Causal Inference in `cplint`

Fabrizio Riguzzi[a,*], Giuseppe Cota[b], Elena Bellodi[b], Riccardo Zese[b]

[a] *Dipartimento di Matematica e Informatica – Università di Ferrara*
Via Saragat 1, 44122, Ferrara, Italy
[b] *Dipartimento di Ingegneria – Università di Ferrara*
Via Saragat 1, 44122, Ferrara, Italy

Abstract

`cplint` is a suite of programs for reasoning and learning with Probabilistic Logic Programming languages that follow the distribution semantics. In this paper we describe how we have extended `cplint` to perform causal reasoning. In particular, we consider Pearl's *do* calculus for models where all the variables are measured. The two `cplint` modules for inference, PITA and MCINTYRE, have been extended for computing the effect of actions/interventions on these models. We also executed experiments comparing exact and approximate inference with conditional and causal queries, showing that causal inference is often cheaper than conditional inference.

Keywords: Probabilistic Logic Programming, Distribution Semantics, Logic Programs with Annotated Disjunctions, ProbLog, Causal Inference, Statistical Relational Artificial Intelligence

1. Introduction

Identifying cause-effect relationships among variables or events is one of the main objectives of science. How to extract such relationships from data and how to use them to make predictions have been fiercely debated. The connection between correlation and causation is particularly subtle and has mislead many authors. The famous sentence "correlation does not imply causation" is often used in statistics to warn against confounding the two. A correlation between two variables means that there is an association between them, but it does not imply that one causes the other. As a matter of fact, it could happen that there is a third factor that causes both producing the correlation.

Pearl in [1] showed that it is possible to represent causality by means of graphical models. Bayesian networks, in particular, are directed acyclic graphs that represent probabilistic dependencies between variables in a very intuitive

*Corresponding author
Email addresses: `fabrizio.riguzzi@unife.it` (Fabrizio Riguzzi), `giuseppe.cota@unife.it` (Giuseppe Cota), `elena.bellodi@unife.it` (Elena Bellodi), `riccardo.zese@unife.it` (Riccardo Zese)

way. In order to represent causality, Pearl [1] extended them into *causal Bayesian networks*, i.e. Bayesian networks where an arc from a node A to a node B means that A *directly causally influences* B. Computing the effect of actions can be performed in causal Bayesian networks by removing the edges that point to the nodes that represent the actions. The probability distribution of some variables E when performing action A, indicated as $P(E|do(A))$, denotes the effect of actions and is at the basis of Pearl's *do calculus*. This distribution can be computed by probabilistic inference on the mutilated network.

Representing probabilistic information in Logic Programming has been pursued by many authors. The distribution semantics [2] is one of the most widely used semantics for Probabilistic Logic Programming (PLP). This semantics is at the basis of many languages, such as Independent Choice Logic [3], PRISM [4], Logic Programs with Annotated Disjunctions (LPADs) [5] and ProbLog [6].

CP-logic [7] is a PLP language for causal reasoning. CP-logic programs (or CP-theories) are syntactically very similar to LPADs and they are given a semantics based on probability trees that represent possible courses of events. The authors proved that their semantics is suitable for representing causation and the effects of causal laws. For legal CP-logic programs, the semantics of LPADs and that of CP-logic coincide.

The authors in [7] also showed that the effect of actions in *do* calculus style can be computed from CP-theories by modifying the theory itself and computing the probability of the query from the modified theory. The modification involves adding facts for positive actions or removing (instantiations) of rules for negative actions.

In this paper we discuss how we implemented this process in practice in `cplint`,[1] a suite of programs for reasoning and learning in PLP. The two main inference modules of `cplint`, PITA and MCINTYRE, have been suitably extended for performing the *do* calculus, thus computing the effects of actions. PITA performs exact inference by knowledge compilation while MCINTYRE performs approximate inference by sampling so their extensions allow to perform both exact and approximate causal inference.

Like [7], we assume that the causal structure of the model is fully known. Pearl's *do* calculus is more general, as it allows to compute the effect of actions also on models with unknown variables. Exploiting the full power of the *do* calculus in PLP is a very interesting direction for future work.

We present two domains to illustrate causal reasoning in PLP: the famous Simpson's paradox and a viral marketing problem. We have also conducted experiments on the latter with an increasing number of members of the social network and compared exact and approximate conditional inference with exact and approximate causal inference. The results show that performing causal inference is often much less expensive than conditional inference, as expected, since actions remove dependencies among random variables.

The paper is organized as follows. Section 2 introduces preliminaries about

[1] `http://sites.unife.it/ml/cplint`

distribution semantics, causal reasoning and PLP. Section 3 and Section 4 describe the modules PITA and MCINTYRE respectively, together with their extension for causal reasoning; Section 5 shows some notable examples of causal inference, namely the Simpson's paradox and the viral marketing problem. Section 6 illustrates related work. Section 7 reports on the experiments performed and Section 8 concludes the paper.

2. Preliminaries

2.1. Probabilistic Logic Programming

The field of PLP has seen many different proposals for integrating logic programming and probability theory. We here concentrate on the Distribution Semantics (DS) [2] because it is one of the most widely used. The basic idea of the DS is that a probabilistic logic program defines a probability distribution over a set of normal logic programs (called *worlds*) that is extended to a joint probability of programs and truth values of a ground query. The probability of the query is then obtained from this joint distribution by marginalization.

We present the DS for LPADs for their general syntax. LPADs are sets of disjunctive clauses in which each atom in the head is annotated with a probability.

Formally a *Logic Program with Annotated Disjunctions* (LPADs) [5] consists of a finite set of annotated disjunctive clauses. An annotated disjunctive clause C_i is of the form

$$h_{i1} : \Pi_{i1}; \ldots ; h_{in_i} : \Pi_{in_i} \leftarrow b_{i1}, \ldots , b_{im_i}.$$

In such a clause the semicolon stands for disjunction, $h_{i1}, \ldots h_{in_i}$ are logical atoms and $b_{i1}, \ldots , b_{im_i}$ are logical literals, $\Pi_{i1}, \ldots , \Pi_{in_i}$ are real numbers in the interval $[0, 1]$ such that $\sum_{k=1}^{n_i} \Pi_{ik} \leq 1$. If $\sum_{k=1}^{n_i} \Pi_{ik} < 1$, the head of the annotated disjunctive clause implicitly contains an extra atom *null* that does not appear in the body of any clause and whose annotation is $1 - \sum_{k=1}^{n_i} \Pi_{ik}$.

Example 1. *The following LPAD T from [8] encodes a very simple model of the development of an epidemic or a pandemic:*

$C_1 = epidemic : 0.6; pandemic : 0.3 \leftarrow flu(X), cold.$
$C_2 = cold : 0.7.$
$C_3 = flu(david).$
$C_4 = flu(robert).$

An epidemic or a pandemic may arise if somebody has the flu and the climate is cold. We are uncertain whether the climate is cold and we know for sure that David and Robert have the flu.

We discuss the DS for the case in which the program does not contain function symbols so that its Herbrand base is finite[2]. An *atomic choice* is a selection of

[2]For the distribution semantics for programs with function symbols see [2, 9, 10].

the k-th atom for a grounding $C_i\theta_j$ of a probabilistic clause C_i and is represented by the triple (C_i, θ_j, k). A *selection* σ is a total set of atomic choices (one atomic choice for every grounding of each probabilistic clause). A set of atomic choices κ is *consistent* if $(C_i, \theta_j, k) \in \kappa, (C_i, \theta_j, m) \in \kappa$ implies $k = m$, i.e., only one head is selected for a ground clause.

A *composite choice* κ is a consistent set of atomic choices. A selection σ identifies a logic program w_σ called a *world*. The probability of w_σ is $P(w_\sigma) = \prod_{(C_i,\theta_j,k)\in\sigma} \Pi_{ik}$. Since the program does not contain function symbols, the set of worlds is finite $W_T = \{w_1, \ldots, w_m\}$ and $P(w)$ is a distribution over worlds: $\sum_{w\in W_T} P(w) = 1$.

A composite choice κ *identifies* a set ω_κ that contains all the worlds associated with a selection that is a superset of κ: i.e., $\omega_\kappa = \{w_\sigma | \sigma \in S_T, \sigma \supseteq \kappa\}$.

