MAP Inference in Probabilistic Answer Set Programs

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

[1] Dipartimento di Scienze dell'Ambiente e della Prevenzione,
Università di Ferrara, Ferrara, Italy
damiano.azzolini@unife.it

[2] Dipartimento di Ingegneria, Università di Ferrara, Ferrara, Italy
elena.bellodi@unife.it

[3] Dipartimento di Matematica e Informatica, Università di Ferrara, Ferrara, Italy
fabrizio.riguzzi@unife.it

Abstract. Reasoning with uncertain data is a central task in artificial intelligence. In some cases, the goal is to find the most likely assignment to a subset of random variables, named query variables, while some other variables are observed. This task is called Maximum a Posteriori (MAP). When the set of query variables is the complement of the observed variables, the task goes under the name of Most Probable Explanation (MPE). In this paper, we introduce the definitions of cautious and brave MAP and MPE tasks in the context of Probabilistic Answer Set Programming under the credal semantics and provide an algorithm to solve them. Empirical results show that the brave version of both tasks is usually faster to compute. On the brave MPE task, the adoption of a state-of-the-art ASP solver makes the computation much faster than a naive approach based on the enumeration of all the worlds.

Keywords: Probabilistic answer set programming · MAP inference · Statistical relational artificial intelligence

1 Introduction

The research field of Probabilistic Logic Programming (PLP) [20] aims to reason with logic programs where some of the facts, called *probabilistic facts*, are considered uncertain [11]. One of the most adopted semantics for these programs, the Distribution Semantics (DS) [21], assigns a meaning to probabilistic logic programs where every *world*, i.e., a logic program identified by the truth values of probabilistic facts, is required to have a total well-founded model [25].

Probabilistic Answer Set Programming (PASP) [10,19] extends the capabilities of Answer Set Programming (ASP) [9] and allows, as PLP, the definition of probabilistic facts. With PASP, however, every world is an answer set program and thus may have multiple answer sets. In this case, a semantics that can be adopted is the *credal semantics*, which assigns a probability range rather than

a sharp probability value, as happens with the DS, to a query. This range is defined by a lower and an upper probability.

Maximum-a-Posteriori (MAP) inference is a central topic in machine learning, where the goal is to find, given a set of evidence variables, the most probable value to a subset of the random variables (called query variables). If the set of query variables is the complement of the set of evidence variables, the problem is called Most Probable Explanation (MPE).

In this paper, we propose an algorithm to perform both *cautious* MAP/MPE and *brave* MAP/MPE inference in probabilistic answer set programs, where we consider respectively the lower and the upper probability bound induced by the query variables. We test this algorithm on two datasets with different configurations. Moreover, we also compare our algorithm with the clingo's [13] `#maximize` statement for the brave MPE task.

The paper is structured as follows: in Sect. 2, we discuss some related works. Section 3 introduces the main concepts of PLP and PASP. Section 4 describes our algorithm to perform brave and cautious MAP/MPE inference in PASP and in Sect. 5 we discuss some experiments to test its performance. Section 6 concludes the paper with some final remarks and possible future works.

2   Related Work

The MAP/MPE task has received relatively small attention in PLP: in [22], the authors introduced an algorithm to compute the MAP/MPE for a given LPAD [26]. The program is converted into a compact form and then the result is computed by analysing it. Similar work can be found in [8]. However, both consider programs where every world has a unique model, so they cannot deal with probabilistic ASP programs, where every world may have multiple models.

The authors of [16] propose a tool to perform inference in ASP programs following the LP^{MLN} [17] semantics. Differently from them, we adopt a different semantics, the credal semantics [10], that we believe being more general and intuitive for PASP. Moreover, we consider the MAP/MPE task, not discussed in their work.

Inference in PASP has been considered in [24], where the authors introduced the PASOCS solver, but they do not explore the MAP/MPE task. Similar considerations can be applied to [23], where the authors discussed how to perform inference in ProbLog [11] programs under the stable model semantics, but still ignoring MAP/MPE.

3   Background

We assume that the reader is familiar with the basic concepts of Logic Programming [18]. Here we consider the Answer Set Programming (ASP) syntax [9] enriched with *aggregate atoms* [3]. An aggregate atom is composed by two *guards* that can be either constants or variables, denoted with g_0 and g_1, two comparison arithmetic operators, δ_0 and δ_1, an aggregate function symbol φ, and a set

of expressions $\epsilon_0, \ldots, \epsilon_n$ where each ϵ_i has the form $t_1, \ldots, t_n : F$ and each t_i is a term whose variables appear in the conjunction of literals F. Given the previous elements, the syntax of an aggregate atom is $g_0 \delta_0 \ \#\varphi\{\epsilon_0; \ldots; \epsilon_n : F\} \ \delta_1 g_1$. An example of aggregate atom is `0 <= #sum{A : p(A)} <= 2`.

