Ann Math Artif Intell (2007) 51:295–331
DOI 10.1007/s10472-008-9089-2

Computing most probable worlds of action probabilistic logic programs: scalable estimation for $10^{30,000}$ worlds

Samir Khuller · M. Vanina Martinez · Dana Nau · Amy Sliva · Gerardo I. Simari · V. S. Subrahmanian

Published online: 26 February 2008
© Springer Science + Business Media B.V. 2008

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

S. Khuller · M. Vanina Martinez · D. Nau · A. Sliva · G. I. Simari · V. S. Subrahmanian (✉)
Department of Computer Science and University of Maryland Institute for Advanced Computer Studies (UMIACS), University of Maryland College Park, College Park, MD 20742, USA
e-mail: vs@cs.umd.edu

S. Khuller
e-mail: samir@cs.umd.edu

M. Vanina Martinez
e-mail: mvm@cs.umd.edu

D. Nau
e-mail: nau@cs.umd.edu

A. Sliva
e-mail: asliva@cs.umd.edu

G. I. Simari
e-mail: gisimari@cs.umd.edu

"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 $10^{30,000}$ worlds can be handled in a few minutes. Subsequently, we develop parallel versions of these algorithms and show that they provide further speedups.

Keywords Uncertainty · Probabilistic logic programs · Most probable worlds · Scalable approximations

Mathematics Subject Classification (2000) 68T37

1 Introduction

Probabilistic logic programs (PLPs) [17] 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 [6, 7], and [15] to induce a set of probability distributions on a space of possible worlds. Past work on PLPs [16, 17] 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 [1, 18] 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 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 [16, 17] and is not claimed as anything dramatically new. We describe the *most probable world* (MPW) problem by immediately using the linear programming methods of [16, 17]—these methods take exponential compute time in the size of the logic program (but polynomial in the number of worlds—which in turn are exponential in the size of the logic program) 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

[1] Our group has thus far built models of approximately 40 groups around the world including tribes such as the Shinwaris and Waziris, terror groups like Hezbollah, PKK, KDPI, 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.

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. We subsequently present some alternative parallel implementations of our algorithms, as well as an algorithm that can be used to extract *ap*-rules from data automatically. Finally, Section 8 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. The parallel algorithms also exhibit appropriate speedups.

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 [16, 17]. We assume the existence of a logical alphabet that consists of a finite set $\mathcal{L}_{\text{cons}}$ of constant symbols, a finite set $\mathcal{L}_{\text{pred}}$ 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 [10]. We assume that a subset $\mathcal{L}_{\text{act}}$ of $\mathcal{L}_{\text{pred}}$ are designated as *action symbols*—these are symbols that denote some action. Thus, an atom $p(t_1, \ldots, t_n)$, where $p \in \mathcal{L}_{\text{act}}$, 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 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. (1) 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 $[p - e, p + e]$. (2) As shown by [6, 17], if we do not know the relationship between events e_1, e_2, even if we know point probabilities for e_1, e_2, we can only infer an interval for the conjunction and disjunction of e_1, e_2. (3) Interval probabilities generalize point probabilities anyway, so our work is also relevant to point probabilities.

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 $F : \mu$, $Body^{\text{act}}(c)$ denotes $A_1 : \mu_1 \wedge A_2 : \mu_2 \wedge ... \wedge A_m : \mu_m$, and $Body^{\text{state}}(c)$ 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.

Figure 1 shows a small rule base consisting of some rules we have derived automatically about Hezbollah using behavioral data from [22]. The behavioral data in [22] has tracked over 300 terrorist groups for about 25 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, as discussed in Section 7. We show four rules we have extracted for the group Hezbollah in Fig. 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.

Example 1 Consider the *ap*-program from Fig. 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.	*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).*

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

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

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 $\{F : \mu \leftarrow Body^{\text{act}} \mid s \text{ satisfies } Body^{\text{state}} \text{ and } F : \mu \leftarrow Body^{\text{act}} \wedge Body^{\text{state}} \text{ is a ground instance of a rule in } \Pi\}$.

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 Let Π be the *ap*-program from Fig. 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 [16, 17] are that (1) *ap*-programs have a bipartite structure (action atoms and state atoms) and (2) they allow arbitrary formulas (including ones with negation) in rule heads ([16, 17] do not). They can easily be extended to include variable annotations and annotation terms as in [16]. Likewise, as in [16], 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* [16, 17] *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 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 do not 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 3 Consider the simple program depicted in Fig. 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 [16]. We use $U_{\Pi_s}(X)$ to set up a linear program $\mathsf{CONS}_U(\Pi, s, X)$ as follows.

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

Fig. 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

Definition 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.

These constraints are similar to those of [6, 7]. The following is an example of how these constraints look.

Example 4 Let Π be the *ap*-program from Fig. 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 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

$$\{F : [\ell(F), u(F)] \mid (\exists \mu)\ F : \mu \in U_{\Pi_s}(X)\} \cup$$

$$\{A : [\ell(A), u(A)] \mid A \textit{ is a ground action atom}\}.$$

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 5 Consider the *ap*-program presented in Fig. 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:

$$U_{\Pi_s}(X) = \{d : [0.52, 0.82], b \wedge a : [0.55, 0.69]\}, \text{ and}$$

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

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.

A frequent critique of the approach presented thus far is that the lower bound and upper bounds computed for each world by minimizing and maximizing the probability of a world subject to the constraints associated with the *ap*-program can be very wide. This is true, but it reflects the fact that the linear program set up earlier reflects *complete lack of knowledge* about the dependencies between the events mentioned in the *ap*-program. For instance, if we additionally know that action atom a and b are independent, then we can expand our set of constraints to include this by inserting the constraint:

$$(\Sigma_{w_i \models a} p_i) \times (\Sigma_{w_j \models b} p_j) = \Sigma_{w_r \models a \wedge b} p_r.$$

Of course, we note that such constraints may be nonlinear.

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 [16] 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, 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 combinations of actions a group might take in a given situation.

Definition 9 (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.

In this section, we include 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 probabilistic logic program (PLP-CONS) whose proof is given below.

Proposition 2 *The problem of deciding if a probabilistic logic program* Π *is consistent in a state s is NP-hard.*

Proof We will perform a reduction of the *boolean formula satisfiability* problem (SAT) to checking consistency of Π with respect to a given state s (PLP-CONS). In order to perform the reduction, we must define a polynomial time computable function R that maps an arbitrary boolean formula f into an instance of PLP-CONS such that f is satisfiable if and only if $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ is solvable (note that $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ is solvable if and only if Π is consistent in state s; this well known property was proved in [17]). The PLP-CONS instance will correspond to a simplified version of the problem, in which only one rule is present, and the upper and lower probabilities are equal to 1. Define:

$$R(f) = \{f : [1, 1] \leftarrow\}$$

to be the PLP built from an arbitrary ground formula f. It is clear that this transformation can be performed in polynomial time. We will now prove that f is satisfiable if and only if Π is consistent with respect to the empty state, (*i.e.*, $\mathsf{CONS}_U(R(f), \emptyset, T^{\omega}_{\Pi_s})$ is solvable):

– f is satisfiable $\Rightarrow$ $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ is solvable: By hypothesis, there exists an assignment of variables in f such that f is true. We use such values to build a world w such that formula f is satisfied by w. Because the upper and lower probabilities in the rule are both 1, we can assign 1 to p_w, the probability of

world w, whereas every other world receives probability 0. We have therefore constructed a solution to the constraints that proves that Π is consistent in s.
– $\mathsf{CONS}(\Pi, s, T^{\omega}_{\Pi_s})$ is solvable $\Rightarrow$ f is satisfiable: By hypothesis, there exists a solution to $\mathsf{CONS}(\Pi, s, T^{\omega}_{\Pi_s})$ that assigns a probability p_{w_i} to each world that satisfies f. Because the set of worlds satisfying f is nonempty, this means it is possible to find at least one assignment for the variables in f such that f is satisfied.

