Technical Communications of the International Conference on Logic Programming, 2010 (Edinburgh), pp. 192–201
http://www.floc-conference.org/ICLP-home.html

ABDUCTIVE INFERENCE IN PROBABILISTIC LOGIC PROGRAMS

GERARDO I. SIMARI AND V.S. SUBRAHMANIAN

Department of Computer Science and UMIACS
University of Maryland College Park
College Park, MD 20742, USA
E-mail address, Gerardo Simari: gisimari@cs.umd.edu
E-mail address, V.S. Subrahmanian: vs@cs.umd.edu

ABSTRACT. Action-probabilistic logic programs (*ap*-programs) are a class of probabilistic logic programs that have been extensively used during the last few years for modeling behaviors of entities. Rules in *ap*-programs have the form "If the environment in which entity E operates satisfies certain conditions, then the probability that E will take some action A is between L and U". Given an *ap*-program, we are interested in trying to change the environment, subject to some constraints, so that the probability that entity E takes some action (or combination of actions) is maximized. This is called the Basic Probabilistic Logic Abduction Problem (Basic PLAP). We first formally define and study the complexity of Basic PLAP and then provide an exact (exponential) algorithm to solve it, followed by more efficient algorithms for specific subclasses of the problem. We also develop appropriate heuristics to solve Basic PLAP efficiently.

1. Introduction

Action probabilistic logic programs (*ap*-programs for short) [Khu07] are a class of the extensively studied family of probabilistic logic programs (PLPs) [Ng92, Ng93, KI04]. *ap*-programs have been used extensively to model and reason about the behavior of groups and an application for reasoning about terror groups based on *ap*-programs has users from over 12 US government entities [Gil08]. *ap*-programs use a two sorted logic where there are "state" predicate symbols and "action" predicate symbols[1] and can be used to represent behaviors of arbitrary entities (ranging from users of web sites to institutional investors in the finance sector) because they consist of rules of the form "if a conjunction C of atoms is true in a given state S, then entity E (the entity whose behavior is being modeled) will take action A with a probability in the interval $[L, U]$."

In this kind of application, it is essential to avoid making probabilistic independence assumptions, since the approach involves *finding out* what probabilistic relationships exist and then exploit these findings in the forecasting effort. For instance, Figure 1 shows a small set of rules automatically extracted from data [Asa08] about Hezbollah's past. Rule 1 says that Hezbollah uses kidnappings as an organizational strategy with probability between 0.5 and 0.56 whenever no political support was provided to it by a foreign state (`forstpolsup`), and the severity of inter-organizational conflict involving it (`intersev1`) is at level "*c*". Rules 2 and 3, also about kidnappings, state that

1998 ACM Subject Classification: I.2.3: Logic Programming, Probabilistic Reasoning.
Key words and phrases: Probabilistic Logic Programming, Imprecise Probabilities, Abductive Inference.

[1]Action atoms only represent the fact that an action is taken, and not the action itself; they are therefore quite different from actions in domains such as AI planning or reasoning about actions, in which effects, preconditions, and postconditions are part of the specification. We assume that effects and preconditions are generally not known.

Technical Communications of the 26th International Conference on Logic Programming, Edinburgh, July, 2010
Editors: Manuel Hermenegildo, Torsten Schaub
LIPIcs - Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.ICLP.2010.192

r_1. kidnap$(1) : [0.50, 0.56] \leftarrow$ forstpolsup$(0) \wedge$ intersev1(c).
r_2. kidnap$(1) : [0.80, 0.86] \leftarrow$ extsup$(1) \wedge$ demorg(0).
r_3. kidnap$(1) : [0.80, 0.86] \leftarrow$ extsup$(1) \wedge$ elecpol(0).
r_4. tlethciv$(1) : [0.49, 0.55] \leftarrow$ demorg(1).
r_5. tlethciv$(1) : [0.71, 0.77] \leftarrow$ elecpol$(1) \wedge$ intersev2(c).

Figure 1: A small set of rules modeling Hezbollah.

this action will be performed with probability between 0.8 and 0.86 when no external support is solicited by the organization (extsup) and either the organization does not advocate democratic practices (demorg) or electoral politics is not used as a strategy (elecpol). Similarly, Rules 4 and 5 refer to the action "civilian targets chosen based on ethnicity" (tlethciv). The first one states that this action will be taken with probability 0.49 to 0.55 whenever the organization advocates democratic practices, while the second states that the probability rises to between 0.71 and 0.77 when electoral politics are used as a strategy and the severity of inter-organizational conflict (with the organization with which the second highest level of conflict occurred) was not negligible" (intersev2). *ap*-programs have been used extensively by terrorism analysts to make predictions about terror group actions [Gil08, Man08].

Suppose, rather than predicting what action(s) a group would take in a given situation or environment, we want to determine what *we can do* in order to induce a given behavior by the group. For example, a policy maker might want to understand what we can do so that a given goal (*e.g.*, the probability of Hezbollah using kidnappings as a strategy is below some percentage) is achieved, given some constraints on what is feasible. The *probabilistic logic abduction problem* (PLAP) deals with finding how to *reach* a new (feasible) state from the current state such that the *ap*-program associated with the group and the new state jointly entail that the goal will be true within a given probability interval.

