Computing Most Probable Worlds for Action Probabilistic Logic Programs [†]

Gerardo Ignacio Simari
`gisimari@cs.umd.edu`

Department of Computer Science
University of Maryland College Park

Abstract

The semantics of probabilistic logic programs (PLPs) is usually given through a possible worlds semantics. We propose a variant of PLPs called *action probabilistic logic programs* or *ap*-programs that use a two-sorted alphabet to describe the conditions under which certain real-world entities take certain actions. In such applications, worlds correspond to sets of actions these entities might take. Thus, there is a need to find the most probable world (MPW) for *ap*-programs. In contrast, past work on PLPs has primarily focused on the problem of entailment.

This paper quickly presents the syntax and semantics of ap-programs and then shows a naive algorithm to solve the MPW problem using the linear program formulation commonly used for PLPs. As such linear programs have an exponential number of variables, we present two important new algorithms, called HOP and SemiHOP to solve the MPW problem exactly. Both these algorithms can significantly reduce the number of variables in the linear programs. Subsequently, we present a "binary" algorithm that applies a binary search style heuristic in conjunction with the Naive, HOP and SemiHOP algorithms to quickly find worlds that may not be "most probable." We experimentally evaluate these algorithms both for accuracy (how much worse is the solution found by these heuristics in comparison to the exact solution) and for scalability (how long does it take to compute). We show that the results of SemiHOP are very accurate and also very fast: more than 10^{27} worlds can be handled in a few minutes.

1 Introduction

Probabilistic logic programs (PLPs) [NS92] 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 [Hai84], [FHM90], and [Nil86] to induce a set of probability distributions on a space of possible worlds. Past work on PLPs [NS91, NS92] 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 [SAM$^+$07, Bha07] 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. Our research group has thus far built models of approximately 30 groups around the world including tribes such as the Shinwaris and Waziris, terror groups like Hezbollah and the PKK, political parties such as the Pakistan People's Party and the Harakat-e-Islami as well as nation states. Of course, all these models only capture a few actions that these entities might take. 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 the current state."*

[†]Submitted to the Department of Computer Science, University of Maryland College Park, in partial fulfilment of the requirements for degree of Master in Science in Computer Science. The work described here is based on work done in collaboration with V.S. Subrahmanian, Dana Nau, Samir Khuller, Maria Vanina Martinez, and Amy Sliva [SSNS06, KMN$^+$07b, KMN$^+$07a, SMSS07, SAM$^+$07].

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 in a given situation. This corresponds precisely to the problem of finding a "most probable world" that is the focus of this paper.

In Section 2, we define the syntax and semantics of action-probabilistic logic programs (*ap*-programs for short). This is a straightforward variant of PLP syntax and semantics from [NS91, NS92] and is not claimed as anything dramatically new. We describe the *most probable world* (MPW) problem by immediately using the linear programming methods of [NS91, NS92] —these methods are exponential because the linear programs are exponential in the number of ground atoms in the language. The new content of this paper starts in Section 4 where we present the *Head Oriented Processing (HOP)* approach; HOP reduces the linear program for *ap*-programs, and 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 are 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 called the *Binary* heuristic that can be applied in conjunction with the *Naive*, HOP, and SemiHOP algorithms. The basic idea is that rather than examining all worlds corresponding to the linear programming variables used by these algorithms, only some fixed number k of worlds is examined. This leads to a linear program whose number of variables is k. Finally, Section 6 describes a prototype implementation of our *ap*-program 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. We show that the SemiHOP algorithm with the binary heuristic is quite accurate (at least when only a small number of worlds is involved) and then show that it scales very well, managing to handle situations with over 10^{27} worlds in a few minutes.

2 Syntax and Semantics of *ap*-programs

Action probabilistic logic programs (*ap*-programs) are an immediate and obvious variant of the probabilistic logic programs introduced in [NS91, NS92]. 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 [Llo87]. 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 atom (resp. action atom) is a well-formed formula (wff) (resp. an action well-formed formula, or action wff). If F, G are wffs (resp. action wffs), then $(F \wedge G), (F \vee G)$ and $\neg F$ are all wffs (resp. action wffs).

Definition 2.1 *If F is a wff (resp. action wff) and $\mu = [\alpha, \beta] \subseteq [0, 1]$, then $F : \mu$ is called a p-annotated wff (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). Notice that wffs are annoted with probability intervals rather than point probabilities. There are three reasons for this. (i) In many cases, we are told that an action formula F is true in state s with some probability p plus or minus some margin of error e — this naturally translates into the interval $[p - e, p + e]$. (ii) As shown by [FHM90, NS92], if we do not know the relationship between events e_1, e_2, even if we know point

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

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

probabilities for e_1, e_2, we can only infer an interval for the conjunction and disjunction of e_1, e_2. (iii) Interval probabilities generalize point probabilities anyway, so our work is also relevant to point probabilities.

Definition 2.2 (*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 2.3 (*ap*-program) *An* action probabilistic logic program *(*ap-*program for short) is a finite set of* ap-*rules.*

Figure 1 shows a small rule base consisting of some rules we have derived automatically about Hezbollah using behavioral data from [WAJ+07]. The behavioral data in [WAJ+07] has tracked over 200 terrorist groups for about 20 years from 1980 to 2004. For each year, values have been gathered for about 150 measurable variables for each group in the sample. These variables include conditions such as 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 developed, but is not covered in this work. We show four 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 2.4 (world/state) *A* world *is any set of ground action atoms. A* state *is any finite set of ground non-action atoms.*

Example 2.1 *Consider the* ap-*program from Figure 1; there are two ground action atoms:* kidnap *and* armed_attacks, *and there are therefore a total of $2^2 = 4$ possible worlds. These are: $w_0 = \emptyset$, $w_1 = \{kidnap\}$, $w_2 = \{armed\_attacks\}$, and $w_3 = \{kidnap, armed\_attacks\}$. The following are two possible states:*

$$s_1 = \{statusMilitaryWing(standing), unDemocratic, internalConflicts\},$$
$$s_2 = \{interOrganizationConflicts, orgPopularity(moderate)\}$$

1.	*d:* $[0.52, 0.82]$	$\leftarrow$	.
2.	$b \wedge a$: $[0.55, 0.69]$	$\leftarrow$	*d:* $[0.48, 0.89]$.

