Probabilistic Answer Set Programming

Eduardo Menezes de Morais
Department of Computer Science
University of São Paulo
São Paulo, Brazil
Email: eduardo.morais@usp.br

Marcelo Finger
Department of Computer Science
University of São Paulo
São Paulo, Brazil
Email: mfinger@ime.usp.br

***Abstract*—This paper introduces a technique called Probabilistic Answer Set Programming (PASP), that allows modeling complex theories and checking its satisfiability with respect to a set of probabilistic data. We propose an algorithm for PASP processing based on a Turing reduction method to ASP.**

***Keywords*-logic programming; probabilistic logic; probabilistic satisfiability (PSAT); answer set programming (ASP).**

I. Introduction

This paper presents a well principled extension of Answer Set Programming (ASP, [1], [2]) to deal with probabilistic constraints. This extension is based on the interaction between logic and probability originally due to Boole [3] and which is now known as Probabilistic Logic [4] or Probabilistic Satisfiability (PSAT, [5]). Recent work has shown that PSAT has industrial applications [6], but for its use in larger problems a better modeling tool is necessary. One possible such tool is PASP.

The goal of this paper is to provide a clear semantics and an effective algorithm to a probabilistic extension of ASP. Due to space limitations, we present only one algorithm for PASP processing; more details can be found in [7].

ASP is a variation of Logic Programming that employs stable model semantics to deal with negation-as-failure, and has been successfully employed in applications ranging from typical AI problems to Software Engineering and Computational Biology.

PASP should be a tool for finding solutions to problems that combines "hard" constraints, that is, logical constraints that can never be broken in a solution, and "soft" constraints that could be broken but ideally would be satisfied. In [6] we see a similar application using PSAT. Since it is easier to model complex theories in ASP than in SAT, with PASP would allow modeling larger and more difficult problems.

The rest of this work is developed as follows. Background on ASP and PSAT is presented briefly in Sections II and III. More details can be found in [7]. The PASP problem is defined in Section IV and a solution method and experimental results in Sections V and VI.

II. Answer Set Programming

Answer Set Programming is a non-monotonic, declarative programming paradigm for solving hard combinatorial problems.

The concept of ***Answer Sets*** (or **stable models** for the case without classical negation) was created by Gelfond and Lifschitz in [1] in an attempt to clarify the semantics of negation in logic programming languages. The idea of using the concept of *Answer Sets* not only as an auxiliary tool for Prolog, but as a proper logic programming paradigm, was introduced by [2]. And so, **Answer Set Programming** (ASP) was born.

While an ASP program can have variables and functions, it must be grounded before it's *Answer Sets* can be found. A Grounded ASP program is a finite set of rules of the form $h \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$, where h and each L_i is a literal (i.e. an atom, a, or its classical negation, also called explicit negation, $\neg a$). The symbol *not* represents **default negation** or **negation as a failure to prove**. In a rule r, "h" is the **head**, represented by $Head(r)$ and "$L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$" is the **body**, represented by $Body(r)$. The literals $L_1, \ldots, L_m$ form the **positive body**, $Body^+(r)$ and $L_{m+1}, \ldots, L_n$ form the **negative body** of the rule, $Body^-(r)$.

Rules without heads are called **restrictions**, "$\leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$".

A set of literals M is a **model** of program P if for every rule r of the program, if $Body^+(r) \subseteq M$ and $Body^-(r) \nsubseteq M$, then $Head(r) \in M$. A model can't have both an atom a and $\neg a$. We say a model M^* is a **minimal model** if it's minimal relative to set inclusion, that is, there is no model M such that $M \subset M^*$. Programs where in all rules the negative body is empty have an unique minimal model [8]. That is not true in general for an arbitrary program.

To define an *Answer Set* of a grounded ASP program, first we must define the reduction of a program.

Definition 1 (from [1])**.** Let M be a finite set of atoms of P, the program P_M, obtained from P by removing:

- all the rules that have a literal A in their negative body if $A \in M$;
- the negative body of the remaining rules

is called **reduction of** P **by** M.