We have therefore shown that SAT is polynomial time reducible to checking consistency in PLP, and therefore this problem is NP-hard. □

Proposition 3 (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 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 above, 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 $s' \Rightarrow \Pi$ 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.

Note that, whenever Π is inconsistent, the value of $low(w)$ is undefined, for any possible world w. 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. □

It is easy to show that, for ground (or propositional) *ap*-programs, BOUNDED-LOW is in EXPTIME as long as we assume that we are only interested in rule heads with at most k atoms in them for some arbitrary, but fixed constant k. Intuitively, this requirement ensures that we do not try to find arbitrarily long (unboundedly long) combinations of ground atoms.

Proposition 4 *Suppose* Π *is a propositional* (*or ground*) ap*-program, and suppose* T_Π *only considers formulas F containing k atoms or less in them. Then, BOUNDED-LOW is in EXPTIME.*

Proof The result follows directly from the following observations. First, we can compute $T^{\omega}_{\Pi_s}$ in exponential time as follows. Computing the $U_{\Pi}()$ operator is clearly polynomial. $\mathsf{CONS}_U(\Pi, s, X)$ takes exponential time to set up because it has an exponential number of variables. As a consequence, solving $\mathsf{CONS}_U(\Pi, s, X)$ takes exponential time in the size of the input (because solving linear programs is polynomial in the size of the linear program). The fixpoint operator terminates after a maximum of an exponential number of iterations because the only linear programs to be solved are those associated with sets of heads of *ap*-rules, and each such linear program is solved at most once per clause. The problem is therefore in EXPTIME. □

Throughout the rest of this paper, we assume that the restrictions in the above proposition applies to all complexity results.

Proposition 5 (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 This proof is very similar to the one for the *NP*-hardness of BOUNDED-LOW. As before, we will reduce the PLP-CONS problem to the problem of deciding if a certain world w is such that $up(w) \leq p_{th}$ for a certain probability value p_{th}. Because PLP-CONS was proven to be *NP*-hard above, this reduction will prove that BOUNDED-UP 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-UP, 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 $p_{th} = 1$. We must now show that this transformation yields a reduction by proving that Π is consistent in state s if and only if $up(w) \leq 1$ with respect to Π' and state s':

- Π is consistent $\Rightarrow up(w) \leq 1$ 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 less than or equal to one.
- $up(w) \leq 1$ with respect to Π' in state $s' \Rightarrow \Pi$ is consistent: If $up(w) \leq 0$ with respect to Π' in state s', this means that w has received a probability value less than or equal to one, 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.

Note that, whenever Π is inconsistent, the value of $up(w)$ is undefined, for any possible world w. To complete the proof, we note that the transformation from a PLP-CONS instance to a BOUNDED-UP instance can be done in polynomial time with respect to the size of the *ap*-program given for PLP-CONS. □

Proposition 6 *BOUNDED-UP is in EXPTIME.*

Proof Analogous to the proof of Proposition 4 □

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.[2]

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 Fig. 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 [17]) 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 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

[2] 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 $Best$.

Fig. 3 The naive algorithm for finding a most probable world

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 10 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 6.

Example 6 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.

Proposition 7 *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*

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

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)*.*

Proof 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 7 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 Fig. 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.

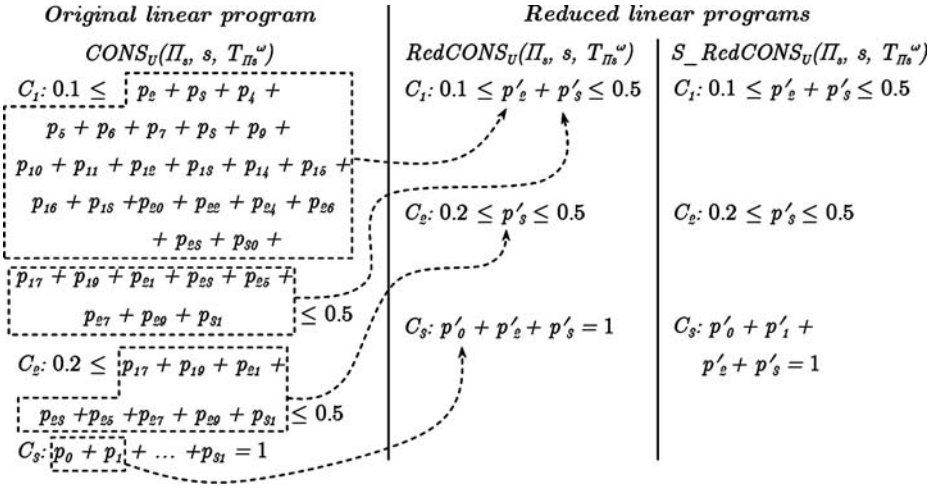

Fig. 4 Reducing $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ by grouping variables. The new LPs are called $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ and $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$, as presented in Definitions 11 and 13

Effectively, what we have done is to modify the number of variables in the linear program from $2^{\mathrm{card}(\mathcal{L}_{\mathrm{act}})}$ to $2^{\mathrm{card}(\Pi_s)}$—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 11 ($\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$) 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 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 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 $low(w_i) = 0$. □

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 $best$.

Fig. 5 The head-oriented processing (HOP) algorithm

Going back to Example 6, 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 (Fig. 5) which works as follows. First we present some simple notation. Let $FixedWff(\Pi, s) = \{F \mid F : \mu \in U_{\Pi_s}(T^{\omega}_{\Pi_s})\}$. 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 $X_1 \sim X_2$ if and only if $Formula(X_1, \Pi, s)$ and $Formula(X_2, \Pi, s)$ are logically equivalent.

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.*

Proof 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 $low(w_i) > low(w_{\text{sol}})$. Suppose HOP does return w_{sol} but that there is a world w_i such that $low(w_i) > low(w_{\text{sol}})$. Clearly, $[w_i]$ and $[w_{\text{sol}}]$ 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 $low(w_{\text{sol}}) \leq low(w_i)$, yielding a contradiction.
- *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 $low(w_{\text{sol}}) = low(w_{\max})$. 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 $\bigwedge_{F\in X} F \wedge \bigwedge_{G\in\bar{X}} \neg G \wedge \left(\bigvee_{a\in w} \neg a \vee \bigvee_{a'\in\bar{w}} a'\right)$ 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 7.

Example 8 Consider the program from Example 7, and suppose $X = \{(a \vee b \vee c \vee d \vee e), (a \wedge e)\}$. In Step (3$a$), 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 $2^{\mathrm{card}(\Pi_s)}$, 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 $2^{2*\mathrm{card}(\Pi_s)}$. Lastly, HOP avoids solving the reduced linear program for all the non-singleton equivalence classes (for instance, in Example 8, 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 8 *Suppose* Π *is an* ap*-program,* s *is a state, and* X *is a subset of* $FixedWff(\Pi, s)$. *Then there exists a world* w_i *such that* $\{w \mid w \mapsto Formula(X, \Pi, s)\} \subseteq [w_i]$.

