Abduction in (Probabilistic) Answer Set Programming

Damiano Azzolini[1], Elena Bellodi[2] and Fabrizio Riguzzi[3]

[1]*Dipartimento di Scienze dell'Ambiente e della Prevenzione, University of Ferrara, Ferrara, Italy*
[2]*Dipartimento di Ingegneria, University of Ferrara, Ferrara, Italy*
[3]*Dipartimento di Matematica e Informatica, University of Ferrara, Ferrara, Italy*

Abstract
Answer Set Programming (ASP) is a branch of Logic Programming particularly useful for representing complex domains. Logic abduction, the reasoning strategy that deals with incomplete data, is tightly related to ASP, and encodes incompleteness through abducibles. The goal of logic abduction is to find the minimal set of abducibles (where minimality is usually considered in terms of set inclusion) that explains a query. Recently, abductive reasoning has been introduced in the context of Probabilistic Logic Programming, but no solutions are available for Probabilistic Answer Set Programming (PASP). In this paper, we close this gap and propose an algorithm to perform abduction both in ASP and in PASP.

Keywords
Abduction, Statistical Relational Artificial Intelligence, Probabilistic Answer Set Programming

1. Introduction

Abductive Logic Programming (ALP) [1, 2] is an extension of Logic Programming (LP) [3] that copes with incomplete data: given a set of *abducible* facts and a set of *integrity constraints*, the goal is to find the *minimal* set, where minimality is usually considered in terms of set inclusion, that explains a given query. This minimal set is often called *abductive explanation*.

Answer Set Programming (ASP) [4] is a powerful formalism to encode complex problems, where the possible solutions are represented as *answer sets*. In the first contribution of this paper, we propose a simple yet effective algorithm to perform abduction in ASP.

One limitation of ASP (and Logic Programming in general) is that it cannot manage uncertain data. Probabilistic Logic Programming (PLP) [5, 6] under the Distribution Semantics (DS) [7] is a possible formalism to express uncertain information using a logic-based language. Recently, the authors of [8] introduced the concept of *probabilistic abductive explanation* and proposed an algorithm to perform abduction in PLP under the DS. A possible extension to ASP that manages uncertainty is *Probabilistic* Answer Set Programming (PASP) under the Credal Semantics (CS) [9, 10]. With this semantics, the probability of a query is not a sharp value, but it is represented by an interval, i.e., it has a lower and an upper probability bound. In a second contribution of this paper, we propose an algorithm to perform abduction in PASP, where the goal is to find, given a query, the minimal set of abducible facts that maximizes the joint *lower probability* of the query and the integrity constraints.

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
damiano.azzolini@unife.it (D. Azzolini); elena.bellodi@unife.it (E. Bellodi); fabrizio.riguzzi@unife.it (F. Riguzzi)

Overall, our contribution is two-fold: first, we provide an algorithm to compute abductive explanations in ASP. Then, we introduce the concept of abductive reasoning in PASP under the CS and propose an algorithm to compute probabilistic abductive explanations. To study the applicability of our approach, we tested both algorithms on three different datasets. The results show that abduction for ASP is orders of magnitude faster than for PASP.

The paper is structured as follows: Section 2 introduces the basic concepts of ASP, PLP, and PASP under the CS. Section 3 describes our algorithm to perform abduction in ASP while our proposal for PASP under the Credal Semantics is presented in 4. The two algorithms are tested in Section 5, related work is discussed in Section 6, and Section 7 concludes the paper.

2. Background

2.1. Answer Set Programming

We assume the reader is already familiar with the basic notions of Logic Programming. For an in-depth treatment of the subject, see [3]. ASP also considers aggregate atoms [11] of the form $g_0 \circ_0 \#f\{e_1; ...; e_n\} \circ_1 g_1$ where g_0 and g_1 are constants or variables and are called *guards*, f is an aggregate function symbol, and $\circ_0$ and $\circ_1$ are arithmetic comparison operators. Each e_i is an expression of the form $t_1, ..., t_l : C$ where each t_i is a term whose variables appear in C, a conjunction of literals. For example, a possible aggregate is $1 < \#count\{X : f(X)\} < 5$. $g_0 \circ_0$ or $\circ_1 g_1$ or both may be omitted.

A rule is of the form $h_1; ...; h_n \leftarrow b_1, ..., b_m$, where each h_i is an atom and each b_i is a literal. $h_1; ...; h_n$ is called the *head* and $b_1, ..., b_m$ is called the *body*. We only consider *safe rules*, i.e., rules where each variable of a rule also appears in a positive literal in the body. If $n = 0$ (no atoms in the head) and $m > 0$, the rule is called an *integrity constraint*. If $n = 1$ and $m = 0$, the rule is called a *fact*, and represents what is known to hold. If a rule does not contain variables, it is called *ground*. The set of groundings of a rule can be obtained by replacing its variables with the constants that appear in the program in all possible ways. An answer set program is a set of rules.