We can verify that the reduction of a program P results in a program where in all rules the negative body is empty.

978-0-7695-5092-3/13 $26.00 © 2013 IEEE
DOI 10.1109/BRACIS.2013.33

Therefore, there is always an unique minimal model for the reduction of a program.

Definition 2. Let P_M be the reduction of the program P by M, M is an ***Answer Set*** of P if the minimal model of P_M is M.

An useful extension to the traditional ASP framework is the weight constraint rule, introduced in [9].

First, let's define weight constraints. A weight constraint has the form:

$$L \leq \{h_1 = w_1, \ldots, \textit{not } h_n = w_n\} \leq U. \tag{1}$$

where h_i are literals or default negated literals, L and U are integers, called **lower limit** and **upper limit** respectively, and $w_1, \ldots, w_n$ are called **weights** associated to a literal, also represented by $weight(h_i)$.

Intuitively, a weight constraint is satisfied if the sum of the weights of the satisfied literals is between L and U.

Example 3. Consider the weight constraints

$$C_1: \quad 2 \leq \{a = 1, b = 2, \textit{not } c = 1\} \leq 3$$

$$C_2: \quad 1 \leq \{\textit{not } a = 2, b = 1, c = 1\} \leq 2$$

The set $\{a, b\}$ satisfies C_2 and not C_1.

A **weight rule** has the form:

$$C_0 \leftarrow C_1, \ldots, C_n \tag{2}$$

where $C_0, \ldots C_n$ are weight constraints and in C_0 there are no default negated literals. A weight constraint rule is satisfied if when $C_1, \ldots, C_n$ are satisfied, C_0 also is.

The reduction (Definition 1) of a weight constraint is a little different. We remove the upper limit and the default negated literals, subtracting the weights of the satisfied default negations.

Definition 4. Let P be a program with weight constraints and M a set of literals appearing on this program, the **reduction of a weight constraint** C of the form (1) is

$$C_M = L' \leq \{h_i = w_i | h_i \in M\}$$

where

$$L' = L - \sum_{h_i \notin M} weight(\textit{not } h_i)$$

For the weight rules, we have:

Definition 5. Let M be a set o literals, the **reduction of a weight rule** of the form (2) is the set of rules.

$$R_M = \begin{cases} \emptyset & \text{if } \exists C_{i \geq 1} : M \not\models C_i \\ \{h \leftarrow C_{iM} | h \in M \cap C_0\} & \text{otherwise} \end{cases}$$

where $M \not\models C$ symbolizes that M does not satisfies the constraint C and C_{iM} is the collection of the reductions of every weight constraint in the body of the rule.

We define a **weight rule's model** and an *Answer Set* for programs with weight rules in an analogous manner of their regular counterparts, with the extra restriction that an *Answer Set* for a program with weight rules must satisfy every rule's upper limit.

The reduction of a weight constraint discards the upper limit but the *Answer Set* definition demands that it is respected. The reason for this strange definition is to allow for more efficient algorithms for discovering *Answer Sets* while maintaining the behavior expected by the intuition. Details can be seen in [10].

As seen in [11], weight rules (or disjunctive rules, that can be written as weight rules) can increase the expressive power of the language and finding *Answer Sets* in programs with weight rules is $\sum_2^P$-complete, while ASP without weight rules is NP-complete.

A result that should be observed is:

Theorem 6 (from [7])**.** *Given an ASP program S with Ψ as the set of all Answer Sets, by adding a new constraint rule, the new program S' will have Answer Sets $\Psi' \subseteq \Psi$.* □

This theorem shows that by adding constraints to an ASP program we do not add new *Answer Sets*, we only remove those that violate the constraint. This holds for both regular constraint rules and weight rules with an empty head. This property will be used in the proposed PASP algorithm.

III. PSAT

The PSAT problem first appeared in 1854 studied by George Boole in [3], and was later rediscovered by several researchers.