Figure 2: A simple example of an *ap*-program with action atoms in the body of the rules, which is already reduced with respect to a certain state.

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 2.5 *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.* The following is an example of a reduction with respect to a state.

Example 2.2 *Let* Π *be the* ap*-program from Figure 1, and suppose we have the following state:*

$$s = \{statusMilitaryWing(standing), unDemocratic, internalConflicts\}$$

The reduction of Π *with respect to state* s *is:*

$$\Pi_s = \{kidnap : [0.60, 0.68], armed\_attacks : [0.93, 1.0]\}.$$

Key differences. The key differences between *ap*-programs and the PLPs of [NS91, NS92] are that (i) *ap*-programs have a bipartite structure (action atoms and state atoms) and (ii) they allow arbitrary formulas (including ones with negation) in rule heads ([NS91, NS92] do not). They can easily be extended to include variable annotations and annotation terms as in [NS91]. Likewise, as in [NS91], 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 this paper. *However, the most important difference between our paper and [NS91, NS92] 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. We first define an intermediate operator $U_{\Pi_s}(X)$.

Definition 2.6 *Suppose* X *is a set of ground* ap*-wffs. We define* $U_{\Pi_s}(X) = \{F : \mu \mid F : \mu \leftarrow A_1 : \mu_1 \wedge \cdots \wedge A_m : \mu_m$ *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 *ap*-wffs 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. The following is an example of the calculation of $U_{\Pi_s}(X)$.

Example 2.3 *Consider the simple program depicted in Figure 2, and let $X = \{d : [0.5, 0.55]\}$. In this case, $U_{\Pi_s}(X) = \{d : [0.52, 0.82], b \wedge a : [0.55, 0.69]\}$.*

In order to assign a probability interval to each ground action atom, we use the same procedure followed in [NS91]. We use $U_{\Pi_s}(X)$ to set up a linear program $\mathsf{CONS}_U(\Pi, s, X)$ as follows.

Definition 2.7 *Let Π be an* ap*-program and s be a state. 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 an Herbrand interpretation (where action symbols are treated like predicate symbols), 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 $\mathsf{CONS}_U(\Pi, s, X)$.*
2. *$\Sigma_{w_i} p_i = 1$ is in $\mathsf{CONS}_U(\Pi, s, X)$.*

We refer to these as constraints of type (1) and (2), respectively.

The following is an example of how these constraints look.

Example 2.4 *Let Π be the* ap*-program from Figure 2, and $X = \{d : [0.5, 0.55]\}$. The possible worlds are: $w_0 = \emptyset$, $w_1 = \{d\}$, $w_2 = \{b\}$, $w_3 = \{a\}$, $w_4 = \{d, b\}$, $w_5 = \{d, a\}$, $w_6 = \{b, a\}$, and $w_7 = \{d, b, a\}$. In this case, the linear program $\mathsf{CONS}_U(\Pi, s, X)$ contains the following constraints:*

$$0.52 \leq p_1 + p_4 + p_5 + p_7 \leq 0.82$$
$$0.55 \leq p_6 + p_7 \leq 0.69$$
$$p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 = 1$$

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 2.8 *Suppose Π is an ap-program, s is a state, and X is a set of ground* ap*-wffs. Our operator $T_{\Pi_s}(X)$ is then defined to be*

$$\{F : [\ell(F), u(F)] \mid (\exists \mu)\ F : \mu \in U_{\Pi_s}(X)\} \cup$$
$$\{A : [\ell(A), u(A)] \mid A \text{ is a ground action atom}\}.$$

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. The following is an example of this process.

Example 2.5 *Consider the* ap*-program presented in Figure 2, with the same state s. For $T_{\Pi_s} \uparrow 0$, we have $X = \emptyset$. We first obtain $U_{\Pi_s}(\emptyset) = \{d : [0.52, 0.82]\}$. Then, $T_{\Pi_s}(\emptyset) = \{d : [0.52, 0.82], a : [0, 1.0], b : [0, 1.0]\}$.*

To obtain $T_{\Pi_s} \uparrow 1 = T_{\Pi_s}(T_{\Pi_s} \uparrow 0)$, let $X = T_{\Pi_s}(\emptyset)$. Then we have:

$U_{\Pi_s}(X) = \{d : [0.52, 0.82], b \wedge a : [0.55, 0.69]\}$, *and*
$T_{\Pi_s}(X) = \{d : [0.52, 0.82], b \wedge a : [0.55, 0.69]\}$
$\quad \cup \{A : [\ell(A), u(A)] \mid A$ *is a ground action atom* $\}$.

In order to infer the probability bounds for all ground action atoms, we need to build a linear program using the formulas from $U_{\Pi_s}(X)$ *and solve it for each ground atom by minimizing and maximizing the objective function of the probabilities of the worlds that satisfy each atom. The possible worlds are:* $w_0 = \emptyset$, $w_1 = \{d\}$, $w_2 = \{b\}$, $w_3 = \{a\}$, $w_4 = \{d, b\}$, $w_5 = \{d, a\}$, $w_6 = \{b, a\}$, *and* $w_7 = \{d, b, a\}$. *The linear program then consists of the following constraints:*

$$0.52 \leq p_1 + p_4 + p_5 + p_7 \leq 0.82$$
$$0.55 \leq p_6 + p_7 \leq 0.69$$
$$p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 = 1$$

In order to obtain $\ell(d)$ *and* $u(d)$ *(that is, bound the probability value for action atom* d*), we minimize and then maximize the objective function* $p_1 + p_4 + p_5 + p_6$ *subject to the linear program above, obtaining:* $d : [0.52, 0.82]$. *Similarly, we use the objective function* $p_3 + p_5 + p_6 + p_7$ *for atom* a*, obtaining* $a : [0.55, 1.0]$*, and* $p_2 + p_4 + p_6 + p_7$ *for* b*, obtaining* $b : [0.55, 1.0]$. *Therefore, we have finished calculating* $T_{\Pi_s} \uparrow 1$*, and we have obtained* $T_{\Pi_s}(X) = \{d : [0.52, 0.82], b \wedge a : [0.55, 0.69], a : [0.55, 1.0], b : [0.55, 1.0]\}$.