We denote a *disjunctive rule* (or simply *rule*) with the syntax

```
h1 ; ... ; hm :- b1, ..., bm.
```

where each `hi` is an atom and each `bi` is a literal. The disjunction of atoms at the left of the neck operator (`:-`) is called *head* while the conjunction of literals at its right is called *body*. If the head is empty and the body is not, the rule is called a *constraint* and if the body is empty and the head is not, the rule is a *fact*. We restrict ourselves to *safe rules*, i.e., rules where every variable in the head also appears in a positive literal in the body. Finally, if a rule does not contain variables it is called *ground*. A program is a finite set of rules.

To provide the definition of answer set, we need to introduce some more concepts. If we consider an answer set program $\mathcal{P}$, with $B_{\mathcal{P}}$ we denote the set of ground atoms that can be constructed with the symbols in $\mathcal{P}$. $B_{\mathcal{P}}$ is also called Herbrand base. An interpretation I of $\mathcal{P}$ is such that $\mathrm{I} \subset B_{\mathcal{P}}$. I satisfies a ground rule if at least one head atom is true in it when all the literals in the body are true in it, and it is called a *model* if it satisfies all the groundings of the rules of $\mathcal{P}$. The *reduct* [12] of a ground program $\mathcal{P}_g$ w.r.t. an interpretation I is obtained by removing from $\mathcal{P}_g$ the rules where at least one literal in the body is false in I. Finally, an *answer set* (or stable model) for a program $\mathcal{P}$ is defined as an interpretation that is a minimal (under set inclusion) model of $\mathcal{P}_g$. We indicate with $AS(\mathcal{P})$ the set of all the answer sets of a program $\mathcal{P}$. Finally, the *projective solutions* [14] onto a set of ground atoms B are given by the set $AS_B(\mathcal{P}) = \{A \cap B \mid A \in AS(\mathcal{P})\}$.

Probabilistic Logic Programming [20] allows the definition of uncertain data in logic programs. For example, ProbLog [11] allows *probabilistic facts*. Each probabilistic fact has the form $\Pi :: f$ where $\Pi \in]0, 1]$ and f is an atom. According to the Distribution Semantics [21], an assignment of truth value, true ($\top$) or false ($\bot$), for every probabilistic fact f_i in the program identifies a *world* w whose probability $P(w)$ can be computed as

$$P(w) = \prod_{i|f_i=\top} \Pi_i \cdot \prod_{i|f_i=\bot} (1 - \Pi_i) \tag{1}$$

If we are given a query q, i.e., a conjunction of ground literals, its probability is the sum of the probability of the worlds where the query is true:

$$P(q) = \sum_{w \models q} P(w) \tag{2}$$

The Distribution Semantics assumes that all the probabilistic facts are independent and that every world is a logic program with a two-valued well-founded model [25]. However, when we consider Probabilistic Answer Set Programming,

the latter condition usually does not hold. For PASP, we consider here the credal semantics (CS) [10,19]. Under the CS, every query q is associated with a probability interval defined by a lower bound $\underline{P}(q)$ and an upper bound $\overline{P}(q)$. A world contributes to the upper probability if the query is present in *at least one* of its answer sets and contributes to the lower probability if the query is present in *all* its answer sets. In formulas,

$$\overline{P}(q) = \sum_{w_i | \exists m \in AS(w_i), \ m \models q} P(w_i)$$

$$\underline{P}(q) = \sum_{w_i | |AS(w_i)| > 0 \ \wedge \ m \in AS(w_i), \ m \models q} P(w_i)$$

These formulas are valid only if every world has at least one answer set, so in this paper we consider only programs that satisfy this requirement. If every world has exactly one answer set, the CS coincides with the DS and the query has a sharp probability value. Consider the following program.

Example 1. Gold example

```
1  0.2:: gold (1).
2  0.3:: gold (2).
3  0.7:: gold (3).
4  valuable (X) ; not_valuable (X):- gold (X).
5  :- #count{X: valuable (X), gold (X)} = VG,
6      #count{X: gold (X)} = G, 10*VG < 6*G.
```

The first three lines introduce three probabilistic facts `gold/1` indicating that the objects identified with 1, 2, and 3 could be made of gold with different probabilities. Line 4 states that an object made of gold may be valuable or not. Line 5 represents a constraint saying that 60% of the objects made of gold are valuable. This program has $2^3 = 8$ worlds listed in Table 1. If we consider the query q `valuable(1)`, $\underline{P}(q) = 0.158$ (corresponding to $P(w_4) + P(w_5) + P(w_6)$) and $\overline{P}(q) = 0.2$ (corresponding to $P(w_4) + P(w_5) + P(w_6) + P(w_7)$).

