Inference in Probabilistic Answer Set Programming Under the Credal Semantics

Damiano Azzolini[1(✉)] and Fabrizio Riguzzi[2]

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

[2] Dipartimento di Matematica e Informatica, University of Ferrara, Ferrara, Italy
fabrizio.riguzzi@unife.it

Abstract. Probabilistic Answer Set Programming under the credal semantics (PASP) describes an uncertain domain through an answer set program extended with probabilistic facts. The PASTA language leverages PASP to express statistical statements. A solver with the same name allows to perform inference in PASTA programs and, in general, in PASP. In this paper, we investigate inference in PASP, propose a new inference algorithm called aspcs based on Second Level Algebraic Model Counting (2AMC), and implement it into the aspmc solver. Then, we compare it with PASTA on a set of benchmarks: the empirical results show that, when the program does not contain aggregates, the new algorithm outperforms PASTA. However, when we consider PASTA statements and aggregates, we need to replace aggregates with a possibly exponential number of rules, and aspcs is slower than PASTA.

Keywords: Second Level Algebraic Model Counting · Probabilistic Answer Set Programming · Inference

1 Introduction

Algebraic Model Counting (AMC) [18] is an umbrella term that comprises several well-known tasks, among the other, SAT, #SAT, weighted model counting, and probabilistic inference. All these tasks require to aggregate the models of a program according to a certain criterion. For instance, model counting requires counting the models while probabilistic inference, e.g., in the probabilistic logic language ProbLog [11], requires summing the probabilities associated with the different models. Other tasks, such as decision theoretic inference [7], MAP and MPE inference [4,5,22], and probabilistic inference under the smProbLog language [23], require to aggregate the results obtained via inference, so they need two levels of aggregations. These tasks they were recently identified as Second Level Algebraic Model Counting (2AMC) tasks [17].

Probabilistic Answer Set Programming under the credal semantics (PASP, for short) [2,8] is one of the possible formalisms to express uncertainty in Answer

R. Basili et al. (Eds.): AIxIA 2023, LNAI 14318, pp. 367–380, 2023.
https://doi.org/10.1007/978-3-031-47546-7_25

Set Programming (ASP) [14], since it extends ASP with ProbLog probabilistic facts [11]. PASP has been recently adopted in the PASTA framework [3] to encode statistical statements [16], that represent statistical information about a given domain. During inference, these are converted into choice rules and constraints with aggregates [1]. Then, the PASTA solver performs projected answer set enumeration [13]. As a first contribution, we discuss how to represent inference in PASP as a 2AMC task. Then, we implement our approach on top of the state of the art aspmc solver [12], that adopts knowledge compilation [10] to compactly represent a program, and we call it aspcs. Tests on different benchmarks show that when programs do not contain aggregates, aspcs is significantly faster than PASTA. However, aspmc, and so aspcs, currently does not support aggregates, so to represent PASTA statistical statements we manually convert constraints with aggregates into a set of ground rules. In this case, aspcs perform worse than PASTA, probably due to the possibly exponential number of rules that come from the conversion of the aggregates.

The paper is structured as follows: Sect. 2 introduces the needed background knowledge. In Sect. 3 we cast inference in PASP as a 2AMC task and in Sect. 4 we test an implementation on top of the aspmc solver against the PASTA solver. Section 5 concludes the paper.

2 Background

ProbLog *probabilistic facts* [11] are one of the most used syntactical constructs to represent uncertainty within a probabilistic logic program [19]: they are of the form $f_i::\Pi_i$ where f_i is a logical atom and $\Pi_i \in [0, 1]$ is its probability value. Its meaning is that: f_i is true with probability Π_i and false with probability $1 - \Pi_i$. These are considered independent. A choice of a truth value for every probabilistic fact defines a *world* w, whose probability is, according to the Distribution Semantics (DS) [21],

$$P(w) = \prod_{f_i \in w} \Pi_i \cdot \prod_{\neg f_i \in w} (1 - \Pi_i).$$

Every ProbLog program has 2^n worlds, where n is the number of probabilistic facts. The DS requires that every world has exactly one model. A *probabilistic clause* is a clause with a probabilistic fact in the head, such as $0.4::f(X) :- b(X, Y)$. The meaning is that $f(X)$ is true with probability 0.4 if the body $b(X, Y)$, which can also be a conjunction, is true. A probabilistic clause can be translated to a normal clause by inserting in the body a fresh probabilistic fact with the same probability and variables X and Y. The previous clause can be rewritten as $f(X) :- b(X, Y), aux_1(X, Y)$, where aux_1 is a new probabilistic fact with an associated probability of 0.4.