In this paper, we first briefly recall *ap*-programs and then formulate PLAP theoretically. We then develop a host of complexity results for PLAP under varying assumptions. We then describe both exact and heuristic algorithms to solve the PLAP problem. We briefly describe a prototype implementation and experiments showing that our algorithm is feasible to use even when the *ap*-program contains hundreds of rules. A brief note on related work before we begin; almost all past work on abduction in such settings have been devised under various independence assumptions [Poo97, Poo93]. We are aware of no work to date on abduction in possible worlds-based probabilistic logic systems such as those of [Hai84], [Nil86], and [Fag90] where independence assumptions are not made.

2. Preliminaries

We now overview the syntax and semantics of *ap*-programs from [Khu07]. We assume the existence of a logical alphabet that consists of a finite set $\mathcal{L}_{cons}$ of constant symbols, a finite set $\mathcal{L}_{pred}$ of predicate symbols (each with an associated arity) and an infinite set $\mathcal{L}_{var}$ of variable symbols; function symbols are not allowed. Terms, atoms, and literals are defined in the usual way [Llo87]. We assume that $\mathcal{L}_{pred}$ is partitioned into disjoint sets: $\mathcal{L}_{act}$ of *action symbols* and $\mathcal{L}_{sta}$ of *state* symbols. Thus, if $t_1, \ldots, t_n$ are terms, and p is an n-ary action (resp. state) symbol, then $p(t_1, \ldots, t_n)$, is called an *action (resp. state) atom*.

Definition 2.1. A (ground) action formula is defined as: *(i)* a (ground) action atom is a (ground) action formula; *(ii)* if F and G are (ground) action formulas, then $\neg F$, $F \wedge G$, and $F \vee G$ are also (ground) action formulas.

The set of all possible action formulas is denoted by *formulas*$(B_{\mathcal{L}_{act}})$, where $B_{\mathcal{L}_{act}}$ is the Herbrand base associated with $\mathcal{L}_{act}$, $\mathcal{L}_{cons}$, and $\mathcal{L}_{var}$.

Definition 2.2. If F is an action formula and $\mu = [\alpha, \beta] \subseteq [0, 1]$, then $F : \mu$ is called an *annotated action formula* (or *ap*-formula), and μ is called the *ap*-annotation of F.

Definition 2.3. A *world* is any finite set of ground action atoms. A *state* is any finite set of ground state atoms.

It is assumed that all actions in the world are carried out more or less in parallel and at once, given the temporal granularity adopted along with the model. Contrary to (related but essentially different) approaches such as stochastic planning, we are not concerned here with reasoning about the effects of actions. We now define *ap*-rules.

Definition 2.4. If F is an action formula, $B_1, \ldots, B_n$ are state atoms, and μ is an *ap*-annotation, then $F : \mu \leftarrow B_1 \ \wedge \ \ldots \wedge \ B_m$ is called an ap-*rule*. If this rule is named r, then *Head*(r) denotes $F : \mu$ and *Body*(r) denotes $B_1 \ \wedge \ \ldots \wedge \ B_n$.

Intuitively, the rule specified above says that if $B_1, \ldots, B_m$ are all true in a given state, then there is a probability in the interval μ that the action combination F is performed by the entity modeled by the *ap*-rule.

Definition 2.5. An *action probabilistic logic program* (*ap*-program for short) is a finite set of *ap*-rules. An *ap*-program Π' such that $\Pi' \subseteq \Pi$ is called a *subprogram* of Π.

Figure 1 shows a small portion of an *ap*-program we derived automatically to model Hezbollah's actions. Henceforth, we use *Heads*(Π) to denote the set of all annotated formulas appearing in the head of some rule in Π. Given a ground *ap*-program Π, we will use *sta*(Π) (resp., *act*(Π)) to denote the set of all state (resp., action) atoms that appear in Π.

Example 2.6. Coming back to the *ap*-program in Figure 1, the following are examples of worlds:

$$\{\texttt{kidnap}(1)\}, \{\texttt{kidnap}(1), \texttt{tlethciv}(1)\}, \{\}$$

The following are examples of states:

$$\{\texttt{forstpolsup}(0), \texttt{elecpol}(0)\}, \{\texttt{extsup}(1), \texttt{elecpol}(1)\}, \{\texttt{demorg}(1)\}.$$

We use $\mathcal{W}$ to denote the set of all possible worlds, and $\mathcal{S}$ to denote the set of all possible states. It is clear what it means for a state to satisfy the body of a rule [Llo87].

Definition 2.7. Let Π be an *ap*-program and s a state. We say that s *satisfies* the body of a rule $F : \mu \leftarrow B_1 \ \wedge \ \ldots \wedge \ B_m$ if and only if $\{B_1, \ldots, B_M\} \subseteq s$.

