equivalence class and the time required to find the $[X_i]$'s.

A variant of the HOP algorithm, which we call the SemiHOP algorithm, tries to avoid computing the full equivalence classes. The SemiHOP algorithm omits finding pairs of sets that represent the same equivalence class, and therefore does not need to compute the checks for logical equivalence of every possible pair, a computation which can be very expensive.

Proposition 5. *Suppose Π is an APLP, s is a state, and X is a subset of $Fixed Wff(\Pi, s)$. Then there exists a world w_i such that $\{w \mid w \mapsto Formula(X, \Pi, s)\} \subseteq [w_i]$.*

We now define the concept of a sub-partition.

Definition 11. *A* sub-partition *of the set of worlds of Π w.r.t. s is a partition $W_1, \ldots, W_k$ where:*

1. *$\bigcup_{i=1}^{k} W_i$ is the entire set of worlds.*
2. *For each W_i, there is an equivalence class $[w_i]$ such that $W_i \subseteq [w_i]$.*

The following result - which follows immediately from the preceding proposition - says that we can generate a subpartition by looking at all subsets of $Fixed Wff(\Pi, s)$.

Proposition 6. *Suppose Π is an APLP, s is a state, and $\{X_1, \ldots, X_k\}$ is the power set of $Fixed Wff(\Pi, s)$. Then the partition $W_1, \ldots, W_k$ where $W_i = \{w | w \mapsto Formula(X_i, \Pi, s)\}$ is a sub-partition of the set of worlds of Π w.r.t. s.*

The intuition behind the SemiHOP algorithm is best presented by going back to constraints 1 and 2 given in Example 1. Obviously, we would like to collapse all three variables p_5, p_7, p_8 into one variable y. However, if we were to just collapse p_7, p_8 into a single variable y', we would still reduce the size of the constraints (through the elimination of one variable), though the reduction would not be maximal. The SemiHOP algorithm allows us to use subsets of equivalence classes instead of full equivalence classes. We first define a *semi-reduced set of constraints* as follows.

Definition 12 (S_RedCONS$_U(\Pi, s, T^\omega_{\Pi_s})$)**.** *Let $W_1, \ldots, W_k$ be a subpartition of the set of worlds for Π and s. For each W_i,* S_RedCONS$_U(\Pi, s, T^\omega_{\Pi_s})$ *uses a variable $p^\star_i$ to denote the summation of the probability of each of the worlds in W_i. For each ap-wff $F : [\ell, u]$ in $T^\omega_{\Pi_s}$,* RedCONS$_U(\Pi, s, T^\omega_{\Pi_s})$ *contains the constraint:*

$$\ell \leq \Sigma_{W_i \mapsto F} p^\star_i \leq u.$$

Here, $W_i \mapsto F$ implies that some world in W_i satisfies F. In addition, S_RedCONS$_U(\Pi, s, T^\omega_{\Pi_s})$ *contains the constraint*

$$\Sigma_{W_i} p^\star_i = 1.$$

Example 4. Returning to Example 1, S_RedCONS$_U(\Pi, s, T^\omega_{\Pi_s})$ could contain the following constraints: $0.7 \leq p_2 + p_3 + p_5 + p_6 + y' \leq 1$, $0.2 \leq p_5 + y' \leq 0.6$, and $p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + y' = 1$ where $y' = p_7 + p_8$.

SemiHOP Algorithm.

1. Compute $T^\omega_{\Pi_s}$.
2. $bestval = 0$; $best = NIL$.
3. For each set $X \subseteq FixedWff(\Pi, s)$ do:
   (a) If there is exactly one interpretation that satisfies $Formula(X, \Pi, s)$ then:
      i. **Minimize $p^\star_i$ subject to** S_RedCONS$_U(\Pi, s, T^\omega_{\Pi_s})$ where W_i is a subpartition of the set of worlds of Π w.r.t. s. Let Val be the value returned.
      ii. If $Val > best$, then $\{best = w_i; bestval = Val\}$.
4. If $bestval = 0$ then return any world whatsoever otherwise return $best$.

