In *Proceedings of the IJCAI'16 Workshop on Autonomous Mobile Service Robots*,
New York City, USA July 2016

Dynamically Constructed (PO)MDPs for Adaptive Robot Planning

Shiqi Zhang, Piyush Khandelwal, and Peter Stone
Department of Computer Science
The University of Texas at Austin
{szhang; piyushk; pstone}@cs.utexas.edu

Abstract

To operate in human-robot coexisting environments, intelligent robots need to simultaneously reason with commonsense knowledge and plan under uncertainty. Markov decision processes (MDPs) and partially observable MDPs (POMDPs), are good at planning under uncertainty toward maximizing long-term rewards; P-LOG, a declarative programming language under *Answer Set* semantics, is strong in commonsense reasoning. In this paper, we present a novel algorithm called DCPARP to dynamically represent, reason about, and construct (PO)MDPs using P-LOG. DCPARP successfully shields exogenous domain attributes from (PO)MDPs so as to limit computational complexity, but still enables (PO)MDPs to adapt to the value changes these attributes produce. We conduct a large number of experimental trials using two example problems in simulation and demonstrate DCPARP on a real robot. Results show significant improvements compared to competitive baselines.

1 Introduction

In order to be fully robust and responsive in real-world environments where frequently humans and robots coexist, intelligent robots need a variety of simultaneous reasoning modalities that were separately developed in the past. In this paper, we focus on robots' needs for: i) commonsense reasoning (both logical and probabilistic), ii) modeling quantitative uncertainties from nondeterministic action outcomes and local, unreliable observations, and iii) planning under such uncertainties toward maximizing long-term rewards. This work, for the first time, uses (logical and probabilistic) commonsense reasoning techniques to dynamically construct probabilistic graphical models (such as MDPs and POMDPs). While traditional hand-coded models implicitly assume the acting agent is the only one that can make changes to the world, we introduce DCPARP, standing for Dynamically Constructed (PO)MDPs for Adaptive Robot Planning, that enables probabilistic planning to be adaptive to exogenous world changes.

An MDP describes a probabilistic transition system under the assumption of full observability. A POMDP extends an MDP by assuming partial observability of underlying states [Kaelbling *et al.*, 1998], and hence models the quantitative uncertainties from robot observations and action outcomes well. (PO)MDP algorithms, e.g., value iteration [Sutton and Barto, 1998], Monte Carlo tree search [Kocsis and Szepesvári, 2006] and SARSOP [Kurniawati *et al.*, 2008], help compute a *policy* that enables planning toward maximizing long-term rewards. MDPs and POMDPs have been used in a variety of robot applications such as [Khandelwal *et al.*, 2015] and [Young *et al.*, 2013], where each (PO)MDP corresponds to a pair of probabilistic transition and reward systems that individually model the nondeterministic action outcomes and planning goal. However, (PO)MDP models are not designed to reason about commonsense knowledge, e.g., office doors are *normally* closed on holidays.

Existing work has investigated modeling exogenous events, e.g., sunlight reduces success rate of a robot navigating through an area (due to the limitations of range-finder sensors), *within* decision-theoretic models [Boutilier *et al.*, 1999]. However, it is often difficult to predict how an exogenous change will affect the system state, and what the distribution for the occurrence of these exogenous events will be. Doing so also presents a trade-off between model correctness and computational tractability (as more domain variables are modeled). Although it is possible to implement domain-specific planners to efficiently handle the exogenous events, we argue that using commonsense reasoning to shield exogenous domain attributes from (PO)MDPs is relatively a much more applicable approach than directly manipulating (PO)MDPs' graphical representations.

Answer set programming (ASP) is a logic programming language that is good at representing and reasoning with logical commonsense knowledge [Baral, 2003; Gelfond and Kahl, 2014] and has been used in robot applications [Chen *et al.*, 2012; Erdem *et al.*, 2015]. Probabilistic extensions of ASP including P-LOG [Baral *et al.*, 2009] and LP^{MLN} [Lee and Wang, 2015] further enable reasoning with probabilistic commonsense knowledge. As a result, ASP's probabilistic extensions can easily represent facts such as office doors are normally closed on holidays and probabilistic models such as a robot has a lower success rate of navigating through an area under sunlight. It should be noted that ASP and its extensions do not support probabilistic planning toward maximizing long-term rewards. For instance, techniques in the ASP family are not suitable for the robot navigation problem

(§ 3.2), while (PO)MDPs are good at such problems. Learning knowledge from the web or through human-robot interaction has been studied in existing research [Samadi *et al.*, 2012; Perera *et al.*, 2015; Myagmarjav and Sridharan, 2015], and is beyond the scope of this paper.

Contemporaneously with ASP, another family of programming languages for probabilistic reasoning are built under First-order logic (FOL) semantics, including BLOG [Milch *et al.*, 2007] and MLNs [Richardson and Domingos, 2006], but the FOL-based ones are not good at representing or reasoning with commonsense knowledge that is *normally* true but not always (more detailed comparisons in [Baral *et al.*, 2009]).

Different methods have been developed to combine commonsense reasoning and probabilistic planning. In [Zhang *et al.*, 2015], ASP and POMDPs are integrated for mobile robots, where the reasoning resulting (answer sets) were used for generating *prior* beliefs for POMDPs (but not for making changes to the POMDP once the robot has started to work on the task). Hanheide *et al.* used a switching planner for deterministic and probabilistic planning and used commonsense knowledge for diagnostic tasks and generating explanations [Hanheide *et al.*, 2015]. In these algorithms, bridging the gap between logical knowledge and probabilistic beliefs requires considerable domain-dependent heuristics. The use of P-LOG and POMDPs in this work enables a principled algorithm that simultaneously allows (both logical and probabilistic) commonsense reasoning and probabilistic planning.