Proof Immediate from Definition 10. □

We now define the concept of a subpartition.

Definition 12 A *subpartition* 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 $FixedWff(\Pi, s)$.

Proposition 9 *Suppose* Π *is an* ap*-program,* s *is a state, and* $\{X_1, \ldots, X_k\}$ *is the power set of* $FixedWff(\Pi, s)$. *Then the partition* $W_1, \ldots, W_k$ *where* $W_i = \{w \mid w \mapsto Formula(X_i, \Pi, s)\}$ *is a subpartition of the set of worlds of* Π *w.r.t.* s.

Proof Immediate from Proposition 8. □

The intuition behind the SemiHOP algorithm is best presented by going back to constraints (1) and (2) given in Example 6. 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.

Definition 13 ($\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$) 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 $p^{\star}_i$ to denote the summation of the probability of each of the worlds in W_i. For each *ap*-wff $F : [\ell, u]$ in $T^{\omega}_{\Pi_s}$, $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ 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, $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ contains the constraint

$$\Sigma_{W_i} p^{\star}_i = 1$$

Example 9 Returning to Example 6, $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ 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 Fig. 6. The following theorem ensures the correctness of this algorithm.

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 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 $best$.

Fig. 6 The SemiHOP algorithm

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.*

Proof The proof is completely analogous to that of Theorem 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 bigger than the size of the set $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$. It is hard to quantify how much bigger $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$ is w.r.t. $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$—in general, the more logically equivalent rule heads we have, the more unnecessary variables will be included in $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$.

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 $\mathcal{V}_{\text{Naive}}, \mathcal{V}_{\mathsf{HOP}}$, 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 $\mathcal{V}^k_{\text{Naive}}, \mathcal{V}^k_{\text{HOP}}, \mathcal{V}^k_{\text{SemiHOP}}$ be some subset of k variables from each of these sets, respectively. Let CONS be one of $\text{CONS}_U(\Pi, s, T^\omega_{\Pi_s})$, $\text{RedCONS}_U(\Pi, s, T^\omega_{\Pi_s})$, or $\text{S\_RedCONS}_U(\Pi, s, T^\omega_{\Pi_s})$. 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 "≤".

Example 10 Consider the program from Example 7, 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, $\text{CONS}'(\Pi, s, T^\omega_{\Pi_s})$ contains the following constraints:

$$0 \leq p_2 + p_4 + p_8 + p_{10} + p_{12} + p_{16} + p_{18} + p_{22} + p_{23} + p_{25} \leq 0.5$$

$$0 \leq p_{23} + p_{25} \leq 0.5$$

$$p_2 + p_4 + p_8 + p_{10} + p_{12} + p_{16} + p_{18} + p_{22} + p_{23} + p_{25} \leq 1$$

Theorem 4 *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* $\text{CONS}_U(\Pi, s, T^\omega_{\Pi_s})$, $\text{RedCONS}_U(\Pi, s, T^\omega_{\Pi_s})$, *or* $\text{S\_RedCONS}_U(\Pi, s, T^\omega_{\Pi_s})$ *and* CONS′ *is constructed according to the above construction.*

Proof 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 Fig. 7. The algorithm is called with $\Pi' = T^{\omega}_{\Pi_s}$, and CONS equal to one of CONS_U, RedCONS, or 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 11 Consider the following very simple program:

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

```
algorithm Binary(Π', m, ε, CONS){
 1.      CONS' = new set of linear constraints;
 2.      W_m = select a set of m worlds in W;
 3.      for each rule r_i in Π {
 4.          let r_i = F : [ℓ, u] ← body;
 5.          add 0 ≤ Σ_{w_i ∈ W_m ∧ w_i ↦ F} p_i ≤ u to CONS';
 6.      }
 7.      for each constraint c_i ∈ CONS'; {
 8.          let L be the lower bound in c_i;
 9.          let L* be c_i's original lower bound in CONS;
10.          while not done(CONS', c_i, ε) {
11.              L' = (L* + L)/2
12.              let c'_i be constraint c_i with lower bound L';
13.              if solvable((CONS' − c_i) ∪ c'_i) {
14.                  CONS' = (CONS' − c_i) ∪ c'_i; L = L'
15.              }
16.              else {
17.                  L* = L';
17.                  L' = (L' − L)/2;
18.                  if solvable((CONS' − c_i) ∪ c'_i) {
19.                      CONS' = (CONS' − c_i) ∪ c'_i; L = L'
20.                  }
21.                  else { L* = L'; }
22.              }
23.          }
24.      }
25.      add Σ_{w_i ∈ W_m} p_i ≤ 1 to CONS';
26.      return CONS';
27. }
```

Fig. 7 The binary heuristic algorithm

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:

$$0.3 \leq p_7 \leq 0.9$$

$$0.15 \leq p_7 \leq 0.3$$

$$p_0 + p_2 + p_6 + p_7 \leq 1$$

In general, in the experiments conducted in this paper, we construct $CONS'$ by randomly sampling worlds. It would be interesting to see if there are techniques which would use non-random sampling and produce higher accuracy within a fixed computation time window. This will be the subject of some of our future work.

6 Towards parallel algorithms

In the previous sections we have described several algorithms that can be used to solve the maximally probable worlds problem. However, even with the given simplifications and heuristic approximation algorithms, the computation time and memory requirements do not always allow us to achieve the desired level of performance. In this section, we will present various parallel versions of the the sequential algorithms presented earlier. Parallelism will not only reduce the computation time of the algorithms for finding the most probable worlds, but will also allow us to examine a greater number of worlds, permitting analysis of larger *ap*-programs, and possibly improving the accuracy of the end result.

6.1 Parallelism for reducing computation time

All of our algorithms given above lend themselves to being parallelized in a straightforward way. This new class of algorithms, the P-MPW (Parallel Maximally Probable World) algorithms, operate identically to the serial algorithms, except that the computation of $\mathsf{low}(w_i)$ or $\mathsf{up}(w_i)$ for all of the worlds w_i is distributed among n nodes of a computing cluster such that m worlds at a time are given to each node. Figure 8 contains the basic P-MPW algorithm in pseudo code.

```
Algorithm P − MPW(Π, s, T^ω_{Π_s}, CONS, m, n){
1.      remainingVars = divideVariables(CONS, m);
2.      while remainingVars not empty {
3.          for each node {
4.          v_j = pop(remainingVars);
5.              for each variable p_i in v_j {
6.                  VAL(p_i) = minimize p_i subject to CONS;
7.                  MAX_j = max VAL(p_i);
8.              }
9.          }
10.     }
11.     BestVal = max MAX_j;
12.     w_k = arg max MAX_j;
13.     return w_k;
}
```

Fig. 8 The P-MPW algorithm. This is the general parallel algorithm for computing the most probable worlds. The variable *remainingVars* is a stack containing the j sets of variables of size m; *divideVariables* is a procedure that performs the division of CONS into these sets

The number m of worlds for which to compute the *low* or *up* values can be determined in several ways. The most obvious is to simply divide the problem evenly across all of the n nodes such that $m = \frac{|\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})|}{n}$ where $|\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})|$ is the number of variables in $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$.

The CONS parameter of the P-MPW algorithm can be either $\mathsf{CONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$, $\mathsf{RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$, $\mathsf{S\_RedCONS}_U(\Pi, s, T^{\omega}_{\Pi_s})$, or the CONS' returned by the binary heuristic. The basic P-MPW and the PAMPW algorithms allow for, in the best case, a computation time improvement of up to a factor of n, where n is the number of nodes in the cluster.