Theorem 3 (correctness of SemiHOP). *Algorithm SemiHOP is correct, i.e. it is guaranteed to return a world whose low probability is greater than or equal to that of any other world.*

The key advantage of SemiHOP over HOP is that we do not need to construct the set $[w_i]$ of worlds, i.e. we do not need to find the equivalence classes $[w_i]$. This is a potentially big saving because there are 2^n possible worlds (where n is the number of ground action atoms) and finding the equivalence classes can be expensive. This advantage comes with a drawback - the size of the set S_RedCONS$_U(\Pi, s, T^\omega_{\Pi_s})$ can be a bit bigger than the size of the set RedCONS$_U(\Pi, s, T^\omega_{\Pi_s})$.

5 Heuristic Methods for finding a Maximally ProbableWorld

In the preceding sections, we have developed three sets of constraints associated, respectively, with the naive algorithm, HOP, and SemiHOP. In all cases, the set of constraint variables can be enormous, even though HOP and SemiHOP try to reduce the number of variables. In this section, we develop a heuristic algorithm to reduce the number of variables even further. To see how the algorithm works, let $\mathcal{C}$ be the set of constraints generated by either Naive, HOP, or SemiHOP. The constraints have one of the forms

$$\ell \leq q_1 + \cdots + q_r \leq u \tag{5}$$

$$q_1 + \cdots + q_m = 1. \tag{6}$$

Suppose we make an *a priori* commitment to only look at some set S_k of k variables from the linear program. In this case, we could eliminate variables not in S_k from any summation in (5). Thus, we might weaken (5) and (6) above to

$$\ell \leq \Sigma_{\{q_1,\ldots,q_r\} \cap S_k} q_i \leq u \tag{7}$$

$$\Sigma_{\{q_1,\ldots,q_m\} \cap S_k} q_i \leq 1. \tag{8}$$

Let $\mathcal{C}'$ be the modification of the constraints in $\mathcal{C}$ derived in this way. It is immediately apparent that as all the lower bounds are set to ℓ, a solution to $\mathcal{C}'$ may or may not exist. Rather than weakening the lower bound from ℓ to 0 (which would guarantee a solution), we wondered how "close" to ℓ one can get while still having a solvable system of equations.

As a consequence, our *binary heuristic* works as follows by *only modifying lower bounds* of such constraints. We start with $\mathcal{C}'$ and see if it is solvable by itself. If so, we minimize each variable in S_k subject to $\mathcal{C}'$ and return the variable (and value) with the highest value. If not, we try to decrease the lower bounds of one or more constraints in $\mathcal{C}'$ as follows. Suppose $c^\star$ is one such constraint of the form

$$\ell^\star \leq \Sigma_{q_i \in S_k} q_i \leq u$$

Furthermore, suppose this constraint was derived from a constraint of the type shown in Equation (5). In this case, we try to replace $\ell^\star$ by $\frac{\ell^\star}{2}$. If this yields a solvable set of equations, we try to replace $\frac{\ell^\star}{2}$ by $\frac{3 \times \ell^\star}{4}$ - if the resulting system of equations is unsolvable, we try to replace it with $\frac{5 \times \ell^\star}{8}$ and so forth. Effectively, we try to keep the lower bounds of constraints as close to those in $\mathcal{C}$ as possible, while still being solvable when terms not in S_k are eliminated from the summations in Equation (5). We will call this the *binary heuristic* due to the fact that it resembles a binary search.

Once we have completed this process of modifying the lower bounds of constraints in $\mathcal{C}'$ (let the resulting system of constraints be called $\mathcal{C}^\bullet$) we minimize each and every variable in S_k subject to the constraints in $\mathcal{C}^\bullet$. The variable with the highest minimal value is returned (together with its value).

Example 5. Suppose we have the same set $\mathsf{CONS}_U(\Pi, s, T^\omega_{\Pi_s})$ as in Example 1. If we now choose the set of four variables $S_k = \{p_2, p_4, p_6, p_8\}$, $\mathcal{C}'$ contains the following constraints:

$$0.7 \leq p_2 + p_6 + p_8 \leq 1$$
$$0.2 \leq p_8 \leq 0.6$$
$$p_2 + p_4 + p_6 + p_8 \leq 1$$

If the algorithm starts by considering the first constraint in $\mathcal{C}'$ it replaces it with $0.35 \leq p_2 + p_6 + p_8 \leq 1$, which yields an unsolvable set of constraints. The lower bound gets succesively replaced by 0.525, 0.4375, and 0.39375, which finally yields a solvable system. At this point, the algorithm decides to accept this value as the lower bound for the constraint. The same process is also carried out for the other constraint.