The work closest to DCPARP is an algorithm called CORPP that unifies the strengths of POMDPs and P-LOG by reasoning with P-LOG to specify the state space of and compute informative priors for POMDP-based planning [Zhang and Stone, 2015]. However, a significant limitation of CORPP is that the reward system and actuating capabilities have to be hand-coded, making it incapable of adapting to exogenous world changes. This paper addresses this limitation by introducing DCPARP that dynamically constructs (PO)MDPs using P-LOG, and, for the first time, shields exogenous attributes from (PO)MDPs while still enabling probabilistic planning to adapt to the exogenous events. To evaluate DCPARP's performance, we have conducted a large number of trials in simulation and demonstrated its effectiveness on a real robot using tasks in an office domain. We observed significant improvements in both efficiency and accuracy compared to CORPP.

2 Background

This work builds on the existing techniques of P-LOG [Baral *et al.*, 2009] and POMDPs [Kaelbling *et al.*, 1998]. A POMDP generalizes an MDP by assuming partial observability of the underlying state and maintains a belief distribution over the set of possible states. Since POMDPs are currently more common in the literature and due to space constraints, we do not discuss the general POMDP framework, but focus on introducing P-LOG, the other key technique this work builds on.

A P-LOG program typically includes both logical and probabilistic rules, where the syntax and semantics of the logical rules are inherited from ASP and the probabilistic reasoning algorithm is based on a causal Bayesian network. An ASP program consists of a set of logical rules, separated by the symbol "←" (as shown below). The left side is called the *head* and the right is called the *body*. A rule is read as "head is true if body is true", and specifically, a rule with an empty body is referred to as a *fact*.

$$\mathtt{l_0 \text{ or } \cdots \text{ or } l_k \leftarrow l_{k+1}, \cdots, l_m, \text{not } l_{m+1}, \cdots, \text{not } l_n}.$$

The $\mathtt{l}$'s in ASP rules are literals, i.e., an expression of $\mathtt{p}$ or $\neg\mathtt{p}$, where $\mathtt{p}$ is an object constant or a variable. Symbol $\mathtt{not}$ is a logical connective called *default negation*; $\mathtt{not\ l}$ is read as "it is not believed that $\mathtt{l}$ is true", which does not imply $\mathtt{l}$ is believed to be false, e.g., $\mathtt{not\ prof(alice)}$ means it is unknown that $\mathtt{alice}$ is a professor. Using default negation, ASP can represent default knowledge with exceptions.

Traditionally, ASP does not explicitly quantify degrees of uncertainty: a literal is either true, false or unknown. P-LOG is an extension to ASP that allows *random functions*, saying that if $\mathtt{B}$, a collection of extended literals (i.e., $\mathtt{l}$ or $\mathtt{not\ l}$) holds, the value of $\mathtt{a(\bar{t})}$ is selected randomly from the set $\{\mathtt{X : q(X)}\} \cap \mathtt{range(a)}$, unless this value is fixed elsewhere, where $\mathtt{q}$ is a predicate:

$$\mathtt{random(a(\bar{t}) : \{X : q(X)\}) \leftarrow B}.$$

Finally, a *probability atom* (or *pr-atom*) states that, if $\mathtt{B}$ holds, the probability of $\mathtt{a(\bar{t}) = y}$ is θ:

$$\mathtt{pr(a(\bar{t}) = y|B)} = \theta, \text{where, } \theta \in [0,1].$$

As a result, we can easily use P-LOG's default negation for logical commonsense reasoning such as office doors are normally closed on holidays and use its probability atom for probabilistic commonsense reasoning such as sunlight reduces the success rate of a robot navigating through an area. Although P-LOG is good at commonsense reasoning, it does not support planning under uncertainty toward maximizing long-term rewards, which motivates the use of P-LOG for dynamically constructing (PO)MDPs in this work.

3 Algorithm

A global state space $\mathcal{S}^G$ can be specified using a set of *endogenous* attributes $\mathcal{V}^{en}$ (whose values can be changed by robot actions) and a set of *exogenous* attributes $\mathcal{V}^{ex}$ (whose values are changed by external factors).

$$\mathcal{S}^G : \mathtt{v}_1^{en} \times \cdots \times \mathtt{v}_n^{en} \times \mathtt{v}_1^{ex} \times \cdots \times \mathtt{v}_m^{ex}$$

where $\mathtt{v}^{en}$'s and $\mathtt{v}^{ex}$'s are *endogenous* and *exogenous* attributes respectively: $\mathtt{v}^{en} \in \mathcal{V}^{en}$ and $\mathtt{v}^{ex} \in \mathcal{V}^{ex}$.

In principle, all of these domain attributes, both endogenous and exogenous, can be modeled within a (PO)MDP. However, in practice there are often too many exogenous events to model all of them. Therefore, we take defaults and facts as the input to reason about all domain attributes in $\mathcal{S}^G$, and then compute a much lower-dimensional state space $\mathcal{S}$ for a (PO)MDP that focuses on a specific task, $\mathcal{S} : \mathtt{v}_1^{en} \times \cdots \times \mathtt{v}_n^{en}$. In case of POMDPs, reasoning with probabilistic commonsense rules associates a probability to each state $s \in \mathcal{S}$ and the probabilities together form a prior belief distribution, so the POMDP-based planning starts with this informative prior when interacting with the environment (similar to CORPP [Zhang and Stone, 2015]). We focus on constructing other components of (PO)MDPs in this paper, especially the state transition system $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ and reward system $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ (§ 3.1), which were both treated as static

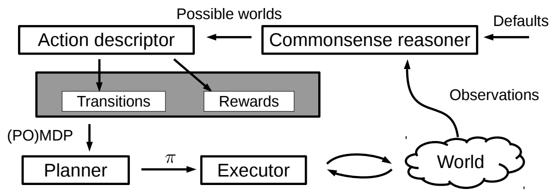


Figure 1: Overview of DCPARP

by CORPP. We use example problems to present the whole process of constructing (PO)MDPs (§ 3.2).