Table 1. Worlds for Example 1. Predicate 'g' stands for `gold`. Column 'mq' indicates whether there is at least one model of the world where the query `valuable(1)` is true and column 'mnq' indicates whether there is at least one model of the world where the query is false.

world	g(1)	g(2)	g(3)	$P(w)$	mq	mnq
0	0	0	0	0.168	F	T
1	0	0	1	0.392	F	T
2	0	1	0	0.392	F	T
3	0	1	1	0.168	F	T
4	1	0	0	0.042	T	F
5	1	0	1	0.098	T	F
6	1	1	0	0.018	T	F
7	1	1	1	0.042	T	T

4 MAP Inference in Probabilistic Answer Set Programming

In PLP, the MAP task [8,22] consists in finding a possible truth value assignment to a subset of probabilistic facts such that a given evidence is satisfied and the sum of the probabilities of the possible worlds identified by the truth values' choices is maximized. More formally, given a probabilistic logic program, a set of ground atoms e, and a set of query random variables (also called query variables) Q, the goal is to solve

$$\arg\max_q P(Q = q \mid e)$$

If all the program variables are query variables, the task is called MPE.

If we consider PASP, every world may have multiple models so the previous definition must be extended. We now introduce the *cautious* MAP and *brave* MAP tasks:

Definition 1. *Cautious and brave MAP/MPE. Given a PASP program $\mathcal{P}$, a set of ground atoms e (call it evidence), and a set of query probabilistic facts Q:*

– *the cautious MAP problem consists in finding a truth assignment q to query facts Q such that $\underline{P}(q \mid e)$ is maximized, i.e., in solving:*

$$\underline{\mathrm{MAP}}(e) = \arg\max_q \underline{P}(Q = q \mid e) = \arg\max_q \sum_{w_i \mid \forall m \in AS(w_i), m \models q \wedge m \models e} P(w_i)$$

– *the brave MAP problem consists in finding a truth assignment q to query facts Q such that $\overline{P}(q \mid e)$ is maximized, i.e., in solving:*

$$\overline{\mathrm{MAP}}(e) = \arg\max_q \overline{P}(Q = q \mid e) = \arg\max_q \sum_{w_i \mid \exists m \in AS(w_i), m \models q \wedge m \models e} P(w_i)$$

The definition of cautious and brave MPE inference for a query e, denoted with $\underline{\mathrm{MPE}}(e)$ and $\overline{\mathrm{MPE}}(e)$ respectively, is similar.

Note that this task is different from computing the conditional probability of a query given evidence. Given the previous definitions, for a query e we have that $P(\underline{\mathrm{MAP}}(e)) \leq P(\overline{\mathrm{MAP}}(e))$ and $P(\underline{\mathrm{MPE}}(e)) \leq P(\overline{\mathrm{MPE}}(e))$.

If we consider all the three probabilistic facts `gold/1` of Example 1 as query variables (denoted by prepending the functor `map`), the cautious MPE state (all the probabilistic facts are query variables) for the query `valuable(1)` is given by {`gold(1), not gold(2), gold(3)`} with an associated probability of 0.098 (world 5 of Table 1). With `not gold(2)` we indicate that the probabilistic fact `gold(2)` should be false. The same state is also the brave MPE state. The cautious MAP/MPE and the brave MAP/MPE state do not necessarily coincide. For example, if we consider `gold(1)` and `gold(3)` as query variables, the cautious MAP state for the evidence `valuable(1)` is {`gold(1), not gold(3)`} with a probability of 0.06 (sum of the probabilities of the worlds 4 and 6 of Table 1) while the brave MAP state is {`gold(1), gold(3)`} with a probability of 0.14 (sum of

the probabilities of the worlds 5 and 7 of Table 1). Finally, there can be multiple cautious/brave MAP/MPE states. If we consider again Example 1 but with all the probabilities set to 0.5 and all the three probabilistic facts as query variables, there are 3 cautious MPE states for the query `valuable(1)`, all with an associated probability of 0.125: {`gold(1), gold(2), not gold(3)`}, {`gold(1), not gold(2), gold(3)`}, and {`gold(1), not gold(2), not gold(3)`}.