If we consider Answer Set Programming [14], the credal semantics [2,8] gives a meaning to answer set programs extended with probabilistic facts. We use the acronym PASP to denote both Probabilistic Answer Set Programming under the credal semantics and a probabilistic answer set program following the credal

semantics. The intended meaning will be clear from the context. Under this semantics, every world w is an answer set program, the one obtained by fixing to true the probabilistic facts true in the world w and by removing the probabilistic facts false in the world w, that has zero or more answer sets (or stable models [14]). Let us denote with $AS(w)$ the set of answer sets for a world w. Furthermore, the probability of a query q, i.e., a conjunction of ground atoms, is characterized by a *lower* and an *upper* probability bound. That is, $P(q) = [\underline{P}(q), \overline{P}(q)]$ where:

$$\underline{P}(q) = \sum_{w_i | \forall m \in AS(w_i),\ m \models q} P(w_i),\ \overline{P}(q) = \sum_{w_i | \exists m \in AS(w_i),\ m \models q} P(w_i). \quad (1)$$

A world w contributes to the lower and upper probability bounds if the query is true in every answer set of w. A world w contributes only to the upper probability bound if the query is true in some of the sets of w. Example 1 shows an example of PASP modeling a scenario where some people buy some products. A crucial point for the credal semantics is that every world must have at least one stable model. If this does not hold, some probability mass is lost. There are alternative semantics that handle worlds without answer sets, such as the credal least undefined semantics [20] or the smProbLog semantics [23], that we do not consider in this paper.

Example 1. This program models a scenario with three different people, Alice, Bob, and Carl, that may shop or not (probabilistic facts *shops*/1), with different probabilities.

```
0.3::shops(alice).
0.2::shops(bob).
0.6::shops(carl).

buy(beans,alice) ; buy(spaghetti,alice) :- shops(alice).
buy(spaghetti,bob) ; buy(steak,bob) :- shops(bob).
buy(tomato,carl) ; buy(garlic,carl) :- shops(carl).

cs(C):- #count{X : buy(spaghetti,X)} = C0,
        #count{X : buy(garlic,X)} = C1,
        C = C0 + C1.
ce(C):- #count{X,Y : buy(Y,X)} = C.

:- cs(S), ce(C), 10* S < 3*C.
```

The three disjuncive rules for *buy*/2 state that each one of the three people can buy different products. For instance, the first disjunctive rule stats that if Alice shops she can buy beans or spaghetti. The *cs*/1 and *ce*/1 rules contain aggregates in the body. The former unifies the number of people that buys spaghetti or garlic to variable C. These two values are computed via the *#count* aggregates. For instance, $\#count\{X : buy(spaghetti, X)\} = C0$ unifies with $C0$ the number of element X such that $buy(spaghetti, X)$ holds. Similarly, *ce*/1 unifies with C the number of pairs (X, Y) such that $buy(Y, X)$ is true. Lastly, a constraint

Table 1. Worlds, number of answer sets with the query $q = buy(spaghetti, alice)$ true (#ASq), and total number of answer sets (#AS) for every world of Example 1.

id	$shops(a)$	$shops(b)$	$shops(c)$	$P(w)$	#ASq	#AS
0	0	0	0	0.224	0	1
1	0	0	1	0.336	0	1
2	0	1	0	0.056	0	1
3	0	1	1	0.084	0	3
4	1	0	0	0.096	1	1
5	1	0	1	0.144	2	3
6	1	1	0	0.024	2	3
7	1	1	1	0.036	4	7

states that at least 30% of the people that buy something buy spaghetti or garlic. We are interested in computing the probability that Alice buys spaghetti. The program has $2^3 = 8$ worlds, listed in Table 1. If we consider the query $q = buy(spaghetti, alice)$, its probability is given by $P(q) = [P(w_4), P(w_4) + P(w_5) + P(w_6) + P(w_7)] = [0.096, 0.096 + 0.144 + 0.024 + 0.036] = [0.096, 0.3]$.

2.1 Statistical Statements

The authors of [3] proposed to represent statistical statements of the form "the fraction of A's that are also C's is between lp and up" with $lp, up \in [0, 1]$ with the syntax $(C \mid A)[lp, up]$ where C is an atom and A a conjunction of literals. All the variables in C also appear in A. They call this language PASTA. For example, if we want to state that at least 60% of the birds ($bird/1$) of a fixed domain fly ($fly/1$) we can write $(fly(X)|bird(X))[0.6, 1]$. To perform inference, a statistical statement is translated into three answer set rules: a disjunctive rule and two constraints with aggregates. The just described example becomes:

$$\begin{aligned} & fly(X); not\_fly(X) :- \ bird(X) \\ & :- \ \#count\{X : fly(X), bird(X)\} = FB, \#count\{X : bird(X)\} = B, \\ & \quad 10 \cdot FB < 6 \cdot B \\ & :- \ \#count\{X : fly(X), bird(X)\} = FB, \#count\{X : bird(X)\} = B, \\ & \quad 10 \cdot FB > 10 \cdot B \end{aligned} \tag{2}$$

The last rule can be omitted since the value of the variable FB cannot be greater than B. The lb and ub values are multiplied by 10 since ASP does not support floating point values. Note that this example can be rewritten with only one *count* aggregate instead of two, but we stick with the previous notation for clarity.