6 Implementation and Experiments

We have implemented four of the algorithms described in this paper—the naive, HOP, SemiHOP, and the binary heuristic algorithms—using approximately 6,000 lines of Java code. The binary heuristic algorithm was applied to each of the ($\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$, $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$, and $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$) constraint sets; we refer to these approximations as the naive$_{bin}$, HOP_{bin}, and $\mathsf{SemiHOP}_{bin}$ algorithms respectively. Our experiments were performed on a Linux computing cluster comprised of 16 dual-core, dual-processor nodes with 8GB RAM. The linear constraints were solved using the QSopt linear programming solver library, and the logical formula manipulation code from the COBA belief revision system and SAT4J satisfaction library were used in the implementation of the HOP and SemiHOP algorithms.

For each experiment, we held the number of rules constant at 10 and did the following: (i) we generated a new ap-program and sent it to each of the three algorithms, (ii) varied the number of worlds from 32 to 16,384, performing at least 4 runs for each value and recording the average time taken by each algorithm, and (iii) we also measured the quality of the SemiHOP and all algorithms that use the binary heuristic by calculating the average distance from the solution found by the exact algorithm. Due to the immense time complexity of the HOP algorithm, we do not directly compare its performance to the naive algorithm or SemiHOP. In the results below we use the metric $ruledensity = \frac{\mathcal{L}_{act}}{card(T^{\omega}_{\Pi_s})}$ to represent the size of the ap-program; this allows for the comparison of the naive, HOP and SemiHOP algorithms as the number of worlds increases.

Running time. Figure 3 shows the running times for each of the naive, SemiHOP, naive$_{binary}$, and $\mathsf{SemiHOP}_{binary}$ algorithms for increasing number of worlds. As expected, the binary search approximation algorithm is superior to the exact algorithms in terms of computation time, when applied to both the naive and SemiHOP contstraint sets. With a sample size of 25%, naive$_{binary}$ and $\mathsf{SemiHOP}_{binary}$ take only about 132.6 seconds and 58.19 seconds for instances with 1,024 worlds, whereas the naive algorithm requires almost 4 hours (13,636.23 seconds). This result demonstrates that the naive algorithm is more or less useless and takes prohibitive amounts of time, even for small instances. Similarly, the checks for logical equivalence required to obtain each $[w_i]$ for HOP cause the algorithm to consistently require an exorbitant amount of time; for instances with only 128 worlds, HOP takes 58,064.74 seconds, which is much greater even than the naive algorithm for 1,024 worlds. Even when using the binary heuristic to further reduce the number of variables, HOP_{bin} still requires a prohibitively large amount of time.

At low rule densities, SemiHOP runs slower than the naive algorithm; with 10 rules, SemiHOP uses 18.75 seconds and 122.44 seconds for 128 worlds, while the naive algorithm only requires 1.79 seconds and 19.99 seconds respectively. However, SemiHOP vastly outperforms naive for problems with higher densities—358.3 seconds versus 13,636.23 seconds for 1,024 worlds—which more accurately reflect real-world problems in which the number of possible worlds is far greater than the number of ap-rules. Because the SemiHOP algorithm uses subpartions rather than unique equivalence classes in the $\mathsf{RedCONS}_U(\Pi, seconds, T^{\omega}_{\Pi_s})$ constraints, the algorithm overhead

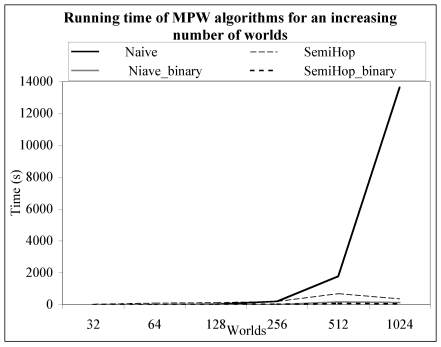

Fig. 3. Running time of the algorithms for increasing number of worlds.

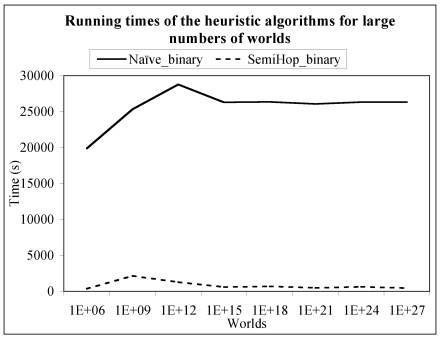

Fig. 4. Running time of naive$_{bin}$ and SemiHOP$_{bin}$ for large number of worlds.

is much lower than that of the HOP algorithm, and thus yields a more efficient running time.