4.1   Algorithm

To solve the cautious/brave MAP/MPE task[1], we developed an algorithm that works in two steps: first, it translates the PASP program into an ASP program by rewriting probabilistic facts and query variables into ASP choice rules. It is shown in Algorithm 1 and it proceeds as follows: first, the function CON-VERTVARIABLES converts probabilistic facts and query variables into an ASP representation. Every probabilistic fact `p::f` and every query variable `map p::f` (note that `f` may also have arguments) is transformed into `0{f}1`. Moreover, we add the rule `not_f:- not f`. Function COMPUTEMINIMALSET [5] extracts the minimal set of probabilistic facts by computing the cautious consequences (intersection of all models). The facts in this set must always be true, so we can remove the choices for them and fix their value. For every element in this set, we add a constraint imposing that it must be true (line 5). This is possible since every world is required to have at least one answer set. Now, if we want to perform brave MAP (i.e., considering the upper probability) given an evidence `e`, we insert the rule `:- not e` (a constraint imposing that the evidence must always be true) to the program and project the solutions on the probabilistic facts (line 9). Otherwise, if we consider cautious MAP (lower probability), we add the rules `q:- e` and `nq:- not e` and still project the solutions on the atoms `q/0` and `nq/0` (line 12). Finally, we extract every world and its contribution to the probability with the function COMPUTECONTRIBUTION and identify the MAP state (function COMPUTEMAPSTATE).

To better understand how the algorithm works, consider the program shown in Example 1 with `gold(1)` and `gold(3)` as query variables and `valuable(1)` as evidence. After the execution of function CONVERTVARIABLES, the probabilistic fact and the two query variables become `0{gold(2)}1`, `0{gold(1)}1`, and `0{gold(3)}1`. The minimal set of atoms, obtained by computing the cautious consequences on the converted program with an additional rule `:- not valuable(1)`, contains `gold(1)`, so we add the constraint `:- not gold(1)` to the program. If we consider *brave* MAP, by adding again `:- not valuable(1)` to the program and projecting the solutions on the probabilistic facts (function PROJECTSOLUTIONS, line 9), we get 4 answer sets:

    AS1 = {gold(1) not_gold(2) not_gold(3)},
    AS2 = {gold(1) not_gold(2) gold(3)},

---

[1] We will usually only write MAP to simplify the notation, since MPE is a special case of MAP.

```
AS3 = {gold(1) gold(2) not_gold(3)}, and
AS4 = {gold(1) gold(2) gold(3)},
```

where with `not_gold(i)` we indicate that the probabilistic fact or query variable is not selected. These four answer sets (worlds) have respectively probability $0.2 \cdot (1-0.3) \cdot (1-0.7) = 0.042$, $0.2 \cdot (1-0.3) \cdot 0.7 = 0.098$, $0.2 \cdot 0.3 \cdot (1-0.7) = 0.018$, and $0.2 \cdot 0.3 \cdot 0.7 = 0.042$, that are computed with the function COMPUTECONTRIBUTION. Finally, if we group these answer sets by query variables (function COMPUTEMAPSTATE), we get two sets representing two different MAP states: `MAP1 = {AS1, AS3}` (`gold(1)` and `not_gold(3)`) and `MAP2 = {AS2, AS4}` (`gold(1)` and `gold(3)`). `MAP1` has probability $0.042 + 0.018 = 0.06$ while `MAP2` has probability $0.098 + 0.042 = 0.14$ so `MAP2` is selected as MAP state since it gives the highest upper probability for the evidence `valuable(1)`.

If we consider instead *cautious* MAP, the process in analogous, but we cannot add the constraint `:- not valuable(1)` since we need to consider the lower probability: in this case, a world contributes to the lower probability if the evidence is true in every answer set. If we add the constraint imposing that the evidence must be true in every answer set, we cannot identify the worlds that have at least one answer set where the evidence is false (and thus do not contribute to the lower probability). We now get 5 answer sets:

```
{gold(1) gold(2) gold(3) nq},
{gold(1) gold(2) gold(3) q},
{gold(1) gold(2) not_gold(3) q},
{gold(1) not_gold(2) gold(3) q}, and
{gold(1) not_gold(2) not_gold(3) q}.
```

The world identified by the first two answer sets is the same (all the three variables true) but in the first there is `nq` and in the second `q`. Thus, the first answer set indicates that there is at least one answer set of this world where the query is false, so it does not contribute to the lower probability (and can be discarded). For the remaining three worlds there is only one answer set each and it has `q` inside, so they contribute to both the lower and the upper probability. By applying, as before, functions COMPUTECONTRIBUTION and then COMPUTEMAPSTATE, we get `{gold(1), not gold(3)}` as MAP state (third and fifth answer set) with an associated probability of $0.2 \cdot 0.3 \cdot (1-0.7) + 0.2 \cdot (1-0.3) \cdot (1-0.7) = 0.06$.