Similar computations with $X = T_{\Pi_s}(T_{\Pi_s}(\emptyset))$ *allows us to conclude that* $T_{\Pi_s} \uparrow 2 = T_{\Pi_s} \uparrow 1$*, which means we reached the fixed point.*

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 [NS91] shows that

Proposition 2.1 *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, given an *ap*-program and a current state, finding the most probable world. As explained through our Hezbollah example, we may be interested in knowing what actions a group might take in a given situation.

Definition 3.1 (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:* $\mathsf{low}(w_i) =$ **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* $up(w_i) =$ **maximize** p_i ***subject to*** $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$.

Thus, the lower 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 (PLP-CONS). The problem is in the class EXPTIME.

Proposition 3.1 (BOUNDED-LOW complexity) *Given a ground* ap*-program* Π*, a state* s*, a world* w*, and a probability threshold* p_{th}*, deciding if* $low(w) \geq p_{th}$ *is* NP*-hard.*

Proof 3.1 *We will reduce the* PLP*-CONS problem to the problem of deciding if a certain world* w *is such that* $low(w) \geq p_{th}$ *for a certain probability value* p_{th}*. Because* PLP*-CONS was proven to be* NP*-hard [FHM90], this reduction will prove that BOUNDED-LOW is* NP*-hard as well.*

Given an instance of PLP*-CONS consisting of a program* Π *and a state* s*, we build an instance of BOUNDED-LOW, consisting of an* ap*-program* Π'*, a state* s'*, a world* w*, and a probability threshold* p_{th} *in the following manner: program* Π' *is equal to* Π *and state* s' *is equal to* s*, world* w *is an arbitrary world, and* $p_{th} = 0$*. We must now show that this transformation yields a reduction by proving that* Π *is consistent in state* s *if and only if* $low(w) \geq 0$ *with respect to* Π' *and state* s'*:*

- Π *is consistent* $\Rightarrow$ $low(w) \geq 0$ *with respect to* Π' *in state* s'*: If* Π *is consistent, this means that* $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ *is solvable. Therefore, it is clear that any possible world will receive a probability value greater than or equal to zero.*

- $low(w) \geq 0$ *with respect to* Π' *in state* s' $\Rightarrow$ Π *is consistent: If* $low(w) \geq 0$ *with respect to* Π *in state* s'*, this means that* w *has received a probability value greater than or equal to zero, subject to* $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$*. This is only possible if* $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ *is solvable, which means that* Π *is consistent (this well known property was proved in [NS92]).*

To complete the proof, we note that the transformation from a PLP*-CONS instance to a BOUNDED-LOW instance can be done in polynomial time with respect to the size of the* ap*-program given for* PLP*-CONS.*

Proposition 3.2 (BOUNDED-UP complexity) *Given a ground* ap*-program* Π*, a state* s*, a world* w*, and a probability threshold* p_{th}*, deciding if* $up(w) \leq p_{th}$ *is* NP*-hard.*

Proof 3.2 *Analogous to that of Proposition 3.1.*

An open problem is to characterize the precise complexity of the BOUNDED-LOW and BOUNDED-UP problems.

The MPW Problem. The *most probable world* problem (MPW for short) is the problem where, given an *ap*-program Π and a state s as input, we are required to find a world w_i where $low(w_i)$ is maximal. [1]

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

```
Naive Algorithm.

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 $Best$.
```

Figure 3: The Naive algorithm for finding a most probable world.

4 Exact Algorithms for finding a Maximally Probable World

In this section, we develop algorithms to find the most probable world for a given *ap*-program and a current state. As the above results show us, there is no unique probability associated with a world w; the probability could range anywhere between $low(w)$ and $up(w)$. Hence, in the rest of this paper, we will assume the worst case, i.e. the probability of world w is given by $low(w)$. We will try to find a world for which $low(w)$ is maximized.

In this section, we study the following problem: given an *ap*-program Π and a state s, find a world w such that $low(w)$ is maximized. If we replace $low(w)$ by $up(w)$, the techniques to find a world w such that $up(w)$ is maximal are similar (though not all apply directly). There may also be cases in which we are interested in using some other value (e.g. the average of $low(w)$ and $up(w)$ and so on).

A Naive Algorithm. The *naive* algorithm to find the most probable world is the direct implementation of the definition of the problem, and it basically consists of the steps described in Figure 3.

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 [NS92]) 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. In this paper, when we say that we are focusing on lowering the computation time of our algorithms, we are referring to improving Step 2(a).

In this section, we will present two algorithms, the HOP and the $\mathsf{SemiHOP}$ algorithms, both of which can significantly reduce the number of variables in the linear program by collapsing multiple linear programming variables into one. They both stem from the basic idea that when variables *always* appear in certain groups in the linear program, these groups can be *collapsed* into a single variable. As we will see, the basic idea can lead to great savings, but being too ambitious in trying to collapse all possible sets can be detrimental to our benefits; this last observation is the root of the second algorithm.

4.1 HOP: Head-Oriented Processing

We can do better than the naive algorithm without losing any precision in the calculation of a most probable world. In this section we present the HOP algorithm, prove its correctness, and propose an enhancement that also provably yields a most probable world.

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 heads are satisfied by w.

Definition 4.1 *Suppose Π is an* ap*-program, 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 if and only if 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 4.1.

Example 4.1 *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 occurs 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 the set $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$. 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 a constraint of type (1) in $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ or does not appear at all. This is what the next result states.