The Herbrand base (B_P) of an answer set program P is the set of all ground atoms that can be constructed using the symbols in the program. An *interpretation* I for P is a subset of B_P. I satisfies a ground rule if at least one h_i is true in I when every b_i is true in I. A *model* is an interpretation that satisfies all the groundings of all the rules of P. If we consider a ground program P_g and an interpretation I, by removing from P_g the rules in which a b_i is false in I we obtain the *reduct* [12] of P_g with respect to I. An *answer set* for P is an interpretation I such that I is a minimal model (in term of set inclusion) of the reduct of P_g. With $AS(P)$ we denote the set of all the answer sets of P.

2.2. Probabilistic Logic Programming (PLP)

Probabilistic Logic Programming [5, 6] extends Logic Programming by incorporating probabilities into facts or rules. If we consider, for instance, ProbLog [13], a probabilistic fact is represented with $\Pi :: f$ where $\Pi \in]0, 1]$ and f is a logical atom. The Distribution Semantics (DS) [7] is at the heart of many PLP languages, such as ProbLog. Following the DS, an *atomic* *choice* is represented with a tuple (f, θ, k) where $k \in \{0, 1\}$: $k = 1$ means that the grounding $f\theta$ for f is selected; $k = 0$ indicates that it is not. A consistent set of atomic choices is called a *composite choice* (identified with κ) and its probability can be computed as

$$P(\kappa) = \prod_{(f_i,\theta,1)\in\kappa} \Pi_i \cdot \prod_{(f_i,\theta,0)\in\kappa} (1 - \Pi_i)$$

If a composite choice contains an atomic choice for every grounding of every probabilistic fact then it is called a *selection*. A selection identifies a *world*, i.e., a logic program composed by the rules of the program and the selected probabilistic facts (those for which $k = 1$). The probability of a world w, $P(w)$, is the probability of the correspondent selection. Finally, the probability of a *query* q (a conjunction of ground atoms) is computed as

$$P(q) = \sum_{w \models q} P(w)$$

Probabilistic facts are considered independent. For example, the following (propositional) program

```
0.3::nosleep.
0.6::lowvitamins.
tired:- nosleep.
tired:- lowvitamins.
```

has 2 probabilistic facts: `nosleep`, that is true with probability 0.3, and `lowvitamins`, that is true with probability 0.6. The program has $2^2 = 4$ worlds; the query `tired` is true in 3 of them (those where at least one of the two probabilistic facts is true) and it has probability $0.3 \cdot 0.6 + 0.3 \cdot (1 - 0.6) + (1 - 0.3) \cdot 0.6 = 0.72$.

2.3. Probabilistic ASP under the Credal Semantics

The Distribution Semantics only considers probabilistic logic programs where every world has a unique two-valued well-founded model [14]. This usually does not hold if we consider ASP programs with probabilistic facts (PASP).

In this case, the Credal Semantics (CS) [9, 10] has been proposed as a possible underlying semantics. Under this semantics, every query q has lower and upper probability bounds, denoted respectively with $\underline{P}(q)$ and $\overline{P}(q)$. In addition to the worlds, the computation of the probability for a query also requires considering the answer sets for each of them: if the query is true in *at least one* answer set of a world w, $P(w)$ contributes to the upper probability; if the query is true in *every* answer set of a world w, $P(w)$ contributes to the lower probability. Clearly, $\underline{P}(q) \leq \overline{P}(q)$. However, every world must have at least one answer set. If we slightly modify the previous program in

```
0.3::nosleep.
0.6::lowvitamins.
tired:- nosleep.
tired; nottired:- lowvitamins.
```

and still consider the query `tired`, the worlds where both probabilistic facts are true and the worlds where `nosleep` is true and `lowvitamins` is false have only 1 answer set each, {`nosleep, tired, lowvitamins`} and {`nosleep, tired`} respectively, and the query is true in them, so we have a contribution of $0.3 \cdot 0.6 + 0.3 \cdot (1 - 0.6)$ to the lower probability. The world where `lowvitamins` is true and `nosleep` is false has 2 answer sets ({`tired, lowvitamins`} and {`nottired, lowvitamins`}) and the query is true in only one of the two, so we get a contribution of $(1 - 0.3) \cdot 0.6$ to the upper probability. In the world where both probabilistic facts are false, the query is false as well so it does not contribute to the probability. Overall, the probability lies in the range $[0.3, 0.72]$.

3. Abductive Answer Set Programming

An *abductive answer set program* is composed of an answer set program and a set of ground atoms called the *abducibles*, that do not appear in the head of any rule. More formally, given an answer set program P, and a possibly empty set of abducibles (also called abducible facts) A, the goal is to find the minimal set of abducibles Δ, called *abductive explanation*, such that a *query* is present in *at least one* answer set. If there are multiple minimal sets, we call them the abductive explanations. Here, minimality is intended in terms of set inclusion [15]. For example, if both $\Delta' = \{\texttt{a}\}$ and $\Delta'' = \{\texttt{a,b}\}$ are explanations for a query q, only Δ' is considered as the abductive explanation, since $\Delta'' \supset \Delta'$, and thus Δ'' is not minimal. We consider integrity constraints (ICs) as normal answer set constraints. To better illustrate these concepts, consider the following example.