For a conditional $(C \mid A)[lp, up]$, if at least one of the literals in A is probabilistic, the program is interpreted as a PASP. Thus, we can compute the probability of a query with Eq. 1, leveraging the PASTA solver [3]. The inference

process of the PASTA solver is the following: first, probabilistic facts are converted into choice rules. For a query q, PASTA introduces two additional rules, $qr :- \ q$ and $nqr :- \ not\ q$. Then, it enumerates the projected answer sets [13] on the (converted) probabilistic facts and $qr/0$ and $nqr/0$ atoms and extracts the probability for every world w where the query is true and the contribution of w to the probability bounds.

2.2 Second Level Algebraic Model Counting

Weighted Model Counting (WMC) consists in summing the weights associated with the models of a given propositional formula (program). Algebraic Model Counting (AMC) [18] generalizes WMC by providing a generic definition based on semirings [15], that can be applied to many different tasks (see [18] for a comprehensive list), such as probabilistic inference. AMC can be solved via knowledge compilation [10], that involves representing the problem in a compact form where the solutions can be efficiently computed. As discussed in [17], some tasks such as decision theory and MAP inference require two levels of AMC, since they need two semirings for two different groups of variables. Thus, they belong to the class of Second Level Algebraic Model Counting (2AMC) [17] problems that can still be solved via knowledge compilation.

Let us introduce 2AMC more formally by following [17]. Given a tuple $A = (\Pi, X_{in}, X_{out}, w_{in}, w_{out}, \mathcal{R}_{in}, \mathcal{R}_{out}, f)$, where X_{in} and X_{out} are a partition of the variables in the propositional theory Π, $\mathcal{R}_{in} = (R^i, \oplus^i, \otimes^i, e^i_\oplus, e^i_\otimes)$ and $\mathcal{R}_{out} = (R^o, \oplus^o, \otimes^o, e^o_\oplus, e^o_\otimes)$ are two commutative semirings, w_{in} and w_{out} are two weight functions associating each atom of the program with a weight, and f is a transformation function from the values of $\mathcal{R}_{in}$ to $\mathcal{R}_{out}$, 2AMC requires solving:

$$2AMC(A) = \bigoplus^{o}_{I_{out} \in \sigma(X_{out})} \bigotimes^{o}_{a \in I_{out}} w_{out}(a) \otimes^o f(\bigoplus^{i}_{I_{in} \in \delta(\Pi | I_{out})} \bigotimes^{i}_{b \in I_{in}} w_{in}(b)) \tag{3}$$

where $\sigma(X_{out})$ are the set of possible assignments to X_{out} and $\delta(\Pi \mid I_{out})$ are the set of possible assignments to Π that satisfy I_{out}. AMC is a special case of 2AMC, where the set X_{out} is empty and the transformation function is the identity function. At a high level, the 2AMC task requires solving an AMC task on the variables X_{in} (inner semiring) for every possible set of assignments of X_{out} (outer semiring). The result of the inner AMC task is converted into an element of the outer semiring (with the function f) and another AMC task is solved, now on X_{out}. 2AMC has been adopted to perform inference in smProbLog [23], an extension of the ProbLog language that allows programs where worlds may have zero or more answer sets. A probability is assigned to every answer set in this way: the probability of a world is equally distributed among its answer sets. The probability of a query is then the sum of the probabilities of the answer sets where the query is true. If we consider a smProbLog program $\Pi = L \cup F$ with Herbrand base H and query q, where F is the set of probabilistic facts and L is the logical

part of the theory, $X_{out} = F$, $X_{in} = H \setminus F$, $\mathcal{R}_{in} = (\mathbb{N}^2, +, \cdot, (0,0), (1,1))$, $\mathcal{R}_{out} = ([0,1], +, \cdot, 0, 1)$ (i.e., the probability semiring), w_{in} associates all the literals to $(1,1)$ except for *not* q, that is mapped to $(0,1)$, w_{out} associates p and $1-p$ to respectively g and *not* g for every probabilistic fact $p::g$ and 1 to all the remaining literals, and the transformation function is $f(n_1, n_2) = n_1/n_2$ where n_2 is the number of models and n_1 the number of models where the query is true. In other words, the inner semiring counts both the number of models (n_2) and the number of models where the query is true (n_1), the transformation function computes the ratio n_1/n_2, and the outer semiring performs probabilistic inference. In the next section we show how we adapted this formulation to perform inference in PASP.