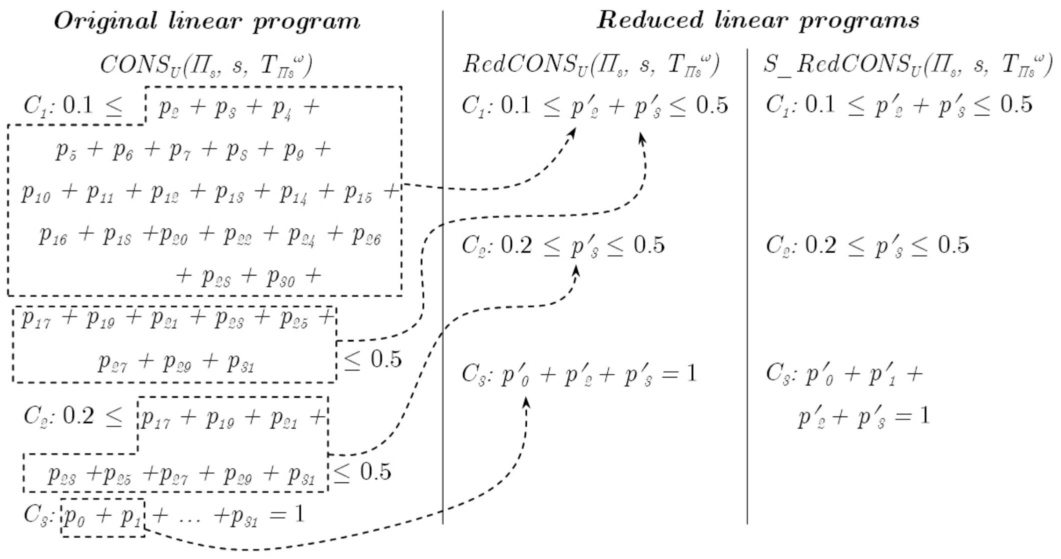
Original linear program
Reduced linear programs
$CONS_U(\Pi_s, s, T_{\Pi s}^{\omega})$
C_1: $0.1 \le p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 + p_9 + p_{10} + p_{11} + p_{12} + p_{13} + p_{14} + p_{15} + p_{16} + p_{18} + p_{20} + p_{22} + p_{24} + p_{26} + p_{28} + p_{30} + p_{17} + p_{19} + p_{21} + p_{23} + p_{25} + p_{27} + p_{29} + p_{31} \le 0.5$
C_2: $0.2 \le p_{17} + p_{19} + p_{21} + p_{23} + p_{25} + p_{27} + p_{29} + p_{31} \le 0.5$
C_3: $p_0 + p_1 + \ldots + p_{31} = 1$
$RedCONS_U(\Pi_s, s, T_{\Pi s}^{\omega})$
C_1: $0.1 \le p'_2 + p'_3 \le 0.5$
C_2: $0.2 \le p'_3 \le 0.5$
C_3: $p'_0 + p'_2 + p'_3 = 1$
$S\_RedCONS_U(\Pi_s, s, T_{\Pi s}^{\omega})$
C_1: $0.1 \le p'_2 + p'_3 \le 0.5$
C_2: $0.2 \le p'_3 \le 0.5$
C_3: $p'_0 + p'_1 + p'_2 + p'_3 = 1$

Figure 4: 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 Definitions 4.2 and 4.4.

Proposition 4.1 *Suppose* Π *is an* ap*-program,* s *is a state, and* $[w_i]$ *is a* $\sim$*-equivalence class. Then for each constraint* c *of the form*

$$\ell \le \Sigma_{w_r \mapsto F}\, p_r \le u \qquad (4)$$

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).*

Proof 4.1 *Let* c *be a constraint of the form* (4) *and suppose for a contradiction that there exist two variables,* p_x *and* p_y *such that* $w_x, w_y \in [w_i]$ *and* p_x *appears in the constraint* c*, while* p_y *does not. In this case,* $w_x \mapsto F$ *and* $w_y \not\mapsto F$*. However, in this case,* $w_x \not\sim w_y$*, and therefore cannot be in the same equivalence class* $[w_i]$*, yielding a contradiction.*

Example 4.2 *Here is a toy example of this situation. Suppose* Π_s *consists of the two very simple rules:*

$$\begin{aligned}(a \vee b \vee c \vee d) : [0.1, 0.5] &\leftarrow \;.\\ (a \wedge e) : [0.2, 0.5] &\leftarrow \;.\end{aligned}$$

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 4, 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 thus corresponds to* 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 the same time.*

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, and in some cases it can actually result in an increase in size). The number of constraints in the linear program stays the same. Formally speaking, we define a *reduced set of constraints* as follows.

Definition 4.2 ($\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$) *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 to find the most probable world, we only need to look at the equivalence classes that are of cardinality 1.

Theorem 4.1 *Suppose* Π *is an* ap*-program,* s *is a state, and* w_i *is a world. If* $card([w_i]) > 1$, *then* $low(w_i) = 0$.

Proof 4.2 *Immediate, by observing that there are no restrictions on the values assigned to the variables that correspond to worlds in the same* $\sim$*-class. If there is more than one world in a class* $[w_x]$, *there is always a solution that assigns zero to each variable* p_i *such that* $w_i \in [w_x]$, *and therefore* $low(w_i) = 0$.

Going back to Example 4.1, 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. First we present some simple notation. Let $\mathit{FixedWff}(\Pi, s) = \{F \mid F : \mu \in U_{\Pi_s}(T^{\omega}_{\Pi_s})\}$. Given a set $X \subseteq \mathit{FixedWff}(\Pi, s)$, we define $Formula(X, \Pi, s)$ to be

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

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

Theorem 4.2 (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.*

Proof 4.3 *We will prove this property in two stages:*

- Soundness: *We wish to show that if HOP returns a world* w_{sol}, *then there is no other world* w_i *such that* $low(w_i) > low(w_{sol})$. *Suppose HOP does return* w_{sol} *but that there is a world* w_i *such that* $low(w_i) > low(w_{sol})$. *Clearly,* $[w_i]$ *and* $[w_{sol}]$ *must be different* $\sim$*-equivalence classes. In this case, step 3 of the HOP algorithm will consider both these equivalence classes. As bestval is set to the highest value of* $low(w_j)$ *for all equivalence classes* $[w_j]$, *it follows that* $low(w_{sol}) \leq low(w_i)$, *yielding a contradiction.*

```
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 $\Pi, s$.
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 $best$.
```

Figure 5: The Head-Oriented Processing (HOP) algorithm.

- Completeness: *We wish to show that if there exists a world w_{max} such that $low(w_{max}) \geq low(w_i) \forall w_i \in \mathcal{W}$, then HOP will return a world w_{sol} such that $low(w_{sol}) = low(w_{max})$. Similar to the case made for soundness, if there exists a world w_{max} with the highest possible* low *value, it is either in the same class as the world that is returned by the algorithm, or in a different class. In the former case, the world returned clearly has the same value as w_{max}; in the latter, this must also be the case, since otherwise the algorithm would have selected the variable corresponding to $[w_{max}]$ instead.*

This concludes the proof, and we therefore have that HOP is guaranteed to return a world whose low probability is greatest.

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 4.2.