Similarly, we define what it means for a world to satisfy a ground action formula:

Definition 2.8. Let F, F_1, F_2 be ground action formulas and w a world. We say that w *satisfies* F if and only if:
– if $F \equiv a$, for some atom $a \in B_{\mathcal{L}_{act}}$, then $a \in w$;
– if $F \equiv F_1 \wedge F_2$, then w satisfies F_1 and w satisfies F_2;
– if $F \equiv F_1 \vee F_2$, then w satisfies F_1 or w satisfies F_2;
– if $F \equiv \neg F'$, for some action formula $F' \in$ *formulas*$(B_{\mathcal{L}_{act}})$, then w does not satisfy F'.

Finally, we will use the concept of *reduction* of an *ap*-program w.r.t. a state:

Definition 2.9. Let Π be an *ap*-program and s a state. The *reduction of* Π *w.r.t.* s, denoted Π_s, is the set $\{F : \mu \mid s \text{ satisfies } \textit{Body} \text{ and } F : \mu \leftarrow \textit{Body} \text{ is a ground instance of a rule in } \Pi\}$. Rules in this set are said to be *relevant* in state s.

The semantics of *ap*-programs uses possible worlds in the spirit of [Hai84, Nil86, Fag90]. Given an *ap*-program Π and a state s, we can define a set $LC(\Pi, s)$ of linear constraints associated with s. Each world w_i expressible in the language $\mathcal{L}_{act}$ has an associated variable v_i denoting the probability that it will actually occur. $LC(\Pi, s)$ consists of the following constraints.
(1) For each $\textit{Head}(r) \in \Pi_s$ of the form $F : [\ell, u]$, we have constraint $\ell \leq \sum_{w_i \in \mathcal{W} \wedge w_i \models F} v_i \leq u$.
(2) $LC(\Pi, s)$ contains the constraint $\sum_{w_i \in \mathcal{W}} v_i = 1$.
(3) All variables are non-negative.
(4) $LC(\Pi, s)$ contains only the constraints described in $1 - 3$.
While [Khu07] provides a more formal model theory for *ap*-programs, we merely provide the definition below. Π_s is *consistent* iff $LC(\Pi, s)$ is solvable over $\mathbb{R}$.

Definition 2.10. Let Π be an *ap*-program, s a state, and $F : [\ell, u]$ a ground action formula. Π_s *entails* $F : [\ell, u]$, denoted $\Pi_s \models F : [\ell, u]$ iff $[\ell', u'] \subseteq [\ell, u]$ where:
$\ell' =$ **minimize** $\sum_{w_i \in \mathcal{W} \wedge w_i \models F} v_i$ **subject to** $LC(\Pi, s)$.
$u' =$ **maximize** $\sum_{w_i \in \mathcal{W} \wedge w_i \models F} v_i$ **subject to** $LC(\Pi, s)$.

The following is an example of $LC(\Pi, s)$ and entailment of an *ap*-formula.

Example 2.11. Consider *ap*-program Π from Figure 1 and state s_2 from Figure 2. The set of possible worlds is as follows: $w_0 = \{\}$, $w_1 = \{\texttt{kidnap}(1)\}$, $w_2 = \{\texttt{tlethciv}(1)\}$, and $w_3 = \{\texttt{kidnap}(1), \texttt{tlethciv}(1)\}$. Suppose we use p_i to denote the variable associated with the probability of world w_i; $LC(\Pi, s_2)$ then consists of the following constraints:

$$0.5 \leq p_1 + p_3 \leq 0.56$$
$$0.49 \leq p_2 + p_3 \leq 0.55$$
$$p_0 + p_1 + p_2 + p_3 = 1$$

One possible solution to this set of constraints is $p_0 = 0$, $p_1 = 0.51$, $p_2 = 0.05$, and $p_3 = 0.44$; another possible distribution is $p_0 = 0.5$, $p_1 = 0$, $p_2 = 0$, and $p_3 = 0.5$; yet another one is $p_0 = 0$, $p_1 = 0.45$, $p_2 = 0.11$, and $p_3 = 0.44$. Finally, formula $\texttt{kidnap}(1) \wedge \texttt{tlethciv}(1)$ (satisfied only by world w_3) is entailed with probability in the interval $[0, 0.55]$, meaning that one cannot assign a probability greater than 0.55 to this formula (this example shows that, contrary to what one might think, the interval $[0, 1]$ is not necessarily a solution).

Note that representing a set of distributions is not possible in many other approaches to probabilistic reasoning, such as Bayesian networks [Pea88], Poole's Independent Choice Logic [Poo97] and related formalisms such as [Poo93]. However, this is a key capability for our approach, as we specifically require a formalism that is not forced to make assumptions about the probabilistic dependence (or independence) of the events we are reasoning about. On the other hand, it is certainly possible to extend our approach in such a way that the key aspects of Bayesian networks and related formalisms are directly expressible, as was shown in [Ng93] when probabilistic logic programs were introduced.