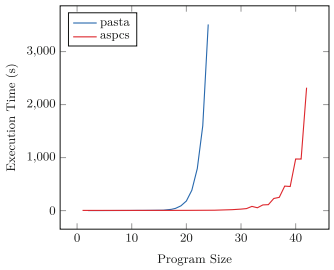
3 Inference in PASP as 2AMC

Recall from Sect. 2 that the probability of a query in a probabilistic answer set program is given by a range. The worlds in which the query is true in all the answer sets contribute to both the lower and upper bound while the worlds in which the query is true in some of the answer sets only contribute to the upper bound. If all the worlds have exactly one answer set, the task reduces to AMC since we just need to sum the probabilities of the worlds where the query is true. In the general case, the inference task is similar to the one of smProbLog described in the previous section: in smProbLog, the probability of a query is a sharp probability value, and every answer set is weighted by the probability of the world divided by the number of its answer sets. In PASP, we have a probability range, so we need to modify both the transformation function and the outer semiring. We consider a transformation function $f(n_1, n_2)$ that returns a pair of values f_{lp} and f_{up} where

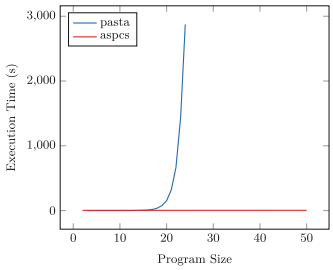
$$f_{lp} = \begin{cases} 1 \text{ if } n_1 = n_2 \\ 0 \text{ otherwise} \end{cases} \qquad f_{up} = \begin{cases} 1 \text{ if } n_1 > 0 \\ 0 \text{ otherwise} \end{cases}$$

where n_2 is the number of models and n_1 the number of models where the query is true. f_{lp} is adopted for the computation of the lower probability while f_{up} for the upper probability. We propose as outer semiring $\mathcal{R}_{out} = ([0,1]^2, +, \cdot, (0,0), (1,1))$, which is the probability semiring extended to two dimensions, where the operations $+$ and $\cdot$ are applied component-wise. Now, w_{out} associates (p,p) and $(1-p, 1-p)$ to respectively g and *not* g for every probabilistic fact $p::g$ and $(1,1)$ to all the remaining literals. In other words, the inner semiring counts the models and computes two values, n_1 and n_2. These are combined according to the above $f(n_1, n_2)$ function that also returns a pair of values. Then, the outer semiring performs the actual probability computation by considering these two values simultaneously and returns the lower and upper probability bounds for the query.

Inference in smProbLog is implemented in the aspmc solver [12] by means of tree decompositions and knowledge compilation with sd-DNNF [9] as target



(a) *qrnqr1* dataset.



(b) *qrnqr2* dataset.

Fig. 1. Results for the *qrnqr1* and *qrnqr2* datasets.

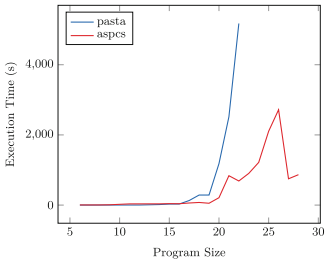
language. At a high level, the goal of tree decomposition is to represent a graph as a tree, where each vertex of the tree is a bag, i.e., a subset of the nodes of the graph. Every graph has one or more tree decompositions. The width of a tree decomposition is the size of its largest possible bag minus one. The treewidth of a graph is the minimum width of all its tree decompositions and represents how close a graph is to being a tree, and it is usually a good indicator of the hardness of a task [6]. We modify aspmc for smProbLog inference by introducing the novel transformation function and semiring. The knowledge compilation is still performed once even if the transformation function returns two values, since both are computed on the same semiring, and so the sd-DNNF is traversed only once. We call this algorithm aspcs. A limitation is that aspmc currently does not support aggregates, which are needed to represent PASTA programs. To overcome this, in the experiments of the following section we manually translate aggregates into ground rules. This, however, results in an exponential number of generated rules.