We can define the conditional probability of a query Q given a world w as: $P(Q|w) = 1$ if Q is true in w and 0 otherwise. The probability of the query can then be obtained by marginalizing over the query

$$P(Q) = \sum_w P(Q, w) = \sum_w P(Q|w)P(w) = \sum_{w\models Q} P(w) \tag{1}$$

Example 2. *For the LPAD T of Example 1, clause C_1 has two groundings, $C_1\theta_1$ with $\theta_1 = \{X/david\}$ and $C_1\theta_2$ with $\theta_2 = \{X/robert\}$, while clause C_2 has a single grounding $C_2\emptyset$. T has $3 \times 3 \times 2$ worlds, the query epidemic is true in 5 of them and its probability is $P(epidemic) = 0.6 \cdot 0.6 \cdot 0.7 + 0.6 \cdot 0.3 \cdot 0.7 + 0.6 \cdot 0.1 \cdot 0.7 + 0.3 \cdot 0.6 \cdot 0.7 + 0.1 \cdot 0.6 \cdot 0.7 = 0.588$.*

2.2. Causal Reasoning

The study of causation was connected to graphical models by Pearl [1], even if diagrams were already used to represent causal models as early as the 1920's [11] Graphical models are used to describe domains characterized by a set of random variables. Bayesian networks, in particular, are directed acyclic graphs where the variables are nodes and probabilistic dependences are represented as arcs: an arc from a node A to a node B means that A probabilistically influences B. An example of a Bayesian network is shown in Figure 1: it describes the domain of a medical study investigating the effects of a new drug on patients. The domain is described by three Boolean variables: Gender (F), Drug (C) and Recovery (E). Gender indicates the gender of the patient, Drug takes value 1 if the drug is administered to the patient under examination and value 0 if a placebo is administered, and Recovery whether the patient recovered from his illness. Gender influences Drug because the decision to administer or not the drug is taken on the basis of the sex of the patient. Gender and Drug influence Recovery because the outcome of the particular illness under examination depends on the sex of the patient and, hopefully, on the treatment.

Pearl [1] introduced *causal Bayesian networks*: these are Bayesian networks where an arc from a node A to a node B means that A directly causally influences B. Causal Bayesian networks can be used to perform causal reasoning, such as for example computing the effect of an action.

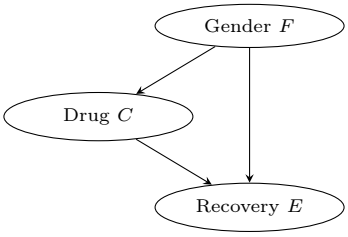

Figure 1: Bayesian network for a drug study domain.

An *action* or *intervention* in this context means setting a variable, say A, to a particular value, say a. The Bayesian network of Figure 1 is causal as we assume that the decision to administer or not the drug is taken on the basis of the sex of the patient. Moreover, the treatment and sex cause the patient to recover or not, as we assume that the illness depends on the gender.

In such a network one could for example ask what is the probability of recovery if we make the action of administering the drug, in other words what is the effect of the drug on recovery, the main aim of medical studies. This corresponds to computing the probability of $E = 1$ when setting C to 1. For answering such queries, Pearl shows that regular probabilistic reasoning cannot be used. So in this case computing $P(E = 1|C = 1)$ does not answer the question of what is the effect of the drug.

Pearl introduces a different calculus, called *do calculus*, to infer the effect of actions. In such a calculus, the action of setting a variable to a value is distinguished from the observation of that value for the variable. Actions appear inside a special *do* operator in the condition part of probabilistic queries. So to compute the effect of the drug on recovery, the query to answer is $P(E = 1|do(C = 1))$.

The *do* calculus reduces a query involving actions to a regular probabilistic query over a *mutilated* Bayesian network obtained by removing all incoming arcs from variables involved in actions. Then the query with actions as observations must be asked from the mutilated network. For example, to answer $P(E = 1|do(C = 1))$, the arc from Gender to Drug must be removed from the network of Figure 1 obtaining the network of Figure 2. Then the query $P(E = 1|C = 1)$ must be asked from the mutilated network. Note that there is no need to specify the conditional probability table (CPT) of the action variables (C in this case) in the mutilated network as the action variables are observed so the CPT does not influence the computation.

Equivalently, we can ask an unconditional query from the mutilated network where the CPTs for the actions are set so that all the probability mass is assigned to the values set by the actions. For the example above, the conditional probability table of C would be given by $P(C = 1) = 1$ and $P(C = 0) = 0$ and the query would be $P(E = 1)$.

It is very important not to confound $P(E|do(C))$ with $P(E|C)$ because the

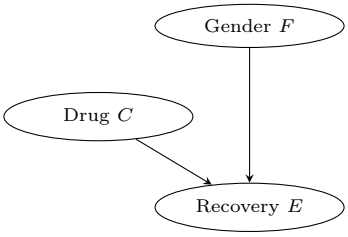

Figure 2: Mutilated version of the Bayesian network of Figure 1 for computing the effect of a drug.

results may be very different, as shown by the famous *Simpson's paradox*.

Example 3 (Simpson's Paradox). *From [1]:*

> *Simpson's paradox* [...] *refers to the phenomenon whereby an event C increases the probability of E in a given population p and, at the same time, decreases the probability of E in every subpopulation of p. In other words, if F and $\neg F$ are two complementary properties describing two subpopulations, we might well encounter the inequalities*
>
> $$P(E|C) > P(E|\neg C)$$
> $$P(E|C,F) < P(E|\neg C,F)$$
> $$P(E|C,\neg F) < P(E|\neg C,\neg F)$$
>
> [...] *For example, if we associate C (connoting cause) with taking a certain drug, E (connoting effect) with recovery, and F with being a female, then* [...] *the drug seems to be harmful to both males and females yet beneficial to the population as a whole.*

Consider the situation exemplified by the following tables from [1]:

Combined	E	$\neg E$		RecoveryRate
Drug(C)	20	20	40	50%
Nodrug($\neg C$)	16	24	40	40%
	36	44	80	

Females	E	$\neg E$		RecoveryRate
Drug(C)	2	8	10	20%
Nodrug($\neg C$)	9	21	30	30%
	11	29	40	

Males	E	$\neg E$		RecoveryRate
Drug(C)	18	12	30	60%
Nodrug($\neg C$)	7	3	10	70%
	25	15	40	

As you can see taking the drug seems to be beneficial overall even if it is not for females and males.

The paradox derives because we must distinguish seeing from doing: we must distinguish observing *that the drug was administered from the* intervention *of administering the drug. The conditioning operator in probability calculus stands for "given that we see", whereas the do operator means "given that we do". So the do operator must be used to infer the effect of actions. If the model of the domain is the network from Figure 1, to compute* $P(E = 1|do(C = 1))$ *and* $P(E = 1|do(C = 0))$ *we must compute* $P(E = 1|C = 1)$ *and* $P(E = 1|C = 0)$ *from the network of Figure 2 by using classical Bayesian inference. For these queries we get respectively 0.4 and 0.5, showing that the drug is not beneficial in the whole population exactly as it is not in the two subpopulations.*

Pearl's *do* calculus also deals with causal Bayesian networks where some of the variables are *unknown*, in the sense that we know that they exert an influence but they are not measurable so it is not possible to quantify this influence, i.e., we don't know how many they are and the CPTs where they are involved, we just know that some exist. When models contain such unknown variables, computing the effect of actions is not always possible, because we can't sum out the contribution of such variables since we don't know their number and CPTs. The *do* calculus provides rules for determining whether it is possible to compute the effect of an action even in the presence of unknown variables and to actually perform the computation. In this paper we consider only the *do* calculus for models with no unknown variables.

2.3. Causal Reasoning in Probabilistic Logic Programming

CP-logic [7] is a PLP language for causal reasoning whose semantics is based on probability trees that represent possible courses of events. The authors proved that their semantics is suitable for representing causation and the effects of causal laws. In particular, they highlighted that the inductive definitions of logic programming and the well-founded semantics of negation [12] produce models respecting most properties of causation, provided the program respects some weak constraints. The semantics of legal CP-logic programs coincides with that of LPADs, but there are LPADs that are not legal CP-theories, i.e., they cannot be assigned a causal semantics. However, these are corner cases in which the stratification level of a couple of atoms in a world is switched in a different world, so that it is not possible to establish a general stratification coherent with temporal precedence in all worlds.

The authors in [7] showed that the effect of actions in *do* calculus style can be computed from CP-theories when there are no unknown variables. In fact, clauses in CP-theories represent causal laws so in order to know the result of intervening on a single causal law, that law should be removed from the theory (and possibly replaced by a different law). For example, to compute the effects of an intervention that prevents a causal law C, that law must be removed from the theory. In case the intervention establishes a new causal law C', that law must be added to the theory. The modularity of CP-logic allows this.