3.1 Algorithm Description

The main idea of DCPARP is to dynamically construct probabilistic graphical models, e.g., (PO)MDPs, using a declarative language that is strong in both logical and probabilistic commonsense reasoning, e.g., P-LOG, and compute policies that are adaptive to exogenous domain changes at runtime. Figure 1 shows a pictorial overview, where this section focuses on the commonsense reasoner and the action descriptor.

Commonsense reasoner (CR) CR includes both logical and probabilistic reasoning rules in P-LOG, and models both exogenous and endogenous domain attributes. Informally, the aim of CR is to understand the parts of the world that may have effects on the robot working on the current task.

Since real-world domains are dynamically changing all the time and robots' observations are partial and unreliable, robots frequently need to reason with *incomplete* domain knowledge[1]. ASP, on which P-LOG is based, well supports CR to take a set of defaults as input and smoothly revise their values using observed "facts" when available, and hence supports reasoning with incomplete domain knowledge well. As an example, a robot using an MDP for indoor navigation may have default knowledge: "area *A* is under sunlight in the mornings". A fact of "no sunlight is observed in area *A*" can smoothly defeat the default. The set of possible worlds, $\mathcal{W}$, is described by a set of *n* endogenous attributes and their values.

To represent state transitions, we define two identical state spaces using predicates `curr_s` and `next_s` in P-LOG:

$$\mathtt{curr\_s}(\mathtt{V}_1,\cdots,\mathtt{V}_\mathtt{n}) \leftarrow \mathtt{v}_1=\mathtt{V}_1,\cdots,\mathtt{v}_\mathtt{n}=\mathtt{V}_\mathtt{n}.$$
$$\mathtt{next\_s}(\mathtt{V}_1,\cdots,\mathtt{V}_\mathtt{n}) \leftarrow \mathtt{v}'_1=\mathtt{V}_1,\cdots,\mathtt{v}'_\mathtt{n}=\mathtt{V}_\mathtt{n}.$$

where `curr_s` and `next_s` specify the current and next states and the `v`'s and `V`'s are endogenous attributes and their variables respectively.

If there is at least one endogenous attribute whose value is not directly observable to the robot, the corresponding task needs to be modeled as a POMDP (otherwise, an MDP).

Action descriptor (AD) We introduce sort `action` and explicitly list a set of *i* actions, $\mathcal{A}$, as a set of *objects* in P-LOG. Random function `curr_a` maps to one of the actions.

$$\mathtt{action}=\{\mathtt{a}_0,\mathtt{a}_1,\cdots,\mathtt{a}_\mathtt{i}\}.\ \ \mathtt{curr\_a}:\mathtt{action}.\ \ \mathtt{random}(\mathtt{curr\_a}).$$

The probabilistic state transitions, $\mathcal{T}(s,a,s')$, can be described using a set of pr-atoms in P-LOG. For instance, the rule below states that the probability of action `A` changing the value of attribute `v` from $\mathtt{V}_1$ to $\mathtt{V}_2$ is 0.9.

$$\mathtt{pr}(\mathtt{v}'=\mathtt{V}_2 \mid \mathtt{v}=\mathtt{V}_1,\ \mathtt{curr\_a}=\mathtt{A})=0.9.$$

For MDPs, the values of endogenous attributes are fully observable to the robot, whereas POMDPs need to model a set of observations, *Z*, for estimating the underlying state. We define `obser` as a sort, and `curr_o` as a random function that maps to an observation object `o`.

$$\mathtt{obser}=\{\mathtt{o}_0,\mathtt{o}_1,\cdots,\mathtt{o}_\mathtt{j}\}.\ \ \mathtt{curr\_o}:\mathtt{obser}.\ \ \mathtt{random}(\mathtt{curr\_o}).$$

The observation function, $\mathcal{O}$, defines the probability of observing `O` given the current state being `s` and current action being `a`. For instance, the rule below states that, if attribute `v`'s current value is `V`, the probability of observing `O` after taking action `A` is 0.8.

$$\mathtt{pr}(\mathtt{curr\_o}=\mathtt{O} \mid \mathtt{curr\_a}=\mathtt{A},\ \mathtt{v}=\mathtt{V})=0.8.$$

The reward function *R* maps a state-action pair to a numeric value. For instance, this rule states that taking action `A` given attribute `v`'s value being `V` yields a reward of 10.0.

$$\mathtt{reward}(10.0,\mathtt{A},\mathtt{V}_1,\cdots,\mathtt{V}_\mathtt{n}) \leftarrow \mathtt{curr\_a}=\mathtt{A},\ \mathtt{curr\_s}(\mathtt{V}_1,\cdots,\mathtt{V}_\mathtt{n}).$$