4 Experiments

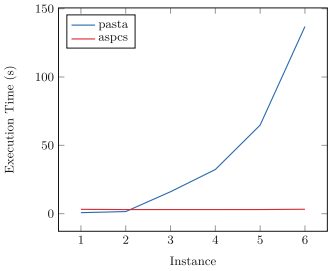
In this section, we compare the PASTA solver[1] with the previously introduced technique based on 2AMC (aspcs) implemented on top of the aspmc solver [12] on 6 different datasets. For all the datasets, except for the cases where we explicitly describe the instance, the number of probabilistic facts defines the size of the instance. All the considered datasets have at least one answer set for each world, i.e., admit the credal semantics. We use the c2d compiler [9] already available in aspmc. We ran the 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* and reported values are from the *real* field.

The first dataset, *qrnqr1*, consists of programs with an increasing number n of probabilistic facts $a(i)$, where $i \in [0, n-1]$, all associated to a probability

[1] Available at: https://github.com/damianoazzolini/pasta.

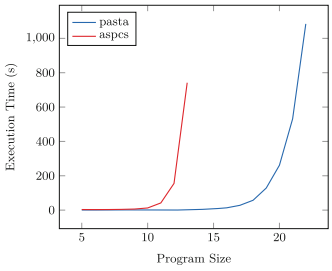


(a) *coloring* dataset.

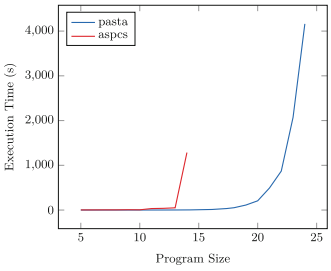


(b) *smoke* dataset.

Fig. 2. Results for the *coloring* and *smoke* datasets.



(a) *bird* dataset.



(b) *viralmarketing* dataset.

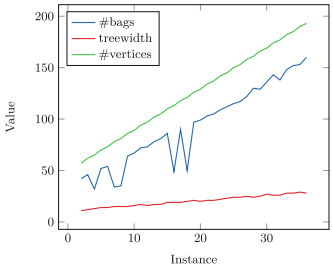
Fig. 3. Results for the *bird* and *viralmarketing* datasets.

of 0.4. This is an arbitrary value, since the probability of a probabilistic fact does not influence the execution time of the algorithm. For each index i, we include a rule $qr :- \ a(i)$ if i is even and two rules $qr :- \ a(1), not\ nqr$ and $nqr :- \ a(1), not\ qr$ (these are equivalent to the disjunctive rule $qr; nqr :- \ a(i)$) if i is odd. The query is qr. For example, the instance of size 4 is:

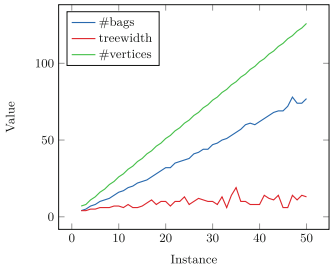
```
0.4::a(0). 0.4::a(1). 0.4::a(2). 0.4::a(3).
qr:- a(0).
qr :- a(1), not nqr. nqr :- a(1), not qr.
qr:- a(2).
qr :- a(3), not nqr. nqr :- a(3), not qr.
```

The *qrnqr2* dataset is similar to *qrnqr1*. Given an instance of size n, we have 3 rules: the first rule has qr in the head and all the probabilistic facts $a(i)$ with $i \in [0, n-1]$ and i even in the body. The second rule has qr in the head and all the probabilistic facts $a(i)$ with $i \in [0, n-1]$ and i odd and $not\ nqr$ in the body. The last rule has nqr in the head, all the probabilistic facts $a(i)$ with $i \in [0, n-1]$ and i odd, and $not\ qr$ in the body. The query is qr. For example, the instance of size 4 is:

```
0.4::a(0). 0.4::a(1). 0.4::a(2). 0.4::a(3).
```

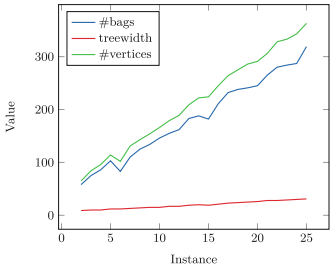


(a) *qrnqr1* dataset.

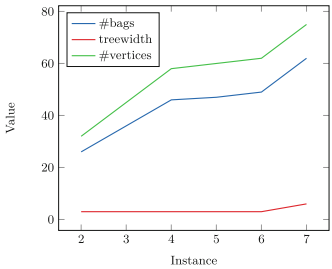


(b) *qrnqr2* dataset.

Fig. 4. Number of bags (#bags), treewdth, and number of vertices (# vertices) for the *qrnqr1* and *qrnqr2* datasets.



(a) *coloring* dataset.