Example 4. *The computation of the effects of interventions is illustrated in [7] with a medical example:*

> *A tumor in a patient's kidney might cause kidney failure, which might cause the death of the patient; however, to make matters even worse, the tumor can also metastasize to the brain, which might also, independently, kill the patient. We can represent this as:*
>
> $kidneyFailure : 0.1 \leftarrow kidneyTumor.$
> $brainTumor : 0.1 \leftarrow kidneyTumor.$
> $death : 0.5 \leftarrow brainTumor.$
> $death : 0.9 \leftarrow kidneyFailure.$

If we want to know what is the effect of putting the patient on a dialysis machine, which allows him to survive kidney failure, we can remove the last law and use the resulting theory for inference.

In this paper we start from the results in [7] and show how we modified inference in `cplint` to allow the computation of the effect of actions of the form $do(A)$ and $do(\backslash{+}A)$ where A is a ground literal. $do(A)$ means that the action A was performed i.e., the action makes A true, whereas $do(\backslash{+}A)$ means that the action makes the atom A false.

3. The PITA Module

PITA [13] computes the probability of a query from a probabilistic program in the form of an LPAD by knowledge compilation [14]. PITA computes explanations for the query and encodes them using Binary Decision Diagrams (BDDs), a language for representing Boolean functions. The probability of the query is given by the probability of the disjunction of the explanations, each explanation being a conjunction of equations of the form $Var = value$, where Var is a random variable associated with a ground clause and $value$ is a possible value (the index of one of the atoms in the head). BDDs encode the disjunction of the explanations using Shannon expansion, which means that the disjunction will be represented as a disjunction of mutually exclusive terms. This permits the computation of the probability with a single visit of the BDD.

PITA computes BDDs for explanations by transforming an LPAD into a normal program containing calls for manipulating BDDs. The idea is to add an extra argument to each subgoal to store a BDD encoding the explanations for the answers of the subgoal. The values of the extra argument of the subgoals are combined using a set of library functions:

- *init, end*: initialize and terminate the data structures for manipulating BDDs;
- *zero(-D), one(-D)*: return BDD D representing the Boolean constant 0 and 1;

- *and(+D1,+D2,-DO), or(+D1,+D2,-DO), not(+D1,-DO)*: Boolean operations between BDDs;
- *get_ var_ n(+R,+S,+Probs,-Var)*: returns the multi-valued random variable associated with rule R with grounding substitution S and list of probabilities *Probs*;
- *equality(+Var,+Value,-D)*: D is the BDD representing *Var=Value*, i.e. that the multivalued random variable *Var* is assigned *Value*;
- *ret_ prob(+D,-P)*: returns the probability P of the BDD D.

These functions are implemented in C as an interface to the CUDD[3] library for manipulating BDDs. A BDD is represented in Prolog as an integer that is a pointer in memory to the root node of the BDD.

The PITA transformation applies to atoms, literals, conjunctions of literals and clauses. The transformation for an atom h and a variable D, $PITA(h, D)$, is h with the variable D added as the last argument. For the sake of simplicity, we consider here only positive literals, but the transformation can be applied also to negative literals (see [13]).

The transformation for a conjunction of literals $b_1, \ldots, b_m$ is

$PITA(b_1, \ldots, b_m, D) = one(DD_0),$
$PITA(b_1, D_1), and(DD_0, D_1, DD_1), \ldots,$
$PITA(b_m, D_m), and(DD_{m-1}, D_m, D).$

The disjunctive clause $C_r = h_1 : \Pi_1 \vee \ldots \vee h_n : \Pi_n \leftarrow b_1, \ldots, b_m.$ where the parameters sum to 1, is transformed into the set of clauses $PITA(C_r)$:

$PITA(C_r, i) = PITA(h_i, D) \leftarrow PITA(b_1, \ldots, b_m, DD_m),$
$get\_var\_n(r, S, [\Pi_1, \ldots, \Pi_n], Var),$
$equality(Var, i, DD), and(DD_m, DD, D).$

for $i = 1, \ldots, n$, where S is a list containing all the variables appearing in C_r. If the parameters do not sum up to 1, the body is empty or the clause is non-disjunctive (a single head with probability 1), the transformation can be optimised.

We assume programs to be range restricted. A program is *range restricted* if all the variables appearing in the head also appear in positive literals in the body. In this case, when the goal $get\_var\_n(r, S, [\Pi_1, \ldots, \Pi_n], Var)$ is called, all the variables of the original clause, listed in S, are instantiated so $get\_var\_n/4$ can associate a random variable with the instantiation of clause C_r.

The PITA transformation applied to clause C_1 of Example 1 yields

[3] http://vlsi.colorado.edu/~fabio/CUDD/

$PITA(C_1, 1) = epidemic(D) \leftarrow$
$one(DD_0), flu(X, D_1), and(DD_0, D_1, DD_1),$
$cold(D_2), and(DD_1, D_2, DD_2),$
$get\_var\_n(1, [X], [0.6, 0.3, 0.1], Var),$
$equality(Var, 1, DD), and(DD_2, DD, D).$
$PITA(C_1, 2) = pandemic(D) \leftarrow$
$one(DD_0), flu(X, D_1), and(DD_0, D_1, DD_1),$
$cold(D_2), and(DD_1, D_2, DD_2),$
$get\_var\_n(1, [X], [0.6, 0.3, 0.1], Var),$
$equality(Var, 2, DD), and(DD_2, DD, D).$

PITA is available for XSB Prolog [15], YAP Prolog [16] and SWI-Prolog [17].

The XSB version, the initial one, uses tabling, a logic programming technique that reduces computation time and ensures termination for a large class of programs [15]. The idea of tabling is simple: keep a store of the subgoals encountered in a derivation together with answers to these subgoals. If one of the subgoals is encountered again, the answers are retrieved from the store rather than recomputing them. Besides saving time, tabling ensures termination for programs without function symbols under the well-founded semantics [12].

PITA also uses a feature of XSB tabling called *answer subsumption* [15] that, when a new answer for a tabled subgoal is found, combines old answers with the new one according to a partial order or lattice. This feature is used to combine the BDDs that are built for different explanations of a goal, using $or/3$ as the join operation of the lattice and $zero/1$ as the predicate returning the bottom element of the lattice. For example, a unary predicate $p/1$ must be declared as tabled by means of the declaration *table p(_,or/3-zero/1).* If an answer $p(a, d_1)$ was found and a new answer $p(a, d_2)$ is derived, the answer $p(a, d_1)$ is replaced by $p(a, d_3)$, where d_3 is obtained by calling $or(d_1, d_2, d_3)$.

To compute the probability of a ground atom A, PITA uses predicate $prob/2$ whose definition is

$prob(A, Prob) \leftarrow$
$add\_bdd\_arg(A, D, A1),$
$call(A1),$
$ret\_prob(D, Prob).$

where $add\_bdd\_arg(A, D, A1)$ performs the PITA transformation $PITA(A, D)$ for a literal A and a variable D, and $A1$ contains the transformed literal. Since YAP and SWI-Prolog do not have answer subsumption in their tabling implementation, the collection of the various explanations for the goal is performed explicitly with this definition of $prob/2$:

$prob(A, Prob) \leftarrow$
$add\_bdd\_arg(A, D, A1),$
$findall(D, A1, L),$
$zero(Zero),$
$foldl(or, L, Zero, DD),$
$ret\_prob(DD, Prob).$

where $foldl/4$ implements the higher order functional programming fold function and is available in the `apply` library of YAP and SWI-Prolog.

3.1. Conditional Exact Inference

To compute the probability of a conjunction of ground goals G given another conjunction of ground goals E, two clauses are added to the knowledge base:

$$\$goal(D) \leftarrow PITA(G, D).$$

$$\$ev(D) \leftarrow PITA(E, D).$$

and the queries $\$goal(DG)$ and $\$ev(DE)$ are asked. DG will contain the BDD representing the explanations for the goal and DE the BDD representing the explanations for the evidence. Then the conjunction of DG and DE is computed obtaining DGE. The probability to be returned is the fraction of the probability of DGE over the probability of DE, as shown in Algorithm 1.

Algorithm 1 Algorithm for computing the conditional probabilities.

```
1: function PROB(T, G, E)                    ▷ Program T, goal G, evidence E
2:     Add $goal(D) ← PITA(G, D). to T
3:     Add $ev(D) ← PITA(E, D). to T
4:     Ask the queries $goal(DG) and $ev(DE)
5:     DGE ← bdd_and(DG, DE)
6:     PGE ← ret_prob(DGE)
7:     PE ← ret_prob(DE)
8:     return PGE/PE
9: end function
```

3.2. Causal Exact Inference

When performing causal inference, evidence E may contain ground literals of the form $do(A)$, meaning that ground literal A is an action rather than an observation.