$s_1 = \{\texttt{forstpolsup}(0), \texttt{intersev1}(c), \texttt{intersev2}(0), \texttt{elecpol}(1), \texttt{extsup}(0), \texttt{demorg}(0)\}$
$s_2 = \{\texttt{forstpolsup}(0), \texttt{intersev1}(c), \texttt{intersev2}(0), \texttt{elecpol}(0), \texttt{extsup}(0), \texttt{demorg}(1)\}$
$s_3 = \{\texttt{forstpolsup}(0), \texttt{intersev1}(c), \texttt{intersev2}(0), \texttt{elecpol}(0), \texttt{extsup}(0), \texttt{demorg}(0)\}$
$s_4 = \{\texttt{forstpolsup}(1), \texttt{intersev1}(c), \texttt{intersev2}(c), \texttt{elecpol}(1), \texttt{extsup}(1), \texttt{demorg}(0)\}$
$s_5 = \{\texttt{forstpolsup}(0), \texttt{intersev1}(c), \texttt{intersev2}(c), \texttt{elecpol}(0), \texttt{extsup}(1), \texttt{demorg}(0)\}$

Figure 2: A small set of possible states

3. The Probabilistic Logic Abduction Problem

Suppose s is a state (the current state), G is a goal (an action formula), and $[\ell, u] \subseteq [0, 1]$ is a probability interval. The basic PLAP problem requires finding a new state s' such that $\Pi_{s'}$ entails $G : [\ell, u]$. However, s' must be *reachable* from s. For this, we merely assume the existence of a reachability predicate $reach$ specifying *direct* reachability from one state to another. $reach^*$ is the reflexive transitive closure of $reach$ and $unReach$ is its complement. We will investigate, in Section 4.2 below, one way in which $reach$ can be specified; when available, knowledge of action effects and preconditions can be encoded into this predicate.

Example 3.1. Suppose, for simplicity, that the only state predicate symbols are those that appear in the rules of Figure 1, and consider the set of states in Figure 2. Then, some examples of reachability are the following: $reach(s_1, s_2)$, $reach(s_1, s_3)$, $reach(s_2, s_1)$, $reach(s_4, s_1)$, $\neg reach(s_2, s_5)$, and $\neg reach(s_3, s_5)$. Note that, if state s_5 is reachable, then the *ap*-program is inconsistent, since both rules 1 and 2 are relevant in that state.

We can now state the Basic PLAP problem formally:
Basic PLAP Problem.
Input: An *ap*-program Π, a state s, a reachability predicate $reach$ and a ground *ap*-formula $G : [\ell, u]$.
Output: "Yes" if there exists a state s' such that $reach^*(s, s')$ and $\Pi_{s'} \models G : [\ell, u]$, and "No" otherwise.

Example 3.2. Consider once again the program in the running example and the set of states from Figure 2. If the goal is $kidnap(1) : [0, 0.6]$ (we want the probability of Hezbollah using kidnappings to be at most 0.6) and the current state is s_4, then the problem is solvable because Example 3.1 shows that state s_1 can be reached from s_4, and $\Pi_{s_1} \models kidnap(1) : [0, 0.6]$.

The following result shows the intractability of Basic PLAP in the general case.

Proposition 3.3. *Basic PLAP is EXPTIME-complete.*

Moreover, this problem is likely to be intractable even under simplifying assumptions.

Proposition 3.4. *Let $\mathcal{L}_{sta}$ be such that $|\mathcal{L}_{sta}| \leq c'$ for some constant $c' \in \mathbb{N}$; the Basic PLAP problem under this assumption is NP-hard.*

Proposition 3.5. *Let $\mathcal{L}_{act}$ be such that $|\mathcal{L}_{act}| \leq c'$ for some constant $c' \in \mathbb{N}$; the Basic PLAP problem under this assumption is NP-hard.*

The above results reveal that the complexity of PLAP is caused by two factors. (P1) We need to find a subprogram Π' of Π such that when the body of all rules in that subprogram is deleted, the resulting subprogram entails the goal, and (P2) Decide if there exists a state s' such that $\Pi' = \Pi_s$ and s is reachable from the initial state.

4. Algorithms for PLAP

In this section, we leverage the above intuition to develop an algorithm for PLAP under the assumption that all goals are of the form $F : [0, u]$ (ensure that F's probability is less than or equal to u) or $F : [\ell, 1]$ (ensure that F's probability is at least ℓ). Finally, we develop a heuristic algorithm.

4.1. Answering Threshold Goals

A *threshold goal* is an annotated action formula of the form $F : [0, u]$ or $F : [\ell, 1]$. In this section, we study how we can devise a better algorithm for Basic PLAP when only threshold goals are considered. This is a reasonable approach, since threshold goals can be used to express the desire that certain formulas (actions) should only be entailed with a certain maximum probability (upper bound) or should be entailed with at least a certain minimum probability (lower bound). The tradeoff lies in the fact that we lose the capacity to express both desires at once. We start by inducing equivalence classes on subprograms that limit the search space, helping address problem P1.

Definition 4.1. Let Π be a ground *ap*-program and F be a ground action formula. We say that subprograms $\Pi_1, \Pi_2 \subseteq \Pi$ are *equivalent* given F, written $\Pi_1 \sim_F \Pi_2$, iff $\Pi_1 \models F : [\ell, u] \Leftrightarrow \Pi_2 \models F : [\ell, u]$ for any $\ell, u \in [0, 1]$. Furthermore, states s_1 and s_2 are *equivalent* given F, written $s_1 \sim_F s_2$, iff $reach(s_1, s_2)$, $reach(s_2, s_1)$, and $\Pi_{s_1} \sim_F \Pi_{s_2}$.