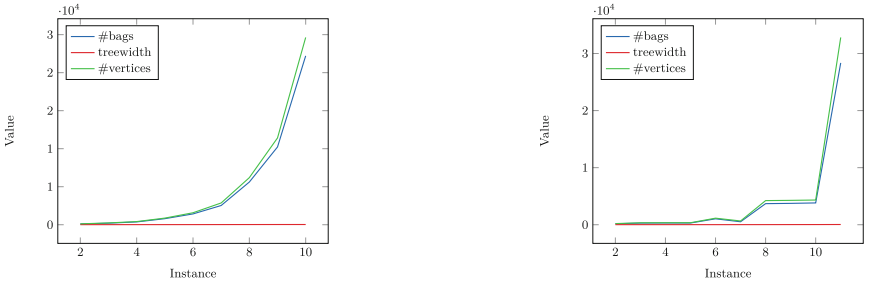
(b) *smoke* dataset.

Fig. 5. Number of bags (#bags), treewdth, and number of vertices (# vertices) for the *coloring* and *smoke* datasets.

```
qr:- a(0), a(2). qr :- a(1), a(3), not nqr.
nqr :- a(1), a(3), not qr.
```

The *coloring* dataset encodes a graph coloring task, where edges in the graph are associated with a random probability. Some nodes have a fixed color. The program of size 7 is:

```
0.6::edge(1,2). 0.1::edge(1,3).
0.4::edge(2,5). 0.3::edge(2,6).
0.3::edge(3,4). 0.8::edge(4,5).
0.2::edge(5,6).
node(1..6).
red(X) :- node(X), not green(X), not blue(X).
green(X) :- node(X), not red(X), not blue(X).
blue(X) :- node(X), not red(X), not green(X).
e(X,Y) :- edge(X,Y). e(Y,X) :- edge(Y,X).
:- e(X,Y), red(X), red(Y).
:- e(X,Y), green(X), green(Y).
:- e(X,Y), blue(X), blue(Y).
```



(a) *bird* dataset. (b) *viralmarketing* dataset.

Fig. 6. Number of bags (#bags), treewdth, and number of vertices (# vertices) for the *bird* and *viralmarketing* datasets.

```
red(1). green(4). green(6).
qr:- blue(3).
```

The query is qr. We generate increasing instances of this dataset by adding random probabilistic edges.

The dataset *smokers* is from [23]: it encodes a network of people, indexed with increasing integers starting from 1, where some have asthma (probabilistic rule $asthma/1$), some are stressed (probabilistic rule $stress/1$), and some smoke due to stress (probabilistic rule $smokes/1$), with probability 0.1, 0.3, and 0.4, respectively. If a person Y smokes and influences (probabilistic fact $influences/2$) a person X, then X will also smoke. Smokers have probability of 0.4 to have asthma. Finally, an asthmatic cannot smoke. The base instance, $i1$, is:

```
0.1::asthma(X) :- person(X).
0.3::stress(X) :- person(X).
0.4::smokes(X) :- stress(X).
smokes(X) :- influences(Y,X), smokes(Y).
0.4::asthma(X) :- smokes(X).
:- smokes(X), asthma(X).
person(1). person(2).
0.3::influences(1,2). 0.6::influences(2,1).
```

We are interested in the probability of $smokes(1)$. Increasing instances are: $i2 = i1 \cup \{person(3).\}$, $i3 = i2 \cup \{person(4).\}$, $i4 = i3 \cup \{0.2::influences(2,3).\}$, $i5 = i4 \cup \{0.7::influences(3,4).\}$, $i6 = i5 \cup \{0.9::influences(4,1).\}$. PASTA does not support probabilistic rules, so we manually ground the rules, remove the probability in the head and add a probabilistic fact in the body, a different one for every grounding.

We tested the two algorithms also on statistical statements. Since aspmc, and so aspcs, currently does not support aggregates, we manually translate them into ground rules. In the worst case, we get 2^n rules ($\sum_k \binom{n}{k} = 2^n$), where n is the size of the instance. The *bird* dataset contains an increasing number of probabilistic facts $0.4::bird(i)$, $i \in [0, n-1]$, and a conditional of the form:

```
(fly(X) | bird(X))[0.6,1].
```

which is equivalent to

```
fly(X) :- bird(X), not not_fly(X).
not_fly(X) :- bird(X), not fly(X).
:- #count{X:fly(X),bird(X)} = FB,
   #count{X:bird(X)} = B, 10*FB<6*B.
```

