Modeling Markov Decision Processes with Imprecise Probabilities Using Probabilistic Logic Programming

Thiago P. Bueno TBUENO@IME.USP.BR
Denis D. Mauá DDM@IME.USP.BR
Leliane N. de Barros LELIANE@IME.USP.BR
Instituto de Matemática e Estatística - Universidade de São Paulo
Rua do Matão, 1010
São Paulo, S (Brazil)

Fabio G. Cozman FGCOZMAN@USP.BR
Escola Politécnica - Universidade de São Paulo
Av. Prof. Mello Moraes, 2231
São Paulo, P (Brazil)

Abstract

We study languages that specify Markov Decision Processes with Imprecise Probabilities (MDPIPs) by mixing probabilities and logic programming. We propose a novel language that can capture MDPIPs and Markov Decision Processes with Set-valued Transitions (MDPSTs); we then obtain the complexity of one-step inference for the resulting MDPIPs and MDPSTs. We also present results of independent interest on the complexity of inference with probabilistic logic programs containing interval-valued probabilistic assessments. Finally, we also discuss policy generation techniques.

Keywords: Markov decision processes; MDP; MDPIP; MDPST; imprecise probabilities; non-determinism; probabilistic logic programming; credal semantics.

1. Introduction

To be able to plan, one must be able to represent the relation between actions and their consequences on the world. Operator-based languages such as STRIPS or PDDL (Fikes and Nilsson, 1971; McDermott et al., 1998) have been devised so as to encode *deterministic* sequential decision problems, with a specific solution in mind (heuristic search). Action languages such as $\mathcal{A}$ or $\mathcal{C}$ (Giunchiglia and Lifschitz, 1998), as well as programming languages such as GOLOG (Levesque et al., 1997), add more expressiveness, but also focus primarily on deterministic problems. Other languages focus on decision under uncertainty; for instance, PPDDL (Younes and Littman, 2004), RDDL (Sanner, 2010), DT-GOLOG (Boutilier et al., 2000). In particular, languages based on probabilistic logic programming (Kersting and De Raedt, 2003; Nitti et al., 2015; Srivastava et al., 2014; Bueno et al., 2016) allow for probabilities, while $\mathcal{C}+$ (Giunchiglia et al., 2004) and $\mathcal{K}$ (Eiter et al., 2004) allow for nondeterminism. There are languages that even allow both probabilities and nondeterminism (Halpern and Tuttle, 1993; Eiter and Lukasiewicz, 2003; Trevizan et al., 2008; Iocchi et al., 2009).

In this paper, we study the properties of planning domain description languages that have enough power so as to encode Markov Decision Processes with Imprecise Probabilities (MDPIPs) (White III and Eldeib, 1994; Delgado et al., 2009, 2011). We propose a novel language based on probabilistic logic programming, enhanced with decision theoretic constructs such as actions, state fluents and utilities. We consider interval-valued probabilities attached to independent facts, and we adopt a semantics given by Lukasiewicz (2007) within the context of probabilistic description logics. The semantics assigns probability measures over answer sets (Gelfond and Lifschitz, 1988). As has been recently noted by Cozman and Mauá (2016), this semantics induces an infinitely-monotone Choquet capacity on intepretations of atoms. We show that our language can be used to specify Markov Decision Processes with Set-valued Transitions (MDPSTs) when all probabilities are point-valued. This class of MDPIPs encompass a wide spectrum of planning tasks ranging from the classical, deterministic case to the probabilistic setting in which actions have stochastic and/or uncertain effects (Trevizan et al., 2007, 2008). We derive the complexity of one-step inference with the resulting languages; we also present results of independent interest on the complexity of inference with probabilistic logic programs containing interval-valued probabilistic assessments. We also discuss how to generate optimal policies from a specification in our language, in this paper focusing on MDPSTs.

The paper is organized as follows. We offer some background knowledge on MDPIPs and MDPSTs, and on probabilistic logic programming, in Section 2. We then present our language in Section 3. We discuss the complexity of one-step inference in Section 4, and describe policy generation algorithms in Section 5. Finally, Section 6 concludes the paper.

2. Background

In this section we review the main concepts behind Markov Decision Processes and some of their variants. We also summarize the main ideas in probabilistic logic programming.

2.1 MDPs, MDPIPs and MDPSTs

Markov Decision Processes (MDP) represent a class of sequential decision-making problems in a stochastic environment (Puterman, 2014). Intuitively, a planning agent has to deliberate over his/her model of the world to choose an optimal action in each decision stage in order to maximize his/her accumulated reward (or minimize the accumulated cost) given the immediate and long-term uncertain effects of available actions.