Example 4.2. Let Π be the *ap*-program from Figure 1, formula $F = \texttt{kidnap}(1)$, $\Pi_1 = \{r_1\}$, $\Pi_2 = \{r_2, r_3\}$ $\Pi_3 = \{r_1, r_4\}$ $\Pi_4 = \{r_1, r_5\}$, and $\Pi_5 = \{r_2, r_3, r_5\}$. Here, $\Pi_1 \sim_F \Pi_3$, $\Pi_1 \sim_F \Pi_4$, $\Pi_3 \sim_F \Pi_4$, and $\Pi_2 \sim_F \Pi_5$. For instance, we can see that $\Pi_1 \sim_F \Pi_3$ because the probability with which $\texttt{kidnap}(1)$ is entailed is given by rule r_1; rule r_4 is immaterial in this case. Clearly, $\Pi_1 \not\sim_F \Pi_2$ since F is entailed with different probabilities in each case.

Next, consider the states from Figure 2 and the reachability predicate from Example 3.1. Since we have that $reach(s_1, s_2)$, $reach(s_2, s_1)$, Π_1 is relevant in s_1, and Π_3 is relevant in s_2, we can conclude that $s_1 \sim_F s_2$.

Relation $\sim$, both between states and between subprograms, is clearly an equivalence relation. The following lemma specifies a way to construct equivalence classes.

Lemma 4.3. *Let Π be an* ap*-program and G be an action formula. Consider two subprograms $\Pi', \Pi'' \subseteq \Pi$ such that $\Pi' = \Pi_a \cup \Pi'_p$ (resp., $\Pi'' = \Pi_a \cup \Pi''_p$), where Π_a is a set of rules whose heads have formulas F such that $F \wedge G \not\models \bot$ and Π'_p (resp., Π''_p) contains rules whose heads have formulas H such that $H \wedge G \models \bot$. Then, $\Pi' \sim_G \Pi''$.*

Lemma 4.4. *Let Π be a consistent* ap*-program and $G : [\ell_G, u_G]$ be a threshold goal. If there exists a rule $r \in \Pi$ such that Head$(r) = F : [\ell_F, u_F]$ and: either (1) if $u_G = 1$, $F \models G$, and $\ell_G \leq \ell_F$; or (2) if $\ell_G = 0$, $G \models F$, and $u_G \geq u_F$; then, $\Pi \models G : [\ell_G, u_G]$.*

The algorithm in Figure 3 first tries to leverage Lemma 4.4 and only proceeds if this is not possible. The way in which the algorithm partitions Π is partly based on Lemma 4.3.

Proposition 4.5. *Given an* ap*-program Π, a state $s \in \mathcal{S}$, and an annotated action formula $G : [\ell, u]$, Algorithm simpleAnnPLAP correctly computes a solution to Basic PLAP. Its worst case running time is in $O\left(2^{|\Pi|} + 2^{|\mathcal{L}_{sta}|} + 2^{|\mathcal{L}_{act}|}\right)$.*

We now present an example of how this algorithm works.

Algorithm 1: simpleAnnPLAP($\Pi, s, G : [\ell_G, u_G]$)

(1) Select rules of the form $r : F : [\ell_r, u_r] \leftarrow s_1 \wedge \ldots \wedge s_n$ such that $F \wedge G \not\models \bot$; call all such rules *active rules*, and the complement set *passive rules*, denoted *active*$(\Pi, G : [\ell_G, u_G])$ and *passive*$(\Pi, G : [\ell_G, u_G])$.
(2) If Lemma 4.4 is applicable, return *true* if there exists a consistent $\Pi' \subseteq$ *candAct*$(\Pi, G : [\ell_G, u_G]) \cup$ *passive*$(\Pi, G : [\ell_G, u_G])$ such that:
  (a) if $u_G = 1$, then at least one rule $r \in \Pi'$ must have head $F : [\ell_F, u_F]$ such that $F \models G$ and $\ell_G \leq \ell_F$; if $\ell_G = 0$), at least one rule $r \in \Pi'$ must have head $F : [\ell_F, u_F]$ such that $G \models F$ and $u_G \geq u_F$;
  (b) state s' for which $\Pi_{s'} = \Pi'$ is such that $reach^*(s, s')$.
(3) Otherwise, for each rule $r_i : F : [\ell_r, u_r] \leftarrow s_1 \wedge \ldots \wedge s_n$ do:
  (a) If $\ell_G = 0$, $F \models G$, and $\ell_r > u_G$ then add r_i to set *conf*$(\Pi, G : [\ell_G, u_G])$
  (b) Otherwise (*i.e.*, $u_G = 1$), if $G \models F$ and $u_r < \ell_G$ then add r_i to set *conf*$(\Pi, G : [\ell_G, u_G])$.
(4) Let *candAct*$(\Pi, G : [\ell_G, u_G])$ = *active*$(\Pi, G : [\ell_G, u_G]) \setminus$ *conf*$(\Pi, G : [\ell_G, u_G])$;
(5) Consider the set *candAct*$(\Pi, G : [\ell_G, u_G]) \cup$ *passive*$(\Pi, G : [\ell_G, u_G])$ and, for each pair of rules of the form $r_i : F_i : [\ell_{r_i}, u_{r_i}] \leftarrow s_1^i \wedge \ldots \wedge s_n^i$ and $r_j : F_j : [\ell_{r_j}, u_{r_j}] \leftarrow s_1^j \wedge \ldots \wedge s_m^j$ such that $F_i : [\ell_{r_i}, u_{r_i}]$ and $F_j : [\ell_{r_j}, u_{r_j}]$ are mutually inconsistent, add the pair (r_i, r_j) to a set called *inc*(Π).
(6) Return *true* if there exists a set of rules $\Pi' \subseteq$ *candAct*$(\Pi, G : [\ell_G, u_G]) \cup$ *passive*$(\Pi, G : [\ell_G, u_G])$ such that $\Pi' \cap$ *candAct*$(\Pi, G : [\ell_G, u_G]) \neq \emptyset$, no pair $\{r_1, r_2\} \subseteq \Pi'$ belongs to *inc*(Π), and:
  (a) $\Pi' \models G : [\ell_G, u_G]$;
  (b) $\exists$ state s' for which $\Pi_{s'} = \Pi'$ such that $reach^*(s, s')$;
(7) If Step 6 is not possible, return *false*;