The reduction in the size of $\mathcal{C}'$ afforded by the binary heuristic algorithm allows us to apply the naive and SemiHOP algorithms to much larger ap-programs. In Figure 4, we examine the running times of the naive$_{bin}$ and SemiHOP$_{bin}$ algorithms for large numbers of worlds (up to about 1.23794×10^{27} possible worlds) with a sample size for the binary heuristic of 2%; this is to ensure that the reduced linear program is indeed tractable. SemiHOP$_{binary}$ consistently takes less time than naive$_{binary}$, though both algorithms still perform rather well. For 1.23794×10^{27} possible worlds, naive$_{binary}$ takes on average 26,325.1 seconds while SemiHOP$_{binary}$ requires only 458.07 seconds. This difference occurs because, $|\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})| < |\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})|$ that is the heuristic algorithm is further reducing an already smaller constraint set. In addition, because SemiHOP only solves the linear constraint problem when there is exactly one satisfying interpretation for a subpartition, it performs fewer computations overall. Figure 5 contains additional experiments running SemiHOP$_{binary}$ on very large ap-programs (from 1,000 to 100,000 ground atoms). Even for such a large number of worlds, the running time is only around 300 seconds for a 2% sample rate.

Quality of solution. Figure 6 compares the accuracy of the probability found for the most probable world by SemiHOP, naive$_{binary}$, and SemiHOP$_{binary}$ to the solution obtained by the naive algorithm, averaged over at least 4 runs for each number of worlds. The results are given as a percentage of the solution returned by the naive algorithm, and are only reported in cases where both algorithms found a solution. The SemiHOP and SemiHOP$_{binary}$ algorithms demonstrate near perfect accuracy; this is significant because in the SemiHOP$_{binary}$ algorithm, the binary heuristic was only sampling 25% of the possible subpartitions. However, in many of these cases, both the naive and the SemiHOP algorithms found most probable worlds with a probability of zero. The most probable world found by the naive$_{binary}$ algorithm can be between 75% and 100% as likely as those given by the regular naive algorithm; however, the naive$_{binary}$ algorithm also was often unable to find a solution.

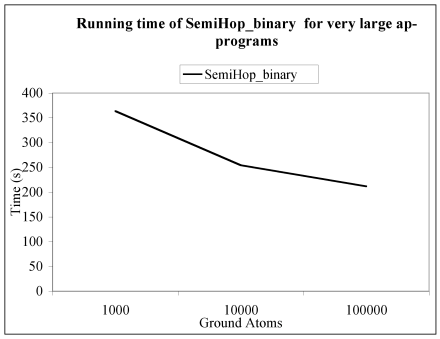

Fig. 5. Running time of the SemiHOP$_{binary}$ algorithm for very large numbers of possible worlds.

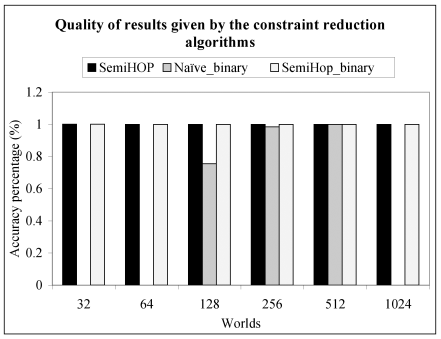

Fig. 6. Quality of the solutions produced by SemiHOP, naive$_{bin}$, and SemiHOP$_{bin}$ as compared to Naive.

7 Conclusions and Related Work

Probabilistic logic programming was introduced in [5, 1] and later studied by several authors [10–13]. This work was preceded by earlier—non-probabilistic—papers on quantitative logic programming of which [14] is an example. [10] presents a model theory, fixpoint theory, and proof procedure for conditional probabilistic logic programming. [11] combines probabilistic LP with maximum entropy. [15] presents a conditional semantics for probabilistic LPs where each rule is interpreted as specifying the conditional probability of the rule head, given the body. [12] develops a semantics for logic programs in which different general axiomatic methods are given to compute probabilities of conjunctions and disjunctions, and [13] presents an approach to a similar problem. gp-programs were implemented by [16], based on the DisLOG system [17].

However, all works to date on probabilistic logic programming have addressed the problem of checking whether a given formula of the form $F : [L, U]$ is entailed by a probabilistic logic program. This usually boils down to finding out if all interpretations that satisfy the PLP assign a probability between L and U to F.