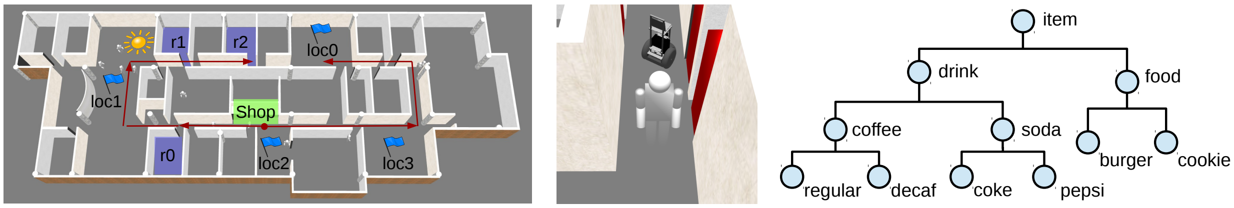
Algorithm 1 Algorithm DCPARP

Require: a set of defaults $\mathcal{D}$; (PO)MDP and P-LOG solvers
1: collect facts $\mathcal{F}^{ex}$ for exogenous attributes $\mathcal{V}^{ex}$
2: **repeat**
3: add $\mathcal{F}^{ex}$ and $\mathcal{D}$ into commonsense reasoner, CR
4: calculate possible worlds $\mathcal{W}$ (each corresponds to a state *s*)
5: **if** $\exists \mathtt{v}^{en} \in \mathcal{V}^{en}$, whose value is not directly observable **then**
6: calculate a prior belief distribution *b* over $\mathcal{W}$
7: **end if**
8: generate $\mathcal{T}$ and $\mathcal{R}$ by reasoning with $\mathcal{W}$ in AD
9: compute policy π for the (PO)MDP specified by $\mathcal{T}$ and $\mathcal{R}$
10: **while** *s* **is not** *term* **and** $\mathcal{F}^{ex}$ is consistent with $\mathcal{W}$ **do**
11: make an observation *z* about endogenous attributes $\mathcal{V}^{en}$
12: update state *s* (or belief state *b*) using *z*
13: select action *a* with π, execute *a*, and update $\mathcal{F}^{ex}$
14: **end while**
15: **until** *s* **is** *term*

Algorithm 1 completely specifies the DCPARP algorithm. The robot first makes observations to collect facts $\mathcal{F}^{ex}$ for exogenous attributes $\mathcal{V}^{ex}$. In Steps 3-4, CR takes defaults $\mathcal{D}$ and facts $\mathcal{F}^{ex}$ as input and computes a set of possible worlds $\mathcal{W}$, where each $w \in \mathcal{W}$ is described by a set of endogenous attributes (and their values). In Steps 5-7, we compute a prior belief *b* over $\mathcal{W}$ for POMDPs. AD takes $\mathcal{W}$ as input and computes transition probabilities $\mathcal{T}$ and reward function $\mathcal{R}$. The planner can compute a policy $\pi : s \rightarrow a$ using algorithms such as SARSOP (for POMDPs) and value iteration or Monte Carlo tree search (for MDPs). Finally, the action executor uses π for interacting with the environment by making observations and taking actions, until a terminal state is reached or exogenous facts lead to inconsistency. In case of inconsistency, we return to Step 3 to recompute the possible worlds.

As an example of exogenous facts causing inconsistency, consider a robot that plans to avoid the area under sunlight (which blinds the sensors) when it was started. An exogenous fact of "current weather not being sunny" causes inconsistency with all possible worlds, so the robot reactivates CR (Step 3) to recompute the MDP state space (and recompute

[1] When we solve an MDP problem, we simply assume the endogenous attributes are fully observable. Robots face a partially observable world in general.



(a) (b) (c)

Figure 2: (a) Simulation environment used in experiments, where the red arrows indicate the delivery routes from the shop to individual rooms; (b) A human walker blocking the way of the robot; and (c) An ontology of available items used in the "shopping" task (Task 1).

the acting policy). Therefore, DCPARP enables the robot's behavior to adapt to the fact of a weather change.

3.2 Algorithm Instantiations on a Mobile Robot

In § 3.1, we describe the transition and reward systems by *enumerating* all the probabilities and rewards, which can be very inefficient. In practice, we use domain-dependent attributes for much more efficient representations. To demonstrate such representations and evaluate DCPARP's performance, we apply DCPARP to two tasks: *shopping request identification* and *robot navigation*. Both task domains have exogenous changes at runtime. Figure 2(a) shows our simulation environment that is constructed using GAZEBO [Koenig and Howard, 2004] and shared by the two tasks.

Task 1: shopping request identification

This task was first introduced in [Zhang and Stone, 2015]. We add more details including distances between rooms and ontology of items (Figure 2) and use this task to evaluate how DCPARP enables the robot to adapt to exogenous domain changes and fine-tune its behaviors. This domain has the following *sorts*, Θ, and each sort has a set of objects.

$$\begin{aligned}
&\texttt{time} = \{\texttt{morning}, \texttt{noon}, \texttt{afternoon}, \texttt{evening}\}\\
&\texttt{room} = \{\texttt{r0}, \texttt{r1}, \texttt{r2}, \texttt{shop}\}.\\
&\texttt{person} = \{\texttt{alice}, \texttt{bob}, \texttt{carol}, \texttt{dan}\}.\\
&\texttt{item} = \{\texttt{regular}, \texttt{decaf}, \texttt{coke}, \texttt{pepsi}, \texttt{burger}, \texttt{cookie}\}.\\
&\texttt{class} = \{\texttt{item}, \texttt{drink}, \texttt{food}, \texttt{coffee}, \texttt{soda}\}.
\end{aligned}$$

We then define predicate set $\mathcal{P}$:{`request`, `subcls`}, where `request(I,R,P)` specifies a shopping request of delivering item `I` to room `R` for person `P`, and $\texttt{subcls}(\texttt{C}_1,\texttt{C}_2)$ claims class $\texttt{C}_1$ to be a subclass of class $\texttt{C}_2$. Figure 2(c) shows the categorical tree that can be represented using the following rules, where $\texttt{C}_1, \texttt{C}_2$, and $\texttt{C}_3$ are variables of classes.

$$\begin{aligned}
&\texttt{subcls}(\texttt{C}_1,\texttt{C}_3) \leftarrow \texttt{subcls}(\texttt{C}_1,\texttt{C}_2),\ \texttt{subcls}(\texttt{C}_2,\texttt{C}_3).\\
&\texttt{is}(\texttt{I},\texttt{C}_1) \leftarrow \texttt{is}(\texttt{I},\texttt{C}_2),\ \texttt{subcls}(\texttt{C}_2,\texttt{C}_1).
\end{aligned}$$