The Probabilistic Satisfiability problem is the problem of deciding if a set probabilities associated to set of logic formulas is satisfiable, as defined in the following paragraphs.

Let's consider the classic propositional logic in the usual way. A PSAT instance is given by a set o formulas in propositional logic over a set of n variables, $S = \{s_1, \ldots, s_k\}$, and a set of probabilities $P = \{p_1, \ldots, p_k\}$ with $0 \leq p_i \leq 1$ for all i.

A **discrete probability distribution** over an enumerable set X is a function π that associates each element of this set with a probability in such a way that $\sum_{x \in X} \pi(x) = 1$.

Given a probability distribution over all S set's formulas' interpretations, we can define the **probability of a formula** as the sum of the probabilities of the interpretations that satisfy this formula. Formally, if π is a probability distribution over the set of interpretations, I_S, and $i(s) = 1$ iff i is a valid interpretation of the formula $s \in S$, formula s's probability is:

$$p(s) = \sum_{i \in I_s} \{\pi(i) | i(s) = 1\} \tag{3}$$

Intuitively, a formula's probability is sum of the probabilities of every "possible world" where the formula holds.

Definition 7. Let $I_S = \{i_1, \ldots, i_{2^n}\}$ be the set of possible interpretations over the variables of S, we say that the probability association $p(s_i) = p_i$ is **satisfiable** if there exists a probability distribution π over I_S such that Equation (3) holds.

Example 8. Consider the following PSAT instance:

$$P(a \vee b \vee c) = 1$$
$$P(a \wedge b) = 0.61; P(a \wedge c) = 0.60; P(b \wedge c) = 0.59$$

And consider the interpretations $v_1 = \{a = b = c = 1\}$, $v_2 = \{a = b = 1; c = 0\}$, $v_3 = \{a = c = 1; b = 0\}$ and $v_4 = \{a = 0; b = c = 1\}$.

The probability distribution π that associates the following values to these interpretations: $\pi(v_1) = 0.4; \pi(v_2) = 0.21; \pi(v_3) = 0.2; \pi(v_4) = 0.19$ is a solution to this PSAT instance, as we can verify by replacing these values in the equations below, that are instances of the Equation 3 for each formula:

$$\pi(v_1) + \pi(v_2) + \pi(v_3) + \pi(v_4) = 1 \quad (4)$$
$$\pi(v_1) + \pi(v_2) = 0.61 \quad (5)$$
$$\pi(v_1) + \pi(v_3) = 0.6 \quad (6)$$
$$\pi(v_1) + \pi(v_4) = 0.59 \quad (7)$$

In matricial form, we have:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.4 \\ 0.21 \\ 0.2 \\ 0.19 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.61 \\ 0.60 \\ 0.59 \end{bmatrix} \quad (8)$$

A PSAT instance can be expressed as a linear programming problem defining the matrix $A_{k \times 2^n} = [a_{ij}]$, such that $a_{ij} = 1$ iff an interpretation j satisfies the formula i and also defining the vector $p_{k \times 1} = [p_i]$. A PSAT instance is satisfied if there exists a vector π that is a solution to:

$$\begin{aligned} A\pi &= p \\ \pi &\geq 0 \\ \textstyle\sum \pi_i &= 1 \end{aligned} \quad (9)$$

The last restriction can be omitted by prefixing in A and p an entire line of ones.

If this system has a solution, thanks to Carathodory's lemma [12], it has a solution with only $k + 1$ nonzero elements, making PSAT a NP-complete problem.

It's possible to transform any PSAT instance into an Atomic Normal Form [13], possibly with new variables. In this form, a PSAT instance is partitioned in two sets, (Γ, Ψ) where Γ is a set of propositional formulas associated with the probability 1 and Ψ only has probabilities associated to atoms. This way we separate a PSAT problem in two problems: a SAT problem and the problem of finding a compatible probability distribution.