Figure 3: An algorithm to solve Basic PLAP assuming a threshold goal.

Example 4.6. Suppose Π is the *ap*-program of Figure 1, the goal is $\texttt{kidnap}(1) : [0, 0.6]$ (abbreviated with $G : [0, 0.6]$ from now on) and the state is that of Example 3.2, $s_{curr} = \{\texttt{forstpolsup}(1), \texttt{intersev1}(c), \texttt{intersev2}(c), \texttt{elecpol}(1), \texttt{extsup}(1), \texttt{demorg}(0)\}$; note that $\Pi_{s_{curr}} = \{r_2, r_5\}$ and that clearly $\Pi_{s_{curr}} \not\models \texttt{kidnap}(1) : [0, 0.6]$. Step 1 of simpleAnnPLAP is simple in this case, since all the heads of rules in Π are atomic – therefore *passive*$(\Pi_{s_{curr}}, G : [0, 0.6]) = \emptyset$, and the set of active rules contains all the rules in Π. The following step checks for the applicability of Lemma 4.4; clearly rule r_1 satisfies the conditions and we only need to verify that some subprogram containing it is reachable. Assuming the same reachability predicate outlined in Example 3.1, $s_1 = \{\texttt{forstpolsup}(0), \texttt{intersev1}(c), \texttt{intersev2}(0), \texttt{elecpol}(1), \texttt{extsup}(0), \texttt{demorg}(0)\}$ is reachable from s_{curr}; this corresponds to choosing subprogram $\Pi' = \{r_1\}$. The only other possibilities are to make both r_1 and r_4, or r_1 and r_5 relevant.

4.2. An Improved PLAP Algorithm

In this section, we show that if we assume reachability/unreachability is specified in a syntactic manner rather than in a very general manner as presented earlier, we can come up with some good heuristics to solve Basic PLAP.

Definition 4.7. Let F and G be first-order formulas over $\mathcal{L}_{sta}$ and $\mathcal{L}_{var}$ with connectives $\wedge$, $\vee$, and $\neg$, and such that the set of variables over F is equal to those over G; all variables are assumed to be universally quantified with scope over both F and G. A *reachability constraint* is of the form $F \not\rightsquigarrow G$; we call F the antecedent and G the consequent of the constraint, and its semantics is:

$$unReach(s_1, s_2) \Leftrightarrow s_1 \models F \text{ and } s_2 \models G$$

where s_1 and s_2 are states in $\mathcal{S}$.

Algorithm 2: simpleAnnPLAP-Heur-RC($\Pi, s, G : [\ell_G, u_G], RC$)

(1) Execute Steps 1, 3, 4, and 5 of *simpleAnnPLAP*
(2) Let *goalState*, *goalStateAct*, *goalStateConf*, and *goalStateInf* be logical formulas over $\mathcal{L}_{sta}$ and $\mathcal{L}_{var}$;
(3) Initialize *goalState* to null, *goalStateAct* to $\bot$, and *goalStateConf*, *goalStateInc* to $\top$;
(4) for each rule $r_i \in$ *candAct*$(\Pi, G : [\ell_G, u_G])$ with *Head*$(r_i) = F : [\ell_F, u_F]$ do
  if $\big[(u_G = 1)$ and $(F \models G$ and $\ell_G \leq \ell_F)\big]$ or $\big[(\ell_G = 0)$ and $(G \models F$ and $u_G \geq u_F)\big]$
    then set *goalStateAct* := *goalStateAct* $\vee$ *getStateFormula*(r_i);
(5) for each rule $r_i \in$ *conf*$(\Pi, G : [\ell_G, u_G])$ do
  set *goalStateConf* := *goalStateConf* $\wedge \neg$*getStateFormula*(r_i);
(6) for each pair of rules $(r_i, r_j) \in$ *inc*(Π) do
  set *goalStateInc* := *goalStateInc* $\wedge \neg$(*getStateFormula*$(r_i) \wedge$ *getStateFormula*(r_j));
(7) set *goalState* := *goalStateAct* $\wedge$ *goalStateConf* $\wedge$ *goalStateInc*;
  // *goalState* describes the states that satisfy the goal
(8) return *decideReachability*$(s,$ *goalState*$, RC)$;

Figure 4: A heuristic algorithm based on Lemma 4.4 to solve Basic PLAP assuming threshold goals and that state reachability is expressed as a set RC of reachability constraints.