A set of random functions describes the possible values of random variables: `curr_time`, `req_item(P)`, `req_room(P)`, and `req_person`. E.g., the two rules below state that if the delivery is for person `P`, the value of `req_item` is randomly selected from the range of `item`, unless fixed elsewhere:

$$\texttt{random}(\texttt{req\_item}(\texttt{P})). \quad \texttt{req\_item} : \texttt{person} \rightarrow \texttt{item}.$$

We can then use a *pr-atom* to specify a probability. For instance, the rule below states that the probability of delivering `coffee` in the morning is 0.8.

$$\texttt{pr}(\texttt{req\_item}(\texttt{P}) = \texttt{coffee} | \texttt{curr\_time} = \texttt{morning}) = 0.8.$$

Such random functions and pr-atoms allow us to represent and reason with probabilistic commonsense knowledge. Finally, the current state (a shopping request) is specified as follows: $\texttt{curr\_s}(\texttt{I},\texttt{R},\texttt{P},\texttt{term}) \leftarrow \texttt{request}(\texttt{I},\texttt{R},\texttt{P}),\ \texttt{term}.$, where predicate `term` identifies the terminal state. The action set is explicitly defined as below.

$$\begin{aligned}
\texttt{action} = \{&\texttt{ask_i}, \texttt{ask_r}, \texttt{ask_p}, \texttt{conf_i0}, \texttt{conf_i1} \cdots, \texttt{conf_r0},\\
&\texttt{conf_r1}, \cdots, \texttt{conf_p0}, \texttt{conf_p1} \cdots, \texttt{del_i0_r0_p0}, \cdots\}
\end{aligned}$$

where, `ask_`'s are general questions (e.g., `ask_r` corresponds to "which room to deliver?"), `conf_`'s are confirming questions (e.g., `conf_r0` corresponds to "is this delivery to `room0`?"), and `del_`'s are actions of deliveries.

For delivery actions, the reward function $\mathcal{R}$ maps a state-action pair to a real number, and is defined as:

$$\mathcal{R}(a^{del}, s) = \begin{cases} R^+, & \textbf{if } a_i \odot s_i \textbf{ and } a_p \odot s_p \textbf{ and } a_r \odot s_r \\ (1 - \lambda_i(a_i, s_i) \cdot \lambda_p(a_p, s_p) \cdot \lambda_r(a_r, s_r))R^-, & \textbf{otherwise} \end{cases}$$

where operator $\odot$ returns true if the action on the left matches the state on the right in the given dimension (subscript). λ in the range of (0, 1] measures the closeness between actual delivery (action) and underlying request (state) in item, person, and room, respectively. R^+ and R^- are the reward and penalty that a robot can get in extreme cases (completely correct or completely incorrect deliveries).

We compute the closeness of two items, $\lambda(I_1, I_2)$ by post-processing the resulting answer set. Specifically, the heuristic closeness function of two items is defined as:

$$\lambda_i(I_1, I_2) = 1 - \frac{max\big(dep(LCA, I_1), dep(LCA, I_2)\big) - 1}{max\big(dep(root, I_1), dep(root, I_2)\big)} \tag{1}$$

where *LCA* is the lowest common ancestor of I_1 and I_2 and *dep(C,I)* is the number of nodes (inclusive) between C and I.

Informally, the closeness of room R_1 to room R_2 is inversely proportional to the effort needed to recover from a delivery to R_1 given the request being to R_2. In Figure 2(a), for instance, a wrong delivery to $r0$ given the request being to $r1$ requires the robot to go back to shop, learn the delivery room being $r1$, and then move to room $r1$. Therefore, the *asymmetric* room closeness function is defined as below:

$$\lambda_r(R_1, R_2) = \frac{dis(shop, R_2)}{2 \cdot dis(shop, R_1) + dis(shop, R_2)} \tag{2}$$

We simply set λ_p to 1. The costs of question-asking actions are stationary: $\mathcal{R}(a^{ask}, s)$=-1, and $\mathcal{R}(a^{conf}, s)$=-2.

Task 2: robot navigation

In this task, the state is fully observable (MDP is used). The robot navigates in a domain shown in Figure 2(a), where moving people can (probabilistically) block its way—Figure 2(b), and sunlight can (probabilistically) blind the robot's laser range-finder, making the robot unrecoverably lost. Planning is done by mapping the domain to a grid, which is defined using the sorts of `row` and `col`, and the predicates `belowof` and `leftof`. We then introduce predicates `near_row` and `near_col` used for specifying if two grid cells are next to each other, where R's (C's) are variables of row (column).

$$\texttt{near\_row}(\texttt{RW}_1,\texttt{RW}_2) \leftarrow \texttt{belowof}(\texttt{RW}_1,\texttt{RW}_2).$$
$$\texttt{near\_row}(\texttt{RW}_1,\texttt{RW}_2) \leftarrow \texttt{near\_row}(\texttt{RW}_2,\texttt{RW}_1).$$
$$\texttt{near\_col}(\texttt{CL}_1,\texttt{CL}_2) \leftarrow \texttt{leftof}(\texttt{CL}_1,\texttt{CL}_2).$$
$$\texttt{near\_col}(\texttt{CL}_1,\texttt{CL}_2) \leftarrow \texttt{near\_col}(\texttt{CL}_2,\texttt{CL}_1).$$

To model the nondeterministic action outcomes, we define random functions `curr_row` and `next_row` that map to the current and next rows, and `curr_col` and `next_col` that map to the current and next columns.

$$\texttt{random}(\texttt{next\_row}:\{\texttt{R\_}:\texttt{near\_row}(\texttt{R\_},\texttt{RW})\}) \leftarrow \texttt{curr\_row}=\texttt{RW}.$$
$$\texttt{random}(\texttt{next\_col}:\{\texttt{C\_}:\texttt{near\_col}(\texttt{C\_},\texttt{CL})\}) \leftarrow \texttt{curr\_col}=\texttt{CL}.$$