The query is $fly(1)$. The instance of size 3 with the constraint with aggregates converted into ground rules is:

```
b1:- bird(1). b1:- bird(2). b1:- bird(3).
:- b1, not fb1.
b2:- bird(1),bird(2). b2:- bird(1),bird(3).
b2:- bird(2),bird(3).
:- b2, not fb2.
b3:- bird(1),bird(2),bird(3).
:- b3, not fb2.
fb1:- fly(1). fb1:- fly(2). fb1:- fly(3).
fb2:- fly(1),fly(2). fb2:- fly(1),fly(3).
fb2:- fly(2),fly(3).
```

Predicate $bk/0$, $k \in \{1,2,3\}$, indicates that at least k probabilistic facts $bird(i)$ are true. Predicate $fbk/0$, $k \in \{1,2\}$ indicates that at least k facts $fly(i)$ are true. Note that the last constraint is $:-\ b3,\ not\ fb2$ because given the lower bound 0.6, we have $:-\ 10 \cdot FB < 6 \cdot 3$, so $:-\ FB < 2$. If the lower bound had been, for example, 0.7, the constraint would have been $:-\ b3,\ not\ fb3$ with an additional rule $fb3 :- fly(1), fly(2), fly(3)$. This translation of the conditionals yields 1420 rules for the instance of size 10, 3048 for size 11, 5694 for size 12, 12301 for size 13, and 22874 rules for size 14.

The *viralmarketing* dataset models a viral marketing scenario, where there is uncertainty on the people (probabilistic facts $person/1$) present in a network. These people advertise (predicate $advertise/2$) a product to other people. A person is reached by the advertisement if it is either directly advertised or advertised by a friend. If advertised, a person can buy or not the product. Finally, a constraint states that at least 70% of the people reached by an advertisement buy an item. The program of size 5 is the following:

```
0.1::person(1). 0.2::person(2).
0.3::person(3). 0.4::person(4). 0.5::person(5).
advertise(1,2):- person(1), person(2).
advertise(2,3):- person(2), person(3).
advertise(2,4):- person(2), person(4).
advertise(3,5):- person(3), person(5).
advertise(4,5):- person(4), person(5).
reach(A,B):- advertise(A,B).
reach(A,B):- advertise(A,C), reach(C,B).
reached(X):- person(X), reach(_,X).
```

```
reached(X):- person(X), advertise(X,_).
{buy(X)} :- reached(X).
:- #count{X:reached(X),buy(X)} = RB,
   #count{X:reached(X)} = R, 10*RB < 7*R.
```

As before, for aspcs, we replace the constraint with aggregates with a set of ground rules. This yields 2016 rules for the instance of size 10, 4055 for size 11, 8141 for size 12, 16322 for size 13, and 32330 for size 14. An instance of size n adds a new individual (person/1 probabilistic fact) and a random connection between two individuals (that are not already connected) to the instance of size $n - 1$. In addition to the execution times (Fig. 1, 2, and 3), we also plotted (Fig. 4, 5, and 6) some statistics of the tree decomposition adopted in aspmc (so aspcs), namely, number of bags, treewidth, and number of vertices.

Overall, when the program does not contain aggregates, aspcs outperforms PASTA in all the datasets. This is due to knowledge compilation adopted in aspmc and so on aspcs. This is particularly evident in Fig. 1b, where the execution time of aspcs seems to be almost constant, possibly because the number of bags, one of the main parameters that drives the construction of the compact form obtained by knowledge compilation, of all the instances is low (77 for the instance of size 50, see Fig. 4). A similar behavior is also present in Fig. 2b. In all the instances, PASTA reaches the time limit. In Fig. 1a, aspcs stops due to the memory limit. When aggregates are present (Fig. 3), aspcs stops due to the memory limit. This can be explained by the statistics of the tree decomposition, where the numbers of bags and vertices increase exponentially (Fig. 6). The biggest instance solved by aspcs for both datasets is 15. For these two datasets, PASTA is faster than aspcs, possibly because enumerating the answer sets is faster than encoding them into a propositional formula and compiling it with knowledge compilation. However, note again that the instance of size 14 of the *viralmarketing* dataset has 32330 rules, so aspcs can handle programs of significant size.

5 Conclusions

In this paper, we proposed aspcs, an algorithm based on aspmc to perform inference in probabilistic answer set programs following the credal semantics via Second Level Algebraic Model Counting. We tested our implementation against the PASTA solver on 6 different datasets. The first four have no aggregates, and aspcs is significantly faster than PASTA. The last two datasets contain statistical statements, and therefore aggregates, which must be manually translated into ground rules, since aspcs does not support them. This translation introduces many more rules, possibly an exponential number, making aspcs slower than PASTA.