For both brave and cautious MAP tasks we need to generate at worst 2^n answer sets, where n is the number of probabilistic facts, thus the algorithm is exponential in n. The reason is that we need to know if there is at least one answer set for every world where the query is true for the brave MAP and if in all the models for every world the query is true for cautious MAP. However, the number of generated models for brave MAP is usually smaller than the number of generated models for cautious MAP, due to the additional constraint removing the models where the query is false. However, this additional constraint plus possibly the constraints given by the elements in the minimal set of atoms does not reduce the complexity of the task.

Algorithm 1. Function COMPUTEMAPSTATE: computation of the MAP/MPE state given a query e in a PASP program $\mathcal{P}$.

```
 1: function COMPUTEMAPSTATE(e, P, mode)
 2:     PASP_p, mapVariables ← CONVERTVARIABLES(P)
 3:     minSet ← COMPUTEMINIMALSET(PASP_p ∪ {: − not e.})
 4:     for all a ∈ minSet do                              ▷ a represents a probabilistic fact
 5:         PASP_p ← PASP_p ∪ {: − not a.}
 6:     end for
 7:     if mode is brave then                                            ▷ Brave MAP
 8:         PASP_p ← PASP_p ∪ {: − not e.}
 9:         AS ← PROJECTSOLUTIONS(PASP_p, probFacts)
10:     else                                                          ▷ Cautious MAP
11:         PASP_p ← PASP_p ∪ {q : − e., nq : − not e.}
12:         AS ← PROJECTSOLUTIONS(PASP_p, probFacts, q ∪ nq)
13:     end if
14:     worldsList ← COMPUTECONTRIBUTION(AS)
15:     return COMPUTEMAPSTATE(worldsList, mapVariables)
16: end function
```

We propose another possible encoding for the brave MPE task. For each query variable `map p::f`, we add: a rule `0{f}1`, a rule `f(lp):- f` and a rule `not_f(nlp):- not f`. `lp` is given by $10^n \cdot log(\mathtt{p})$ and `nlp` is given by $10^n \cdot log(1-\mathtt{p})$, where n is an integer that denotes its scale. The multiplications by 10^n are needed since ASP does not handle floating points. For example, if we set n to 3, the fact `0.2::gold(1)` of Example 1 is expanded in: `0{gold(1)}1`, `gold(1,-1609):- gold(1)`, and `not_gold(1,-223):- not gold(1)`, where $10^3 \cdot log(0.2) = -1609$ and $10^3 \cdot log(0.8) = -223$. With this log-encoding, we can leverage the property $log(a \cdot b) = log(a) + log(b)$ and thus use the `#sum` aggregate. By multiplying by 10^n, it is not straightforward to obtain the original probabilities once we have the brave MPE state. However, once we get the combination of variables in this state, we can simply look up the initial probabilities in the program. Finally, since we have the (converted) probability as argument of the atoms, we can use the clingo [13] `#maximize` to find the combination of query variables resulting in the brave MPE state. If we consider again Example 1, with all the probabilistic facts converted as previously described, we can compute the brave MPE state with `#maximize{ P : wp(P) }` where `wp/1` is defined as

```
1  wp(P):-
2     PS = #sum{X,Y : gold(Y,X)},
3     PNS = #sum{X,Y : not_gold(Y,X)},
4     P = PS + PNS.
```

This is a naive encoding that requires the enumeration of all the answer sets. An alternative ASP encoding, we call it *improved*, for the solution of the brave MPE task for the same example, that does not require the enumeration of all the answer sets, is `#maximize{X,Y:gold(Y,X); X,Y:not_gold(Y,X)}`. In the next section, we test our algorithm for cautious and brave MAP and MPE and compare the execution time between our brave MPE proposal and the clingo `#maximize` statement.

5   Experiments

We implemented the algorithm in Python and we used the clingo APIs [13] to compute the answer sets[2]. To test the performance, we ran some experiments on a computer with Intel® Xeon® E5-2630v3 running at 2.40 GHz with 8 Gb of RAM and a time limit of 8 h. Execution times are computed with the bash command `time`. The reported values are from the `real` field.