We use predicates `near_window` and `sunny` to define the cells that are near to window and the cells that are actually under sunlight. The rule below is a default stating that: in the mornings, a cell near window is believed to be under sunlight, unless defeated elsewhere.

$$\texttt{sunny}(\texttt{RW},\texttt{CL}) \leftarrow \texttt{near\_window}(\texttt{RW},\texttt{CL}),\ \texttt{not}\ \neg\texttt{sunny}(\texttt{RW},\texttt{CL}),\ \texttt{curr\_time}=\texttt{morning}.$$

While navigating in areas under sunlight, there is a large probability of becoming lost (0.9), which deterministically leads to the end of an episode.

$$\texttt{pr}(\texttt{next\_term}=\texttt{true} \mid \texttt{curr\_row}=\texttt{RW},\ \texttt{curr\_col}=\texttt{CL},\ \texttt{sunny}(\texttt{RW},\texttt{CL}))=0.9.$$
$$\texttt{pr}(\texttt{next\_term}=\texttt{true} \mid \texttt{curr\_term}=\texttt{true})=1.0.$$

The robot can take actions to move to a grid cell next to its current one: $\texttt{action}=\{\texttt{left},\texttt{right},\texttt{up},\texttt{down}\}$. For instance, given action `up`, the probability of successfully moving to the above grid cell is 0.9, given no obstacle in the above cell.

$$\texttt{pr}(\texttt{next\_row}=\texttt{RW}_2 \mid \texttt{curr\_row}=\texttt{RW}_1,\ \texttt{curr\_col}=\texttt{CL}_1,\ \texttt{belowof}(\texttt{RW}_1,\texttt{RW}_2),\ \neg\texttt{sunny}(\texttt{RW}_2,\texttt{CL}_1),\ \neg\texttt{blocked}(\texttt{RW}_2,\texttt{CL}_1),\ \texttt{curr\_a}=\texttt{up})=0.9.$$

Finally, the current state is specified by endogenous attributes `curr_row`, `curr_col`, and `curr_term`:

$$\texttt{curr\_state}(\texttt{RW},\texttt{CL},\texttt{TM}) \leftarrow \texttt{curr\_row}=\texttt{RW},\ \texttt{curr\_col}=\texttt{CL},\ \texttt{curr\_term}=\texttt{TM}.$$

The goal of visiting room (`r0`, `c3`) can be defined as below, where an early termination has a penalty of -100.0.

$$\texttt{pr}(\texttt{next\_term}=\texttt{true} \mid \texttt{curr\_row}=\texttt{r0},\texttt{curr\_col}=\texttt{c3})=1.0.$$
$$\texttt{reward}(50.0,\texttt{A},\texttt{r0},\texttt{c3},\texttt{true}) \leftarrow \texttt{curr\_state}(\texttt{r0},\texttt{c3},\texttt{true}).$$
$$\texttt{reward}(-100.0,\texttt{A},\texttt{RW},\texttt{CL},\texttt{true}) \leftarrow \texttt{curr\_state}(\texttt{RW},\texttt{CL},\texttt{true}),\ \texttt{RW} <> \texttt{r0}.$$
$$\texttt{reward}(-100.0,\texttt{A},\texttt{RW},\texttt{CL},\texttt{true}) \leftarrow \texttt{curr\_state}(\texttt{RW},\texttt{CL},\texttt{true}),\ \texttt{CL} <> \texttt{c3}.$$

Informally, DCPARP decomposes a (PO)MDP problem into two subproblems, commonsense reasoning and probabilistic planning, that respectively focus on "curse of dimensionality" and "curse of history" (elaborated in [Kurniawati *et al.*, 2010]), aiming at significantly reducing the complexity of (PO)MDP planning compared to its one-shot solution.

4 Experimental Results

DCPARP has been implemented in simulation and on real robots. Experiments in simulation focus on statistical analysis and robot experiments are mostly for demonstrating the effectiveness on specific test cases. We evaluate two hypotheses that DCPARP enables the robot to: (I) fine-tune its behavior; and (II) adapt to exogenous domain changes. We take CORPP as the baseline algorithm unless specified otherwise.

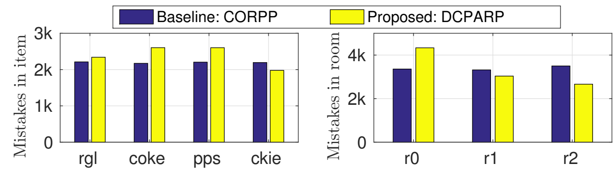


Figure 3: DCPARP enables the robot to fine-tune its behavior in delivering different items to different rooms. The x-axis and y-axis correspond to the *incorrect* deliveries and the number of mistakes (over 100*k* trials). For instance, the `r0` in the right bars represents the numbers of deliveries to `r0` given `r1` or `r2` being requested.

Experiments in simulation were conducted using GAZEBO [Koenig and Howard, 2004] on a desktop machine with 16G memory and Intel Core i7 CPU at 3.40GHz. We used a solver introduced in [Zhu, 2012] for P-LOG programs (except that reasoning about reward was manually conducted), the APPL solver for POMDPs [Kurniawati *et al.*, 2008], and value iteration for MDPs [Sutton and Barto, 1998].

Hypothesis-I (Task 1) We use Task 1 with four items, three rooms and two persons for comparing DCPARP to CORPP (the baseline). The hidden shopping request was randomly selected in each trial. Speech recognition errors are modeled, e.g., 0.8 accuracy in recognizing answers of confirming questions and a lower accuracy for general questions (depending on the number of that sort's objects). The value of $[R^+, R^-]$ is $[20, -20]$. Since CORPP gives full penalties to partially correct deliveries whereas DCPARP does not, we adjust the value of R^- for DCPARP to make the overall reward of delivery actions, $\sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}^{del}}\mathcal{R}(a,s)$, comparable to each other.