In this case, evidence E is partitioned into two conjunctions, EO containing only the observation atoms and EA containing all the literals A for which E contains $do(A)$. Let $remove\_do$ be the function taking as input a conjunction of do literals and returning $remove\_do(EA) = \{A | do(A) \in EA\}$.

The knowledge base is extended with

$$\$goal(D) \leftarrow PITA(G, D).$$

as for non causal inference, plus

$$\$ev(D) \leftarrow PITA(EO, D).$$

Then Algorithm 2 is used to obtain a new program on which conditional inference as in PITA is performed. The algorithm considers every action of the form $do(A) \in EA$ with $A = p(t_1, \ldots, t_n)$ or $A = \backslash{+}p(t_1, \ldots, t_n)$ and, for each

clause with $p(u_1, \ldots, u_n, D)$ in the head, it adds to the body the conjunction of constraints $dif(u_1, t_1), \ldots, dif(u_n, t_n)$. Then the clause

$$p(t_1, \ldots, t_n, D) \leftarrow one(D).$$

is added to the program for every action of the form $do(p(t_1, \ldots, t_n))$ (positive actions).

$dif/2$ is a coroutine predicate that expresses disequality of terms. The actual test is delayed until the terms are sufficiently instantiated to be found different, or have become identical. The predicate is available in most Prolog systems and is usually implemented by means of attributed variables [18].

By using $dif/2$, the body of the clause fails as soon as a disequality is violated. If we had used the disunification predicate $\backslash=/2$, we should have inserted the disequality constraints at the end of the body, just before the call to $get\_var\_n/4$, because at the beginning of the body some variables may not be instantiated. This would have resulted in a waste of computation, as failure would be obtained only after having resolved all the literals in the body. With $dif/2$ failure may be obtained earlier.

The result is correct as shown by Theorem 1.

Algorithm 2 Algorithm for preparing the knowledge base for exact causal inference.

```
1: function PrepPITAKB(T, EA)                ▷ Program T, set of literals
   appearing as do actions in the evidence EA
2:     for all do(A) ∈ EA with A = p(t_1, ..., t_n) or A = \+p(t_1, ..., t_n) do
3:         for all clauses C = p(u_1, ..., u_n, D) ← B do
4:             Remove C from T
5:             Add p(u_1, ..., u_n, D) ← dif(u_1, t_1), ..., dif(u_n, t_n), B to T
6:         end for
7:     end for
8:     for all do(A) atom in EA with A = p(t_1, ..., t_n) do
9:         Add p(t_1, ..., t_n, D) ← one(D). to T
10:    end for
11:    return T
12: end function
```

Theorem 1. *Given a a goal G and an evidence E, the probability for G to be true given that E holds $P(G|E)$ on program T has the same value as*

$$\text{prob}(\text{PreparePITAKB}(T, EA), G, EO).$$

Proof 1. *By including the $dif/2$ constraints in the body, we effectively make sure that, when evaluating the body of the clauses (causal laws), all groundings of the clauses whose head matches with one of the action atoms produce failure, resulting in the same effect as removing the ground causal law from the theory.*

The addition of clauses $p(t_1, \ldots, t_n, D) \leftarrow one(D).$ *for every positive action* $do(p(t_1, \ldots, t_n))$ *then ensures that* $p(t_1, \ldots, t_n)$ *is forced to true, and the absence of any clause for* $p(t_1, \ldots, t_n)$ *for negative actions* $do(\backslash{+}p(t_1, \ldots, t_n))$ *ensures that* $p(t_1, \ldots, t_n)$ *is forced to false.*

In this way we adopt the strategy of [7] for representing interventions in CP-logic.

4. The MCINTYRE Module

MCINTYRE [8] performs approximate inference by sampling. It first transforms the program and then queries the transformed program. The disjunctive clause $C_i = h_{i1} : \Pi_{i1} \vee \ldots \vee h_{in} : \Pi_{in_i} \leftarrow b_{i1}, \ldots, b_{im_i}.$ where the parameters sum to 1, is transformed into the set of clauses $MC(C_i)$:

$$\begin{aligned} MC(C_i, 1) = \quad & h_{i1} \leftarrow b_{i1}, \ldots, b_{im_i}, \\ & sample\_head(ParList, i, S, NH), NH = 1. \\ \ldots & \\ MC(C_i, n_i) = \quad & h_{in_i} \leftarrow b_{i1}, \ldots, b_{im_i}, \\ & sample\_head(ParList, i, S, NH), NH = n_i. \end{aligned}$$

where S is a list containing each variable appearing in C_i and $ParList$ is $[\Pi_{i1}, \ldots, \Pi_{in_i}]$. If the parameters do not sum up to 1, the last clause (the one for *null*) is omitted. Basically, we create a clause for each head and we sample a head index at the end of the body with `sample_head/4`. If this index coincides with the head index, the derivation succeeds, otherwise it fails. Thus failure can occur either because one of the body literals fails or because the current clause is not part of the sample.

For example, clause C_1 of Example 1 becomes

$$\begin{aligned} MC(C_1, 1) = \quad & epidemic \leftarrow flu(X), cold, \\ & sample\_head([0.6, 0.3, 0.1], 1, [X], NH), NH = 1. \\ MC(C_1, 2) = \quad & pandemic \leftarrow flu(X), cold, \\ & sample\_head([0.6, 0.3, 0.1], 1, [X], NH), NH = 2. \end{aligned}$$

The predicate `sample_head/4` samples an index from the head of a clause and uses the built-in Prolog predicates `recorded/3` and `recorda/3` for respectively retrieving or adding an entry to the internal database. Since `sample_head/4` is at the end of the body and since we assume the program to be range restricted, at that point all the variables of the clause have been grounded. If the rule instantiation had already been sampled, `sample_head/4` retrieves the head index with `recorded/3`, otherwise it samples a head index with `sample/2`:

$$\begin{aligned} & sample\_head(\_ParList, R, VC, NH) \leftarrow \\ & \quad recorded(exp, (R, VC, N), \_), !, NH = N. \\ & sample\_head(ParList, R, VC, NH) \leftarrow \\ & \quad sample(ParList, NH), \\ & \quad recorda(exp, (R, VC, NH), \_). \end{aligned}$$

Tabling can be effectively used to speed up the computation. To sample a truth value for a ground atom *Goal* from the program we use the following predicate

$sample(Goal) \leftarrow$
  $abolish\_all\_tables$,
  $eraseall(exp)$,
  $call(Goal)$.

To compute the probability of a query, a number N of samples is taken and the probability is given by S/N where S is the number of times that `sample/1` succeeds.

4.1. Conditional Approximate Inference

Similarly to PITA, to compute the probability of a conjunction of ground goals G given another conjunction of ground goals E, two clauses are added to the knowledge base:

$$\$goal \leftarrow G.$$

$$\$ev \leftarrow E.$$

Conditional inference in MCINTYRE can be performed either by rejection sampling or by Metropolis-Hastings Markov Chain Monte Carlo (MCMC) [19]. In rejection sampling [20], you take a sample by first querying the evidence (with $sample(\$ev)$) and, if the query is successful, query the goal in the same sample (with $sample(\$goal)$), otherwise the sample is discarded.

In Metropolis-Hastings MCMC [21], a Markov chain is built by taking an initial sample and by generating successor samples. A sample corresponds to a composite choice, which in turn corresponds to a set of worlds (see Subsection 2.1). The initial sample κ_0 is built by randomly sampling choices so that the evidence is true. A successor sample κ is obtained by deleting a fixed number of sampled probabilistic choices, i.e. $\kappa_0 \supset \kappa$. Then the evidence is queried by taking a sample κ' starting with the undeleted choices with $\kappa \subset \kappa'$. If the query succeeds, the goal is queried by taking a sample κ'' with $\kappa' \subset \kappa''$, otherwise κ' is discarded. The sample is accepted with a probability of $\min\{1, \frac{|\kappa|}{|\kappa''|}\}$ where $|\kappa|$ is the number of choices (i.e. atomic choices) sampled in the previous sample and $|\kappa''|$ is the number of choices sampled in the current sample. Then the number of successes of the query is increased by 1 if the query succeeded in the last accepted sample. The final probability is given by the number of successes over the number of samples.

4.2. Causal Approximate Inference

As for PITA, the evidence E is partitioned into the conjunctions EO of observation atoms and EA of action atoms. Then the knowledge base is extended with

$$\$goal \leftarrow G.$$

as for non causal inference, plus

$$\$ev \leftarrow EO.$$

Then Algorithm 3, MCINTYRE's version of Algorithm 2, is used to preprocess the program before using MCINTYRE algorithms for conditional inference. You

can notice that in Alg. 3 the variable D is missing (see predicates in Alg. 2), this is because variable D in exact inference is used to contain the BDD, but in approximate inference we just use sampling without building any BDDs. It

Algorithm 3 Algorithm for preparing the knowledge base for approximate causal inference.