Example 1 (Smoke). *The program and the graph of Figure 1 describe a network with 5 people (nodes) connected by a friendship relation (edges). The friendship relation exists if the corresponding abducible (denoted by prepending the functor `abducible` to the terms `e/2`) is selected. Some individuals smoke, some others do not. In particular, `b` and `d` smoke. A disjunctive rule states that a person `X` can either smoke or not smoke given that she is a friend of someone (`Y`) who smokes. The integrity constraint states that at least 80% of the people smoke. The goal is to compute the abductive explanation(s) for the query `smokes(c)`.*

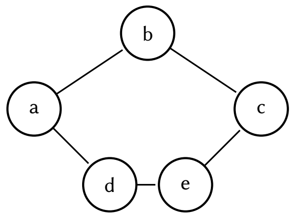
We propose an algorithm to perform abduction in ASP.

Every abducible fact `abducible a` is replaced by a choice rule of the form `0{a}1`, stating that `a` can be included or not in every answer set. To encode the query `query`, we add a constraint `:- not query`. In this way, we impose that the query is true in every answer set. By applying this transformation to the program shown in Figure 1a, we obtain a new program with 62 answer sets. Every answer set represents a possible explanation but only some of them (2) are abductive explanations, i.e., minimal (in terms of set inclusion), for the query `smokes(c)`: {{`e(b,c)`},{`e(d,e),e(e,c)`}}.

To compute the abductive explanations, a first solution could consist in enumerating all the answer sets and removing the dominated ones. However, the number of abductive explanations is usually orders of magnitude smaller than the possible answer sets (in this example, there are 2 abductive explanations, while the total number of answer sets is 62). For this reason we adopt an alternative approach. First, we compute the cautious consequences (intersection of all the

```
abducible e(a,b). abducible e(b,c).
abducible e(a,d). abducible e(d,e).
abducible e(e,c).

friend(X,Y):- e(X,Y).
friend(X,Y):- e(Y,X).

smokes(b). smokes(d).

smokes(X) ; nosmokes(X):-
    friend(X,Y), smokes(Y).

:- #count{X:nosmokes(X)} = N,
   #count{X:smokes(X)} = S,
   10*S < 8*(N+S).
```

(a) Program.



(b) Network of 5 people (nodes) connected by a friendship relation (edges).

Figure 1: Example of an abductive answer set program and its graph representation.

models) of the whole program, and project [16] them on the abducibles. For every abducible a in the cautious consequences, we add a constraint `:- a` in the program, since these are present in every answer set, so we can avoid generating a choice for them. After this process, we add an additional rule with an argument that stores the number of abducibles selected in the computed answer sets. For example, for the program shown in Example 1a, we add `c(C):- #count{Y,X : e(X,Y)} = C`. After that, we iteratively generate answer sets and, at each step, we add a constraint to the program that imposes that the number of abducibles in the generated answer sets is N, where N ranges from 0 to the total number of abducibles in the program. We go from 0 to N, so we can keep track of the dominated explanations. The answer sets obtained at each iteration are the abductive explanations. In other words, we call multiple times the solver to generate the answer sets and, at each call, we generate only the answer sets with a fixed number of abducibles. Moreover, at each iteration, we also impose a constraint to avoid the generation of answer sets that are supersets of the already computed ones. This process is described in Algorithm 1: first, abducibles are converted as previously described (line 2). Then, we compute the minimal set of abducible facts (line 3) and add each fact of this minimal set into the program, as integrity constraint (line 5). The loop at line 9 handles the generation of sets of abducibles of increasing size. The function AddConstraintSizeAndComputed adds a constraint to the program to limit the number of abducibles and a constraint to remove the already computed solutions. At the end of the loop, we check whether there are some solutions that are not minimal with the function Least and return the abductive explanation for the query.

To clarify the overall process, let us apply Algorithm 1 on the program of Example 1. There are no cautious consequences projected on the abducibles, so no constraint is added to the program. Then, we begin generating the answer sets, starting from 0 abducibles with the constraint `:- c(X), X != 0`. This program is unsatisfiable, so no further constraints are added. In the