Formally, an MDP consists of (i) a finite set of *states* $\mathcal{S}$; (ii) a finite set of applicable *actions* $\mathcal{A}(s)$ for each state s; (iii) a Markovian *transition model* $\mathcal{T}(s,a,s') = \mathbb{P}(s'|s,a)$ specifying the probability that after executing action a in state s the next state is s'; (iv) a *reward model* $\mathcal{R}(s,a,s')$ specifying the reward (or cost) of executing action a in state s and transitioning to state s'; and (v) a set of decision stages $D = 1, ..., H$. The solution of an MDP with infinite horizon (i.e., $H \to \infty$) is a stationary, deterministic *optimal policy* $\pi^* : \mathcal{S} \to \mathcal{A}(s)$ that prescribes an optimal action a in state s in order to maximize the expected cumulative reward of state s defined by the *optimal value function* $V^* : \mathcal{S} \to \mathbb{R}$ given by:

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \left\{ \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)(\mathcal{R}(s,a,s') + \gamma\, V^*(s')) \right\}, \tag{1}$$

where $\gamma \in [0,1[$ is the discount factor necessary for convergence.

There are situations in which it is not easy (or even possible) to define a precise probability measure for a given transition. In this case, it is necessary to consider a more general version of an MDP known as **Markov Decision Processes with Imprecise Probabilities** (MDPIP) (White III and Eldeib, 1994; Satia and Lave Jr, 1973). In this model, the probability parameters are imprecise and therefore the transition model cannot be specified by a single conditional distribution, but it must be defined by *sets of probabilities* for each state transition. These sets are commonly referred to as *transition credal sets* $\mathcal{K}(\cdot|s, a)$ (Delgado et al., 2009). All other components of the MDP are unchanged (i.e., finite state and action space, reward function).

There are several objective criteria for solving an MDPIP with infinite horizon. In this paper, we only consider the Γ-maximin criterion (Delgado et al., 2009) which selects a robust policy that yields the supremum of the lower expected reward. The optimal value function of state s is:

$$V^*(s) = \max_{a\in\mathcal{A}(s)} \left\{ \min_{\mathbb{P}(\cdot|s,a)\in\mathcal{K}(\cdot|s,a)} \sum_{s'\in\mathcal{S}} \mathbb{P}(s'|s,a)(\mathcal{R}(s,a,s') + \gamma\, V^*(s')) \right\}. \tag{2}$$

Finally, another interesting variant model of MDP is the **Markov Decision Process with Set-valued Transition** (MDPST). This model is a particular instance of an MDPIP aimed at representing the transition model of an MDP with (separate) components for probabilistic and non-deterministic action effects (Trevizan et al., 2007, 2008). In an MDPST, the transition model is defined by the probability mass function $m(k|s, a)$ and the non-deterministic function $F(s, a) \subseteq 2^{\mathcal{S}}$, such that $k \in F(s, a)$. Its semantics is that after applying action a to state s the probability that the next state s' is in the *reachable set* $k \in F(s, a)$ is given by $m(k|s, a)$. These components together induce the imprecise probabilities over next states constrained by the following set of inequalities:

$$0 \leq m(\{s'\}|s,a) \leq \mathbb{P}(s'|s,a) \leq \sum_{k\in F(s,a) \text{ s.t. } s'\in k} m(k|s,a) \leq 1, \tag{3}$$

$$0 \leq \sum_{s'\in\mathcal{D}(k,s,a)} \mathbb{P}(s'|s,a) \leq m(k|s,a) \leq \sum_{s'\in k}\mathbb{P}(s'|s,a) \leq 1, \tag{4}$$

where $\mathcal{D}(k, s, a) = k - \bigcup_{k'\in F(s,a), k'\neq k} k'$.

Inequalities 3 and 4 define a transition credal set $\mathcal{K}(\cdot|s, a)$ as demonstrated by Trevizan et al. (2007) therefore proving that an MDPST is indeed an MDPIP. Though, the contrary does not necessarily holds since the class of MDPIPs is much more general than that of MDPSTs.

The solution of an MDPST under the minimax criteria is an optimal policy with respect to the optimal value function, given by:

$$V^*(s) = \max_{a\in\mathcal{A}(s)} \left\{ \sum_{k\in F(s,a)} m(k|s,a) \min_{s'\in k} (\mathcal{R}(s,a,s') + \gamma\, V^*(s')) \right\}. \tag{5}$$

Throughout the paper we assume a factored representation of the state in which a state s is given by a set of state fluents $\{x_1, ..., x_2\}$ which are state properties whose truth value changes with the actions; the factored transition function is $\mathbb{P}(s'|s, a) = \prod_{i=1}^{n} \mathbb{P}(x_i'|x_1, ..., x_n, a)$ and the reward function is also factored. This representation implies a dynamic Bayesian network in which next-state fluents are independent given the current-state fluents and action.