6.2 Parallelism for improving solution accuracy of heuristics

We can also utilize parallel algorithms to improve the quality of the final solutions. In this section we will describe explicitly parallel algorithms that are able to take into account additional samples of worlds for the heuristic approximations, propagating the most probable worlds from each sample throughout the successive computations and allowing more thorough comparisons between the most probable worlds found by each iteration of the parallel computation.

The first algorithm, PAMPW-MS (parallel approximation of the maximally probable worlds—multi-sample), allows us to examine a greater proportion of the possible worlds in computing the most probable world. With this method, each parallel computation investigates a distinct sample of possible worlds for the binary heuristic constraint selection algorithm; the resulting most probable worlds from each sample are then compared to find the most probable world overall. Using the PAMPW-MS algorithm we are able to look at larger samples of the possible worlds and thereby have a better chance of finding an approximate solution that is more accurate with

```
Algorithm PAMPW − MS(Π, s, T^ω_{Π_s}, m, n, ϵ, r, CONS){
1.      for each node {
2.          CONS_{n_j} = Binary(T^ω_{Π_s}, r, ϵ, CONS)
3.          for each variable p_i in CONS_{n_j} {
4.              VAL(p_i) = minimize p_i subject to CONS_{n_j}
5.              MAX_{n_j} = max VAL(p_i)
6.          }
7.      }
8.      BestVal = max MAX_{n_j}
9.      w_k = arg max MAX_{n_j}
10.      return w_k
}
```

Fig. 9 The PAMPW-MS algorithm. This is a parallel algorithm for computing an approximation of the most probable world using the binary heuristic. On each node, the binary heuristic algorithm is used to select a different reduced set of constraints containing *r* variables. The most probable world is then that with the maximum low probability across all of the nodes

respect to the solutions returned by the naive, HOP or SemiHOP algorithms. Figure 9 contains pseudo code for the PAMPW-MS algorithm.

Using PAMPW-MS we can further generalize the functionality of the algorithm to find the *k* most probable worlds from each node and compare these sets of worlds to find the *k* worlds that are the most probable overall.

To further increase our ability to examine a larger sample of the possible worlds, we have also developed an iterative version of the PAMPW-MS algorithm, called the iPAMPW-MS. In iPAMPW-MS, we first compute the *k* most probable worlds on each node of the cluster as described in PAMPW-MS. Then, we again generate a set of constraints for each node, propagating the *k* most probable worlds from the first iteration into the second sample set. For example, if our world selection method selects 1,000 worlds and *k* is chosen to be 20, then in the second iteration we only select 980 worlds and automatically include the 20 most probable worlds from previous computation. Using this new set of constraints, the *k* most probable worlds are again computed and propagated into the next iteration. We continue this process until we have completed *I* iterations of the algorithm, choosing the final *k* most probable worlds from the last sets of *k* obtained across all processors. This iterative process allows us to progressively refine the solution set of the *k* most probably worlds, improving the accuracy of the approximation heuristic algorithms. The steps in the iPAMPW-MS algorithm are given in the pseudo code in Fig. 10.

6.3 Parallelism for increasing computation capacity

Last, but not least, rather than simply distributing the MPW algorithms and performing the same computation in a shorter amount of time, we can also design a "pleasantly parallel" algorithm for finding the most probable world of larger *ap*-programs, i.e. programs with a larger number of ground atoms. To accomplish this task, we must be able to distribute the set of constraints among nodes in a

```
Algorithm iPAMPW − MS(Π, s, T^ω_{Π_s}, m, n, ϵ, r, i, k, CONS){
1.      for iteration=1 to iteration=i {
2.          for each node {
3.              prevWorlds_{n_j,i} = ∅
4.              CONS_{n_j,i} = modConstraints(prevWorlds_{n_j,i},
                  Binary(T^ω_{Π_s}, r, ϵ, CONS))
5.              for each variable p_i in CONS_{n_j,i} {
6.                  VAL(p_i) = minimize p_i subject to CONS_{n_j,i}
7.                  MAX_{n_j,i} = max_k VAL(p_i)
8.                  prevWorlds_{n_j,i} = k arg max MAX_{n_j,i}
9.              }
10.         }
11.     }
12.     BestVal = max MAX_{n_j,i}
13.     w = arg max MAX_{n_j,i}
14.     return w
}
```

Fig. 10 The iPAMPW-MS algorithm. This algorithm solves multiple iterations of the PAMPW-MS, using the *modConstraints* function to perform the Binary heuristic, replacing *k* of the *r* constraints with the variables in the set prevWorlds. The most probable world is then that with the maximum low probability across all of the nodes and iterations

cluster, rather than simply dividing the task of computing *low* or *up* for each of the worlds w_i.

An *ap*-program can be represented as a graph in which the vertices are literals in the program and an edge indicates that its two endpoints occur together in a *ap*-rule.

Definition 14 (Literal Relationship (LR) Graph) Let Π be a ground *ap*-program. The *literal relationship graph* $G^\Pi = (V, E)$ is an *undirected* graph defined as follows.
$V = \{l \mid l$ is a literal (positive or negative) appearing in a rule in $\Pi\}$.
$E = \{(l_i, l_j) \mid l_i, l_j \in V$ and l_i and l_j are either complementary literals or they both appear in a rule in $\Pi\}$.
We use $G^\Pi = (V, E)$ to denote the *Literal Relationship Graph* for the program Π.

Consider the simple *ap*-program Π:

$$\begin{array}{ll} (a \vee b) & : [0.7, 1] \quad \leftarrow . \\ ((a \wedge b) \vee (b \wedge c)) & : [0.2, 0.6] \leftarrow . \\ (a) & : [0.4, 0.4] \leftarrow . \end{array}$$

Figure 11 shows the LR-graph associated with Π.

The *rank* of an LR-graph is the maximum cardinality of the connected components of the graph.

Definition 15 (Rank of an LR-Graph) Let Π be a ground *ap*-program, and $G^\Pi = (V, E)$ be the LR-Graph for Π. We say that graph G^Π has *rank k*, if *k* is the maximum cardinality of any connected component in G^Π.

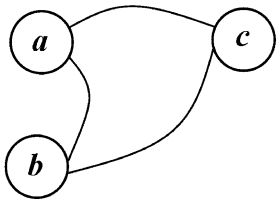

Fig. 11 The literal-relationship graph G^{Π} for a simple *ap*-program Π

For example, when the rank of the LR-graph G^{Π} is 1, this means that all rules in Π are only literals, and there are no complementary literals. Note that the graph in our example has rank 3. On the other hand, if we deleted the second probabilistic statement from the program Π for Fig. 11, we would have a graph of rank 2. Note that we can compute the rank of an LR-graph in polynomial time (w.r.t. the size of the graph), and we can also compute the LR-graph itself in polynomial time. As a consequence, checking if the LR-graph's rank is below some *a priori* set bound b is a polynomial-time operation.