Example 4.3 *Consider the program from Example 4.2, 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 worst-case complexity of HOP is, as its Naive counterpart, exponential. However, HOP can sometimes (but not always) be preferable to the Naive algorithm. The number of variables in $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ is $2^{card(\Pi_s)}$, 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 $2^{2*card(\Pi_s)}$. Lastly, HOP avoids solving the reduced linear program for all the non-singleton equivalence classes (for instance, in Example 4.3, 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. We will now explore a way in which we can trade off computation time against how many of these savings we obtain, again without giving up obtaining an exact solution.

4.2 Enhancing HOP: The SemiHOP Algorithm

A variant of the HOP algorithm, which we call the SemiHOP algorithm, tries to avoid computing the full equivalence classes. As in the case of HOP, SemiHOP is also guaranteed to return a most probable world. The SemiHOP algorithm avoids 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 prove to be very expensive.

Proposition 4.2 *Suppose* Π *is an* ap*-program,* s *is a state, and* X *is a subset of* $\mathit{FixedWff}(\Pi, s)$*. Then there exists a world* w_i *such that* $\{w \mid w \mapsto \mathit{Formula}(X, \Pi, s)\} \subseteq [w_i]$*.*

Proof 4.4 *Immediate from Definition 4.1.*

We now define the concept of a sub-partition.

Definition 4.3 *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 $\mathit{FixedWff}(\Pi, s)$.

Proposition 4.3 *Suppose* Π *is an* ap*-program,* s *is a state, and* $\{X_1, \ldots, X_k\}$ *is the power set of* $\mathit{FixedWff}(\Pi, s)$*. Then the partition* $W_1, \ldots, W_k$ *where* $W_i = \{w \mid w \mapsto \mathit{Formula}(X_i, \Pi, s)\}$ *is a sub-partition of the set of worlds of* Π *w.r.t.* s*.*

Proof 4.5 *Immediate from Proposition 4.2.*

The intuition behind the SemiHOP algorithm is best presented by going back to constraints 1 and 2 given in Example 4.1. 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 (because we could have eliminated two variables). The SemiHOP algorithm allows us to use subsets of equivalence classes instead of full equivalence classes. We define a *semi-reduced set of constraints* as follows.

```
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 $p_i^\star$ subject to $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$
            where $W_i$ is a subpartition of the set of worlds of $\Pi$ 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 $best$.
```

Figure 6: The SemiHOP algorithm.

Definition 4.4 ($\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$) *Let $W_1, \ldots, W_k$ be a subpartition of the set of worlds for Π and s. For each W_i, $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ uses a variable $p_i^\star$ to denote the summation of the probability of each of the worlds in W_i. For each* ap*-wff $F : [\ell, u]$ in $T^{\omega}_{\Pi_s}$, $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ contains the constraint:*

$$\ell \leq \Sigma_{W_i \mapsto F} p_i^\star \leq u.$$

Here, $W_i \mapsto F$ implies that some world in W_i satisfies F. In addition, $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ contains the constraint

$$\Sigma_{W_i} p_i^\star = 1$$

Example 4.4 *Returning to Example 4.1, $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ 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 $y' = p_7 + p_8$.*

The pseudo-code for the SemiHOP algorithm is depicted in Figure 6. The following theorem ensures the correctness of this algorithm.

Theorem 4.3 (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.*

Proof 4.6 *The proof is completely analogous to that of Theorem 4.2, with the only difference in this case being that some of the equivalence classes will be partitioned.*

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. However, this advantage comes with a drawback, since the size of the set $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ can be a bit bigger than the size of the set $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$.

5 The Binary Heuristic

In this section, we introduce a heuristic called the *Binary Heuristic* that can be utilized in conjunction with any of the three exact algorithms described thus far (Naive, HOP, and SemiHOP) in the paper. The basic idea behind the Binary Heuristic is to limit the number of variables in the linear programs associated with the Naive, HOP, and SemiHOP algorithms to a fixed number k that is chosen by the user.

Suppose we use $\mathcal{V}_{Naive}, \mathcal{V}_{\mathsf{HOP}}$, and $\mathcal{V}_{\mathsf{SemiHOP}}$ to denote the set of variables occurring in the linear programs $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$, $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ and $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$, respectively. Note that all these linear programs contain two kinds of constraints:

- Interval constraints which have the form $\ell \leq p_{i_1} + \cdots + p_{i_m} \leq u$ and
- A single equality constraint of the form $p_1 + \cdots + p_n = 1$.

Let $\mathcal{V}^k_{Naive}, \mathcal{V}^k_{\mathsf{HOP}}, \mathcal{V}^k_{\mathsf{SemiHOP}}$ be some subset of k variables from each of these sets, respectively. Let CONS be one of $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$, $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$, or $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$. We now construct a linear program CONS' from CONS as follows.

- For all constraints of the form
$$\ell \leq p_{i_1} + \cdots + p_{i_m} \leq u$$
remove all variables in the summation that do not occur in the selected set of k variables and re-set the lower bound to 0.
- For the one constraint of the form $p_1 + \cdots + p_n = 1$, remove all variables in the summation that do not occur in the selected set of k variables and replace the equality "=" by "$\leq$".

Example 5.1 *Consider the program from Example 4.2, and suppose $m = 10$ and CONS refers to the constraints associated with the naive algorithm which has 32 worlds altogether. Then, we can select a sample of ten worlds such as*

$$\mathcal{W}_m = \{w_2, w_4, w_8, w_{10}, w_{12}, w_{16}, w_{18}, w_{22}, w_{23}, w_{25}\}$$

Now, $\mathsf{CONS}'(\Pi, s, T^{\omega}_{\Pi_s})$ *contains the following constraints:*

$$\begin{aligned}
&0 \leq p_2 + p_4 + p_8 + p_{10} + p_{12} + p_{16} + p_{18} + p_{22} + p_{23} + p_{25} \leq 0.5\\
&0 \leq p_{23} + p_{25} \leq 0.5\\
&p_2 + p_4 + p_8 + p_{10} + p_{12} + p_{16} + p_{18} + p_{22} + p_{23} + p_{25} \leq 1
\end{aligned}$$

Theorem 5.1 *Let Π be an* ap*-program, $m > 0$ be an integer, and s be a state. Then every solution of CONS is also a solution of CONS' where CONS is one of $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$, $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$, or $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ and CONS' is constructed according to the above construction.*

Proof 5.1 *(i) Suppose σ is a solution to CONS. For any interval constraint*

$$\ell \leq p_{i_1} + \cdots + p_{i_m} \leq u$$

deleting some terms from the summation preserves the upper bound and clearly the summation still is greater than or equal to 0. Hence, σ is a solution to the modified interval constraint in CONS'. For the equality constraint $p_1 + \ldots + p_n = 1$, removing some variables from the summation causes the resulting sum (under the solution σ) to be less than or equal to 1 and hence the corresponding constraint in CONS' is satisfied by σ.