2.2 Probabilistic Logic Programming and the Credal Semantics

Probabilistic Logic Programming (PLP) extends Logic Programming (LP) by assigning *probability measures* to logical facts. It is typically assumed a fixed vocabulary of constants and relations. An atom is a predicate $r(t_1, ..., t_n)$ representing a n-arity relation over terms $t_1, ..., t_n$ where a term is either a logical variable or a constant from the vocabulary. We use lowercase to denote constants and uppercase to denote variables. A ground atom is an atom with no variables as one of its terms.

A **probabilistic logic program** is a pair $L_p = \langle \mathbf{BK}, \mathbf{PF} \rangle$ consisting of a set of logical rules **BK** called *background knowledge* and a set of probabilistic facts **PF**. A logical rule is of the form $h :- b_1, ..., b_m, \mathit{not}\ b_{m+1}, ..., \mathit{not}\ b_n.$, where atom h is called the *head* and the atoms $b_i, i = 1, ..., n$ are called the body. The reserved symbol not is to be interpreted as *negation as failure*, i.e., $\mathit{not}\ b_i$ is *true* in the absence of information that justifies b_i being *true*. A probabilistic fact denoted by $\alpha :: f.$ is an atom f annotated with probability $\alpha \in [0, 1]$. All probabilistic facts are probabilistically independent and cannot be unified with any rule's head atom.

A *total choice* denoted by θ is a complete truth assignment to the probabilistic facts of L_p. Each total choice θ induces a logical program denoted by L^θ containing the background knowledge of L_p and only the facts with a *true* value in θ. This semantics defines each probabilistic fact $\alpha_i :: f_i$ as a boolean random variable f_i distributed accordingly to the Bernoulli distribution with mean α_i. Since the probabilistic facts $\alpha_i :: f_i$ are independent, the probability of the induced logic program L^θ is given by:

$$\mathbb{P}(L^\theta | L_p) = \prod_{f_i \in \theta} \alpha_i \prod_{f_i \notin \theta} (1 - \alpha_i) \, . \tag{6}$$

The semantics of a probabilistic logic program L_p is given by the set of all probability models of L_p, accordingly to its **credal semantics** (Lukasiewicz, 2007; Cozman and Mauá, 2016). A *probability model* for a program L_p is a probability measure $\mathbb{P}$ over logical interpretations of its atoms such that (i) every interpretation I with $\mathbb{P}(I) > 0$ is a stable model of the induced program L^θ for the total choice θ that is consistent with I on the set **PF**; and (ii) the probability of the induced program L^θ is given by Equation 6. If the probabilistic logic program L_p is acyclic or stratified (Lloyd, 2012) then the *credal set* for program L_p consists of a single probability model related to its unique stable model.

An interpretation I over the set of atoms of a logic program L is a **stable model** if and only if I is the minimal model of the reduct program L^I. The reduct program L^I is the set of positive rules $\{H(r) :- B^+(r) \mid r \in L \text{ and } B^-(r) \cap I = \emptyset\}$ where $H(r)$ is the head of rule r; $B^+(r)$ and $B^-(r)$ are the sets of positive and negative atoms in the body of rule r. A typical logic program with more than one stable model is the non-stratified program $L = \{p :- \mathit{not}\ q.\ q :- \mathit{not}\ p.\}$ which has two stable models, namely the set of models $\{\{p\}, \{q\}\}$.

Given a probabilistic logic program L_p whose credal semantics is given by the credal set $\mathcal{K}_{L_p}$, the inference tasks of computing the *lower conditional probability* of query **Q** given evidence **E** denoted by $\underline{\mathbb{P}}(\mathbf{Q}|\mathbf{E})$ and respectively the *upper conditional probability* denoted by $\overline{\mathbb{P}}(\mathbf{Q}|\mathbf{E})$ are given by:

$$\underline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) = \inf_{\mathbb{P} \in \mathcal{K}_{L_p}} \mathbb{P}(\mathbf{Q}|\mathbf{E}) \tag{7}$$

$$\overline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) = \sup_{\mathbb{P} \in \mathcal{K}_{L_p}} \mathbb{P}(\mathbf{Q}|\mathbf{E}) \tag{8}$$

where **Q** and **E** are consistent sets of literals and it is assumed that $\mathbb{P}(\mathbf{E}) > 0$.

3. A Language to Specify MDPIPs and MDPSTs

One can specify an MDP through a probabilistic logic program, by annotating atoms with special meanings so as to distinguish actions, state fluents and rewards. This has been, for instance, the approach taken by LOMDP (Kersting and De Raedt, 2003), DTBLOG (Srivastava et al., 2014), and DDC (Nitti et al., 2015). In a previous work, we devised the MDP-PROBLOG specification language for sequential decision problems based on the PROBLOG language (Bueno et al., 2016). Here we extend the language so as incorporate incomplete and imprecise assessments.