The first dataset, `gold`, contains a set of programs with the structure of Example 1. The size of a program is given by the number of probabilistic facts `gold/1`. Example 1 has size 3. For the MAP task, 50% of the `gold/1` facts are considered query variables. We randomly set the probability of probabilistic facts. The query is `valuable(1)`. Results are shown in Fig. 1a. We removed the results for size less than 19 since their execution times were negligible. The computation of the brave MAP state seems the fastest one, followed by the brave MPE state. This is due to the additional constraint inserted into the program, which removes some of the possible answer sets. Cautious MAP and cautious MPE have comparable execution times. In all the cases, for size greater than 25 we get a memory error.

The second dataset, `smoke`, describes a network of friends where some of them smoke. An example of program of size (number of people) 4 is:

```
1  0.73::e(0,1).  0.59::e(0,2).
2  0.08::e(0,3).  0.19::e(2,3).
3
4  smokes(0).  smokes(2).
5  friend(X,Y):- e(X,Y).  friend(X,Y):- e(Y,X).
6  smokes(X); no_smokes(X):- friend(X,Y),smokes(Y).
7
8  :- #count{X:no_smokes(X)} = N,
9     #count{X:smokes(X)} = S, 10*S < 8*(N+S).
```

A person `X` smokes if she has at least one friend `Y` that smokes. The constraint imposes that at least 80% of the people smoke. The goal is to compute the MAP/MPE state for the query `smokes(n)` where `n` is the number of people involved (here 4). Half of the people of the network certainly smoke. If the number of people is odd, we round the result to the next integer. As before, for the MAP experiments, 50% of `e/2` facts are query The number of probabilistic facts follows a Barabási-Albert preferential attachment model generated with the `networkx` [15] Python package. We set as initial number of nodes of the graph (n) the size of the instance and as the number of edges that connect a new node to an existing one (m) 2. Results are shown in Fig. 1b. As for the `gold` dataset, also here brave MAP and brave MPE seem the fastest, and their execution times are similar (the red and black curves in the plot overlap). In all cases, for size greater than 14 we get a memory error.

In a second set of experiments we verified whether and how the execution time of the algorithm varies when there is an increasing number of MAP/MPE states.

---

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

To do this, we generated two versions of the `gold` dataset, one with random probabilities and one with all the probabilities set to 0.5. The remaining parts of the programs are equal to Example 1. Figure 2a shows the execution times of the cautious and brave MAP and MPE task on the dataset with all the probabilities set to 0.5. As before, brave MAP/MPE are the fastest. Also here, datasets with size larger than 25 cause a memory error, except for brave MPE that stops at size 23. Execution times for cautious MPE/MAP are almost identical. In Fig. 2b we compare the two versions of the datasets on the brave MPE task. Brave MAP with all probabilities set to 0.5 and brave MPE with random probabilities seem to take the same time to complete. Execution times for random probabilities are slightly smaller since there is usually only one MAP/MPE state in this case. Moreover, the MAP/MPE task where all the probabilities are equal gives a memory error starting from size 24, while, when the probabilities are all different, we get a memory error starting from size 26. A similar trend (exponential) was observed in the case of cautious MAP/MPE, but with the same differences found in Fig. 2a.