Each connected component c in an LR-graph G^{Π} represents a subprogram Π_c of Π that utilizes only the literals in that component. Therefore, each connected component comprises a separate set of linear constraints $\mathsf{CONS}(\Pi_c, s, T^{\omega}_{\Pi_s})$, $\mathsf{RedCONS}_U(\Pi_c, s, T^{\omega}_{\Pi_s})$, $\mathsf{S\_RedCONS}_U(\Pi_c, s, T^{\omega}_{\Pi_s})$, or $CONS'$. By finding the maximally probable world in each component, we can compare these individual solutions and find the maximally probable world across all components of the original set of constraints for Π. The PAMPW-LR algorithm uses this methodology to divide a much larger *ap*-program into smaller pieces that can be computed in parallel.

```
Algorithm PAMPW − LR(Π, s, T^ω_{Π_s}, m, n, ϵ, r, CONS){
1.      G = buildLR(Π)
2.      components = getComponents(G)
3.      while components not empty {
4.          for each node {
5.          c_j = pop(components)
6.          CONS_{c_j} = Binary(c_j, s, r, ϵ, CONS)
7.              for each variable p_i in CONS_{c_j} {
8.                  VAL(p_i) = minimize p_i subject to CONS_{c_j}
9.                  MAX_{c_j} = max VAL(p_i)
10.             }
11.         }
12.     }
13.     BestVal = max MAX_{c_j}
14.     w_k = arg max MAX_{c_j}
15.     return w_k
}
```

Fig. 12 The PAMPW-LR Algorithm. This is a parallel algorithm for computing an approximation of the most probable world. The variable *components* is a stack containing the *j* connected components in G^{Π}; *getComponents* is a procedure that returns these connected components

On a cluster with n nodes, the PAMPW-LR algorithm assigns a connected component of G^{Π} to each node and then computes the most probable world of each component. The algorithm will then aggregate the results and return the most probable world overall. While the PAMPW-LR approach does not provide any time savings with regards to the computation time, it does allow the analysis of much larger *ap*-programs that can be divided into computationally feasible parallel components. Figure 12 contains pseudo code for the PAMPW-LR algorithm.

7 Applications of *ap*-programs

We have developed approximately 40 *ap*-programs in our lab to date that are carefully constructed models of 40 different groups from around the world. The groups modeled include tribes (e.g. Shinwari, Waziri, Mohmand tribes in along the Pakistan/Afghanistan border), several terrorist groups (e.g. Hezbollah, Fatah Revolutionary Council—Abu Nidal Organization, the Kurdish group PKK and others), as well as political parties (e.g. Jamaat-i-Ulema Islami, Pakistan People's Party). For each of these groups, we identified a small set of actions that the group has taken in the past. For each such action, we tried to find conditions that are good predictors of when those groups would take those actions, and when they would not. These led to rules in the *ap*-program syntax.

The rules themselves have been developed in three ways: by manually having students (and in the case of about 20 groups, terrorism experts) code them, and by automatically extracting them from certain data sets. We started with the manual coding strategy and later transitioned to the use of an automatic extractor that works on a specialized data set called the "Minorities at Risk Organizational Behavior" data set [22]. This data set has identified around 150 parameters to monitor for about 300 groups around the world that are either involved in terrorism or are at risk of becoming full-fledged terrorist organizations. The 150 attributes describe aspects of these groups, such as whether or not the group engaged in violent attacks, if financial or military support was received from foreign governments, and the type of leadership the group has. The data was easy to divide into outcome conditions—or actions that could be taken by the group (i.e. bombings, kidnappings, armed attacks, etc.)—and environmental conditions (i.e. the type of leadership, the kind and amount of foreign support, whether the group has a military wing, etc.). Values for these 150 parameters are available for up to 20 years per group, though it is less for some groups (e.g. groups that have been around for a shorter duration). For each group, MAROB provides a table whose columns correspond to the 150 parameters and the rows correspond to the years.

The automated extraction has been applied thus far to about 15 groups (such as Hezbollah, FRC-ANO, PKK, Baath Party, Kurdistan Democratic Party of Iran). The automatic *ap*-program extraction (APEX) algorithm requires that we assume that the MAROB columns can be divided into action parameters—those attributes which will form the heads of the *ap*-rules—and environmental parameters—those attributes that will appear in the body of the rules. The APEX algorithm for extracting *ap*-rules consists of three main steps:

1. Select an action condition (an action parameter with an instantiated value) to be the head of the rule,

2. Fix one environmental condition as part of the body of the rule,
3. Add varying combinations of the remaining environmental conditions to the body to determine if significant correlations exist between the body conditions and the outcome condition.

We then use the standard measurements of support and confidence from the literature.

Definition 16 (Support) For an action condition A, an environmental condition E, and a database DB, the *support* is defined as:

$$S_{A,E} = \frac{|t \; s.t. \; t \in DB \wedge (A = true \wedge E = true)|}{|DB|}.$$

Definition 17 (Confidence) For an action condition A, an environmental condition E, and a database DB, the *confidence* is defined as:

$$C_{A,E} = \frac{|\{t \; s.t. \; t \in DB \wedge (A = true \wedge E = true)\}|}{|\{t \; s.t. \; t \in DB \wedge E = true\}|}.$$

The APEX algorithm calculates the difference between the confidence value produced by an environmental condition and by its negation. If this difference is above a given threshold, then an *ap*-rule is extracted. To obtain the probability range for the extracted rule, we use the confidence value initially obtained, plus/minus the standard deviation σ of the values involved in its calculation.

```
Algorithm APEX(DB, AC, EC, k, t) {
1.      set Rules = ∅;
2.      for each action a_i ∈ AC {
3.          set head = a_i;
1.          for each condition e_j ∈ EC {
4.              set fixed_condition = e_j;
5.              for each combination, varied_condition, of
                        1, 2, ..., and k of remaining conditions
                        v_1, v_2, ..., v_k ∈ EC
6.                  set body = fixed_condition ∧ varied_condition;
7.                  compute PosConf = C_{head,body};
8.                  set body = fixed_condition ∧ ¬varied_condition;
9.                  compute NegConf = C_{head,body};
10.                  set prob = |PosConf − NegConf|;
11.                  if prob >= t
12.                      add (head : [prob − σ, prob + σ] ← body) to Rules
13.              }
14.          }
15.      }
16.      Return Rules;
}
```

Fig. 13 The APEX algorithm

The complete APEX Algorithm for a database DB with a set of action conditions AC, environmental conditions EC, and confidence difference threshold t is summarized in Fig. 13. Note that this algorithm is not a novel one, and simply performs calculations to capture interesting variations in the data in order to build rules.

For instance, we have extracted approximately 14,000 *ap*-rules for Hezbollah. Some examples of the *ap*-rules extracted from the data for Hezbollah are given in Fig. 14.

1. $(ARMATTACK = 1) : [0.01, 0.79] \leftarrow$
   $(ORGSUCIMPL = 1) \wedge (STATEVIOLENCE = 1) \wedge$
   $(AUTHORG = 0) \wedge (ORGST4 = 1)$
   Armed attacks are carried out if Lebanon has not come to agreement with Hezbollah, the state is not using lethal violence against Hezbollah, Lebanon is not authoritarian, and Hezbollah solicits external support only as a minor/infrequent strategy.

2. $(DSECGOV = 1) : [0.16, 0.84] \leftarrow$
   $(ORGLOC = 1) \wedge (DIASUP = 0) \wedge$
   $(INTERORGCON = 1) \wedge (MILITIAFORM = 2)$
   Domestic government/state lives and security are targets of terrorism if Hezbollah is in Lebanon, Hezbollah has not received support from the Lebanese diaspora, there is inter-organizational conflict, and Hezbollah has a standing military wing.

3. $(KIDNAP = 1) : [0.34, 1.0] \leftarrow$
   $(ORGLOC = 1) \wedge (ORGDOMGOALS = 2) \wedge (ORGST4 = 1)$
   Hezbollah carries out kidnappings if Hezbollah is located in Lebanon, the major goal of Hezbollah is focused on creating or increasing remedial policies, and Hezbollah solicits external support only as a minor/infrequent strategy.