An **MDP-PROBLOG program** consists of three parts: a program $L_{\text{MDP}}^{\text{SPACE}}$ declaring state fluents and actions, a program $L_{\text{MDP}}^{\text{TRANSITION}}$ encoding a transition model, and a program $L_{\text{MDP}}^{\text{REWARD}}$ encoding the reward model.

The *dependency graph* of an MDP-PROBLOG program is the signed directed graph over the ground atoms of the program; there is a positive (resp., negative) arc $B \to A$ if there is a rule with B in the body and A in the head, and B is non-negated (resp., negated). In our previous work, we showed that MDP-PROBLOG programs with acyclic dependency graphs represents a factored MDP, whose transition model for each action is a dynamic Bayesian network: each ground atom is a variable; probabilistic facts are root nodes associated with corresponding probabilities and non-probabilistic facts are internal nodes associated with deterministic functions. We also showed that MDP-PROBLOG with *positive* cycles in its dependency graph still represent factored MDP (note that dynamic Bayesian networks do not allow cycles). We did not define the semantics of programs with cycles; we close this gap here.

We use the following running example to illustrate concepts:

Example 1 *In the **Viral Marketing** (VM) domain, we are given an information about individuals and their trust relationships, and we are interested in selecting individuals to market a certain product. The goal is to maximize the long-term profit by increasing the likelihood of sales while decreasing the cost of marketing. We assume that a person might buy the product after being marketed or because she trusts someone who already bought it. Also, if a person has not been the target of a marketing action in the current step, but she has been marketed in the past, then the delayed effect of past marketing actions should be accounted for.*

The program $L_{\text{MDP}}^{\text{SPACE}}$ consists of (invariant) facts and two types of rules: *state fluent declarations* and *action fluent declarations*. State fluent declarations are of the form $\mathsf{state\_fluent}(A) \text{ :- } B_1, \ldots, B_n.$, where A is an atom representing a state fluent and $B_1, \ldots, B_n$ are literals mentioning action fluents (actions that may or may not occur) or non state fluents (state properties whose truth value does not change, i.e. invariants). Action fluent declarations are of the form $\mathsf{action\_fluent}(A) \text{ :- } B_1, \ldots, B_n.$, where A is an atom representing an action and $B_1, \ldots, B_n$ are as before. The state fluents are distinguished between *current state* and *next state*. Current-state fluents take an extra argument 0 to indicate the current stage, while next-state fluents take an extra argument 1 to indicate the next stage.

Consider our running example. We declare individuals by a set of (invariant) ground facts $\mathsf{person}(p_i)$, a state and action fluents by:

$$\begin{aligned} \mathsf{state\_fluent}(\mathsf{marketed}(P)) &\text{ :- } \mathsf{person}(P). \\ \mathsf{state\_fluent}(\mathsf{buys}(P)) &\text{ :- } \mathsf{person}(P). \\ \mathsf{action\_fluent}(\mathsf{market}(P)) &\text{ :- } \mathsf{person}(P). \end{aligned}$$

Given persons p_1 and p_2, we have 4 state fluents: $\mathsf{marketed}(p_1)$, $\mathsf{marketed}(p_2)$, $\mathsf{buys}(p_1)$ and $\mathsf{buys}(p_2)$. Thus, the program above defines 2^4 states. For example, we have a state where $\mathsf{marketed}(p_1)$

is true, and all of $\mathsf{marketed}(p_2)$, $\mathsf{buys}(p_1)$ and $\mathsf{buys}(p_2)$ are false. Similarly, we have 2 actions fluents: $\mathsf{market}(p1)$ and $\mathsf{market}(p2)$. Thus, the program defines 2^2 actions. For example, we have an action where $\mathsf{market}(p1)$ is true and $\mathsf{market}(p2)$ is true [1].

The program $L^{\text{TRANSITION}}_{\text{MDP}}$ contains a set of rules such that no action fluents nor current-state fluents unify with head atoms.

The transition model of our running example is given by the program:

$$\begin{aligned}
0.5 &:: \ \mathsf{decay}(Person).\\
\mathsf{marketed}(Person,\ 1) &:-\ \mathsf{market}(Person).\\
\mathsf{marketed}(Person,\ 1) &:-\ \mathit{not}\ \mathsf{market}(Person),\ \mathsf{marketed}(Person,\ 0),\ \mathsf{decay}(Person).
\end{aligned}$$

$$\begin{aligned}
0.2 &:: \ \mathsf{buy\_from\_marketing}(Person).\\
0.3 &:: \ \mathsf{buy\_from\_trust}(Person).\\
\mathsf{buys}(Person,\ 1) &:-\ \mathsf{marketed}(Person,\ 1),\ \mathsf{buy\_from\_marketing}(Person).\\
\mathsf{buys}(Person,\ 1) &:-\ \mathsf{trusts}(Person,\ Person2),\ \mathsf{buys}(Person2,\ 1),\ \mathsf{buy\_from\_trust}(Person).
\end{aligned}$$

According to this program, an individual is under the effect of a marketing action if she has either been targeted in the current stage, or, with probability 0.5, if she was under the effect in a previous stage. There is also the idea that a person buys the product with a certain probability if she has been the target of marketing, and with a different probability if some of her trustees was the target of marketing.