A major problem with the above result is that CONS' is always satisfiable because setting all variables to have value 0 is a solution. The binary algorithm tries to tighten the lower bound in the interval constraints involved so that we have a set of solutions that more closely mirror the original set. It does this by looking at each interval constraint in CONS' and trying to set the lower bound of that constraint first to $\ell/2$ where ℓ is the lower bound of the corresponding constraint in CONS. If the resulting set of constraints is satisfiable, it increases it to $3\ell/4$, otherwise it reduces it to $\ell/4$. This is repeated for different interval constraints until reasonable tightness is achieved. It should be noted that the order in which the constraints are processed is important - different orders can lead to different CONS' being generated. The detailed algorithm is shown in Figure 7. The algorithm is called with $\Pi' = T^{\omega}_{\Pi_s}$, and CONS equal to one of CONS_U, $\mathsf{RedCONS}$, or $\mathsf{S\_RedCONS}$.

The Binary algorithm takes a chance. Rather than use a very crude estimate of the lower bound in the constraints (such as 0, the starting point), it tries to "pull" the lower bounds as close to the original lower bounds as possible in the expectation that the revised linear program is closer in spirit to the original linear program. Here is an example of this process.

Example 5.2 *Consider the following very simple program:*

$a \wedge b : [0.8, 0.9] \leftarrow .$
$a \wedge c : [0.2, 0.3] \leftarrow .$

Let $\mathcal{W} = \{w_0 = \emptyset, w_1 = \{a\}, w_2 = \{b\}, w_3 = \{c\}, w_4 = \{a,b\}, w_5 = \{a,c\}, w_6 = \{b,c\}, w_7 = \{a,b,c\}\}$, but suppose $m = 4$ and we select a sample of four worlds $\mathcal{W}_m = \{w_0, w_2, w_6, w_7\}$. Now, assuming $s = \emptyset$, $\mathsf{CONS}'(\Pi, s, T^{\omega}_{\Pi_s})$ contains the following constraints:

$0 \leq p_7 \leq 0.9$
$0 \leq p_7 \leq 0.3$
$p_0 + p_2 + p_6 + p_7 \leq 1$

which is clearly solvable, but yielding the all-zero solution. The binary heuristic will then modify the first constraint so that its lower bound is 0.4 and, since this new program is unsolvable, will subsequently adjust it to 0.2. At this point, the program is now back to being solvable, and one more iteration leaves the lower bound at $(0.4 + 0.2)/2 = 0.3$, which results once again in a solvable program. At this point, we decide to stop, and the final value of the lower bound is thus 0.3. The algorithm then moves on to the following constraint, and adjusts its lower bound first to 0.1 and then to 0.15, and decides to stop. The final set of constraints is then:

$0.3 \leq p_7 \leq 0.9$
$0.15 \leq p_7 \leq 0.3$
$p_0 + p_2 + p_6 + p_7 \leq 1$

```
algorithm $Binary(\Pi', m, \epsilon, \mathsf{CONS})$ {
 1.     $\mathsf{CONS}'$ = new set of linear constraints;
 2.     $\mathcal{W}_m$ = select a set of $m$ worlds in $\mathcal{W}$;
 3.     for each rule $r_i$ in $\Pi$ {
 4.         let $r_i = F : [\ell, u] \leftarrow body$;
 5.         add $0 \leq \Sigma_{w_i \in \mathcal{W}_m \wedge w_i \mapsto F} p_i \leq u$ to $\mathsf{CONS}'$;
 6.     }
 7.     for each constraint $c_i \in \mathsf{CONS}'$; {
 8.         let $L$ be the lower bound in $c_i$;
 9.         let $L^*$ be $c_i$'s original lower bound in $\mathsf{CONS}$;
10.         while not $done(\mathsf{CONS}', c_i, \epsilon)$ {
11.             $L' = (L^* + L)/2$
12.             let $c_i'$ be constraint $c_i$ with lower bound $L'$;
13.             if $solvable((\mathsf{CONS}' - c_i) \cup c_i')$ {
14.                 $\mathsf{CONS}' = (\mathsf{CONS}' - c_i) \cup c_i'$; $L = L'$
15.             }
16.             else {
17.                 $L^* = L'$;
17.                 $L' = (L' - L)/2$;
18.                 if $solvable((\mathsf{CONS}' - c_i) \cup c_i')$ {
19.                     $\mathsf{CONS}' = (\mathsf{CONS}' - c_i) \cup c_i'$; $L = L'$
20.                 }
21.                 else { $L^* = L'$; }
22.             }
23.         }
24.     }
25.     add $\Sigma_{w_i \in \mathcal{W}_m} p_i \leq 1$ to $\mathsf{CONS}'$;
26.     return $\mathsf{CONS}'$;
27. }
```

Figure 7: The Binary Heuristic Algorithm.

6 Implementation and Experiments

We have implemented several 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 P-MPW algorithm has also been implemented, and is described in more detail below. 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 64 8-core, 8-processor nodes with between 10GB and 20GB of 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, where each rule consisted of an empty body (we assume they are the rules that are relevant in the state, and after computing the fixpoint) and a number of clauses in the head distributed uniformly between 1 and 5. The probability intervals were also generated randomly, making sure that the lower bound was less than or equal to the upper bound. All random number selection were implemented using the random number generator provided by JAVA. The experiments then consisted of the following: (i) generate a new *ap*-program and send it to each of the three algorithms, (ii) vary the number of worlds from 32 to 16,384, performing at least 10 runs for each value and recording the average time taken by each algorithm, and (iii) measure the quality of 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 discussion 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 and HOP and SemiHOP algorithms as the number of worlds increases.

Running time Figure 8 shows the running times for each of the naive, SemiHOP, naive$_{binary}$, and 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 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 1024 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 $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ constraints, the algorithm overhead is much lower than that of the HOP algorithm, and thus yields a more efficient running time.