Algorithm 1 Function AbductionASP: computation of the abductive explanations for a query *query* and an ASP program $\mathcal{P}$.

```
 1: function Abduction($query, \mathcal{P}$)
 2:     $probFacts, abducibles, P_p \leftarrow$ ConvertProgram($\mathcal{P}$)
 3:     $minSet \leftarrow$ ComputeMinimalSet($P_p \cup \{$:- $not\ query.\}$)
 4:     for all $f \in minSet$ do
 5:         $P_p \leftarrow P_p \cup \{$:- $not\ a.\}$
 6:     end for
 7:     $alreadyComputed \leftarrow \emptyset$
 8:     $abduciblesSet \leftarrow \emptyset$
 9:     for $i \in range(0, len(abducibles))$ do
10:         $P_p^c \leftarrow$ AddConstraintSizeAndComputed($P_p, i, alreadyComputed$)
11:         $P_p^{qc} \leftarrow P_p^c \cup \{$:- $not\ query.\}$
12:         $projectSet \leftarrow abducibles$
13:         $AS \leftarrow$ ProjectSolutions($P_p^{qc}, projectSet$)        ▷ Computation of the answer sets.
14:         for all $as \in AS$ do
15:             $abduciblesSet \leftarrow abduciblesSet \cup as$
16:             $alreadyComputed \leftarrow alreadyComputed \cup as$
17:         end for
18:     end for
19:     $abduciblesSet \leftarrow$ Least($abduciblesSet$)
20:     return $abduciblesSet .elements$
21: end function
```

case the program is satisfiable with answer sets of size 0, this means that there is no need to keep searching for abductive explanations since the empty explanation is sufficient, and we can stop the search. We now consider answer sets with 1 abducible, we remove the constraint for 0 abducibles and replace it with `:- c(X), X != 1`. There is one answer set, and thus an abductive explanation, {`e(b,c)`}, so a constraint of the form `:- e(b,c)` is added to the program. In this way, in the next iterations, we do not generate answer sets that contain this abducible, since it is already in a smaller explanation. That is, if a is an abductive explanation, all the subsequent explanations that contain a will be dominated, so they can be ignored. At the third iteration, we replace the constraint for 1 abducible with `:- c(X), X != 2`, while the constraints on the already discovered abducibles are kept. We obtain a new solution, {`e(d,e)`, `e(e,c)`}, so we add a new constraint to the program `:- e(d,e), e(e,c)`. We continue this process until considering 5 abducibles (all). At the end, we get {{`e(b,c)`}, {`e(d,e) e(e,c)`}} as abductive explanations. By default, abducibles not included into the abductive explanations are not selected. Note that, if we consider as minimality measure the number of abducibles in an answer set, we can solve this task by simply setting an optimization problem where the goal is to minimize the number of abducibles in the answer sets. That is, if we use, for example, the ASP system clingo [17], we can add the two rules:

```
c(C):- C = #count{X,Y : e(X,Y)}.
#minimize{C : c(C)}.
```

and get, as result, the minimal sets in terms of cardinality (`e(b,c)`).

In the next section we show how to extend this abductive framework in the case of *Probabilistic* Answer Set Programming.

4. Abductive Reasoning in Probabilistic Answer Set Programming

A *probabilistic* integrity constraint [8] is of the form

$$\Pi \leftarrow l_1, \ldots, l_n$$

where $\Pi \in]0,1]$ and each l_i is a literal (abducibles are allowed). Here, we also allow l_i to be an aggregate atom. Probabilistic facts and probabilistic integrity constraints identify the worlds, each of which may have multiple answer sets. We obtain a world by adding or not each ground probabilistic fact and each grounding of each probabilistic integrity constraint. For the grounding of the integrity constraints, we consider the standard concept of global and local variable [11]. In particular, a global variable of a rule is such that it appears in at least one literal not involved in aggregations. Variables only appearing in aggregates are called local. A ground instance of a rule with aggregates is obtained by first replacing global variables and then local variables with ground terms.

The probability of a world is given by the product of the atomic choices for the probabilistic facts with a factor Π_i for every probabilistic integrity constraint i selected and a factor $(1 - \Pi_j)$ for every probabilistic integrity constraint j not selected.

Definition 1 (Probabilistic abductive answer set program). *An answer set program P, a set of probabilistic facts F, a set of abducibles A, and a (possibly empty) set of probabilistic integrity constraints IC define a probabilistic abductive answer set program.*

Given a probabilistic abductive answer set program, the *lower joint* probability of the query q and the constraints IC given an explanation Δ is the sum of the probabilities of the worlds w where Δ is an explanation for the query q, all the constraints are satisfied, and q is true in all the answer sets. In formula:

$$\underline{P}(q, IC \mid \Delta) = \sum_{w:\forall m \in AS(P_w \cup \Delta), m \models q, m \not\models IC_{P_w}} P(w)$$

where P_w is the abductive answer set program identified by a word w and IC_{P_w} is the set of IC involved in P_w.