The transition program induces a transition credal set $\mathcal{K}(s'|s, a)$, where s is an interpretation of current-state fluents, a is an interpretation of action fluents and s' is an interpretation of next-state fluents. Each conditional distribution in the transition credal set specifies a transition model $\mathcal{T}(s, a, s')$ assigned with probability $\mathbb{P}(s'|s, a)$ given by the credal semantics of the program.

The program $L^{\text{REWARD}}_{\text{MDP}}$ contains a set of rules of the form $\mathsf{utility}(A, c)\ :-\ B_1, \ldots, B_n$, where A is state or action fluent, c is a value denoting reward/cost, and each B_i is a literal.

In our running example, every product bought contributes with a reward of 5, and every marketing action costs -1:

$$\begin{aligned}
&\mathsf{utility}(\mathsf{buys}(Person, 1),\ 5).\\
&\mathsf{utility}(\mathsf{market}(Person),\ -1).
\end{aligned}$$

Finally, the program $L^{\text{REWARD}}_{\text{MDP}}$ specifies an additive reward model $\mathcal{R}(s, a, s')$ over current states (interpretation of current-state fluents), actions (interpretation of action fluents) and next states (interpretations of next-state fluents). A rule $\mathsf{utility}(A, c)\ :-\ B_1, \ldots, B_n$ contributes with (additive) reward c if and only if $A, B_1, \ldots, B_n$ are all true in the interpretation.

Since we adopted the credal semantics for the transition program, the transition credal set is the dominating credal set of an infinitely monotone Choquet capacity (Cozman and Mauá, 2016); that is, each transition is governed by a probabilistic transition into a reachable set that consists of the stable models. To get some intuition on this result, consider that for each fixed total choice, we obtain a logic program that may have more than one stable model (if it has no stable model, the whole probabilistic logic program has no semantics). And recall that over the total choices we have

1. Note that the semantics of the probabilistic logic programming allows concurrent actions just like in RDDL (Sanner, 2010).

a product measure. Hence we have a multi-valued mapping form one sample space endowed with a probability measure (the space of total choices) into another space (the space of stable models); this implies that over the latter space we have an infinitely monotone Choquet capacity (Augustin et al., 2014). Thus we have the following surprising (and pleasant) consequence:

Theorem 1 *An* MDP-PROBLOG *program specifies a factored MDPST.*

Although an MDPST is a particular case of MDPIP, so far we have assumed that every probability value is known with absolute precision. This is obviously unrealistic in practice. The natural solution then is to allow a fact to be associated with a probability interval. We denoted these extended probabilistic facts by $[\alpha, \beta] :: p$. where p is an atom and parameters α and β are probability bounds such as $0 \leq \alpha \leq \beta \leq 1$. In the case of $\alpha = \beta$, we have a standard probabilistic fact. The semantics of a probabilistic logic program with interval-valued facts is the credal set that consists of all probability distributions that satisfy the constraints (that is, whose marginal probabilities for facts lies within given intervals).

For example, in the viral marketing domain, we might be uncertain about the probabilities that an individual will buy a product given different scenarios:

$$[0.1, 0.3] :: \ \mathsf{buy\_from\_marketing}(Person).$$
$$[0.2, 0.4] :: \ \mathsf{buy\_from\_trust}(Person).$$

Now suppose we have an MDP-PROBLOG program, possibly with interval-valued probabilistic facts and negative cycles [2]. Suppose also the current state $\mathbf{S}_0$ is given, and possibly an additional set of grounded atoms $\mathbf{E}$ on the current time step; finally suppose we have a set of grounded atoms $\mathbf{Q}$ of next state, and we wish to compute $\overline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}, \mathbf{S}_0)$. By using arguments that apply to inference in credal networks, we have that the value of $\overline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}, \mathbf{S}_0)$ is attained at a selection of extreme points of the probability intervals, together with a selection of reachable set for all resulting probabilities (Augustin et al., 2014). That is, to compute an upper probability, we must go through all extreme points of probability intervals, and all possible extreme points of the induced infinitely monotone Choquet capacities. The same result obtains for the computation of lower probabilities. We will use these results in Section 5.