Example 9. The Example 8 in Atomic Normal Form is:

$$\Gamma = \left\{ \begin{array}{cccc} \neg x \vee a & \neg y \vee a & \neg z \vee b & a \vee b \vee c \\ \neg x \vee b & \neg y \vee c & \neg z \vee c & \\ \neg a \vee \neg b \vee x & \neg a \vee \neg c \vee y & \neg b \vee \neg c \vee z & \end{array} \right\}$$

$$\Psi = \{P(x) = 0.61; P(y) = 0.60; P(z) = 0.59\}$$

Furthermore, we can observe that in the matrix representation of a problem in the Atomic Normal Form, each element of the n-th column represents the truth value of a variable in Ψ in the n-th interpretation, except for the first line. This property is better explained in [7]. In the Example 8, the value of the variables of Ψ in each interpretation are $v_1 = \{x = y = z = 1\}$, $v_2 = \{x = 1; y = z = 0\}$, $v_3 = \{y = 1; x = z = 0\}$, and $v_4 = \{z = 1; x = y = 0\}$. And indeed, the values in each column of the matrix of the Equation 8 correspond to the values of the variables x, y and z respectively.

IV. PASP

In this paper, we extend Answer Set Programming with information about probability. We call this extension Probabilistic Answer Set Programming, or PASP.

A. Definition

Like PSAT, PASP is a decision problem where we must decide whether a set of probabilities is **satisfiable** by the rules of a program. The input of a PASP problem is a grounded ASP program, S, without classical negation and with a Herbrand base $HB_s = \{a_1, \ldots, a_n\}$, and a set of probabilities $P = \{p_1, \ldots, p_k\}, k \leq n$, associated to a subset of HB_S. We assume the probabilities are rational numbers.

The formal definition of satisfiability in PASP resembles PSAT's concept of satisfiability.

Definition 10. Let $2^{HB_S} = \{v_1, \ldots, v_N\}$ be the set of all subsets of HB_S (the candidates to *Answer Set* of S).

We say a set of probabilities $P = \{P(a_i) = p_i | 1 \leq i \leq k\}$ is **satisfied** by S if there is a probability distribution π over 2^{HB_S} where $p_i = \sum \{\pi(v_l) | a_i \in v_l$ and v_l is *Answer Set* of $S\}$.

If the set P contains a single literal, PASP reduces to finding one *Answer Set* with that literal.

Like the PSAT, we can write a PASP problem in matrix form, defining $A_{k \times 2^{\|HB_S\|}} = [a_{ij}]$, such that $a_{ij} = 1$ if the j-th atom subset contains the i-th atom; we force $\pi_j = 0$ if the j-th column of A is not an *Answer Set* of P. This way the criterion for deciding the satisfiability of a PASP instance becomes:

$$\begin{aligned} A\pi &= p \\ \pi &\geq 0 \\ \textstyle\sum \pi_i &= 1 \end{aligned} \quad (10)$$

Once again we can eliminate the last criteria by adding a line of ones.

PASP closely relates to PSAT instances in Atomic Normal Form. In both problems, probabilities are associated only with atoms/literals and an instance can be partitioned in two parts: a SAT/ASP problem and finding a compatible probability distribution. Furthermore, the property that the n-th column represents the truth value of the variables in the n-th interpretation still holds.

Proposition 11. *If there exists an solution for a PASP instance, there is a solution with at most $k+1$ nonzero elements in π.*

This Proposition follows directly from Carathodory's Lemma.

B. Related work

There have been other works that extend Answer Set Programming with probabilities, such as **P-log** [14], [15].

The way P-log tackles probabilities is radically different form PASP's. P-log works with causal probabilities (intuitively, probabilities that are independent of any factor other that it's cause), observations, and actions. Its focus is knowledge representation and inference mechanism over Bayesian networks.

On the other hand, PASP's focus is hard and soft constraints and there is no probability independence or dependence hypothesis.

Therefore, even though both PASP and P-log are both methods of extending ASP with probabilities, their focus are very different and they should be used for different tasks.

V. Resolution methods

We present a PASP-solving algorithm based on linear programming with column generation.