Acknowledgements. This work has been partially supported by the Spoke 1 "FutureHPC & BigData" of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 - Next

Generation EU (NGEU), by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No. 952215, and by the "National Group of Computing Science (GNCS-INDAM)".

References

1. 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
2. Azzolini, D.: A brief discussion about the credal semantics for probabilistic answer set programs. In: Arias, J., et al. (eds.) Proceedings of the International Conference on Logic Programming 2023 Workshops co-located with the 39th International Conference on Logic Programming (ICLP 2023). CEUR Workshop Proceedings, vol. 3437, pp. 1–13. CEUR-WS.org (2023)
3. Azzolini, D., Bellodi, E., Riguzzi, F.: Statistical statements in probabilistic logic programming. In: Gottlob, G., Inclezan, D., Maratea, M. (eds.) LPNMR 2022. LNCS, vol. 13416, pp. 43–55. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15707-3_4
4. Azzolini, D., Bellodi, E., Riguzzi, F.: MAP inference in probabilistic answer set programs. In: Dovier, A., Montanari, A., Orlandini, A. (eds.) AIxIA 2022. LNCS, vol. 13796, pp. 413–426. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-27181-6_29
5. Bellodi, E., Alberti, M., Riguzzi, F., Zese, R.: MAP inference for probabilistic logic programming. Theory Pract. Logic Program. **20**(5), 641–655 (2020). https://doi.org/10.1017/S1471068420000174
6. Bliem, B., Morak, M., Moldovan, M., Woltran, S.: The impact of treewidth on grounding and solving of answer set programs. J. Artif. Intell. Res. **67**, 35–80 (2020). https://doi.org/10.1613/jair.1.11515
7. Van den Broeck, G., Thon, I., van Otterlo, M., De Raedt, L.: DTProbLog: a decision-theoretic probabilistic Prolog. In: Fox, M., Poole, D. (eds.) Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, pp. 1217–1222. AAAI Press (2010)
8. Cozman, F.G., Mauá, D.D.: The joy of probabilistic answer set programming: semantics, complexity, expressivity, inference. Int. J. Approximate Reasoning **125**, 218–239 (2020). https://doi.org/10.1016/j.ijar.2020.07.004
9. Darwiche, A.: New advances in compiling CNF into decomposable negation normal form. In: de Mántaras, R.L., Saitta, L. (eds.) 16th European Conference on Artificial Intelligence (ECAI 2004), pp. 328–332. IOS Press (2004)
10. Darwiche, A., Marquis, P.: A knowledge compilation map. J. Artif. Intell. Res. **17**, 229–264 (2002). https://doi.org/10.1613/jair.989
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 (2007)
12. Eiter, T., Hecher, M., Kiesel, R.: Treewidth-aware cycle breaking for algebraic answer set counting. In: Bienvenu, M., Lakemeyer, G., Erdem, E. (eds.) Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, pp. 269–279 (2021). https://doi.org/10.24963/kr.2021/26
13. 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
14. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: 5th International Conference and Symposium on Logic Programming (ICLP/SLP 1988), vol. 88, pp. 1070–1080. MIT Press, USA (1988)
15. Gondran, M., Minoux, M.: Graphs, Dioids and Semirings: New Models and Algorithms. Operations Research/Computer Science Interfaces Series, 1st edn. Springer, New York (2008). https://doi.org/10.1007/978-0-387-75450-5
16. Halpern, J.Y.: An analysis of first-order logics of probability. Artif. Intell. **46**(3), 311–350 (1990). https://doi.org/10.1016/0004-3702(90)90019-V
17. Kiesel, R., Totis, P., Kimmig, A.: Efficient knowledge compilation beyond weighted model counting. Theory Pract. Logic Program. **22**(4), 505–522 (2022). https://doi.org/10.1017/S147106842200014X
18. Kimmig, A., Van den Broeck, G., De Raedt, L.: Algebraic model counting. J. Appl. Logic **22**(C), 46–62 (2017). https://doi.org/10.1016/j.jal.2016.11.031
19. Riguzzi, F.: Foundations of Probabilistic Logic Programming Languages, Semantics, Inference and Learning, 2nd edn. River Publishers, Gistrup (2023)
20. Rocha, V.H.N., Gagliardi Cozman, F.: A credal least undefined stable semantics for probabilistic logic programs and probabilistic argumentation. In: Kern-Isberner, G., Lakemeyer, G., Meyer, T. (eds.) Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, pp. 309–319 (2022). https://doi.org/10.24963/kr.2022/31
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., De Raedt, L., Kimmig, A.: smProbLog: stable model semantics in ProbLog for probabilistic argumentation. Theory Pract. Logic Program. **23**, 1198–1247 (2023). https://doi.org/10.1017/S147106842300008X