4. $(TLETHCIV = 1) : [0.13, 1.0] \leftarrow (ORGLOC = 1) \wedge (ORGST3 = 1)$
   Transnational targets of terrorism are chosen based on ethnicity/ascriptive feature(s) of individuals if Hezbollah is in Lebanon and Hezbollah uses electoral politics only as a minor/infrequent strategy.

5. $(TTSECGOV = 1) : [0, 0.68] \leftarrow$
   $(ORGCULTGR = 0) \wedge (INTERORGCON = 1) \wedge (DIASUP = 0)$
   Transnational government/state lives and security are targets of terrorism if Hezbollah expresses no cultural grievances, there is inter-organizational conflict, and Hezbollah has not received support from the Lebanese diaspora

Fig. 14 A sample of the rules extracted by APEX from the Hezbollah dataset. The atoms in the rules are represented as a variable and its value. The English translation of each rule is also provided

8 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 $\text{naive}_{\text{bin}}$, $\mathsf{HOP}_{\text{bin}}$, and $\mathsf{SemiHOP}_{\text{bin}}$ algorithms respectively. Our experiments were performed on a Linux computing cluster comprised of 64 8-core, 8-processor nodes with between 10 and 20 GB 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: (1) generate a new *ap*-program and send it to each of the three algorithms, (2) vary the number of worlds from 32 to 16,384, performing at least ten runs for each value and recording the average time taken by each algorithm, and (3) 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 $ruledensity = \frac{\mathcal{L}_{\text{act}}}{card(T^{\omega}_{\Pi_s})}$ 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 15 shows the running times for each of the naive, SemiHOP, $\text{naive}_{\text{binary}}$, and $\mathsf{SemiHOP}_{\text{binary}}$ 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 constraint sets. With a sample size of 25%, $\text{naive}_{\text{binary}}$ and $\mathsf{SemiHOP}_{\text{binary}}$ take only about 132.6 and 58.19 s for instances with 1,024 worlds, whereas the naive algorithm requires almost 4 h (13,636.23 s). 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 s, 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, $\mathsf{HOP}_{\text{bin}}$ still requires a prohibitively large amount of time.

At low rule densities, SemiHOP runs slower than the naive algorithm; with ten rules, SemiHOP uses 18.75 and 122.44 s for 128 worlds, while the naive

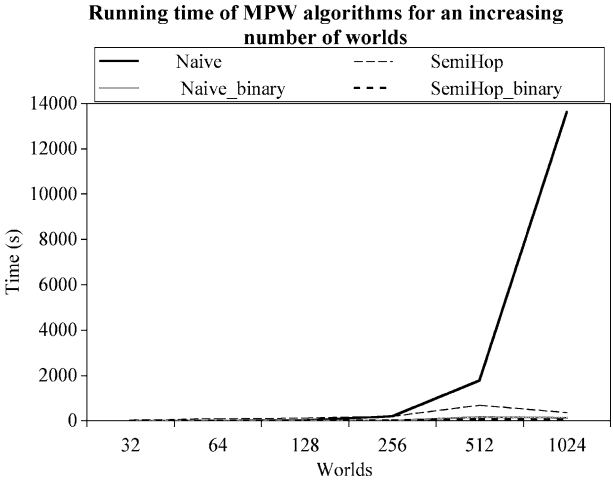

Fig. 15 Running time of the algorithms for increasing number of worlds

algorithm only requires 1.79 and 19.99 s respectively. However, SemiHOP vastly outperforms naive for problems with higher densities—358.3 s versus 13,636.23 s 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 subpartitions 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 Fig. 16, we examine the running times of the $\text{naive}_{\text{bin}}$ and $\mathsf{SemiHOP}_{\text{bin}}$ 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. $\mathsf{SemiHOP}_{\text{binary}}$ consistently takes less time than $\text{naive}_{\text{binary}}$, though both algorithms still perform rather well. For 1.23794×10^{27} possible worlds, $\text{naive}_{\text{binary}}$ takes an average 26,325.1 s while $\mathsf{SemiHOP}_{\text{binary}}$

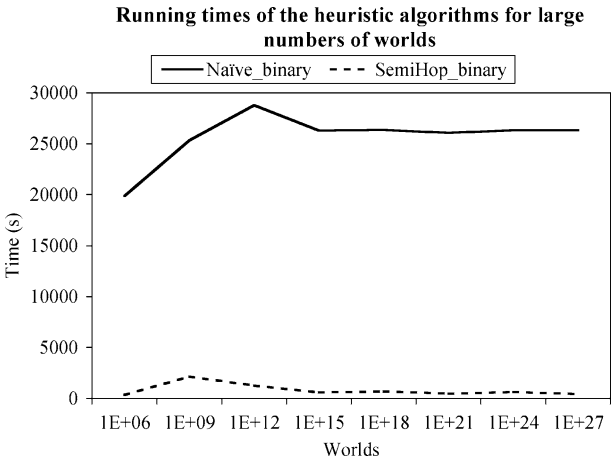

Fig. 16 Running time of $\text{naive}_{\text{bin}}$ and $\mathsf{SemiHOP}_{\text{bin}}$ for large number of worlds

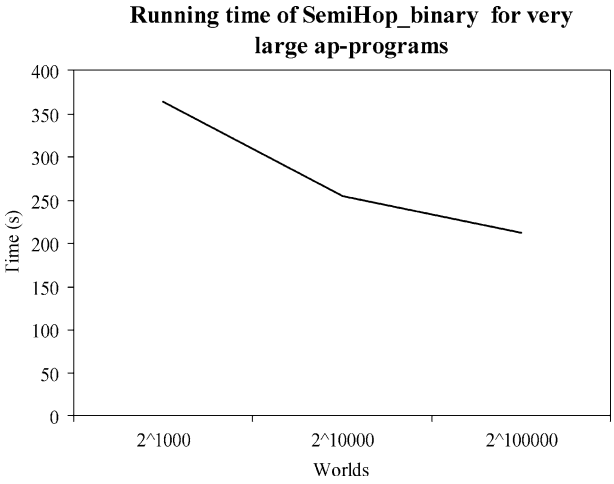

Fig. 17 Running time of the $\mathsf{SemiHOP}_{\text{binary}}$ algorithm for very large numbers of possible worlds

requires only 458.07 s. 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 $\mathsf{SemiHOP}_{\text{binary}}$ on problems with very large *ap*-programs (from 1,000 to 100,000 ground atoms) only take around 300 s using a 2% sample rate. However, this aspect of the SemiHOP algorithm can also lead to some anomalous behavior, where the running time will appear to decrease as the number of worlds increases. Figure 17 illustrates this anomaly, as the computation time appears to decrease with very large numbers of worlds. This occurs when we 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

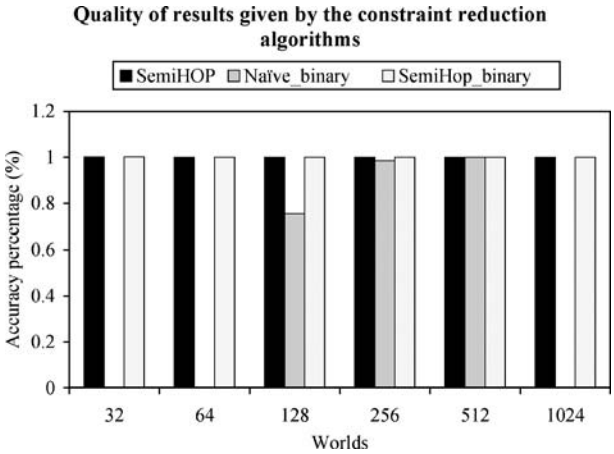

Fig. 18 Quality of the solutions produced by SemiHOP, $\text{naive}_{\text{bin}}$, and $\mathsf{SemiHOP}_{\text{bin}}$ as compared to Naive