A. Solving via Linear Programming

This method is similar to one of the methods in [13], but adapted to account for PASP's peculiarities, such as non-monotonicity.

Since a PASP instance can be written as a Linear Programming problem without a cost function, we can reduce the problem of solving the PASP instance to the problem of finding a basic viable solution where the columns correspond to *Answer Sets*.

For this end, we will use the first phase of the Simplex algorithm [16]. We introduce $n+1$ artificial variables, ψ, where n is the size of the probability vector. Note that those are not variables from ASP's point of view, but from the Linear Programming point of view.

In the Simplex algorithm we use the Identity matrix as a basic matrix. While it's possible to use it, a better choice in dealing with probability constraints is a matrix of form (11).

$$I^* = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 0 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \tag{11}$$

Since the matrix $[I^*|\pi]$ is in Echelon form [17], the system always has a solution. If the probability vector is decreasingly ordered, this solution is non-negative.

We need a way to select columns to enter the base so that we can minimize the probability mass associated to the artificial variables that are not *Answer Sets*, $\sum \psi_i$. The selected column must represent an *Answer Set* and must have negative reduced cost. In Section V-B we see methods for selecting these columns. Considering the existence of a $selectColumn$ method, to be detailed later, Algorithm 1 presents this resolution method.

Algorithm 1 Method for solving PASP

Require: An ASP Program S, a set of k positive literals l, and a set of k probabilities, p.
  $B \leftarrow I^*_{(k+1)\times(k+1)}$ /* Initial Base */
  $c_B \leftarrow [1 \dots 1]_{k+1}$
  for $j = 1$ **to** k **do**
    if $Satisfied(B_j, S)$ **then**
      $c_{B_j} \leftarrow 0$
    end if
  end for
  $\pi_B \leftarrow B^{-1}p$ /* Initial Basic Feasible Solution */
  while $c'_B\pi_B > 0$ **do**
    $z \leftarrow SelectColumn(S, l, B, c_B)$
    if $z = \emptyset$ **then**
      return UNSAT
    end if
    $ChangeBase(B, z, \pi, c_B)$
    $\pi \leftarrow B^{-1}p$ /* New Basic Feasible Solution */
  end while
  return B, π_B

The function $ChangeBase$ replaces some column of B with column z and updates the cost vector, c_B. This is a standard procedure in linear programming algorithms and represents that the *Answer Set* represented by column z is assigned a non-zero probability.

$Satisfied$ verifies if the column B_j represents an *Answer Set*. This is made easier because the values in each column represent the values of the variables in the corresponding set. But since not all literals are associated with probabilities, to implement $Satisfied$ we cannot simply verify if a set is an *Answer Set* of S. Instead we must use an ASP Solver to find an *Answer Set* with or without a certain literal. Due the non-monotonicity of ASP, according Theorem 6 a possible way is to use the following restrictions.

$$
\begin{array}{l}
\leftarrow L_1 \\
\vdots \\
\leftarrow L_m \\
\leftarrow \textit{not } L_{m+1} \\
\vdots \\
\leftarrow \textit{not } L_k
\end{array}
$$

where $L_1, \ldots, L_m$ are the literals that should not appear on the *Answer Set*, i.e. the literals that are 0 in B_j, and $L_{m+1}, \ldots, L_k$ are the literals that must appear on the *Answer Set*, i.e. the literals that are 1. We add these restrictions to the ASP program S, run the ASP Solver, and get the result.

The main loop of Algorithm 1 persists while $c'_B\pi$ is positive. If stops if the problem is unsatisfiable or when a zero cost has been reached. Let illustrate this process with an example.

Example 12. Consider the following ASP program S:

$$
\begin{array}{rcl}
0 \leq \{a = 1\} \leq 1. & & \\
b, c & \leftarrow & a. \\
b & \leftarrow & \textit{not } a. \\
c & \leftarrow & \textit{not } a.
\end{array}
$$

Associated with the probabilities $p(b) = 0.7$ and $p(c) = 0.4$.