Reachability constraints simply state that if the antecedent is satisfied in a certain state, then no states that satisfy the consequent are reachable from it. We now present an example of a set of reachability constraints.

Example 4.8. Consider again the setting and *ap*-program from Figure 1. The following are examples of reachability constraints[2]:

$$\texttt{forstpolsup}(1) \not\rightarrow \texttt{intersev1}(c)$$
$$\texttt{intersev1}(c) \vee \texttt{intersev2}(c)) \wedge \texttt{demorg}(0) \not\rightarrow \texttt{demorg}(1)$$

Algorithm *simpleAnnPLAP-Heur-RC* (Figure 4) takes advantage of the structure added by the presence of reachability constraints. The algorithm starts out by executing the steps of *simpleAnnPLAP* that compute the sets *active*, *passive*, *candAct*, *conf*, and *inc*. It then builds formulas generated by reachability constraints that any solution state must satisfy; the algorithm uses a subroutine *formula*(s) which returns a formula that is a conjunction of all the atoms in state s and the negations of those not in s. In Step 4, the formula describes the fact that at least one of the states that make relevant "candidate active" rules (as described in Algorithm *simpleAnnPLAP*) must be part of the solution; similarly, Step 5 builds a formula ensuring that none of the conflicting active rules can be relevant if the problem is to have a solution. Finally, Step 6 describes the constraints associated with making relevant rules that are probabilistically inconsistent. Noticeably absent are the "passive" rules from the previous algorithm; such rules impose no constraints on the solution. The last two steps put subformulas together into a conjunction of constraints, and the algorithm must decide if there exist any states that model formula *goalState* and are eventually reachable from s. Eventual reachability can be decided by means of a *SAT-based* method as follows: if the current state does not satisfy *goalState*, it starts by initializing formula *Reachable* which will be used to represent the set of eventually reachable states at each step. The initial formula describes state s, and the algorithm then proceeds to select all the constraints whose antecedents are entailed by *Reachable*. Once we have this set, *Reachable* is updated to the conjunction of the negations of all the consequents of constraints in the set. We are done whenever either *Reachable* models *goalState*, or the old version of *Reachable* is modeled by the new one (no new reachable states exist).

[2]If available, knowledge of action effects and preconditions can be represented with similar constraints.

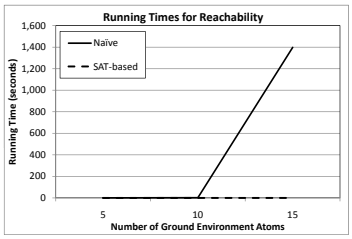

Figure 5: 5 rules, 25 ground action atoms, 5 reachability constraints, and atomic queries.

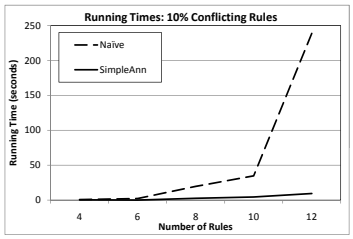

Figure 6: 10% goal-conf. rules, 25 action atoms, 5 state atoms (ground), and atomic queries.

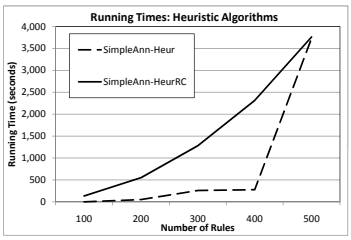

Figure 7: Larger *ap*-programs; 25 action atoms, 5 state atoms (ground), and 5 reach. constr.

5. Experimental Results

We conducted experiments using a prototype JAVA implementation consisting of roughly 2,500 lines of code. All experiments were run on multiple multi-core Intel Xeon E5345 processors at 2.33GHz, 8GB of memory, running the Scientific Linux distribution of the GNU/Linux operating system, kernel version 2.6.9-55.0.2.ELsmp.[3] Numbers reported are averages over at least 20 runs.

No. of State Atoms. In Figure 5 we show the running times of the different approaches to deciding reachability; the naive approach becomes intractable very quickly, while the (still exact) SAT-based algorithm approach has negligible cost for these runs.

No. of Rules. Figure 6 reports the running times of the SimpleAnn rule selection algorithm vs. the naive approach for programs in which 10% of the rules were forced to be in probabilistic conflict with the goal. This experiment shows how SimpleAnn leverages the presence of these rules, greatly reducing its running time w.r.t. that of the naive algorithm.