1: **procedure** PREPAREMCKB(T, EA) ▷ Program T, set of literals appearing as *do* actions in the evidence EA
2: **for all** $do(A) \in EA$ with $A = p(t_1, \ldots, t_n)$ or $A = \backslash{+}p(t_1, \ldots, t_n)$ **do**
3: **for all** clauses $C = p(u_1, \ldots, u_n) \leftarrow B$ **do**
4: Remove C from T
5: Add $p(u_1, \ldots, u_n) \leftarrow dif(u_1, t_1), \ldots, dif(u_n, t_n), B$ to T
6: **end for**
7: **end for**
8: **for all** $do(A)$ atom in EA with $A = p(t_1, \ldots, t_n)$ **do**
9: Add A to T
10: **end for**
11: **end procedure**

is easy to see that Theorem 1 holds also for MCINTYRE. Figure 3 shows the architecture of the cplint system with their module and algorithms used for causal inference.

5. Notable Examples

In this section we illustrate the implementation in `cplint` of two famous problems: the Simpson's paradox and the viral marketing problem.

5.1. Simpson's Paradox

The medicine study of Example 3 can be represented with the program of Figure 4[4]. Here, `:- action drug/0.` means that `drug/0` is a predicate that can be used to specify actions. We need the `:- action p/n` directive because the predicate `p` should be declared as *dynamic* in order to perform `retract/1` (execution of line 4 in Algorithm 2). In PITA the directive `:- action p/n` makes the predicate `p/n+2` *dynamic*. In MCINTYRE, instead, it has the effect to make `p/n` *dynamic*.

We query the conditional probabilities of recovery given treatment on the whole population and on the two subpopulations with:

```
?- prob(recovery,drug,P).
?- prob(recovery,\+ drug,P).
?- prob(recovery,(drug,female),P).
?- prob(recovery,(\+drug,female),P).
```

[4] Also available at `http://cplint.lamping.unife.it/example/inference/simpson.swinb`.

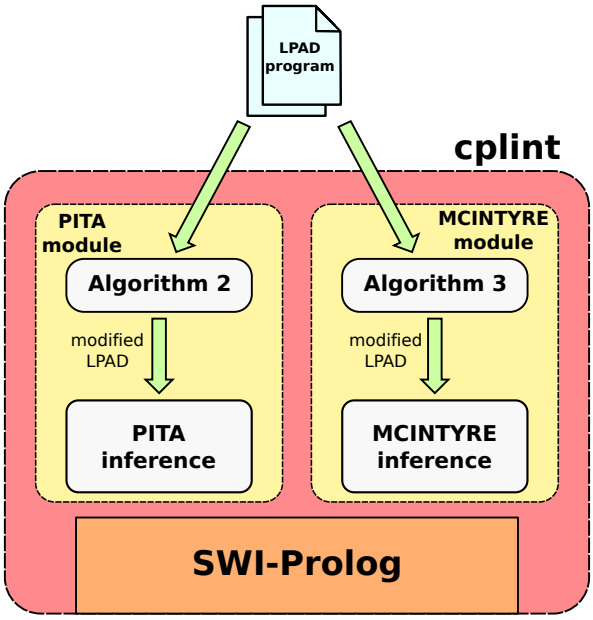

Figure 3: Architecture of cplint for causal inference.

```
?- prob(recovery,(drug,\+ female),P).
?- prob(recovery,(\+ drug,\+ female),P).
```

The results of these queries are those in the tables of Example 3.

If instead we want to know the probability of recovery given the action treatment (taking a drug), we must ask

```
?- prob(recovery,do(drug),P).
?- prob(recovery,do(\+ drug),P).
?- prob(recovery,(do(drug),female),P).
?- prob(recovery,(do(\+drug),female),P).
?- prob(recovery,(do(drug),\+ female),P).
?- prob(recovery,(do(\+ drug),\+ female),P).
```

The results of the last four queries are the same as the last four conditional queries, so the probability of recovery in the two subpopulations is the same as that for the case of seeing rather than doing, as the observation of sex makes the arc from sex to drug irrelevant.

The results of the first two *do* queries instead differ from the conditional ones: they are respectively 0.4 and 0.5, showing that the drug is not beneficial and that the probability of recovery on the whole population is now in accordance with that in the subpopulations, in particular it is the weighted average of the probability of recovery in the subpopulations.

5.2. Viral Marketing

Let us now consider a viral marketing scenario inspired by [22]. A firm is interested in marketing a new product to its customers. These are connected in a social network that is known to the firm: the network represents the trust relationships between customers. The firm has decided to adopt a marketing strategy that involves giving the product for free to a number of its customers, in the hope that these influence the other customers and entice them to buy the product. The firm wants to choose the customers to which marketing is applied so that its return is maximised. This involves computing the probability that the non-marketed customers will acquire the product given the action to the marketed customers.

We can model this domain with an LPAD where the predicate `trust/2` encodes the links between customers in the social network and the predicate `has/1` is true for customers that possess the product, either received as a gift or bought. Predicate `trust/2` is defined by a number of certain facts, while predicate `has/1` is defined by two rules, one expressing the prior probability of a customer to buy the product and one expressing the fact that if a trusted customer has the product, then there is a certain probability that the trusting customer buys the product. The complete LPAD is shown in Figure 5[5]. The social network encoded by the program is represented in Figure 6. If the firm wants to estimate the effect of giving the product for free to customer 3 on the probability of customer 2 buying the product, the query to ask is

```
?- prob(has(2),do(has(3)),P).
```

This query on the program above returns 0.136. If instead we query

```
?- prob(has(2),has(3),P).
```

we get 0.407, showing that not distinguishing seeing from doing leads to an overly optimistic estimate.

6. Related Work

P-log [23] is a probabilistic logic programming language that is equipped with a system capable of handling causal reasoning. Differently from LPADs, the semantics of P-log programs is based on Answer Set Programming (ASP) and the possible worlds are the models of the program interpreted as an ASP program. As such, multiple worlds are generated not only because of probabilistic constructs but also because of logical constructs, negation in particular. The viral marketing program of Section 5.2 can be encoded in P-log as in Figure 7.

The P-log system performs reasoning on such program by computing the whole set of possible worlds using an ASP reasoner. This means enumerating all possible worlds, which can be very expensive. For example, the program

[5] Also available at `http://cplint.lamping.unife.it/example/inference/viral.swinb`.

above has $4 + 4 * 4 = 20$ Boolean random variables generating 2^{20} possible worlds, making the computation of the example queries of Section 5.2 much more expensive than with PITA: P-log was stopped before the end after several minutes of computation, while PITA returns the results almost instantaneously (less than one second) on the same machine. By comparison, P-log achieves a similar running time only when the program above is restricted to three nodes, for a total of $3 + 3 * 3 = 12$ Boolean random variables and $2^{12} = 4096$ worlds. These results indicate that P-log is more suited for programs mixing probabilistic and advanced non-monotonic constructs. If these features are not needed, `cplint` can achieve better results.

Some languages, such as ICL [9] and ProbLog [24], only allow facts as probabilistic clauses. This does not limit the expressiveness, as it is possible to transform an LPAD into an ICL or ProbLog program. For example, the viral marketing program translated into ProbLog is shown in Figure 8.

Considering ProbLog as an example, if an action involves a predicate defined only by probabilistic facts, causal inference can be performed by conditional inference. Since probabilistic facts have no parents, in the program above $P(has(2)|do(apriori(3)))$ is equal to $P(has(2)|apriori(3))$ and, at the same time, $P(has(2)|do(\backslash+apriori(3)))$ is equal to $P(has(2)|\backslash+apriori(3))$. On the other hand, if actions involve predicates defined by rules, as for example in $P(has(2)|do(has(3)))$, the previous simple approach does not apply. In fact, for the action $do(A)$, one should look for all groundings of all probabilistic facts on which A depends and include them in the evidence. This requires a partial evaluation of the program. For the example above one could compute $P(has(2)|do(has(3)))$ by computing $P(has(2)|apriori(3), viral(3,1), viral(3,2))$ but in general the partial evaluation may be costly.

Anyway, in case a program can be rewritten by having all predicates for actions defined by facts only, then causal inference can be performed by conditional inference or unconditional inference on simple modifications of the program. This is the approach taken for example in [9], which describes a scenario where there is a robot and a key, the robot can pick up or put down the key and move to different locations[6]. In this example actions are defined only by (certain) facts so their effects can be computed by adding or removing the facts encoding the actions.