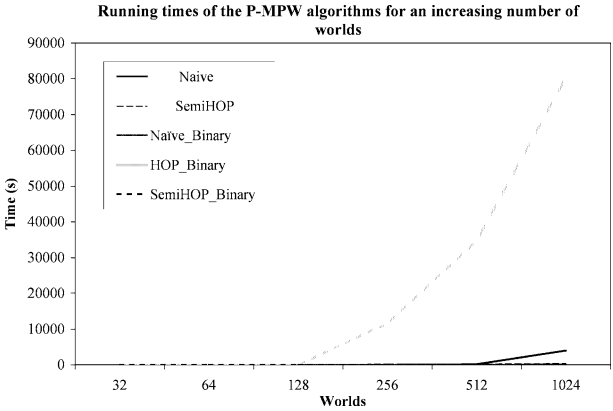

Fig. 19 Running time of the P-MPW versions of the naive, SemiHOP, $\text{naive}_{\text{bin}}$,$\text{HOP}_{\text{bin}}$, and $\text{SemiHOP}_{\text{bin}}$ algorithms for an increasing number of worlds

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 18 compares the accuracy of the probability found for the most probable world by SemiHOP, $\text{naive}_{\text{binary}}$, and $\text{SemiHOP}_{\text{binary}}$ 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 $\text{SemiHOP}_{\text{binary}}$ algorithms demonstrate near perfect accuracy; this is significant because in the $\text{SemiHOP}_{\text{binary}}$ 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 $\text{naive}_{\text{binary}}$ algorithm can be between 75% and 100% as likely as those given by the regular naive algorithm; however, the $\text{naive}_{\text{binary}}$ algorithm also was often unable to find a solution.

8.1 Parallel implementations

In addition to the basic MPW algorithms, we also implemented the P-MPW algorithm and the PAMPW-MS iterative algorithm, using about 6,700 lines of Java code. The above experiments were repeated running the parallel algorithms on the same

Table 1 Percent speedup achieved with P-MPW algorithms

Worlds	Naive	SemiHOP	$\text{Naive}_{\text{binary}}$	$\text{HOP}_{\text{binary}}$	$\text{SemiHOP}_{\text{binary}}$
32	50.48	02.73	111.21	94.40	76.64
64	66.15	40.88	123.72	97.78	80.26
128	83.23	47.97	121.58	97.10	78.06
256	89.47	52.01	86.80	13.61	44.73
512	90.29	55.84	89.95	29.15	9.68
1024	72.45	49.16	89.81	1.86	27.88
Avg	75.35	41.43	103.84	55.65	52.87

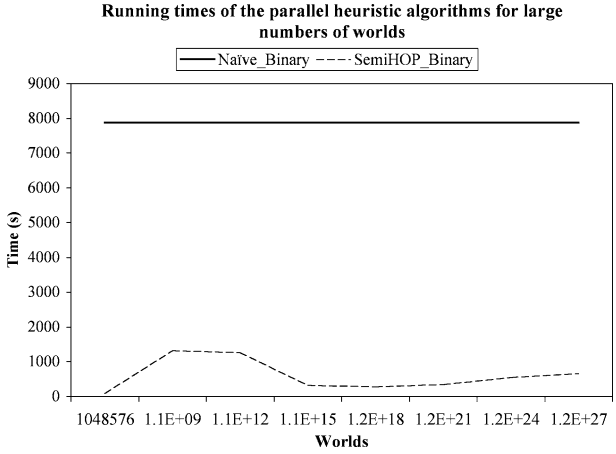

Fig. 20 Running time of the P-MPW versions of the $\text{naive}_{\text{bin}}$ and $\text{SemiHOP}_{\text{bin}}$ algorithms for large numbers of worlds

computing cluster. Each run utilized 16 processors in parallel to solve a single MPW problem. As expected, the P-MPW algorithms produce a marked speedup in the computation time for finding the most probable world (Fig. 19). Where the basic naive algorithm requires almost 4 h (13,636.23 s) for problems with 1,024 possible worlds, the naive P-MPW algorithm completed the same computation in only about an hour (4016.83 s). This is a very promising result for situations where an exact solution is necessary. We see a similar speedup for the SemiHOP and heuristic algorithms; the P-MPW SemiHOP algorithm uses slightly under 6 min (339.65 s) as opposed to 33.47 min (2,008.1 s) to solve for 1,024 worlds, and the naive heuristic improves from 136.08 to 21.78 s. In some cases, however, the P-MPW version of the SemiHOP algorithm actually performs worse as compared to the serial SemiHOP algorithm. This anomaly occurs in those instances where there are no subpartitions with only a single satisfying interpretation; in such cases, we do not actually need to solve an MPW computation (as described in Section 4.1), so the overhead of managing parallel threads is greater than the running time of the serial version. In most instances, though, the P-MPW algorithm greatly improves the efficiency of computing the most probable world. Table 1 contains the average speedup achieved by using the P-MPW algorithms compared to their serial counterparts. Similar improvements can be seen when using the P-MPW heuristic algorithms on large numbers of worlds, providing an average speedup of about 66%. These running times are shown in Fig. 20.

9 Related work

Probabilistic logic programming was introduced in [16, 17] and later studied by several authors [2, 3, 9, 11, 14]. This work was preceded by earlier—non-probabilistic—papers on quantitative logic programming of which [21] is an example. Ngo and Haddawy [14] presents a model theory, fixpoint theory, and proof procedure for conditional probabilistic logic programming. Lukasiewicz and Kern-Isberner [9] combines probabilistic LP with maximum entropy. Lukasiewicz [13] presents a conditional semantics for probabilistic LPs where each rule is interpreted as specifying the

conditional probability of the rule head, given the body. Lakshmanan and Shiri [11] develops a semantics for logic programs in which different general axiomatic methods are given to compute probabilities of conjunctions and disjunctions. Dekhtyar and Subrahmanian [3] presents an approach to a similar problem. Damasio et al. [2] 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 [2]). 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 [16]. Our framework modifies gp-programs in three ways: (1) we do not allow extensional predicates to occur in rule heads, while gp-programs do allow them, (2) we allow arbitrary formulas to occur in rule heads, whereas gp-programs only allow the so-called "basic formulas" to appear in rule heads. (3) Most importantly, of all, we solve the problem of finding the most probable model whereas [16] solve the problem of entailment.

A related work is that of *optimal models* of disjunctive logic programs [12]. An optimal model of a disjunctive logic program tries to find either a model, a minimal model, or a stable model of the DLP that maximizes an objective function. The techniques there assume no knowledge of the objective function (except for monotonicity). In contrast, our techniques use *ap*-programs which are a form of probabilistic logic program (not considered in [12]). Moreover, our techniques set up a linear program that is associated with an *ap*-program, while their work has no such linear program. The complexities associated with finding the most probable world arise from the linear programming formulation because the linear programs are exponential, containing one variable for each world. This does not occur in the non-probabilistic framework of [12]. Leone et al. [12] gives sound and complete ways to find optimal models by doing a generate and test of models, along with some intelligent pruning. This paper focuses directly on how to solve the linear program to find appropriate worlds. Moreover, this paper provides the first heuristic methods that scale to large scenarios—we developed our algorithms and showed how they perform when there are about $2^{100,000}$, or about $10^{30,000}$ worlds.