Let's solve it using Algorithm 1. First we initialize the matrix

$$
\begin{bmatrix}
1 & 1 & 1 \\
0 & 1 & 1 \\
0 & 0 & 1
\end{bmatrix}
$$

The second line represents b and the third line represents c, the atoms with probabilities. We see that only the second and third columns are consistent, for S has an *Answer Set* where b occurs and c does not (namely, $\{a, b\}$) and where both occur (namely, $\{b, c\}$), but not where both b and c are absent; thus the cost vector c_B has the value $[1, 0, 0]'$. We compute π such that:

$$
\begin{bmatrix}
1 & 1 & 1 \\
0 & 1 & 1 \\
0 & 0 & 1
\end{bmatrix}
.
\begin{bmatrix}
0.3 \\
0.3 \\
0.4
\end{bmatrix}
=
\begin{bmatrix}
1 \\
0.7 \\
0.4
\end{bmatrix}
$$

which has a total cost $c'_B \cdot \pi = 0.3$.

The implementation of $SelectColumn$ will be describe ahead. Assume it return the *Answer Set* $\{a, c\}$. The function $ChangeBase$ would swap the first column with column $[1, 0, 1]'$ representing this *Answer Set*:

$$
\begin{bmatrix}
1 & 1 & 1 \\
0 & 1 & 1 \\
1 & 0 & 1
\end{bmatrix}
.
\begin{bmatrix}
0.3 \\
0.6 \\
0.1
\end{bmatrix}
=
\begin{bmatrix}
1 \\
0.7 \\
0.4
\end{bmatrix}
$$

that has a total cost of 0. Therefore, the problem is solved, in this case with only one interaction.

$SelectColumn$ can be written as simply calculating the reduced cost of all *Answer Set* of S, but let's see a more efficient method ahead.

B. Column generation

To obtain columns with negative reduced cost, we can use a **Turing Reduction** [18] to ASP. This method is based on the method in [5] to solve the PSAT problem.

To say an *Answer Set* j have a negative reduced cost is equivalent to say that $\bar{c}_j = c_j - c'_B B^{-1} A_j < 0$. Since $c_j = 0$ for all *Answer Sets* we only need that

$$
c'_B B^{-1} A_j > 0 \tag{12}
$$

For this *Answer Set*.

To obtain such *Answer Set* we generate a new ASP program S' in whose *Answer Sets* also are *Answer Sets* of S and furthermore, the reduced cost is negative. Because of Theorem 6, if we can express this requirement as a set of restrictions, we can simply concatenate S with these restrictions.

We show two ways of expressing the inequality (12) as a set of restrictions.

1) Weight rules: Let u be the vector $c'_B B^{-1}$ and $l_1, \ldots, l_k$ be the value of S's literals associated with probabilities. Then the inequality $c'_B B^{-1} A_j > 0$ can be expressed as:

$$
u_0 + \sum_{i=1}^{k} u_i l_i > 0 \tag{13}
$$

u_0 always has a coefficient of 1 because of the value 1 in the first position of the vector A_j.

Considering the semantics of weight constraints, the rule that must be add to S so that any *Answer Set* of the resulting program have negative reduced cost is:

$$
\leftarrow \{l_1 = u_1, \ldots, l_k = u_k\} \leq -u_0
$$

Some ASP solvers only allow integer weights. Nevertheless, since $u = c'_B B^{-1}$, all elements of c_B are 0 or 1, $B^{-1} = \frac{Adj(B)}{det(B)}$ (where $Adj(B)$ is the adjunct matrix of B and $det(B)$ is it's determinant) and the elements of $Adj(B)$ are integers, if we multiply u by $det(B)$ we obtain integer weights.

2) Inequalities as SAT instances: It was shown in [19] how to transform any binary linear inequality with integer non-negative coefficients into propositional logic formulas in linear time.

The coefficients on Inequality (13) can be turned into integers by multiplying then by $det(B)$, as shown in the previous section. We need to know the maximum number of bits to represent the coefficients. Using the Hadamard inequality [20, problem 523] we know that the number of bits of sum of the coefficients is limited by $\log_2 k^2 (k+1)^{\frac{k+1}{2}} / 2^k$.