The authors of [25] proposed an approach to perform the full *do* calculus on propositional causal models using Answer Set Programming. Moreover, they present an algorithm for inducing models from data. Our approach differs from this because we consider inference for relational causal models, albeit in a restricted case. Therefore our causal random variables can be parameterized by logical variables, as $has(P)$ in the viral marketing example: we have a different causal Boolean variable $has(p)$ for each person p and the rules defining the predicate $has/2$ serve as a template for building a complex propositional model

[6] Available also in ProbLog at `https://dtai.cs.kuleuven.be/problog/tutorial/various/14_robot_key.html`.

of the dependence of $has(p)$ from its causes.

7. Experiments

In this section we aim to evaluate the performance of causal reasoning with `cplint` while comparing exact inference, performed with PITA, with approximate inference, performed with Metropolis-Hastings of MCINTYRE. Given the different focus of P-log, a comparison with this system would be unfair. Therefore we compare the performance of causal reasoning in `cplint` with regular probabilistic reasoning. All the experiments here presented were executed on a Linux machine equipped with a Intel Xeon E5-2630 v3 @ 2.40 GHz CPU with 8 GB of main memory.

In particular, we considered the viral marketing domain. We generated random social networks of increasing size and we evaluated random probabilistic and causal queries with an increasing number of evidence literals. The random social networks were generated as scale-free graphs according to the Barabasi-Albert model [26]. We used the `sample_pa`[7] function of the `igraph` R library to generate the graphs with parameter `m` set to 2 (the number of edges to be added at each time step is 2). We considered a number of nodes from 10 to 100 in steps of 10 and, for each number of nodes, we generated 10 graphs (for a total of 100 different generated graphs). For each number of nodes, we generated conjunctions of literals of the form $has(n)$ where n is a node sampled uniformly at random from the set of nodes. For each number of literals from 2 to 8 in steps of 2 we generated 10 random conjunctions with that number of literals. For each conjunction C_l with l literals, we sampled uniformly a node m and we prepared the queries $P_l = P(has(m)|C_l)$ and $Q_l = P(has(m)|do(C_l))$, where $do(C_l) = \{do(A)|A \in C_l\}$.

Then we posed the queries P_l and Q_l to each of the 10 graphs for each number of nodes and we measured the execution time. The computed time was averaged over the 10 graphs with the same number of nodes and the 10 conjunctions with the same number of literals. Hence we have 100 queries for each number of nodes. We set a timeout of 600 seconds for each query and we set to 1000 the number of samples for MCINTYRE.

The average runtime for conditional and causal queries are then plotted in Figures 9-12 as a function of the number of nodes. Tables 1-4 show the average timings with their 95% confidence intervals.

In particular, Figure 10 shows that with 4 evidence literals and a graph size larger than 60 nodes at least one conditional query with exact inference has encountered the timeout. Whereas causal queries (both with exact and approximate inference) and conditional queries with approximate inference are still feasible. Figures 11 and 12 show that at least one conditional query with exact inference has encountered the timeout for graphs with more than 10 nodes and queries with 6 evidence literals or more. In all the figures we can see that the

[7] `http://igraph.org/r/doc/sample_pa.html`

running time of conditional inference increases with the size of the graphs, while the runtime of causal inference is roughly constant. In these experiments the average running time for causal approximate inference is below 130 milliseconds for every graph size, whereas causal exact inference is surprisingly faster than the approximate one and the average running time is below 4 milliseconds for every graph size. The causal exact inference is faster than the approximate one because, in our example, there is a small number of explanations for each causal query, therefore it takes less time to compute all the explanations than it does to sample the probabilistic logic program 1000 times. Table 5 reports the mean squared error of approximate causal inference (caus mcint). We can notice that the errors are less than $4{\cdot}10^{-3}$, proving that the proposed approximate approach gives results close enough to the exact ones.

Inference method	**Size of the dataset**				
	10	**20**	**30**	**40**	**50**
cond exact	0.32 ± 0.09	0.89 ± 0.63	3.01 ± 1.59	2.73 ± 1.40	2.19 ± 1.52
caus exact	3.15 ± 0.55	2.92 ± 0.05	2.97 ± 0.05	2.98 ± 0.07	2.93 ± 0.06
cond mcint	168.62 ± 8.04	185.28 ± 6.15	197.05 ± 8.32	201.6 ± 5.91	204.22 ± 6.77
caus mcint	46.11 ± 3.19	57.42 ± 2.5	64.94 ± 4.08	73.96 ± 4.2	63.57 ± 3.44
Inference method	**Size of the dataset**				
	60	**70**	**80**	**90**	**100**
cond exact	11.41 ± 4.96	13.91 ± 7.31	13.91 ± 5.65	20.24 ± 8.34	37.29 ± 28.30
caus exact	3.03 ± 0.08	3.01 ± 0.10	3.00 ± 0.06	2.98 ± 0.09	3.36 ± 0.46
cond mcint	225.34 ± 8.68	227.66 ± 8.91	232.13 ± 10.47	237.46 ± 9.24	252.91 ± 12.9
caus mcint	74.27 ± 4.36	78.98 ± 6.6	77.81 ± 5.32	77.06 ± 6.77	81.26 ± 7.9

Table 1: Execution time (in milliseconds) and 95% confidence intervals for conditional and causal queries with 2 evidence literals. The size of the datasets is expressed in number of nodes of the graph.

8. Conclusions

While performing causal reasoning in PLP has been discussed before, no existing system allows to perform causal reasoning in an easy, user-friendly and fast way. This paper discusses how we have implemented causal reasoning in the `cplint` system, thus providing a practical point of view on causal inference. Causal queries on models with no unknown variables can now be answered with exact and approximate inference by exploiting the PITA and MCINTYRE modules respectively. We conducted experiments on the viral marketing problem with random social networks of increasing size. We compared the performance of causal reasoning in `cplint` with regular probabilistic reasoning. The results show that the modification of the inference algorithms do not impact on the execution time and that causal reasoning is in effect cheaper than conditional inference, as expected, thus showing that causal inference is suitable for real life

Inference method	Size of the dataset				
	10	**20**	**30**	**40**	**50**
cond exact	4.43 ± 1.27	66.98 ± 29.64	811.73 ± 446.72	936.18 ± 626.14	854.23 ± 682.3
caus exact	2.84 ± 0.07	2.95 ± 0.05	2.95 ± 0.07	3.04 ± 0.06	2.9 ± 0.06
cond mcint	236.18 ± 8.89	289.33 ± 11.77	350.97 ± 20.46	381.16 ± 25.24	364.11 ± 21.96
caus mcint	41.91 ± 3.25	56.88 ± 4.14	65.93 ± 4.48	79.38 ± 4.48	63.48 ± 3.38

Inference method	Size of the dataset				
	60	**70**	**80**	**90**	**100**
cond exact	2372.11 ± 1291.21	–	–	–	–
caus exact	3.04 ± 0.08	3.06 ± 0.08	3.03 ± 0.07	3.07 ± 0.06	3.37 ± 0.38
cond mcint	438.48 ± 36.96	453.89 ± 34.03	464.05 ± 35.7	482.93 ± 32.17	522.62 ± 39.74
caus mcint	71.89 ± 5.27	82.19 ± 6.04	78.64 ± 5.98	94.64 ± 7.01	103.73 ± 10.01

Table 2: Execution time (in milliseconds) and 95% confidence intervals for conditional and causal queries with 4 evidence literals. The dash means that the timeout was reached. The size of the datasets is expressed in number of nodes of the graph.

applications. The new inference algorithm are available in the `cplint` pack of SWI-Prolog and can be tried online at `http://cplint.lamping.unife.it` [27].

In the future, we plan to extend the system for performing the full *do* calculus, in order to deal with models where the causal influences are partly unknown.

Acknowledgments:

This work was supported by the "National Group of Computing Science (GNCS-INDAM)".