Figure 3 shows the numbers of mistakes made by the robot. We can see CORPP makes no difference in either item (**Left**) or room (**Right**), because it does not reason about the reward system—incorrect deliveries are not differentiated and all receive the same penalty. In contrast, DCPARP enables the robot to behave in such a way that the robot makes the fewest mistakes in `cookie` (**Left**) and room `r2` (**Right**). Such behaviors match our expectations: `cookie` is "very different" from the other three items and `r2` has the greatest distance from the shop, so the robot should make effort to avoid delivering `cookie` (or delivering to `r2`) when that is not requested. The results support Hypothesis-I that using DCPARP the robot is able to fine-tune its behavior.

To better understand the robot's behavior (specifically, the **Right** of Figure 3), we manually remove the uncertainties in `item` and `person` in the initial belief, and visualize which action the POMDP policy suggests given different initial beliefs in `room`. In the **Right** of Figure 4, we see the robot is relatively more cautious in delivering to `r1` and `r2` (the green and yellow areas in the top and left corners are smaller than the red one in the right). It is very difficult to achieve such fine-tuned behaviors from hand-coded models.

Hypothesis-II (Task 1) Figure 5 shows the results of the "shopping" task when exogenous changes are added: items can be temporarily unavailable. Since CORPP cannot adapt to such exogenous changes, it has to model all items all the

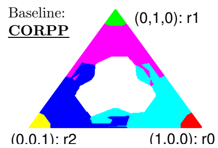


Figure 4: A visualization of CORPP and DCPARP policies, where the person wants to deliver to one of the three rooms. Each point corresponds to a belief. Each color corresponds to an action: white corresponds to the general question of "which room to deliver"; the colors in the corners correspond to delivery actions; and the remaining three colors correspond to confirming questions.

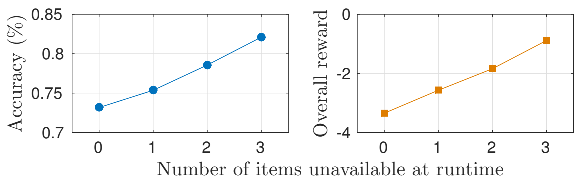


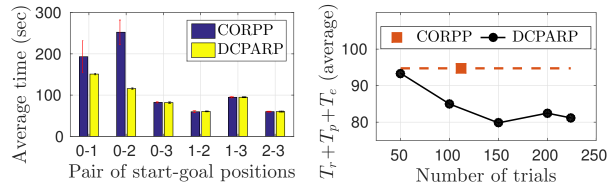
Figure 5: DCPARP performs increasingly better in accuracy and overall reward in the shopping task when more items are known to be unavailable: CORPP corresponds to the left ends of the two curves (CORPP uses a static model so it has to include all items).

time. In contrast, DCPARP dynamically constructs POMDPs: when items are known to be unavailable, states of these items being requested and actions of delivering these items are removed from the POMDP. For instance, when three items are unavailable, the numbers of states and actions are reduced from $(37, 50)$ to $(18, 29)$. As a result, DCPARP performs increasingly better in both accuracy and overall reward (y-axes in Figure 5) when more items are known to be unavailable (x-axes in Figure 5). In contrast, CORPP has to use a static POMDP that includes all items (assuming no item unavailable), so its performance corresponds to the left ends of the two curves. Results shown in Figure 5 supports the hypothesis that DCPARP enables the robot to adapt to dynamic changes.

Hypothesis-II (Task 2) We further evaluate Hypothesis-II using the "navigation" task: the testing environment and the robot are shown in Figure 2(a) and 2(b). We limit the number of random walkers to be 1 and its speed to be one fifth of the robot's. A goal room is randomly selected from the four flag rooms. Reasoning happens only after the current episode is terminated (goal room is reached). The walker's position is the only exogenous domain change (by temporarily setting the time to be "evening"). We cached policies for both CORPP (4 policies) and our approach (56 policies) at runtime.

Figure 6(a) shows the robot's traveling time given start-goal pairs: once the robot arrives at its current goal, the next one is randomly selected. The walker moves slowly near the door of room $r1$. Without adaptive planning developed in this work, the robot follows the "optimal" path and keeps trying to bypass the walker for a fixed length of time. If the low-level motion planner does not find a way to bypass the walker within the time, the robot will take the other way to navigate to the other side of the walker and continues executing the "optimal" plan generated by the outdated model. We can see when the robot navigates between $loc0$ and $loc2$, DCPARP reduces the traveling time from about 250 seconds to about 110 seconds, producing a significant improvement.

Results over 8.5 hours of experiments are shown in Figure 6(b): 224 trials using DCPARP and 112 trials using CORPP.



(a) (b)

Figure 6: (a) Average time consumed in navigating between location pairs when a walker moves near the door of room $r1$ (error bars represent standard deviations); **(b)** DCPARP enables the robot to adapt to exogenous domain changes (the walker's position). Results are processed in batches (each has 50 trials, when available).

Without caching, we find the time consumed by DCPARP (over 54 trials) is distributed over P-LOG reasoning (T_r, 28%), MDP planning (T_p, $<1\%$), and execution (T_e, 72%). Compared to CORPP, DCPARP enables the robot to spend much less time in execution (T_e) in all phases. At the beginning phase, DCPARP requires more reasoning time for dynamically constructing MDPs, which together with the less execution time makes the overall time comparable to CORPP (left ends of Figure 6(b)). Eventually, the low execution time (T_e) dominates the long-term performance (right ends of Figure 6(b)), supporting the hypothesis that DCPARP enables the robot to adapt to exogenous domain changes.