The reduction in the size of the set of constraints afforded by the binary heuristic algorithm allows us to apply the naive and SemiHOP algorithms to much larger *ap*-programs. In Figure 9, we examine the running times of the naive$_{bin}$ and SemiHOP$_{bin}$ algorithms for large numbers of worlds (up to 2^90 or 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 an 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. Because of this property, experiments running SemiHOP$_{binary}$ on problems with very large *ap*-programs (from 1,000 to 100,000 ground atoms) only take around 300 seconds using a 2% sample rate. However, this aspect of the SemiHOP algorithm can also lead to some anomolous behavior, where the running time will appear to decrease as the number of worlds increases. Figure 10 illustrates this anomaly, as the computation time appears to decrease with very large numbers of worlds. This occurs when we

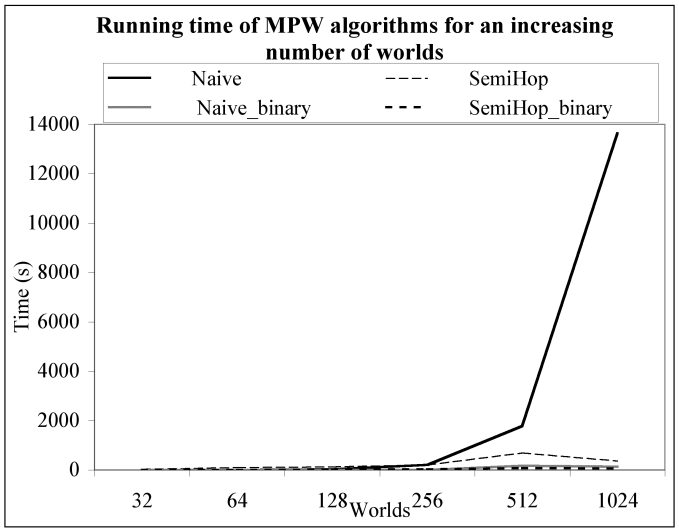

Figure 8: Running time of the algorithms for increasing number of worlds.

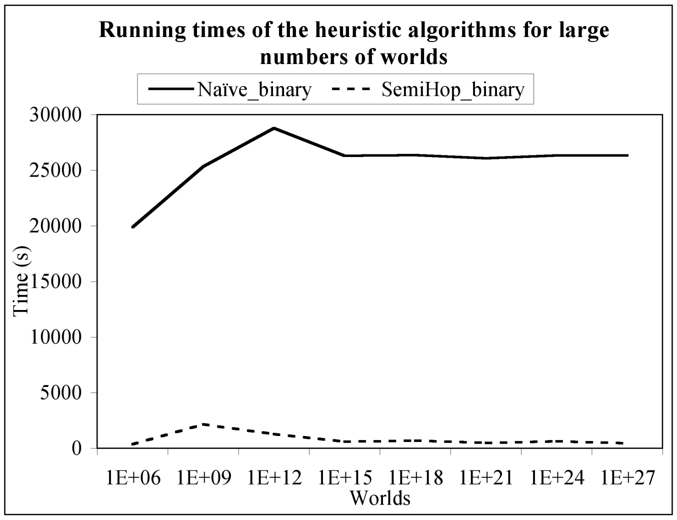

Figure 9: Running time of naive$_{bin}$ and SemiHOP$_{bin}$ for large number of worlds.

have taken a small sample of subpartitions in a problem with very high rule density, and there are no subpartitions with a single satisfying interpretation; as a result, no "most probable world" computations are performed, which obviously leads to a drastic reduction in the running time. Further experimentation is necessary to determine the optimal balance between an efficient running time and a sample large enough to produce meaningful results.

Quality of solution Figure 11 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 10 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

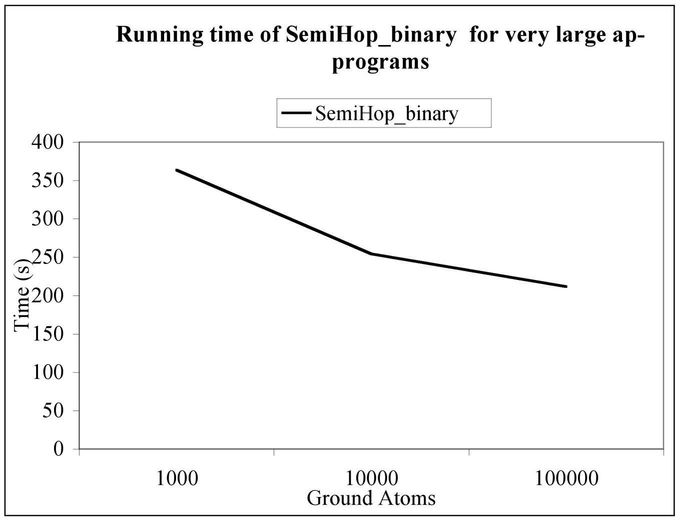

Figure 10: Running time of the SemiHOP$_{binary}$ algorithm for very large numbers of possible worlds.

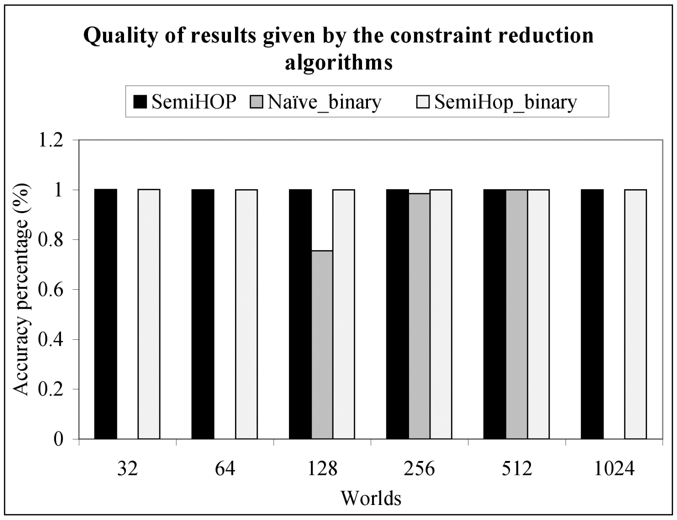

Figure 11: Quality of the solutions produced by SemiHOP, naive$_{bin}$, and SemiHOP$_{bin}$ as compared to Naive.

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.