Finally, the goal of (cautious) abduction in probabilistic answer set programming is to find the minimal set of abducibles $\Delta \subseteq A$ such that $\underline{P}(q, IC \mid \Delta)$ is maximized, i.e., solve

$$\mathrm{least}(\arg\max_{\Delta} \underline{P}(q, IC \mid \Delta))$$

where, as in [8], the function least removes the sets that are not minimal. The main goal is to maximize the lower joint probability so, even if there are, for example, two solutions $\Delta' \subset \Delta''$ that yield respectively probabilities $P_{\Delta'} < P_{\Delta''}$, we only consider Δ'' as abductive explanation (despite being not minimal), since it gives the highest probability. Note that if a set Δ maximizes the lower joint probability, it may not maximize the upper joint probability. For example, if we consider the program:

```
abducible a.
abducible b.
0.5::fa.
0.5::fb.
query:- a,fa.
query;notquery:- b,fb.
```

the abductive explanation for the query `query` is $\Delta = \{\texttt{a}\}$ that yields a lower joint probability of 0.5, while the set of abducibles that maximizes the lower upper probability is $\{\texttt{a},\texttt{b}\}$ that yields an upper probability of 0.75. The goal of (brave) abduction in probabilistic answer set programming can be defined in a similar way by considering the upper probability. However, here we focus on cautious abduction and every time we write abduction in probabilistic answer set programming we consider the goal of cautious abduction.

Example 2 (Probabilistic Smoke). *Suppose now that the relationships are uncertain. To model this, we add a probabilistic fact* `fe/2` *for every abducible in Example 1. Moreover, we suppose that the information provided by the integrity constraint is also uncertain, and has an associated probability of 0.2. The program became:*

```
abducible e(a,b). abducible e(b,c).
abducible e(a,d). abducible e(d,e).
abducible e(e,c).

0.5::fe(a,b). 0.5::fe(b,c). 0.5::fe(a,d).
0.5::fe(d,e). 0.5::fe(e,c).

friend(X,Y):- e(X,Y), fe(X,Y).
friend(X,Y):- e(Y,X), fe(Y,X).

smokes(b). smokes(d).

smokes(X) ; nosmokes(X):-
    friend(X,Y), smokes(Y).

0.2:- #count{X:nosmokes(X)} = N,
      #count{X:smokes(X)} = S,
      10*S < 8*(N+S).
```

The goal is to compute the abductive explanation for the query `smokes(c)`*.*

Probabilistic integrity constraints require a particular conversion. For every probabilistic IC of the form `p:- body` we add a probabilistic fact `p::f`, a rule `ic :- body`, and two constraints `:- f, ic`, and `:- not f, not ic` imposing respectively that, if the fact is selected, the constraint must be true and, if the fact is not selected, the constraint must be false. This new probabilistic fact is parsed as previously described.

To find the abductive explanations in PASP we modified the algorithm described in the previous section. We cannot impose the constraint that removes an already found explanation e since another explanation $e^{'} \supset e$ with a higher associated probability can exist. Moreover, we cannot add the constraint `:- not query`, since we need to consider the lower probability, and so ensure that the query is true in *all* the models for a world. Finally, we need to identify the choices made for the abducibles at each iteration and compute the probability for each world. If we consider Example 2 with the query `smokes(c)`, the only probabilistic abductive explanation is the set {`e(b,c)`,`e(d,e)`,`e(e,c)`} that gives a lower probability of 0.125. This process is summarized in Algorithm 2: first, the probabilistic facts, and the probabilistic integrity constraints are converted as previously explained (line 2). Then, we compute the minimal set of probabilistic and abducible facts (line 3) and add each fact of this minimal set into the program, as integrity constraint (line 5). The function AddConstraintSize adds a constraint to the program to limit the number of abducibles. The functions ExtractAbdChoices and ExtractWorlds respectively extracts the set of abducibles and the set of probabilistic facts for every answer set. The function ComputeContribution computes the contribution to both lower and upper probability [18] for every choice of abducibles and the function UpdateSet keeps track of the best solutions found so far. At the end of the loop, we check whether there are some solutions that are not minimal with the function Least and return the probabilistic abductive explanation and the probability range for the query.

Algorithm 2 Function AbductionPASP: computation of the probabilistic abductive explanations for a query $query$ and a PASP program $\mathcal{P}$.

```
1: function Abduction($query, \mathcal{P}$)
2:     $probFacts, abducibles, P_p \leftarrow$ ConvertProgram($\mathcal{P}$)
3:     $minSet \leftarrow$ ComputeMinimalSet($P_p \cup \{$:- $not\ query.\}$)
4:     for all $f \in minSet$ do
5:         $P_p \leftarrow P_p \cup \{$:- $not\ a.\}$
6:     end for
7:     $alreadyComputed \leftarrow \emptyset$
8:     $abduciblesSet \leftarrow \emptyset$
9:     for $i \in range(0, len(abducibles))$ do
10:        $P_p^c \leftarrow$ AddConstraintSize($P_p, i$)
11:        $P_p^{qc} \leftarrow P_p^c \cup \{q$:- $query., nq$:- $not\ query.\}$
12:        $projectSet \leftarrow probFacts \cup abducibles \cup \{q\} \cup \{nq\}$
13:        $AS \leftarrow$ ProjectSolutions($P_p^{qc}, projectSet$)
14:        for all $as \in AS$ do
15:            $wa \leftarrow$ ExtractAbdChoices($as$)          ▷ Identify choices for the abducibles
16:            $wp \leftarrow$ ExtractWorlds($wa$)                ▷ Extract worlds
17:            $contributionsList \leftarrow$ ComputeContribution($wp$)
18:            $abduciblesSet \leftarrow$ UpdateSet($contributionsList$)
19:        end for
20:    end for
21:    $abduciblesSet \leftarrow$ Least($abduciblesSet$)
22:    return $abduciblesSet.lp, abduciblesSet.up, abduciblesSet.elements$
23: end function
```