With these information, we can generate SAT formulas in CNF that are satisfiable iff the reduced cost of the model is negative. Let the generated formula be in the form:

$$(a_{11} \vee \cdots \vee a_{n1}) \wedge \cdots \wedge (a_{1m} \vee \cdots \vee a_{jm})$$

We express it in the S' program as the restrictions bellow:

$$\leftarrow \textit{not } a_{11}, \ldots, \textit{not } a_{n1}.$$
$$\vdots$$
$$\leftarrow \textit{not } a_{1m}, \ldots, \textit{not } a_{jm}.$$

VI. Experimental results

An implementation of the resolution method described in V-A utilizing weight rules is available[1] under GPLv3 license.

We run several tests to analyze the behavior of this method. The tests were run on a computer with 12 Intel Core i7 processors and 48Gb of RAM running Ubuntu Linux 12.04. The test consists of randomly generated PASP instances. The ASP rules where generated according to the procedure show in [21] to create 3-LP programs varying in the number of literals and number of rules. The generated probabilities vary in the number of literals with probabilities and in their range. For each possible configuration, 200 instances were generated resulting in a total of 70,200 tests.

From all these tests, 7 instances, or 0.009% of the tested instances, result in a degenerated matrix and loop forever. Anti-cycling techniques for the simplex algorithm are well know but the statistical insignificance of the cases where it occurred led us to believe that an anti-cycling mechanism was not needed for our tests.

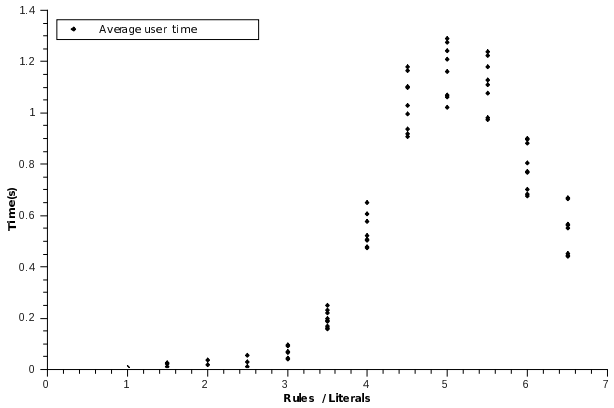

Figure 1. Dispersion graph of the tests with 150 literals

The complete results can be seen found [7]. In all configurations a trend similar to what can be seen in the Graph 1 appear. This trend is an "easy-hard-easy" pattern, characteristic of phase transitions.

[1]at the repository git://gitorious.org/pasp/pasp-asp.git.

The phase transition is a phenomenon found in many NP-complete problems where a region surrounding certain critical values of some parameter is much more difficult than the adjacent regions, usually when the probability of a "accepting" answer is 50%. In [21] we see that for 3-LP ASP programs with 150 literals the harder instances are found when the ratio between the number of rules and the number of literals is close to 5, but this value does not correspond to a 50% probability of being "accepted". This seems to repeat in the PASP case with 3-LP programs.

From the 70,200 tests, only 175 PASP instances, 0.249%, are satisfiable and almost all of then have a rule-literal ratio of 0.5, not being significantly influenced by the value of the probabilities.

Since, unless the probabilities equal 1, more than one *Answer Set* is needed for a PASP instance to be satisfiable and according to [21], the probability of the existence of an *Answer Set* abruptly falls from 100% to 30% when the rules and literals ration goes from 0 to 1, it's possible that showing the existence of a single *Answer Set* was most of the work for these instances, which explains the similarity between the time curves for ASP and PASP.

Therefore, for the tested instances, the run time was dominated by the time of running a single ASP Solver instance.

VII. Conclusion

In this work, we have defined the problem of probabilistic answer set programming, proposed an algorithm for it and shown the feasibility of its implementation.