Inference method	Size of the dataset				
	10	**20**	**30**	**40**	**50**
cond exact	158.37 ± 38.36	–	–	–	–
caus exact	2.81 ± 0.08	2.95 ± 0.06	2.91 ± 0.06	3.13 ± 0.08	2.97 ± 0.07
cond mcint	331 ± 14.59	506.29 ± 40.93	558.82 ± 68.84	686.51±91.9	795.54 ± 122.04
caus mcint	44.5 ± 1.4	51.95 ± 4.5	66.74 ± 5.25	92.36 ± 7.05	72.96 ± 4.31
Inference method	**Size of the dataset**				
	60	**70**	**80**	**90**	**100**
cond exact	–	–	–	–	–
caus exact	3.12 ± 0.08	3.1 ± 0.08	3.1 ± 0.07	3.09 ± 0.07	3.51 ± 0.54
cond mcint	939.05 ± 248.19	1075.76 ± 149.54	1015.65 ± 160.3	1260.15 ± 199.99	1240.61 ± 208.2
caus mcint	86.13 ± 6.21	99.42 ± 7.7	83.02 ± 7.53	96.71 ± 8.15	109.95 ± 12.87

Table 3: Execution time (in milliseconds) and 95% confidence intervals for conditional and causal queries with 6 evidence literals. The dash means that the timeout was reached. The size of the datasets is expressed in number of nodes of the graph.

```
:- use_module(library(pita)).
:- pita.
:- begin_lpad.
:- action drug/0.
female:0.5.
recovery:0.6:- drug,\+ female.
recovery:0.7:- \+ drug,\+ female.
recovery:0.2:- drug,female.
recovery:0.3:- \+ drug,female.
drug:30/40:- \+ female.
drug:10/40:-female.
:-end_lpad.
```

Figure 4: LPAD for Simpson's paradox.

```
:- use_module(library(pita)).
:- pita.
:- begin_lpad.
:- action has/1.
has(_) : 0.1.
has(P) : 0.4 :- trusts(P, Q), has(Q).
trusts(2,1).
trusts(3,1).
trusts(3,2).
trusts(4,1).
trusts(4,3).
:- end_lpad.
```

Figure 5: LPAD for viral marketing.

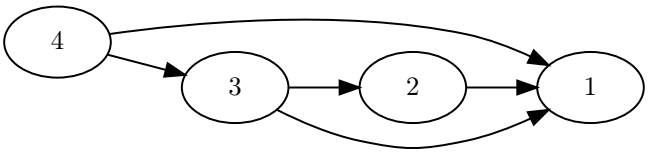

Figure 6: Social network for the viral marketing example.

```
bool={t,f}.
node={1..4}.
#domain node(P;Q).
has1: node -> bool.
has2: node,node -> bool.
[ri(P)] random(has1(P)).
[ri(P)] pr(has1(P,t))=1/10.
[rn(P,Q)] random(has2(P,Q)).
[rn(P,Q)] pr(has2(P,Q,t))=4/10.
has(P):- has1(P,t).
has(P):- has2(P,Q,t),trusts(P,Q),has(Q).
trusts(2,1).
trusts(3,1).
trusts(3,2).
trusts(4,1).
trusts(4,3).
```

Figure 7: P-log program for viral marketing.

```
has(P):- apriori(P).
has(P):- trusts(P, Q), has(Q), viral(P,Q).
apriori(_):0.1.
viral(_,_):0.4.
trusts(2,1).
trusts(3,1).
trusts(3,2).
trusts(4,1).
trusts(4,3).
```

Figure 8: ProbLog program for viral marketing.

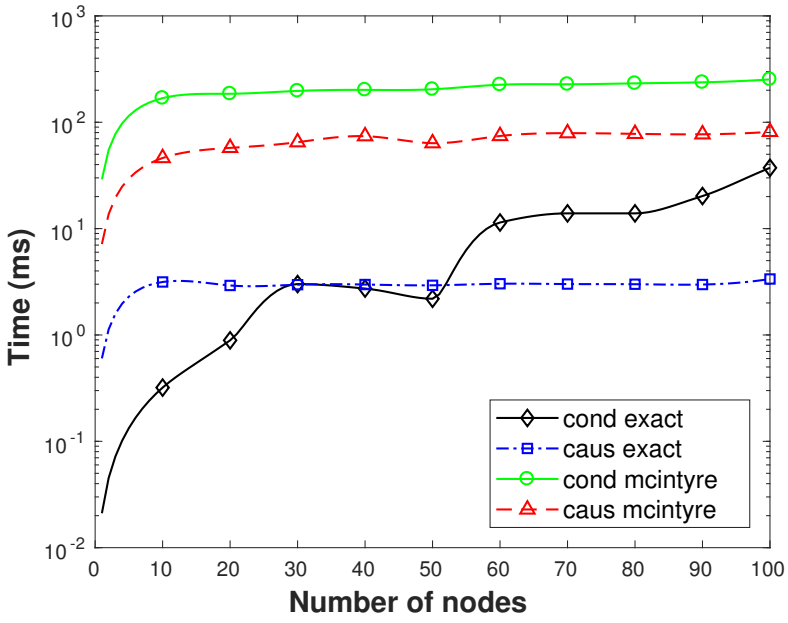

Figure 9: Average time for conditional and causal queries with 2 evidence literals.

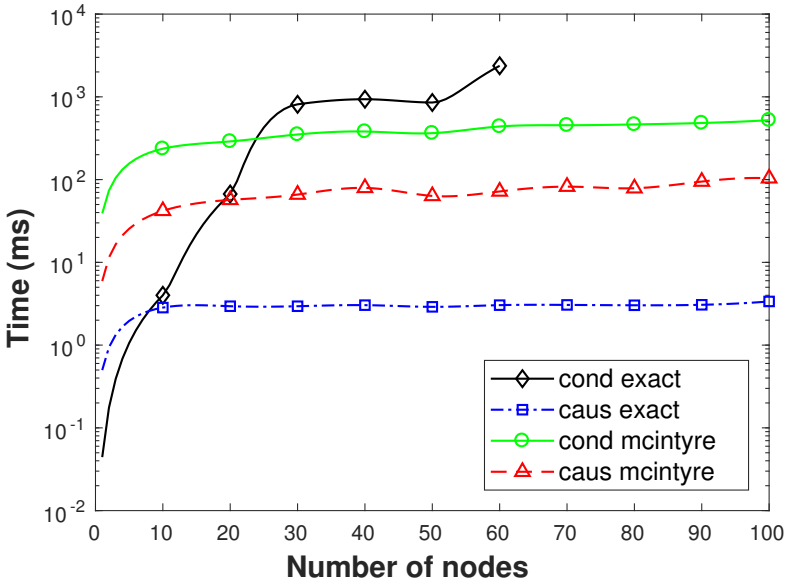

Figure 10: Average time for conditional and causal queries with 4 evidence literals.

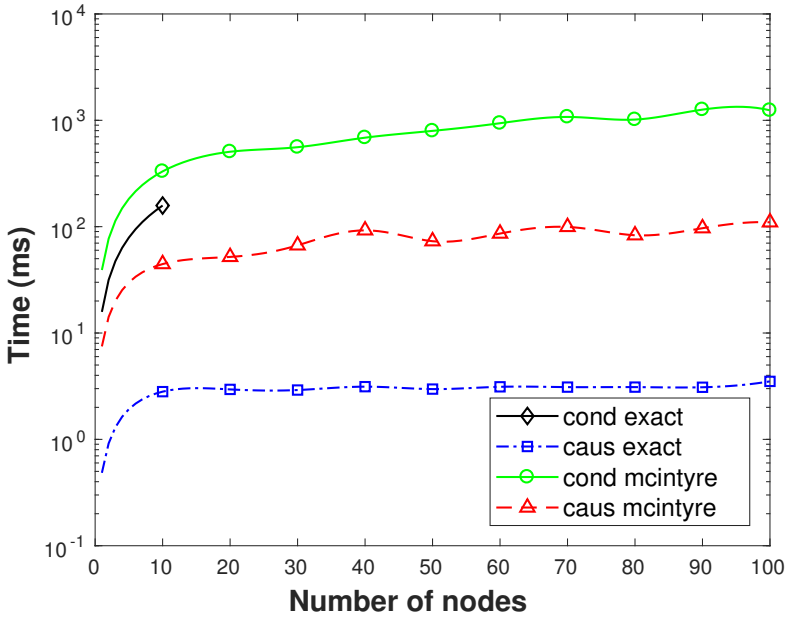

Figure 11: Average time for conditional and causal queries with 6 evidence literals.