Lastly, we compared our algorithm with the `#maximize` statement of clingo on the brave MPE task for the `gold` dataset. As before, we generate a set of programs with random probabilities and a set of programs with all the probabilities set to 0.5. For a fair comparison, we set all the elements of the minimal set of atoms to be true in the program that will use the clingo statement and we add the constraint imposing that the query must be true. We ran two tests: one that outputs only one brave MPE state (even if there may be more) and one that outputs all the states, by using the flag `--opt-mode=optN`. We only considered the naive encoding, since the improved one is order of magnitude faster than the other and than our tool. For example, with 30 probabilistic facts and the improved encoding, the result is computed in a fraction of a second. Results in Fig. 3a show that the execution time for the computation of the brave MPE state oscillates when we want only one solution when probabilities are all equals. The computation of all the solutions when the probabilities are all set to 0.5 is the fastest one. For random probabilities, in both cases (the two curves overlap) the programs of size larger than 12 give a memory error. Figure 3b shows that clingo's `#maximize` statement is slower than our algorithm but it can handle larger instances when we want to compute all the solutions of the brave MPE task when all the probabilities of the states are equal (red and yellow curves). This may be due to a better memory management of the program and a possibly better search strategy. Moreover, the computation of 1 MPE state in clingo (blue curve) stops for the time limit, rather than the memory limit as the others.
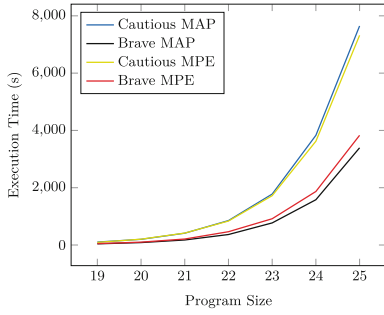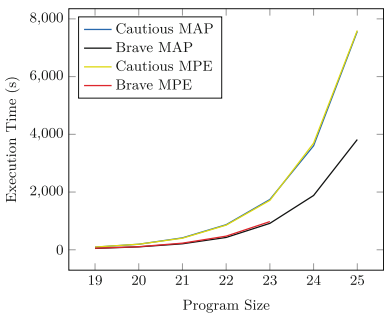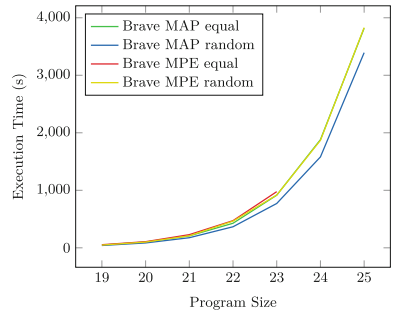
(a) Results for the `gold` dataset.
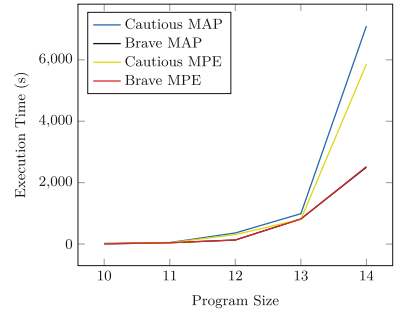
(b) Results for the `smoke` dataset.

Fig. 1. Results for cautious and brave MAP and MPE tasks for the `gold` and `smoke` datasets in terms of inference time as the program size (number of probabilistic facts) increases.
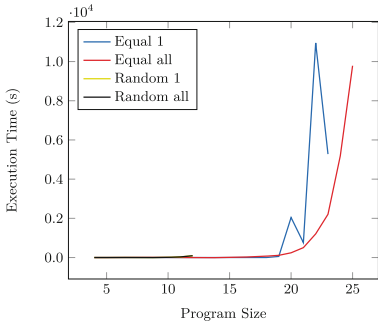

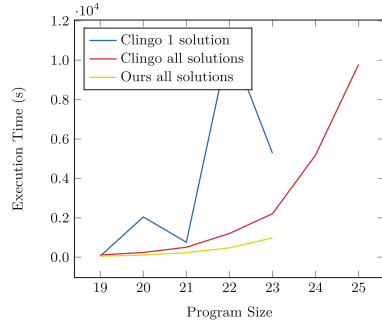(a) Results for the `gold` dataset with all the probabilities set to 0.5.

(b) Comparison between random and equal probabilities for the brave MAP/MPE task for the `gold` dataset.

Fig. 2. Comparisons between the two `gold` dataset versions.

(a) Results obtained using the `#maximize` aggregate in clingo on the `gold` dataset.

(b) Comparison between clingo `#maximize` statement and our algorithm on the `gold` dataset with probabilities set to 0.5.

Fig. 3. Results for the brave MPE task computed with clingo's `#maximize` statement using the naive encoding and comparison with our algorithm. '1' means that we compute only 1 solution while 'all' means that we compute all the solutions.

6   Conclusions

In this paper, we proposed the concepts of cautious and brave MAP/MPE inference in probabilistic answer set programming and developed an algorithm to solve these tasks. We ran some experiments on multiple datasets and we obtained that, generally, cautious MAP/MPE is slower than brave MAP/MPE, due to the necessity to enumerate all the possible answer sets needed to compute the lower probability. We also proposed two alternative encodings for the brave MPE task and compare the clingo `#maximize` statement with our approach. The encoding that does not require the enumeration of all the answer sets is order of magnitude faster than the other and than our tool. However, if we consider the naive encoding, when all the probabilities are set to 0.5, clingo is slower than our algorithm but it seems to be able to solve larger instances with less memory requirements. In the future, we plan to test other ASP solvers such as WASP [1,2], adopt approximate algorithms based on sampling [6,7], and consider the concept of abduction [4] in PASP.

