Modeling Markov Decision Processes with Imprecise Probabilities Using Probabilistic Logic Programming

Thiago P. Bueno  Denis D. Mauá  Leliane N. de Barros

Fabio G. Cozma

Universidade de São Paulo, Brazil

ISIPTA 2017
Goals

- To introduce a modeling language that can capture Markov Decision Processes with Imprecise Probabilities (MDPIPs),

- by employing Probabilistic Logic Programming (PLP).
Markov Decision Processes

A Markov Decision Process (MDP) consists of:

- a set of **states** $S$;
- a set of **actions** $A(s)$ for each state $s$;
- a **transition model** $P(s'|s, a)$ specifying the probability of next state $s'$ after executing action $a$ in state $s$;
- a **reward model** $R(s, a, s')$ specifying the reward (or cost) of executing action $a$ in state $s$ and transitioning to state $s'$;
- a set of decision stages $D = 1, ..., H$. 
Optimal policy, optimal value function

- The solution of an MDP with infinite horizon (i.e., $H \to \infty$) is a stationary, deterministic **optimal policy** $\pi^* : S \to A(s)$ that maximizes

\[
\sum_{t=0}^{\infty} \gamma^t R(s_t, a, s_{t+1}).
\]

- The optimal policy produces the **optimal value function** $V^* : S \to \mathbb{R}$ satisfying the Bellman equation

\[
V^*(s) = \max_{a \in A(s)} \left\{ \sum_{s' \in S} \mathbb{P}(s'|s,a)(R(s,a,s') + \gamma V^*(s')) \right\},
\]
Suppose there is a set of probabilities modeling each state transition.

These sets are referred to as transition credal sets $\mathcal{K}(\cdot|s, a)$.

The $\Gamma$-maximin criterion selects a policy such that

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \left\{ \min_{\mathbb{P}(\cdot|s,a) \in \mathcal{K}(\cdot|s,a)} \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)(R(s,a,s') + \gamma V^*(s')) \right\}.$$
A **Markov Decision Process with Set-valued Transition (MDPST)** is a special MDPIP.

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 $m(k|s, a)$.

Policy is obtained by simplified equation:

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \left\{ \sum_{k \in F(s, a)} m(k|s, a) \min_{s' \in k} (R(s, a, s') + \gamma V^*(s')) \right\}.$$
There are languages to specify MDPs; several combine logical expressions with probabilities.

- The PPDDL language can even encode MDPSTs.
- But not intuitive at all.
A probabilistic logic program is a pair \( L_p = \langle BK, PF \rangle \) where:

- \( BK \) is a set of logical rules, and
- \( PF \) is a set of independent probabilistic facts.
A **probabilistic logic program** is a pair \( L_p = \langle \text{BK}, \text{PF} \rangle \) where:

- \( \text{BK} \) is a set of logical rules, and
- \( \text{PF} \) is a set of *independent* probabilistic facts.

A logical rule is of the form

\[
h_1; \ldots; h_l : - b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n.
\]

A probabilistic fact is denoted \( \alpha :: f \). where \( f \) is an atom annotated with probability \( \alpha \in [0, 1] \).
Example: Viral Marketing

0.2 :: buy_from_marketing(Person).
0.3 :: buy_from_trust(Person).

buys(Person) :- buy_from_marketing(Person).
buys(Person) :- buy_from_trust(Person),
    trusts(Person, Person2), buys(Person2).

trusts(alice, eve). trusts(eve, bob).

What is the probability of Alice buying the product?

\[ \mathbb{P}(\text{buys}(alice)) = ? \]
Example: Viral Marketing (continued)

...  