7 Related Work

Probabilistic logic programming was introduced in [NS91, NS92] and later studied by several authors [NH95, LKI99, LS01, DS97, DPS99]. This work was preceded by earlier—non-probabilistic—

papers on quantitative logic programming of which [vE86] is an example. [NH95] presents a model theory, fixpoint theory, and proof procedure for conditional probabilistic logic programming. [LKI99] combines probabilistic LP with maximum entropy. [Luk98] presents a conditional semantics for probabilistic LPs where each rule is interpreted as specifying the conditional probability of the rule head, given the body. [LS01] develops a semantics for logic programs in which different general axiomatic methods are given to compute probabilities of conjunctions and disjunctions. [DS97] presents an approach to a similar problem. [DPS99] present a well-founded semantics for annotated logic programs and show how to compute this well-founded semantics.

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 or is true in a specific model (e.g., the well-founded model [DPS99]). 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 [NS91]. Our framework modifies gp-programs in three ways: (i) we do not allow extensional predicates to occur in rule heads, while gp-programs do allow them, (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 [NS91] solve the problem of entailment.

8 Conclusions

In this work, we have presented the theory and algorithms of *ap*-programs. *ap*-programs are a variant of probabilistic logic programs and their syntax and semantics is not very different from them. What we have done, however, is to present the following contributions:

1. Dealing with the problem of reducing the size of the linear programs that are generated by *ap*-programs - this problem has not been addressed in the literature for this problem, for any kind of PLPs.

2. Studying the problem of finding the most probable world, given an *ap*-program - this problem has not been addressed in the literature either, for any kind of PLPs.

3. Three algorithms to find the most probable world, along with the Binary heuristic that can be used in conjunction with any of them.

4. Our theory has produced tangible results of use to US military officers [Bha07, Sub07].

5. Our implementation is the only one we are aware of that can work for large numbers of ground atoms with reasonable accuracy and levels of efficiency much superior to past efforts (we could only evaluate accuracy in cases with small numbers of ground atoms).

On the other hand, there are many problems that remain open. First, we need an accurate estimation of the computational complexity of the MPW problem. We have proven NP-hardness results, but were unable to establish membership in NP. A more accurate classification would be desirable. Moreover, it would be desirable to come up with efficient parallel algorithms. Third, it would be nice to get some concrete theoretical results about the accuracy of solutions produced by the binary heuristic. It is possible also that a judicious selection of variables in the binary heuristic may yield better results.

References

[Bha07] Yudhijit Bhattacharjee. Pentagon asks academics for help in understanding its enemies. *Science Magazine*, 316(5824):534–535, 2007.

[DPS99] Carlos Viegas Damasio, Luis Moniz Pereira, and Terrance Swift. Coherent well-founded annotated logic programs. In *Logic Programming and Non-monotonic Reasoning*, pages 262–276, 1999.

[DS97] Alex Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. In *International Conference on Logic Programming*, pages 391–405, 1997.

[FHM90] Ronald Fagin, Joseph Y. Halpern, and Nimrod Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(1/2):78–128, 1990.

[Hai84] T. Hailperin. Probability logic. *Notre Dame Journal of Formal Logic*, 25 (3):198–212, 1984.

[KMN$^+$07a] Samir Khuller, Maria Vanina Martinez, Dana Nau, Gerardo Simari, Amy Sliva, and VS Subrahmanian. Action probabilistic logic programs. *Annals of Mathematics and Artificial Intelligence*, (To Appear), 2007.

[KMN$^+$07b] Samir Khuller, Maria Vanina Martinez, Dana Nau, Gerardo Simari, Amy Sliva, and VS Subrahmanian. Finding most probable worlds of probabilistic logic programs. In *Proceedings of the First International Conference on Scalable Uncertainty Management (SUM 2007)*, volume 4772, pages 45–59. Lecture Notes in Computer Science, Springer-Verlag, 2007.

[LKI99] Thomas Lukasiewicz and Gabriele Kern-Isberner. Probabilistic logic programming under maximum entropy. *Lecture Notes in Computer Science (Proc. ECSQARU-1999)*, 1638, 1999.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming, Second Edition.* Springer-Verlag, 1987.

[LS01] Laks V. S. Lakshmanan and Nematollaah Shiri. A parametric approach to deductive databases with uncertainty. *IEEE Trans. on Knowledge and Data Engineering*, 13(4):554–570, 2001.

[Luk98] Thomas Lukasiewicz. Probabilistic logic programming. In *European Conference on Artificial Intelligence*, pages 388–392, 1998.

[NH95] Liem Ngo and Peter Haddawy. Probabilistic logic programming and bayesian networks. In *Asian Computing Science Conference*, pages 286–300, 1995.

[Nil86] Nils Nilsson. Probabilistic logic. *Artificial Intelligence*, 28:71–87, 1986.

[NS91] Raymond T. Ng and V. S. Subrahmanian. A semantical framework for supporting subjective and conditional probabilities in deductive databases. In Koichi Furukawa, editor, *Proceedings of the Eighth International Conference on Logic Programming*, pages 565–580. The MIT Press, 1991.

[NS92] Raymond T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[SAM$^+$07] V.S. Subrahmanian, M. Albanese, V. Martinez, D. Reforgiato, G.I. Simari, A. Sliva, O. Udrea, and J. Wilkenfeld. CARA: A Cultural Reasoning Architecture. *IEEE Intelligent Systems*, 22(2):12–16, 2007.

[SMSS07] Amy Sliva, Vanina Martinez, Gerardo Ignacio Simari., and VS Subrahmanian. Soma models of the behaviors of stakeholders in the afghan drug economy: A preliminary report. In *First International Conference on Computational Cultural Dynamics (IC-CCD 2007)*. ACM Press (To Appear), 2007.

[SSNS06] Gerardo Simari, Amy Sliva, Dana Nau, and V. S. Subrahmanian. A stochastic language for modelling opponent agents. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 244–246, New York, NY, USA, 2006. ACM.

[Sub07] V. S. Subrahmanian. Cultural Modeling in Real Time. *Science*, 317(5844):1509–1510, 2007.

[vE86] M.H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 4:37–53, 1986.

[WAJ$^+$07] Jonathan Wilkenfeld, Victor Asal, Carter Johnson, Amy Pate, and Mary Michael. The use of violence by ethnopolitical organizations in the middle east. Technical report, National Consortium for the Study of Terrorism and Responses to Terrorism, February 2007.