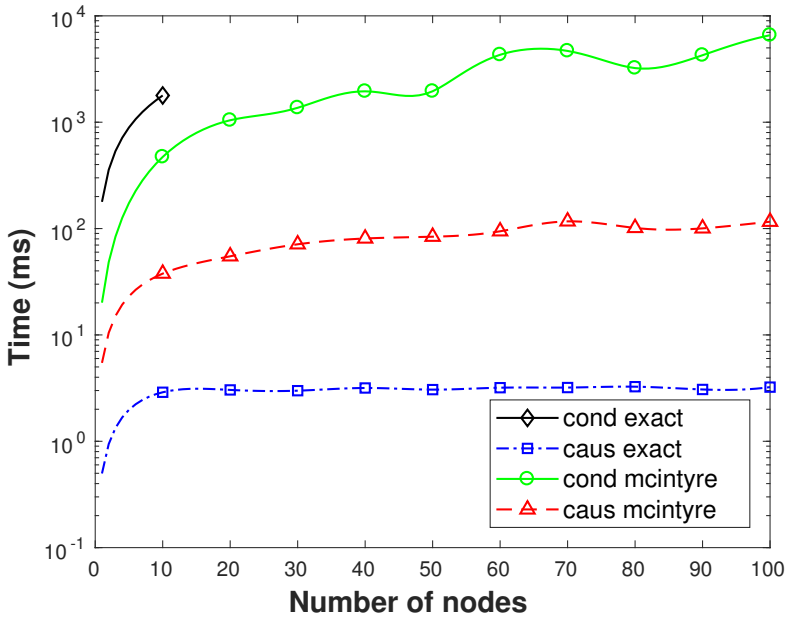

Figure 12: Average time for conditional and causal queries with 8 literal evidences.

Inference method	**Size of the dataset**				
	10	**20**	**30**	**40**	**50**
cond exact	1784.48 ± 451.27	–	–	–	–
caus exact	2.89 ± 0.07	3.04 ± 0.06	2.99 ± 0.07	3.17 ± 0.08	3.06 ± 0.07
cond mcint	471.56 ± 34.17	1043.48 ± 255.12	1366.8 ± 232.7	1952.12 ± 382.1	1954.56 ± 336.25
caus mcint	37.71 ± 2.28	54.86 ± 3.42	71.15 ± 4.29	80.67 ± 8.1	83.86 ± 5.85
Inference method	**Size of the dataset**				
	60	**70**	**80**	**90**	**100**
cond exact	–	–	–	–	–
caus exact	3.19 ± 0.08	3.2 ± 0.08	3.26 ± 0.09	3.07 ± 0.06	3.23 ± 0.09
cond mcint	4306.83 ± 1759.19	4679.31 ± 1304.03	3227.32 ± 1234.65	4265.53 ± 1375.54	6576.63 ± 2313.33
caus mcint	94.28 ± 7.36	116.85±9.25	101.36±9.51	100.42±7.34	115.94±9.33

Table 4: Execution time (in milliseconds) and 95% confidence intervals for conditional and causal queries with 8 evidence literals. The dash means that the timeout was reached. The size of the datasets is expressed in number of nodes of the graph.

Evidence literals	Size of the dataset									
	10	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
2	2.5	1.7	2.4	2.4	1.6	2.3	2.3	3.0	2.0	2.6
4	0.9	1.8	2.4	2.7	1.6	2.6	2.6	2.2	2.7	2.9
6	1.2	1.4	2.5	3.9	1.7	2.7	2.3	2.3	2.1	2.3
8	0.9	2.2	1.5	3.2	2.3	2.3	3.1	2.0	2.5	2.0

Table 5: Mean Squared Error for approximate causal inference (caus mcintyre). All the values must be multiplied by 10^{-3}. The size of the datasets is expressed in number of nodes of the graph.

References

[1] J. Pearl, Causality, Cambridge University Press, 2000.

[2] T. Sato, A Statistical Learning Method for Logic Programs with Distribution Semantics, in: L. Sterling (Ed.), 12th International Conference on Logic Programming, Tokyo, Japan, MIT Press, Cambridge, Massachusetts, 1995, pp. 715–729.

[3] D. Poole, The Independent Choice Logic for modelling multiple agents under uncertainty, Artificial Intelligence 94 (1997) 7–56.

[4] T. Sato, Y. Kameya, PRISM: a language for symbolic-statistical modeling, in: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97), Vol. 97, 1997, pp. 1330–1339.

[5] J. Vennekens, S. Verbaeten, M. Bruynooghe, Logic Programs With Annotated Disjunctions, in: B. Demoen, V. Lifschitz (Eds.), Logic Programming, 24th International Conference, ICLP 2004, Saint-Malo, France, Proceedings, Vol. 3131 of Lecture Notes in Computer Science, Springer, Berlin Heidelberg, Germany, 2004, pp. 431–445. doi:10.1007/978-3-540-27775-0_30.

[6] L. De Raedt, A. Kimmig, H. Toivonen, ProbLog: A probabilistic Prolog and its application in link discovery, in: M. M. Veloso (Ed.), Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India (IJCAI-07), Vol. 7, AAAI Press, Palo Alto, California USA, 2007, pp. 2462–2467.

[7] J. Vennekens, M. Denecker, M. Bruynooghe, CP-logic: A language of causal probabilistic events and its relation to logic programming, Theory and Practice of Logic Programming 9 (3) (2009) 245–308.

[8] F. Riguzzi, MCINTYRE: A Monte Carlo system for probabilistic logic programming, Fundamenta Informaticae 124 (4) (2013) 521–541. doi:10.3233/FI-2013-847.

[9] D. Poole, Abducing through negation as failure: Stable models within the independent choice logic, J. Logic Program. 44 (1-3) (2000) 5–35.

[10] F. Riguzzi, The distribution semantics for normal programs with function symbols, International Journal of Approximate Reasoning 77 (2016) 1 – 19. doi:10.1016/j.ijar.2016.05.005.

[11] S. Wright, Correlation and causation, Journal of agricultural research 20 (7) (1921) 557–585.

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

[13] F. Riguzzi, T. Swift, The PITA system: Tabling and answer subsumption for reasoning under uncertainty, Theory and Practice of Logic Programming 11 (4–5) (2011) 433–449. doi:10.1017/S147106841100010X.

[14] A. Darwiche, P. Marquis, A knowledge compilation map, Journal of Artificial Intelligence Research 17 (2002) 229–264.

[15] T. Swift, D. S. Warren, XSB: Extending prolog with tabled logic programming, Theory and Practice of Logic Programming 12 (1-2) (2012) 157–187. doi:10.1017/S1471068411000500.

[16] V. Santos Costa, R. Rocha, L. Damas, The YAP Prolog system, Theory and Practice of Logic Programming 12 (1-2) (2012) 5–34.

[17] J. Wielemaker, T. Schrijvers, M. Triska, T. Lager, SWI-Prolog, Theory and Practice of Logic Programming 12 (1-2) (2012) 67–96. doi:10.1017/S1471068411000494.

[18] C. Holzbaur, Metastructures vs. attributed variables in the context of extensible unification, in: M. Bruynooghe, M. Wirsing (Eds.), Programming Language Implementation and Logic Programming: 4th International Symposium, PLILP'92 Leuven, Belgium, August 26–28, 1992 Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 1992, pp. 260–268. doi:10.1007/3-540-55844-6_141.

[19] M. Alberti, E. Bellodi, G. Cota, F. Riguzzi, R. Zese, cplint on SWISH: Probabilistic logical inference with a web browser, Intelligenza Artificiale 11 (1) (2017) 47–64.

[20] J. Von Neumann, Various techniques used in connection with random digits, Nat. Bureau Stand. Appl. Math. Ser. 12 (1951) 36–38.

[21] A. Nampally, C. Ramakrishnan, Adaptive MCMC-based inference in probabilistic logic programs, arXiv preprint arXiv:1403.6036.

[22] G. Van den Broeck, I. Thon, M. van Otterlo, L. De Raedt, Dtproblog: A decision-theoretic probabilistic prolog, in: M. Fox, D. Poole (Eds.), 24th AAAI Conference on Artificial Intelligence, AAAI'10, Atlanta, Georgia, USA, July 11-15, 2010, AAAI Press, 2010, pp. 1217–1222.

[23] C. Baral, M. Gelfond, N. Rushton, Probabilistic reasoning with answer sets, Theory and Practice of Logic Programming 9 (1) (2009) 57–144. doi:10.1017/S1471068408003645.

[24] A. Kimmig, B. Demoen, L. De Raedt, V. S. Costa, R. Rocha, On the implementation of the probabilistic logic programming language ProbLog, Theory and Practice of Logic Programming 11 (2-3) (2011) 235–262.

[25] A. Hyttinen, F. Eberhardt, M. Järvisalo, Do-calculus when the true graph is unknown., in: 31st International Conference on Uncertainty in Artificial Intelligence (UAI-15), 2015, pp. 395–404.

[26] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.

[27] F. Riguzzi, E. Bellodi, E. Lamma, R. Zese, G. Cota, Probabilistic logic programming on the web, Software: Practice and Experience 46 (10) (2016) 1381–1396. doi:10.1002/spe.2386.