4. The Complexity of One-Step Inference

In this section we will need a number of concepts from complexity theory; most of them are standard: we use *languages*, *decision problems*, *many-one reductions*, and *complexity classes* such as P and NP (Papadimitriou, 2003). The complexity class PP consists of those languages $\mathcal{L}$ such that: there is a polynomial time nondeterministic Turing machine M such that $\ell \in \mathcal{L}$ if and only if more than half of the computations of M on input ℓ end up accepting). We consider oracle machines and complexity classes such as Σ_i^{P}, recursively defined as $\Sigma_i^P = \mathsf{NP}^{\Sigma_{i-1}^P}$ with $\Sigma_0^P = \mathsf{P}$. We also use classes from Wagner's *polynomial counting hierarchy*: that is, the smallest set of classes containing P and, recursively, for any class C in the polynomial counting hierarchy, the classes PP^{C}, NP^{C}, and $\mathsf{coNP}^{\mathsf{C}}$ (Torán, 1991; Wagner, 1986).

2. One could suppose that an MDP-PROBLOG program with interval-valued probabilities defines a BMDP (Givan et al., 1997), however in our language the imprecision is over state fluents while in BMDPs the imprecision is over states.

We are interested here in the complexity of *one-step inference*; that is, if we have the state at time t, then what is the computational cost of computing the probability of $\{X_{t+1} = x\}$? We start by analyzing a problem of independent interest: the complexity of inferences in probabilistic logic programs with interval-valued probabilistic facts (Section 4.1) and then we look at the complexity of one-step inference (Section 4.2).

4.1 Credal logic programs with interval-valued probabilistic facts

Suppose we have a credal logic program, possibly disjunctive and non-stratified, but not necessarily aimed at modeling planning scenarios. That is, we just have a disjunctive logic program associated with a number of interval-valued probabilistic facts. The only restriction we impose is that there is a bound on predicate arity. Suppose that additionally we have, as *input*, a set $\mathbf{Q}$ of truth assignments to grounded atoms, and another set $\mathbf{E}$ of truth assignments to grounded atoms; additionally we have a rational number γ in $[0, 1]$. We refer to $(\mathbf{Q}, \mathbf{E})$ as the *query*, and to $\mathbf{E}$ as the *evidence*. As *output* we have the decision as to whether $\overline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) > \gamma$ where the probabilities are computed with respect to the input credal logic program. Consider the strings describing a credal logic program, a query, and a rational, and denote by $\mathcal{C}$ the language consisting of all such strings that satisfy $\overline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}) > \gamma$. Note that if we restrict our programs to be non-disjunctive and acyclic, then they specify credal networks (Cozman, 2005), and therefore deciding $\mathcal{C}$ is at least a $\mathsf{NP}^{\mathsf{PP}}$-hard problem. It is remarkable that we can also decide $\mathcal{C}$ in $\mathsf{NP}^{\mathsf{PP}}$; that is:

Theorem 2 *Deciding whether a string is in $\mathcal{C}$ is a $\mathsf{NP}^{\mathsf{PP}}$-complete problem.*

Proof Hardness follows, as already noted, from the fact that inference with credal networks is $\mathsf{NP}^{\mathsf{PP}}$-complete (De Campos and Cozman, 2005). Membership is a consequence of the following construction. First, guess the extreme point of each interval-valued probability assessment (this requires a nondeterministic Turing machine, but given that predicate arity is bounded, there is a polynomial number of guesses to be made). Then call, as an oracle, a counting Turing machine that guesses the truth assignment for all grounded probabilistic facts; by counting the number of such assignments that leads to satisfaction of $\mathbf{Q}$ and $\mathbf{E}$, we can decide whether the base nondeterministic choice satisfies or not the inequality of interest. The problem is that, for each selected truth assignment for ground probabilistic facts, we must decide whether it is possible to satisfy the query; for a disjunctive logic program this can be made using a Σ_3^P oracle. That is, our problem can be solved in $\mathsf{NP}^{\mathsf{PP}^{\Sigma_3^P}}$. However, due to a remarkable result by Toda and Watanabe (Toda and Watanabe, 1992), we have that $\mathsf{P}^{\mathsf{PP}^{\Sigma_k^P}} = \mathsf{P}^{\mathsf{PP}}$; consequently, our decision problem is in $\mathsf{NP}^{\mathsf{PP}}$ and the proof is finished. ■

4.2 One-step transitions

Now consider the specification of a planning problem using a credal logic program as described in Section 3. That is, we have a logic program with added interval-valued probabilistic facts. Denote by $\mathcal{PC}$ the language that consists of strings encoding a credal logic program with a bound on predicate arity, a query, and a rational as in Section 4.1, but now the credal logic program is the description of a planning scenario as in Section 3, and with the following additional restrictions. The query must now refer only to grounded atoms at the next time step (not at current time step), and a string is in the language if and only if $\overline{\mathbb{P}}(\mathbf{Q}|\mathbf{E}, \mathbf{S}_0) > \gamma$ where $\mathbf{S}_0$ is the current state. That is, we focus on one-step, from current to next state, and we wish to compute an inference about the time step.