Hypothesis-II (Task 2 on a robot) To test the robot's behavior adapting to sunlight change, we placed a Segway-based robot (**Right** of Figure 7) at the start position shown in the **Left** of Figure 7, and left the robot two routes that lead to the goal, with Route 2 being shorter. DCPARP enables the robot to select the safer route (Route 1), even though it is longer. Demo videos of simulated and real-robot trials are available (anonymously) at: `goo.gl/n6apCt`

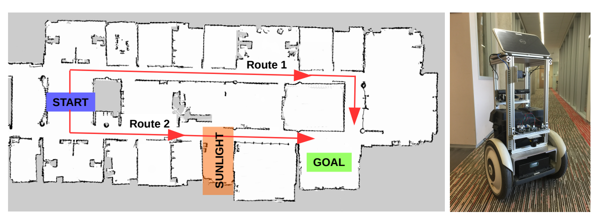


Figure 7: Floor map and the Segway-based robot used in experiments. DCPARP enables the robot to select "Route 1", successfully avoiding the "sunlight" area along "Route 2".

5 Conclusions

This paper introduces a novel algorithm called DCPARP that uses commonsense reasoning to dynamically construct (PO)MDPs for adaptive robot planning. We use declarative language P-LOG, a probabilistic extension of answer set programming, for reasoning with logical and probabilistic commonsense knowledge, and use probabilistic graphical models, such as (PO)MDPs, for probabilistic planning. This paper, for the first time, enables robot behaviors to adapt to exogenous domain changes without including these exogenous attributes in probabilistic planning models. DCPARP has been evaluated both in simulation and on a real robot. We observed significant improvements comparing to competitive baselines (including CORPP), based on experiments on two tasks in an office environment.

Acknowledgements

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CNS-1330072, CNS-1305287), ONR (21C184-01), and AFOSR (FA9550-14-1-0087). Peter Stone serves on the Board of Directors of, Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

[Baral *et al.*, 2009] Chitta Baral, Michael Gelfond, and Nelson Rushton. Probabilistic Reasoning with Answer Sets. *Theory and Practice of Logic Programming*, 9(1):57–144, 2009.

[Baral, 2003] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[Boutilier *et al.*, 1999] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11(1):94, 1999.

[Chen *et al.*, 2012] Xiaoping Chen, Jiongkun Xie, Jianmin Ji, and Zhiqiang Sui. Toward open knowledge enabling for human-robot interaction. *Journal of Human-Robot Interaction*, 1(2):100–117, 2012.

[Erdem *et al.*, 2015] Esra Erdem, Volkan Patoglu, and Zeynep G Saribatur. Integrating hybrid diagnostic reasoning in plan execution monitoring for cognitive factories with multiple robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2007–2013, 2015.

[Gelfond and Kahl, 2014] Michael Gelfond and Yulia Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press, 2014.

[Hanheide *et al.*, 2015] Marc Hanheide, Moritz Göbelbecker, Graham S Horn, Andrzej Pronobis, Kristoffer Sjöö, Alper Aydemir, Patric Jensfelt, Charles Gretton, Richard Dearden, Miroslav Janicek, et al. Robot task planning and explanation in open and uncertain worlds. *Artificial Intelligence*, 2015.

[Kaelbling *et al.*, 1998] Leslie Kaelbling, Michael Littman, and Anthony Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101:99–134, 1998.

[Khandelwal *et al.*, 2015] Piyush Khandelwal, Samuel Barrett, and Peter Stone. Leading the way: An efficient multi-robot guidance system. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1625–1633, 2015.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.

[Koenig and Howard, 2004] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[Kurniawati *et al.*, 2008] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, 2008.

[Kurniawati *et al.*, 2010] Hanna Kurniawati, Yanzhu Du, David Hsu, and Wee Sun Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research*, 2010.

[Lee and Wang, 2015] Joohyung Lee and Yi Wang. A probabilistic extension of the stable model semantics. In *AAAI Spring Symposium on Logical Formalizations of Commonsense easoning, Stanford, USA*, 2015.

[Milch *et al.*, 2007] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L Ong, and Andrey Kolobov. Blog: Probabilistic models with unknown objects. *Statistical relational learning*, page 373, 2007.

[Myagmarjav and Sridharan, 2015] Batbold Myagmarjav and Mohan Sridharan. Incremental knowledge acquisition for human-robot collaboration. In *Robot and Human Interactive Communication (RO-MAN), 2015 24th IEEE International Symposium on*, pages 809–814. IEEE, 2015.

[Perera *et al.*, 2015] Vittorio Perera, Robin Soetens, Thomas Kollar, Mehdi Samadi, Yichao Sun, Daniele Nardi, René van de Molengraft, and Manuela Veloso. Learning task knowledge from dialog and web access. *Robotics*, 4(2):223–252, 2015.

[Richardson and Domingos, 2006] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[Samadi *et al.*, 2012] Mehdi Samadi, Thomas Kollar, and Manuela Veloso. Using the web to interactively learn to find objects. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[Young *et al.*, 2013] Steve Young, Milica Gasic, Blaise Thomson, and Jason D. Williams. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013.

[Zhang and Stone, 2015] Shiqi Zhang and Peter Stone. CORPP: Commonsense reasoning and probabilistic planning, as applied to dialog with a mobile robot. In *Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI)*, pages 1394–1400, 2015.

[Zhang *et al.*, 2015] Shiqi Zhang, Mohan Sridharan, and Jeremy L. Wyatt. Mixed logical inference and probabilistic planning for robots in unreliable worlds. *IEEE Transactions on Robotics*, 31(3):699–713, June 2015.

[Zhu, 2012] Weijun Zhu. *PLOG: Its Algorithms and Applications*. PhD thesis, Texas Tech University, USA, May 2012.