References

1. Alviano, M., Dodaro, C., Leone, N., Ricca, F.: Advances in WASP. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.) LPNMR 2015. LNCS (LNAI), vol. 9345, pp. 40–54. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23264-5_5

2. Alviano, M., Dodaro, C., Marques-Silva, J., Ricca, F.: Optimum stable model search: algorithms and implementation. J. Log. Comput. **30**(1), 863–897 (2020). https://doi.org/10.1093/logcom/exv061

3. Alviano, M., Faber, W.: Aggregates in answer set programming. KI-Künstliche Intelligenz **32**(2), 119–124 (2018). https://doi.org/10.1007/s13218-018-0545-9

4. Azzolini, D., Bellodi, E., Ferilli, S., Riguzzi, F., Zese, R.: Abduction with probabilistic logic programming under the distribution semantics. Int. J. Approx. Reason. **142**, 41–63 (2022). https://doi.org/10.1016/j.ijar.2021.11.003

5. Azzolini, D., Bellodi, E., Riguzzi, F.: Statistical statements in probabilistic logic programming. In: Gottlob, G., Inclezan, D., Maratea, M. (eds.) Logic Programming and Nonmonotonic Reasoning, pp. 43–55. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-15707-3_4

6. Azzolini, D., Riguzzi, F., Lamma, E.: An analysis of Gibbs sampling for probabilistic logic programs. In: Dodaro, C., et al. (eds.) Workshop on Probabilistic Logic Programming (PLP 2020). CEUR-WS, vol. 2678, pp. 1–13. Sun SITE Central Europe, Aachen, Germany (2020)

7. Azzolini, D., Riguzzi, F., Masotti, F., Lamma, E.: A comparison of MCMC sampling for probabilistic logic programming. In: Alviano, M., Greco, G., Scarcello, F. (eds.) AI*IA 2019. LNCS (LNAI), vol. 11946, pp. 18–29. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35166-3_2

8. Bellodi, E., Alberti, M., Riguzzi, F., Zese, R.: MAP inference for probabilistic logic programming. Theor. Pract. Log. Prog. **20**(5), 641–655 (2020). https://doi.org/10.1017/S1471068420000174

9. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011). https://doi.org/10.1145/2043174.2043195

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

11. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: a probabilistic Prolog and its application in link discovery. In: Veloso, M.M. (ed.) IJCAI 2007. vol. 7, pp. 2462–2467. AAAI Press/IJCAI, California (2007)

12. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: semantics and complexity. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 200–212. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30227-8_19

13. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot asp solving with clingo. Theory Pract. Logic Program. **19**(1), 27–82 (2019). https://doi.org/10.1017/S1471068418000054

14. Gebser, M., Kaufmann, B., Schaub, T.: Solution enumeration for projected Boolean search problems. In: van Hoeve, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 71–86. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01929-6_7

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

16. Hahn, S., Janhunen, T., Kaminski, R., Romero, J., Rühling, N., Schaub, T.: Plingo: a system for probabilistic reasoning in clingo based on LPMLN (2022). https://doi.org/10.48550/ARXIV.2206.11515

17. Lee, J., Wang, Y.: Weighted rules under the stable model semantics. In: Baral, C., Delgrande, J.P., Wolter, F. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, 25–29 April 2016. pp. 145–154. AAAI Press (2016)

18. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer, Heidelberg (1987). https://doi.org/10.1007/978-3-642-83189-8

19. Mauá, D.D., Cozman, F.G.: Complexity results for probabilistic answer set programming. Int. J. Approx. Reason. **118**, 133–154 (2020). https://doi.org/10.1016/j.ijar.2019.12.003

20. Riguzzi, F.: Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning. River Publishers, Gistrup (2018)

21. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) ICLP 1995, pp. 715–729. MIT Press (1995). https://doi.org/10.7551/mitpress/4298.003.0069

22. Shterionov, D., Renkens, J., Vlasselaer, J., Kimmig, A., Meert, W., Janssens, G.: The most probable explanation for probabilistic logic programs with annotated disjunctions. In: Davis, J., Ramon, J. (eds.) ILP 2014. LNCS (LNAI), vol. 9046, pp. 139–153. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23708-4_10

23. Totis, P., Kimmig, A., Raedt, L.D.: Smproblog: stable model semantics in problog and its applications in argumentation. arXiv abs/2110.01990 (2021). https://doi.org/10.48550/ARXIV.2110.01990

24. Tuckey, D., Russo, A., Broda, K.: PASOCS: a parallel approximate solver for probabilistic logic programs under the credal semantics. arXiv abs/2105.10908 (2021). https://doi.org/10.48550/ARXIV.2105.10908

25. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM **38**(3), 620–650 (1991)

26. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3132, pp. 431–445. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27775-0_30