Another related effort in the agent's world is that of optimal feasible status sets [19]. Optimal status sets build upon the status set semantics for agent programs [5]. Status sets are sets of actions that an agent can take in a given situation. They bear the same relationship to agent programs that models bear to logic programs. The work on optimal status sets improves upon work such as that in [12] because it provides a *non-ground* framework which avoids grounding out agent programs till required. This is a big contribution that we would like to extend *ap*-programs to. However, the work of [19] has the same differences from our present work as [12].

The closest work to solving large linear programs is that of [8] who use column generation methods to solve PSAT, CONDSAT, and minimal modifications to ensure satisfiability. They report experiments showing that problems of up to 140 variables and 300 clauses can be solved in reasonable time (about 190 min). Even though this suggests that the Column Generation method could be of use for our work, it should be noted that the relevant problem (PSAT) that the authors are

solving in this paper is much easier computationally than most probable worlds, since most probable world computations require solving the linear program once for each world, whereas they only solve the linear program once. Moreover, we deal with $10^{30,000}$ worlds and solve this problem in a few minutes, while they deal with a much smaller number and solve it in 190 min, though some 17 years have gone by since the publication of those results.

10 Conclusions

In this paper, we have developed 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 make the following contributions:

1. This is the first paper that deals with the problem of reducing the size of the linear programs that are generated by *ap*-programs and shown that these reductions are practical; past work on doing this by Lukaseiwicz [9, 13] were not demonstrated to handle the very large problem sizes ($10^{30,000}$ worlds) described in this paper.
2. This is the first paper that studies the problem of finding the most probable world, given an *ap*-program; we have not seen a single paper addressing this problem for any kind of PLPs.
3. We have developed three algorithms to find the most probable world and developed the Binary heuristic that can be used in conjunction with any of them.
4. We have developed the first parallel algorithms for *ap*-programs (and these are the first parallel algorithms for any kind of PLP).
5. Our theory has produced tangible results of use to US military officers [1, 20].
6. 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).

There is ample scope for further work, even without expanding the current paper too much. It is clear in our binary algorithm that there are many different ways of selecting the worlds to consider. Can we identify how different selection policies on worlds change the accuracy and computation time of the worlds that are returned as being most probable? Knowing the answer to this question can have a potentially large impact on the quality of the solutions found by our algorithms. A related question is to come up with better methods to estimate the quality of the solutions found by our algorithms when a large number of worlds is processed. We currently do our accuracy estimates on small numbers of worlds.

Many other problems remain open as well. First, we need an accurate estimation of the computational complexity of the MPW problem. We have proven NP-hardness results here, but were unable to establish membership in NP. A more accurate classification would be desirable. Moreover, it would be desirable to come up with even more efficient parallel algorithms—the currently scaling offered is not proportional to the number of CPUs used. 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.

Another major problem is that of *grounding*. Most existing works on probabilistic logic programs assume a *ground* program. However, there has been work on non-ground computations in non-monotonic logic programs [4]. A major way of further scaling our system is to support non-ground representations of worlds. For example, the single atom $p(X)$ may represent all worlds that are instances of $p(X)$. Likewise, the set $\{p(X), q(X)\}$ might represent all worlds consisting of two atoms, each of which is an instance of $p(X)$ and $q(X)$, respectively. It would be interesting to see if these intuitions can lead to enhanced scalability.

Acknowledgements This work was supported by AFOSR grants FA95500610405 and FA95500510298, by ARO grant DAAD190310202, by NSF grant 0540216 and by DoD grant N6133906C0149.

References

1. Bhattacharjee, Y.: Pentagon asks academics for help in understanding its enemies. Science **316**(5824), 534–535 (2007)
2. Damasio, C.V., Pereira, L.M., Swift, T.: Coherent well-founded annotated logic programs. In: Logic Programming and Non-monotonic Reasoning, pp. 262–276 (1999)
3. Dekhtyar, A., Subrahmanian, V.S.: Hybrid probabilistic programs. In: International Conference on Logic Programming, pp. 391–405 (1997)
4. Eiter, T., Lu, J.J., Subrahmanian, V.S.: Computing non-ground representations of stable models. In: Logic Programming and Non-monotonic Reasoning, pp. 198–217 (1997)
5. Eiter, T., Subrahmanian, V.S., Pick, G.: Heterogeneous active agents, i: semantics. Artif. Intell. **108**(1–2), 178–255 (1999)
6. Fagin, R., Halpern, J.Y., Megiddo, N.: A logic for reasoning about probabilities. Inf. Comput. **87**(1–2), 78–128 (1990)
7. Hailperin, T.: Probability logic. Notre Dame J. Form. Log. **25**(3), 198–212 (1984)
8. Jaumard, B., Hansen, P., de Aragão, M.P.: Column generation methods for probabilistic logic. In: Proceedings of the First Integer Programming and Combinatorial Optimization Conference, pp. 313–331. University of Waterloo Press, Waterloo, Ontario, Canada (1990)
9. Lukasiewicz, T., Kern-Isberner, G.: Probabilistic logic programming under maximum entropy. In: Lecture Notes in Computer Science (Proceedings of ECSQARU 1999) 1638 (1999)
10. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer-Verlag (1987)
11. Lakshmanan, L.V.S., Shiri, N.: A parametric approach to deductive databases with uncertainty. IEEE Trans. Knowl. Data Eng. **13**(4), 554–570 (2001)
12. Leone, N., Scarcello, F., Subrahmanian, V.S.: Optimal models of disjunctive logic programs: semantics, complexity, and computation. IEEE Trans. Knowl. Data Eng. **16**(4), 487–503 (2004)
13. Lukasiewicz, T.: Probabilistic logic programming. In: European Conference on Artificial Intelligence, pp. 388–392 (1998)
14. Ngo, L., Haddawy, P.: Probabilistic logic programming and bayesian networks. In: Asian Computing Science Conference, pp. 286–300 (1995)
15. Nilsson, N.: Probabilistic logic. Artif. Intell. **28**, 71–87 (1986)
16. Ng, R.T., Subrahmanian, V.S.: A semantical framework for supporting subjective and conditional probabilities in deductive databases. In: Furukawa, K. (ed.) Proceedings of the Eighth International Conference on Logic Programming, pp. 565–580. The MIT Press (1991)
17. Ng, R.T., Subrahmanian, V.S.: Probabilistic logic programming. Inf. Comput. **101**(2), 150–201 (1992)
18. Subrahmanian, V.S., Albanese, M., Martinez, V., Reforgiato, D., Simari, G.I., Sliva, A., Udrea, O., Wilkenfeld, J.: CARA: a cultural reasoning architecture. IEEE Intell. Syst. **22**(2), 12–16 (2007)
19. Stroe, B., Subrahmanian, V.S., Dasgupta, S.: Optimal status sets of heterogeneous agent programs. In: AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 709–715. ACM, New York, NY, USA (2005)

20. Subrahmanian, V.S.: Cultural modeling in real-time. Science **1507**, 309–310 (2007)
21. van Emden, M.H.: Quantitative deduction and its fixpoint theory. J. Log. Program. **4**, 37–53 (1986)
22. 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 (February 2007)