Using the result in the previous section we immediately have:

Theorem 3 *Deciding whether a string is in $\mathcal{PC}$ is a $\mathsf{NP}^{\mathsf{PP}}$-complete problem.*

Proof Note that when we fix $\mathbf{S}_0$, we obtain a decision problem for a credal probabilistic program. Then Theorem 2 implies the result. ■

Now suppose we restrict ourselves to point-valued probabilistic assessments; that is, every probabilistic facts is of the form $\alpha :: A.$. As discussed in Section 3, such assessments allow us to define MDPSTs when programs can be disjunctive/non-stratified. Now denote by $\mathcal{PM}$ the language defined exactly as $\mathcal{PC}$, with the difference that every probabilistic assessment is point-valued. It is known that the complexity of inference in non-disjunctive probabilistic logic programs that can be non-stratified is $\mathsf{PP}^{\Sigma_2^P}$-complete, while the complexity of inference in disjunctive probabilistic logic programs is $\mathsf{PP}^{\Sigma_3^P}$-complete (Cozman and Mauá, 2017), submitted. Hence we obtain, as a direct consequence:

Theorem 4 *Deciding whether a string is in $\mathcal{PM}$ is a $\mathsf{NP}^{\Sigma_3^P}$-complete problem.*

5. Dynamic Programming for MDP-PROBLOG programs

In this section, we discuss how dynamic programming can be applied to solve sequential decision problems specified by MDP-PROBLOG programs. To emphasize: we allow programs with (negative and positive) cycles in the dependency graph and interval-valued probabilistic facts.

For simplicity, we consider grounded programs. So consider a (ground) MDP-PROBLOG program, a current state s (i.e., an interpretation of state fluents) and action $a \in \mathcal{A}(s)$ (i.e., an interpretation of actions). Due to Theorem 1, given evidence s, a, the transition model induces a set of probability mass functions $m(k|s, a)$ over sets of stable models $k \in F(s, a)$. One can show that the robust (i.e., maximin) policy is given by the argument of the following modified Bellman equation:

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \left\{ \min_{m(\cdot|s,a) \in \mathcal{K}(\cdot|s,a)} \sum_{k \in F(s,a)} m(k|s, a) \min_{s' \in k} \left(\mathcal{R}(s, a, s') + \gamma\, V^*(s') \right) \right\} \tag{9}$$

The outer (i.e., leftmost) minimization can be solved by considering all extremes of the interval-valued probabilities; after each choice is made, the resulting program specifies an MDPST whose transition is governed by the stable models of the transition program: this is the inner (rightmost) minimization in the equation above.

When all reachable sets are singletons (i.e., $\forall k \in F(s, a), |k| = 1$) there is no need to perform the inner minimization over the states in k and then we have the traditional case of MDPIPs given by Equation 2. On the other hand, if all interval-valued probabilistic facts degenerate to point-valued standard probabilistic facts the outer minimization over the probabilistic models of the credal set $\mathcal{K}(\cdot|s, a)$ is not need and then we have the Equation 5 for precise MDPSTs. Finally, when both assumptions hold we are back to the classical MDP case of Equation 1.

The traditional dynamic programming scheme for solving the set of equations defining the state value function is the **Value Iteration** algorithm (Puterman, 2014). Essentially, it assigns an initial value to all states and iteratively updates all state values until the convergence by using Equation 9 as an update rule known as Bellman backup. A number of optimizations exist for avoiding redundant calculations and restricting the computation for only the most promising states regarding the optimal policy. Nevertheless, virtually all of these techniques has to deal one or more backup calculations.

6. Conclusion

In this paper, we addressed the problem of modeling MDPIPs and MDPSTs using probabilistic logic programming. Our contributions are:

- an extension of the MDPPROBLOG language that aimed at representing imprecise probabilities and non-determinism;
- novel results about the complexity of one-step inference in credal logic programs with interval-valued probabilistic facts (and on the complexity of probabilistic logic programs with interval-valued probabilistic facts); and
- a scheme for generating optimal policy for MDPIPs and MDPSTs encoded by probabilistic logic programming.

For the future, we plan to implement and test algorithms for policy generation. In order to do so, it would be valuable to maximize expected values with respect to the credal sets encoding transitions. Given that heuristics are important in state-of-art algorithms for MDPs, we believe that similar heuristics must be developed for MDPIPs and MDPSTs. In particular, it should be important to import techniques from logical reasoning into the realm of probabilistic logic programming.

Acknowledgments

This work was partially supported by CNPq (grants 870666/1998-3, 308433/2014-9) and FAPESP (grants 2015/01587-0, 2016/01055-1, 2016/22900-1).