5. Experiments

To evaluate our approach, we ran the proposed algorithm on a computer with Intel® Xeon® E5-2630v3 running at 2.40 GHz with 16Gb of ram and a time limit of 8 hours. We used the ASP solver clingo [17]. We tested three datasets, both for ASP and PASP[1].

The first dataset, `clauses`, encodes a domain with an increasing number of abducibles `ab/1`, a clause for every one of them and a constraint imposing that at least two abducibles should be selected. The structure for the program of size 4 (where the size indicates the number of abducibles) for ASP is the following:

```
abducible ab(1). abducible ab(2).
abducible ab(3). abducible ab(4).
qry:- ab(1). qry:- ab(2). qry:- ab(3). qry:- ab(4).
:- #count{X : a(X)} = C, C != 2.
```

The query is `qry`. For the PASP version we considered deterministic and probabilistic integrity constraints. In the first case half of the `ab/1` atoms are abducibles and half are probabilistic facts with an associated probability of 0.5. In the second case, the IC has an associated probability of 0.5. In both cases, the integrity constraints only involves the number of abducibles that can be selected. The remaining part of the program is the same. The goal of this experiment is to test the inference time of the algorithm on programs with an increasing number of clauses that must be considered for the computation of the (probabilistic) abductive explanation.

Results are shown in Figure 2. The execution times for PASP with deterministic and probabilistic integrity constraints are similar, and, in both cases, we get a memory error for size 26. Instead, the time required for the computation of abductive explanations in ASP is negligible with respect to PASP.

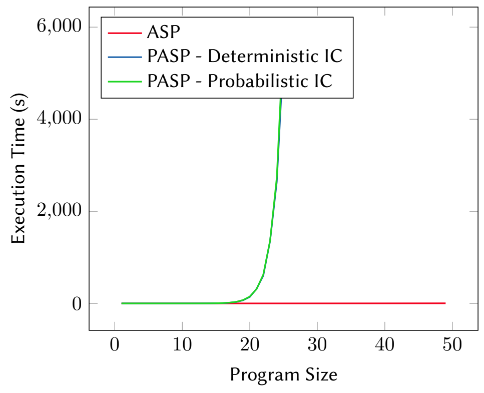


Figure 2: Inference times for the `clauses` experiments.

The second dataset, `bird`, encodes a small biological domain composed of birds. Each bird can either fly or not fly, and there are at least 60% of birds that fly. An example of program of size 4 (with 4 birds) is the following:

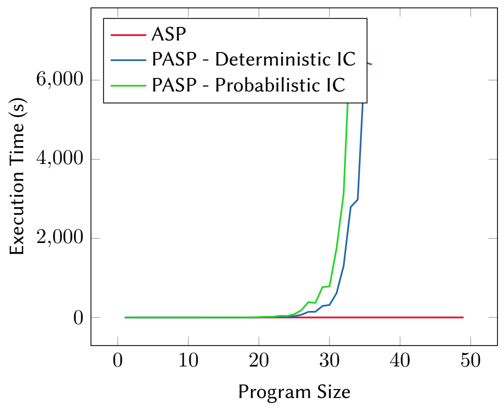
[1]Source code and datasets available at: https://github.com/damianoazzolini/pasta.

```
bird(1). bird(2). bird(3). bird(4).
fly(X);nofly(X):- bird(X).
:- #count{X:fly(X),bird(X)} = FB,
   #count{X:bird(X)} = B, 10*FB<6*B.
```

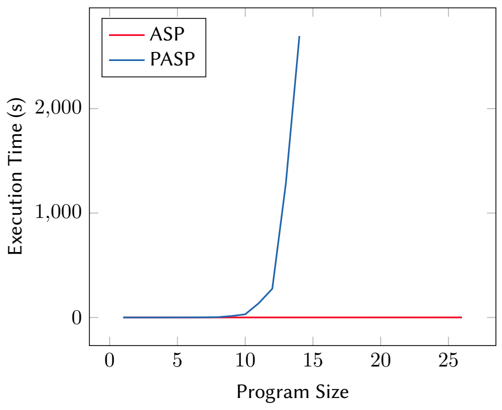
the query is `fly(1)`.

For the ASP version, we considered an increasing number of `bird/1` facts as abducibles. For PASP with deterministic constraints 1/3 of the `bird/1` facts are certain, 1/3 are probabilistic with an associated probability of 0.5, and 1/3 are abducibles. For PASP with probabilistic integrity constraints the split is the same, but the constraint imposing that 60% of birds fly has an associated probability of 0.5.