Our work builds on top of the gp-program paradigm [5]. Our framework modifies gp-programs in three ways: (i) we do not allow non-action predicates to occur in rule heads, while gp-programs do, (ii) we allow arbitrary formulas to occur in rule heads, whereas gp-programs only allow the so-called "basic formulas" to appear in rule heads. (iii) Most importantly, of all, we solve the problem of finding the most probable model whereas [5] solve the problem of entailment.

This is justified because in certain classes of applications, a p-program describes probabilities on possible worlds, and we are interested in finding that world which has the highest probability. Such an example could be a market model of bidding in a specialized auction, such as an electricity auction, which contains rules specifying what actions a potential competitor might take in a given situation. Here, the organization coming up with the model might want to know the most likely scenarios (worlds) that they have to face. We have been working on an economic application about what may

occur in a given market when certain actions such as "reduce price of fruit by 10%" are taken (e.g., by increasing supply). Of course, this can be viewed as an entailment problem (take the conjunction of positive atoms in a world, conjoin that with the conjunction of negative atoms in that world, solve a linear program for each world, and choose the best one). This corresponds to the naive (exact) solution in this paper which is easily shown to not work at all once the amount of worlds exceeds a small number. What we do in this paper is provide algorithms to find the world that has the maximal probability, and to our knowledge, we are the first to do this. We further provide two approximation algorithms that have been experimentally shown to produce solutions within about 10-15% of the optimal solution in a small fraction of the time required to find the best solution.

References

1. Ng, R.T., Subrahmanian, V.S.: Probabilistic logic programming. Information and Computation **101**(2) (1992) 150–201
2. Hailperin, T.: Probability logic. Notre Dame Journal of Formal Logic **25 (3)** (1984) 198–212
3. Fagin, R., Halpern, J.Y., Megiddo, N.: A logic for reasoning about probabilities. Information and Computation **87**(1/2) (1990) 78–128
4. Nilsson, N.: Probabilistic logic. Artificial Intelligence **28** (1986) 71–87
5. Ng, R.T., Subrahmanian, V.S.: A semantical framework for supporting subjective and conditional probabilities in deductive databases. In Furukawa, K., ed.: Proc. of the 8th Int. Conf. on Logic Programming, The MIT Press (1991) 565–580
6. Subrahmanian, V., Albanese, M., Martinez, V., Reforgiato, D., Simari, G.I., Sliva, A., Udrea, O., Wilkenfeld, J.: CARA: A Cultural Adversarial Reasoning Architecture. IEEE Intelligent Systems **22**(2) (2007) 12–16
7. Lloyd, J.W.: Foundations of Logic Programming, Second Edition. Springer-Verlag (1987)
8. Wilkenfeld, J., Asal, V., Johnson, C., Pate, A., Michael, M.: The use of violence by ethnopolitical organizations in the middle east. Technical report, National Consortium for the Study of Terrorism and Responses to Terrorism (2007)
9. Ernst, J., Martinez, V., Simari, G.I., Sliva, A.: Mining rules about behaviors of terror groups, In preparation for submission to a conference (2007)
10. Ngo, L., Haddawy, P.: Probabilistic logic programming and bayesian networks. In: Asian Computing Science Conf. (1995) 286–300
11. Lukasiewicz, T., Kern-Isberner, G.: Probabilistic logic programming under maximum entropy. LNCS (Proc. ECSQARU-1999) **1638** (1999)
12. Lakshmanan, L.V.S., Shiri, N.: A parametric approach to deductive databases with uncertainty. IEEE Trans. on Knowledge and Data Engineering **13**(4) (2001) 554–570
13. Dekhtyar, A., Subrahmanian, V.S.: Hybrid probabilistic programs. In: Int. Conf. on Logic Programming. (1997) 391–405
14. van Emden, M.: Quantitative deduction and its fixpoint theory. Journal of Logic Programming **4** (1986) 37–53
15. Lukasiewicz, T.: Probabilistic logic programming. In: European Conference on Artificial Intelligence. (1998) 388–392
16. Pillo, A.: Implementation and investigation of probabilistic reasoning in deductive databases. Diploma Thesis, University of Wuerzburg (1998)
17. Seipel, D., Thöne, H.: DISLOG - A system for reasoning in disjunctive deductive databases. In: DAISD. (1994) 325–343