0.15 :: invited_party(Person).

\[\text{buys}(\text{Husband}) : - \text{invited}_\text{party}(\text{Husband}),\]
\[\quad \text{married}(\text{Husband}, \text{Wife}), \text{not} \ \text{buys}(\text{Wife}).\]
\[\text{buys}(\text{Wife}) : - \text{invited}_\text{party}(\text{Wife}),\]
\[\quad \text{married}(\text{Husband}, \text{Wife}), \text{not} \ \text{buys}(\text{Husband}).\]

\[\text{married}(\text{alice}, \text{bob}).\]

- How to compute the probability of \( \mathbb{P}(\text{buys}(\text{alice})) \) now?  
  In some situations, there is more than a (stable) model...
Credal Semantics

- Propositional probabilistic facts $\alpha_i :: f_1$, $\alpha_2 :: f_2$, etc.
- Each total choice of probabilistic facts has probability

$$\prod_{f_i \in \theta} \alpha_i \prod_{f_i \not\in \theta} (1 - \alpha_i).$$

- But some total choices may produce more than one stable model!
- Credal semantics of a program is the set of all joint distributions that can be produced this way.
  
  - Important: this set is the dominating set of an infinitely monotone Choquet capacity (!).
We need to extend the ProbLog language to define:

- special-purpose predicates for state variables and actions;
- syntax and semantics for specifying the transition function; and
- the dependencies of reward function and its utility attributes.

An **MDP-ProbLog program** consists of three parts:

(i) a program $L_{\text{SPACE MDP}}$ declaring state variables and actions;
(ii) a program $L_{\text{TRANSITION MDP}}$ encoding a transition model; and
(iii) a program $L_{\text{REWARD MDP}}$ encoding the reward model.
Viral Marketing (revisited)

state_fluent(marked(P)) :- person(P).
state_fluent(buys(P)) :- person(P).
action_fluent(market(P)) :- person(P).
Viral Marketing (revisited)

Transition model

\[
\text{marketed}(\text{ann}, 0) \rightarrow \text{marketed}(\text{ann}, 1) \\
\text{market}(\text{ann}, 0) \\
\text{trusts}(\text{ann}, \text{bob}) \rightarrow \text{buys}(\text{ann}), 1 \\
\text{trusts}(\text{bob}, \text{ann}) \rightarrow \text{buys}(\text{bob}), 1 \\
\text{market}(\text{bob}, 0) \\
\text{marketed}(\text{bob}, 0) \rightarrow \text{marketed}(\text{bob}, 1)
\]

\[
0.5 :: \text{forget}(\text{Person}). \\
\text{marketed}(\text{Person}, 1) :- \text{market}(\text{Person}). \\
\text{marketed}(\text{Person}, 1) :- \text{not} \ \text{market}(\text{Person}), \ \text{marketed}(\text{Person}, 0), \ \text{forget}(\text{Person}).
\]
Viral Marketing (revisited)

0.2 :: buy_from_marketing(Person).
0.3 :: buy_from_trust(Person).

buys(Person, 1) :- marketed(Person, 1), buy_from_marketing(Person).

buys(Person, 1) :- trusts(Person, Person2), buys(Person2, 1), buy_from_trust(Person).

utility(buys(Person, 1), 5). utility(market(Person), −1).
Theorem

An MDP-ProbLog program specifies an MDPST.

- Now suppose there is indeterminacy on probability values.
- For instance,

\[ [0.1, 0.3] :: \text{buy}\_{\text{from\_marketing}}(\text{Person}). \]
Complexity of One-Step Inference

If we have the state at time $t$, then what is the computational cost of computing the upper probability of $\{X_{t+1} = x\}$?

More precisely: what is the cost of deciding whether $\overline{P}(Q|E) > \gamma$? (Note: reject if $\overline{P}(E) = 0$...)

As input, a program with a bound on predicate arity, and the elements of the query.
Complexity of One-Step Inference

**Theorem**

\[ \text{Deciding one-step inference is an } \text{NP}^{\text{PP}} \text{-complete problem.} \]

**Theorem**

\[ \text{Deciding one-step inference when all probabilities are point-valued is } \text{PP}^{\Sigma_3^P} \text{-complete problem.} \]
Conclusion

- Main goal: to introduce a language that can specify MDPIPs and MDPSTs by combining probabilities with logic programming.

- Besides the language, main contribution is complexity analysis for one-step inference.

- In the paper, a discussion of dynamic programming algorithm to build $\Gamma$-maximin policies.

- Thanks to CNPq and FAPESP for support.