Finally, Figure 7 shows the running times for the SimpleAnn heuristic step (that is, assuming the algorithm only tries to apply the heuristic and pessimistically returns *false* otherwise) and the SimpleAnn-HeurRC algorithm for larger programs. It is interesting to see the different shapes of the curves: as programs get larger, the SAT formulas associated with SimpleAnn-HeurRC become larger as well, leading to the gradual increase in the running time; on the other hand, we can see that the strategy of only focusing on certain "heuristic rules" pays off for the SimpleAnn heuristic step, but there is a spike in running time when the size grows from 400 to 500 rules. This is likely due to the appearance of more such rules, which means that the algorithm has many more subprograms to verify for entailment of the goal.

6. Related Work and Conclusions

Abduction has been extensively studied in diagnosis [Con91], reasoning with non-monotonic logics [Eit95], probabilistic reasoning [Pea91, Poo97], argumentation [Koh02], planning [Esh88], and temporal reasoning [Esh88]; furthermore, it has been combined quite naturally with different variants of logic programs [Den02]. David Poole *et al*. combined probabilistic and non-monotonic reasoning, leading to the development of the Independent Choice Logic [Poo97]. Though this model is related to our work, it makes general assumptions of pairwise independence of probabilties of events; other related models are based on the class of graphical models including Bayesian Networks

[3]We note that this implementation makes use of only one processor and one core.

(BNs). The main difference between graphical model-based work and our work is that we *make no assumptions on the dependence or independence of probabilities of events*.

While AI planning may seem relevant, there are several differences. First, we are not assuming knowledge of the effects of actions; second, we assume the existence of a probabilistic model underlying the behavior of the entity being modeled. In this framework, we want to find a state such that *when the atoms in the state are added to the* ap*-program, the resulting combination* entails *the desired goal with a given probability.* While the italicized component of the previous sentence can be achieved within planning, it would require a state space that is exponentially larger than the one we use (the search space would be the set of all sets of atoms that are jointly entailed by any subprogram of the *ap*-program and any state).

To the best of our knowledge, this is the first paper that tackles the problem of abductive reasoning in probabilistic logic programming under no independence assumptions, in the tradition of [Ng92] for probabilistic logic programming, and [Hai84, Nil86, Fag90] for probabilistic logic.

Acknowledgements. The authors were funded in part by AFOSR grant FA95500610405 and ARO grant W911NF0910206.

References

[Asa08] V Asal, J Carter, and J Wilkenfeld. Ethnopolitical violence and terrorism in the middle east. In J Hewitt, J Wilkenfeld, and T Gurr (eds.), *Peace and Conflict 2008*. Paradigm, Boulder, CO, 2008.

[Con91] Luca Console and Pietro Torasso. A spectrum of logical definitions of model-based diagnosis. *Comput. Intell.*, 7(3):133–141, 1991.

[Den02] Marc Denecker and Antonis C. Kakas. Abduction in logic programming. In *Computational Logic: Logic Programming and Beyond, Part I*, pp. 402–436. Springer-Verlag, London, UK, 2002.

[Eit95] Thomas Eiter and Georg Gottlob. The complexity of logic-based abduction. *J. ACM*, 42(1):3–42, 1995.

[Esh88] Kave Eshghi. Abductive planning with event calculus. In *ICLP/SLP*, pp. 562–579. 1988.

[Fag90] Ronald Fagin, Joseph Y. Halpern, and Nimrod Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(1/2):78–128, 1990.

[Gil08] Jim Giles. Can conflict forecasts predict violence hotspots? *New Scientist*, (2647), 2008.

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

[Khu07] Samir Khuller, Maria Vanina Martinez, Dana S. Nau, Amy Sliva, Gerardo I. Simari, and V. S. Subrahmanian. Computing most probable worlds of action probabilistic logic programs: scalable estimation for 10^30,000 worlds. *AMAI*, 51(2-4):295–331, 2007.

[KI04] Gabriele Kern-Isberner and Thomas Lukasiewicz. Combining probabilistic logic programming with the power of maximum entropy. *Artif. Intell.*, 157(1-2):139–202, 2004.

[Koh02] J. Kohlas, D. Berzati, and R. Haenni. Probabilistic argumentation systems and abduction. *AMAI*, 34(1-3):177–195, 2002.

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

[Man08] A. Mannes, M. Michael, A. Pate, A. Sliva, V. S. Subrahmanian, and J. Wilkenfeld. Stochastic opponent modelling agents: A case study with Hezbollah. In Huan Liu and John Salerno (eds.), *Proc. of IWSCBMP*. 2008.

[Ng92] Raymond T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[Ng93] Raymond T. Ng and V. S. Subrahmanian. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *J. Autom. Reasoning*, 10(2):191–235, 1993.

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

[Pea88] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[Pea91] Judea Pearl. Probabilistic and qualitative abduction. In *AAAI Spring Symposium on Abduction*, pp. 155–158. AAAI Press, Stanford, CA, 1991.

[Poo93] David Poole. Probabilistic horn abduction and bayesian networks. *Artif. Intell.*, 64(1):81–129, 1993.

[Poo97] David Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artif. Intell.*, 94(1-2):7–56, 1997.