Future work involves improving algorithms and modeling of larger problems involving hard and soft constraints, in the line of [6].

References

[1] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," in **International Conference on Logic Programming/Joint International Conference and Symposium on Logic Programming**, 1988, pp. 1070–1080.

[2] V. W. Marek and M. Truszczyński, "Stable models and an alternative logic programming paradigm," in **In The Logic Programming Paradigm: a 25-Year Perspective**. Springer, 1999, pp. 375–398.

[3] G. Boole, **An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities**. Walton and Maberly, 1854. [Online]. Available: http://books.google.com/books?id=SWgLVT0otY8C

[4] N. J. Nilsson, "Probabilistic logic," **Artif. Intell.**, vol. 28, no. 1, pp. 71–87, 1986.

[5] M. Finger and G. De Bona, "A logic based algorithm for solving probabilistic satisfiability," in **Proceedings of the 12th Ibero-American conference on Advances in artificial intelligence**, ser. IBERAMIA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 453–462.

[6] M. Finger, R. L. Bras, C. Gomes, and B. Selman., "Solutions for hard and soft constraints using optimized probabilistic satisfiability," **Accepted for SAT2013**, 2013.

[7] E. M. de Morais, "Probabilistic answer set programming (in portuguese)," Master's thesis, Instituto de Matemática e Estatística da Universidade de São Paulo, 2012.

[8] M. H. Van Emden and R. A. Kowalski, "The semantics of predicate logic as a programming language," **J. ACM**, vol. 23, no. 4, pp. 733–742, Oct. 1976.

[9] I. Niemel, P. Simons, and T. Soininen, "Stable model semantics of weight constraint rules," in **Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR99), Volume 1730 of lecture**. Springer-Verlag. LNAI, 1999, pp. 317–331.

[10] T. Syrjnen, **Lparse 1.0 User's Manual**, 2000, disponvel em: ¡http://www.tcs.hut.fi/Software/smodels/lparse.ps¿. Acesso em: 6 Nov. 2010.

[11] R. Ben-Eliyahu and R. Dechter, "Propositional sematics for disjunctive logic programs." in **JICSLP'92**, 1992, pp. 813–827.

[12] V. Prasolov and V. Tikhomirov, **Geometry**. American Mathematical Society, julho 2001.

[13] G. de Bona, "Probabilistic satisfiability (in portuguese)," Master's thesis, Instituto de Matemática e Estatística da Universidade de São Paulo, 2011.

[14] C. Baral, M. Gelfond, and N. Rushton, "Probabilistic reasoning with answer sets," in **LPNMR**, ser. Lecture Notes in Computer Science, V. Lifschitz and I. Niemel, Eds., vol. 2923. Springer, 2004, pp. 21–33.

[15] ——, "Probabilistic reasoning with answer sets," **Theory Pract. Log. Program.**, vol. 9, no. 1, pp. 57–144, Jan. 2009.

[16] D. Bertsimas and J. N. Tsitsiklis, **Introduction to Linear Optimization**. Belmont, Massachusetts, EUA: Athena Scientific, 1997.

[17] E. W. Weisstein, "Echelon form. From MathWorld—A Wolfram Web Resource," 2012, http://mathworld.wolfram.com/EchelonForm.html.

[18] H. Rogers, **Theory of recursive functions and effective computability**, ser. McGraw-Hill series in higher mathematics. McGraw-Hill, 1967.

[19] J. P. Warners, "A linear-time transformation of linear inequalities into conjunctive normal form," **Information Processing Letters**, vol. 68, no. 2, pp. 63–69, 1998.

[20] D. Faddeev and I. Somins'kyi, **Book of Problems in Higher Algebra**. Vyshcha Shkola, 1971.

[21] Y. Zhao and F. Lin, "Answer set programming phase transition: A study on randomly generated programs," in **Logic Programming**, ser. Lecture Notes in Computer Science, C. Palamidessi, Ed. Springer Berlin / Heidelberg, 2003, vol. 2916, pp. 239–253.