Results are shown in Figure 3a. As expected, the introduction of a probabilistic integrity constraint in the PASP program increases the execution time with respect to the same program with a deterministic IC, since an additional probabilistic fact must be considered. As before, the execution time to compute the abductive explanation in ASP is constant and almost negligible up to size 50.



(a) Inference times for the `bird` experiments.



(b) Inference times for the `smoke` experiments.

Figure 3: Results for the `bird` and `smoke` experiments.

A third dataset, `smoke`, encodes a network where nodes represent people and edges represent relationships. A person can either smoke or not smoke. Other people can either smoke or not smoke if they are influenced by others. This benchmark has no integrity constraints. An example program of size 4 is the following:

```
smokes(X) ; nosmokes(X) :- smokesFact(X).
smokes(X) ; nosmokes(X) :- smokes(Y), influences(X,Y).

smokesFact(1). smokesFact(2). smokesFact(3).
smokesFact(4). influences(0,1). influences(0,2).
influences(0,3). influences(1,3).
```

The goal is to compute the (probabilistic) abductive explanations for the query `smokes(1)`.

For abduction in ASP, half of the `smokesFact/1` facts are abducibles and half are certain. Similarly, half of the `influences/2` facts are abducibles and half are certain. For abduction in PASP, half of the `smokesFact/1` facts are probabilistic with an associated probability of 0.5 and half are certain, and half of the `influences/2` facts are abducibles and half are certain. The generation of `influences/2` facts follow a Barabási-Albert model (we used the method `barabasi_albert_graph` from the networkx [19] python package).

Results are shown in Figure 3b. The inference time trend in the two cases coincides with the previous experiments.

The gap of execution time between abduction in ASP and PASP is too big to be analyzed. Thus, we decided to run a separate experiment for abduction in ASP, with larger programs for all the datasets. Results are shown in Figure 4. The `bird` and `smoke` datasets have similar running time while the `clauses` one is the slowest among the three: this is probably because each program has a significant number of abductive explanations: for example, the program of size 10 has 45 while the program of size 360 has 64620 abductive explanations.

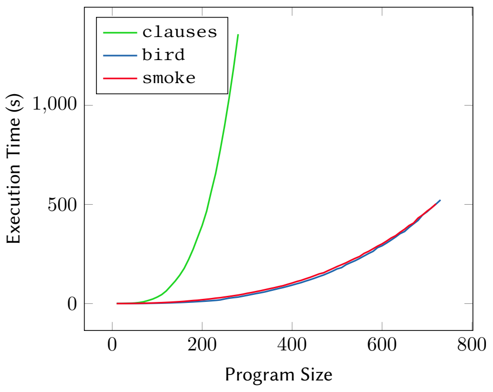


Figure 4: Inference times for abduction in ASP with programs of different sizes for the 3 datasets.

6. Related Work

Abduction and Answer Set Programming are strongly related [20]. In [21] the authors propose a specialized framework to perform abductive reasoning under the stable model semantics by defining a special T_p operator. Differently from them, we leverage an existing ASP solver (clingo) to perform abduction, and do not develop specialized operators. ABDUAL [22], later refined in TABDUAL [23], is a framework that performs abduction in the context of well-funded semantics with the capability to compute stable models. It is implemented in XSB Prolog [24] and leverages common Logic Programming techniques such as tabling. Differently from them, we use an ASP framework, easily supporting the whole ASP syntax. Recently, the authors of [8] proposed an algorithm to perform abduction in probabilistic logic programs under the Distribution Semantics. To the best of our knowledge, no existing frameworks can perform abduction in Probabilistic Answer Set Programming under the Credal Semantics.

7. Conclusions

In this paper we proposed a new algorithm to perform abduction in Answer Set Programming and Probabilistic Answer Set Programming under the Credal Semantics. For the former, the goal is to find the minimal set, where minimality is defined in terms of set inclusion, of abducible facts that explains a query, i.e., such that the query has at least one answer set. For the latter, the goal is to find the minimal set which maximizes the lower joint probability of the query and the constraints. Results on three datasets show that abduction for ASP is orders of magnitude faster than for PASP, since the generation of all the worlds is not needed. As future work, we plan to apply approximate inference [25] to speed up the computation in PASP. A further direction for future work could consist in better exploring the relation between lower and upper probability for abduction in PASP.

References

[1] A. C. Kakas, P. Mancarella, Abductive logic programming, in: V. W. Marek, A. Nerode, D. Pedreschi, V. S. Subrahmanian (Eds.), Proceedings of the Workshop Logic Programming and Non-Monotonic Logic, Austin, TX, USA, November 1–2, 1990, 1990, pp. 49–61.

[2] A. C. Kakas, P. Mancarella, Database updates through abduction, in: Proceedings of the 16th VLDB, Morgan Kaufmann, 1990, pp. 650–661.