References

T. Augustin, F. P. Coolen, G. de Cooman, and M. C. Troffaes. *Introduction to imprecise probabilities*. 2014.

C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-Theoretic, High-Level Agent Programming in the Situation Calculus. In *AAAI/IAAI*, pages 355–362, 2000.

T. Bueno, D. Mauá, L. N. de Barros, and F. Cozman. Markov Decision Processes Specified by Probabilistic Logic Programming: Representation and Solution. In *BRACIS*, pages 337–342, 2016.

F. G. Cozman. Graphical models for imprecise probabilities. *International Journal of Approximate Reasoning*, 39(2-3):167–184, 2005.

F. G. Cozman and D. D. Mauá. The structure and complexity of credal semantics. In *Workshop on Probabilistic Logic Programming*, pages 3–14, 2016.

F. G. Cozman and D. Mauá. The Complexity of Inferences and Explanations in Probabilistic Logic Programming. *Proceedings ISIPTA, Lugano, Switzerland*, 2017.

C. P. De Campos and F. G. Cozman. The inferential complexity of bayesian and credal networks. In *IJCAI*, volume 5, pages 1313–1318, 2005.

K. V. Delgado, L. N. de Barros, F. G. Cozman, and R. Shirota. Representing and solving factored Markov decision processes with imprecise probabilities. *ISIPTA*, pages 169–178, 2009.

K. V. Delgado, S. Sanner, and L. N. De Barros. Efficient solutions to factored MDPs with imprecise transition probabilities. *Artificial Intelligence*, 175(9-10):1498–1527, 2011.

T. Eiter and T. Lukasiewicz. Probabilistic Reasoning About Actions in Nonmonotonic Causal Theories. In *UAI*, pages 192–199, 2003.

T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A Logic Programming Approach to Knowledge-state Planning: Semantics and Complexity. *ACM Trans. Comput. Logic*, 5(2):206–263, 2004.

R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080, 1988.

E. Giunchiglia and V. Lifschitz. An Action Language based on Causal Explanation: Preliminary Report. In *AAAI/IAAI*, pages 623–630, 1998.

E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artificial Inteligence*, 153:49–104, 2004.

R. Givan, S. Leach, and T. Dean. Bounded parameter Markov decision processes. In *European Conference on Planning*, pages 234–246, 1997.

J. Y. Halpern and M. R. Tuttle. Knowledge, Probability, and Adversaries. *J. ACM*, 40(4):917–960, 1993.

L. Iocchi, T. Lukasiewicz, D. Nardi, and R. Rosati. Reasoning About Actions with Sensing Under Qualitative and Probabilistic Uncertainty. *ACM Trans. Comput. Logic*, 10(1):5:1–5:41, 2009.

K. Kersting and L. De Raedt. Logical Markov decision programs. In *Proceedings of the IJCAI'03 Workshop on Learning Statistical Models of Relational Data*, pages 63–70, 2003.

H. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–83, 1997.

J. W. Lloyd. *Foundations of logic programming*. 2012.

T. Lukasiewicz. Probabilistic description logic programs. *International Journal of Approximate Reasoning*, 45(2):288–307, 2007.

D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL-the Planning Domain Definition Language. 1998.

D. Nitti, V. Belle, and L. De Raedt. Planning in discrete and continuous Markov decision processes by probabilistic programming. In *ML and KD in Databases*, pages 327–342. 2015.

C. H. Papadimitriou. *Computational complexity*. 2003.

M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. 2014.

S. Sanner. Relational dynamic influence diagram language (RDDL): Language description. *Unpublished ms. Australian National University*, page 32, 2010.

J. K. Satia and R. E. Lave Jr. Markovian decision processes with uncertain transition probabilities. *Operations Research*, 21(3):728–740, 1973.

S. Srivastava, S. J. Russell, P. Ruan, and X. Cheng. First-Order Open-Universe POMDPs. In *UAI*, pages 742–751, 2014.

S. Toda and O. Watanabe. Polynomial-time 1-turing reductions from #PH to #P. *Theoretical Computer Science*, 100(1):205–221, 1992.

J. Torán. Complexity classes defined by counting quantifiers. *Journal of the ACM (JACM)*, 38(3): 752–773, 1991.

F. W. Trevizan, F. G. Cozman, and L. N. de Barros. Planning under Risk and Knightian Uncertainty. In *IJCAI*, pages 2023–2028, 2007.

F. W. Trevizan, F. G. Cozman, and L. N. De Barros. Mixed probabilistic and nondeterministic factored planning through Markov decision processes with set-valued transitions. In *Workshop on A Reality Check for Planning and Scheduling Under Uncertainty at ICAPS*, 2008.

K. W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta informatica*, 23(3):325–356, 1986.

C. C. White III and H. K. Eldeib. Markov decision processes with imprecise transition probabilities. *Operations Research*, 42(4):739–749, 1994.

H. L. Younes and M. L. Littman. PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*, 2004.