[3] J. W. Lloyd, Foundations of Logic Programming, 2nd Edition, Springer, 1987.

[4] G. Brewka, T. Eiter, M. Truszczyński, Answer set programming at a glance, Communications of the ACM 54 (2011) 92–103. doi:10.1145/2043174.2043195.

[5] L. De Raedt, P. Frasconi, K. Kersting, S. Muggleton (Eds.), Probabilistic Inductive Logic Programming, volume 4911 of *LNCS*, Springer, 2008.

[6] F. Riguzzi, Foundations of Probabilistic Logic Programming: Languages, semantics, inference and learning, River Publishers, Gistrup, Denmark, 2018.

[7] T. Sato, A statistical learning method for logic programs with distribution semantics, in: L. Sterling (Ed.), ICLP 1995, MIT Press, 1995, pp. 715–729. doi:10.7551/mitpress/4298.003.0069.

[8] D. Azzolini, E. Bellodi, S. Ferilli, F. Riguzzi, R. Zese, Abduction with probabilistic logic programming under the distribution semantics, International Journal of Approximate Reasoning 142 (2022) 41–63. doi:10.1016/j.ijar.2021.11.003.

[9] F. G. Cozman, D. D. Mauá, On the semantics and complexity of probabilistic logic programs, J. Artif. Intell. Res. 60 (2017) 221–262. doi:10.1613/jair.5482.

[10] F. G. Cozman, D. D. Mauá, The joy of probabilistic answer set programming: Semantics, complexity, expressivity, inference, Int. J. Approx. Reason. 125 (2020) 218–239. doi:10.1016/j.ijar.2020.07.004.

[11] M. Alviano, W. Faber, Aggregates in answer set programming, KI-Künstliche Intelligenz 32 (2018) 119–124. doi:10.1007/s13218-018-0545-9.

[12] W. Faber, N. Leone, G. Pfeifer, Recursive aggregates in disjunctive logic programs: Semantics and complexity, in: European Workshop on Logics in Artificial Intelligence, Springer, 2004, pp. 200–212. doi:10.1007/978-3-540-30227-8_19.

[13] L. De Raedt, A. Kimmig, H. Toivonen, Problog: A probabilistic prolog and its application in link discovery, in: M. M. Veloso (Ed.), IJCAI, 2007, pp. 2462–2467.

[14] A. Van Gelder, K. A. Ross, J. S. Schlipf, The well-founded semantics for general logic programs, J. ACM 38 (1991) 620–650.

[15] T. Eiter, G. Gottlob, The complexity of logic-based abduction, J. ACM 42 (1995) 3–42. doi:10.1145/200836.200838.

[16] M. Gebser, B. Kaufmann, T. Schaub, Solution enumeration for projected boolean search problems, in: W.-J. van Hoeve, J. Hooker (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer-Verlag, 2009, pp. 71–86. doi:10.1007/978-3-642-01929-6_7.

[17] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot asp solving with clingo, Theory and Practice of Logic Programming 19 (2019) 27–82. doi:10.1017/S1471068418000054.

[18] D. Azzolini, E. Bellodi, F. Riguzzi, Statistical statements in probabilistic logic programming, in: G. G. D. Inclezan, M. Maratea (Eds.), 16th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2022), 2022.

[19] A. A. Hagberg, D. A. Schult, P. J. Swart, Exploring network structure, dynamics, and function using networkx, in: G. Varoquaux, T. Vaught, J. Millman (Eds.), Proceedings of the 7th Python in Science Conference, Pasadena, CA USA, 2008, pp. 11–15.

[20] M. Denecker, A. Kakas, Abduction in logic programming, Lecture Notes in Computer Science 2407 (2002) 402–436.

[21] K. Inoue, C. Sakama, Computing extended abduction through transaction programs, Ann. Math. Artif. Intell. 25 (1999) 339–367. doi:10.1023/A:1018926021566.

[22] J. J. Alferes, L. M. Pereira, T. Swift, Abduction in well-founded semantics and generalized stable models, CoRR cs.LO/0312057 (2003).

[23] A. Saptawijaya, L. M. Pereira, Tabdual: a tabled abduction system for logic programs, FLAP 2 (2015) 69–124.

[24] T. Swift, D. S. Warren, XSB: Extending prolog with tabled logic programming, Theor. Pract. Log. Prog. 12 (2012) 157–187. doi:10.1017/S1471068411000500.

[25] D. Azzolini, F. Riguzzi, E. Lamma, F. Masotti, A comparison of MCMC sampling for probabilistic logic programming, in: M. Alviano, G. Greco, F. Scarcello (Eds.), Proceedings of the 18th Conference of the Italian Association for Artificial Intelligence (AI*IA2019), Rende, Italy 19-22 November 2019, volume 11946 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, 2019, pp. 18–29. doi:10.1007/978-3-030-35166-3_2.