Preference Planning for Markov Decision Processes

Meilun Li and **Zhikun She**
School of Mathematics and Systems Science
Beihang University, China

Andrea Turrini and **Lijun Zhang**
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing, China

Abstract

The classical planning problem can be enriched with quantitative and qualitative user-defined preferences on how the system behaves on achieving the goal. In this paper, we propose the probabilistic preference planning problem for Markov decision processes, where the preferences are based on an enriched probabilistic LTL-style logic. We develop P4Solver, an SMT-based planner computing the preferred plan by reducing the problem to quadratic programming problem, which can be solved using SMT solvers such as Z3. We illustrate the framework by applying our approach on two selected case studies.

Introduction

Classical planning problem focuses on finding a sequence of actions leading from initial state to the user-defined set of goal states. For many problems in reality, however, users may have preferences over some special actions than other ones, or willingness to touch some special group of states before reaching the goal. This leads to the promotion of preference-based planning to integrate planning with user preferences (Baier and McIlraith 2009).

Planning with preferences has been studied for decades and becomes even more attractive in recent years. Many researchers have proposed preference languages to effectively and succinctly express user's preferences (Coste-Marquis et al. 2004; Delgrande, Schaub, and Tompits 2004; Boutilier et al. 2004), algorithms for designing preference-based planners corresponding to the different preference languages, or algorithms to increase planning efficiency (Bacchus and Kabanza 2000; Edelkamp 2006; Sohrabi, Baier, and McIlraith 2009; Tu, Son, and Pontelli 2007; Baier, Bacchus, and McIlraith 2009).

The above mentioned work is essentially based on nondeterministic transition systems. For most systems in reality, however, the effect of system actions is often unpredictable. Even though we cannot precisely know what will follow after an action, some event is more (or less) frequent than other events. This motivates incorporating probability to planning problems to capture this kind of uncertainty. Markov Decision Processes (MDPs) (Bellman 1957) provide a mathematical framework for modelling decision making in situations where outcomes are partly stochastic and partly under the control of a decision maker. It is a unifying model for many classes of planning problems studied in AI (Boutilier, Dean, and Hanks 1999).

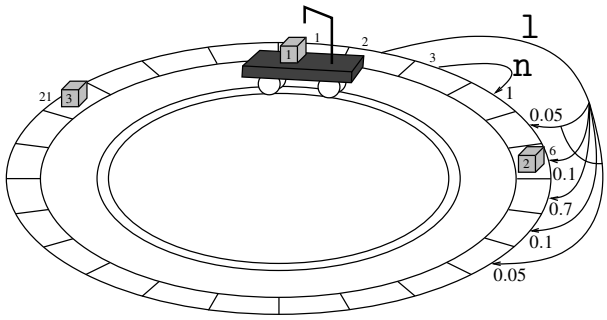

Figure 1: The robot MDP

In this paper, we propose the probabilistic preference-based planning problem on MDPs, which we refer to as P4. Derived from probabilistic linear-time temporal logic (probabilistic LTL) (Hansson and Jonsson 1994; Baier and Katoen 2008), we propose a preference language allowing us to specify the goal and preferences in terms of probabilistic formulas. Using the language, we can for instance express the property "the goal states are eventually reached with probability at least 0.95" (in probabilistic LTL style, $\mathcal{P}_{\geq 0.95}(\mathbf{F}(goal))$) and, among all ways, we prefer those such that "with probability at least 0.99 the system never enters a forbidden state" ($\mathcal{P}_{\geq 0.99}(\mathbf{G}(\neg forbidden))$). Further, we propose P4Solver, a planner for P4, in which we reduce P4 to a quadratic programming problem. We use the nonlinear real arithmetic of Z3 (de Moura and Bjørner 2008) to solve the problem, after having encoded it in the SMT-LIB format (Barrett, Stump, and Tinelli 2010). The experimental results confirm in general the effectiveness of the approach.

A Motivating Example

As an example of problems our technique addresses, consider a robot moving in one direction on a ring rail surrounded by N loading and unloading areas (or positions), depicted in Figure 1. The task of the robot is to sort B boxes between these areas, with $B < N$. The robot alternates be-

tween two modes: control ($mode(\mathtt{c})$) and action ($mode(\mathtt{a})$). In the control mode, the robot decides what to do next: either to move on the rail (m) or to use its arm to load/unload boxes (a). As movement, either it can go to the next area (n), or it can go quickly 5 areas ahead (l); the actual reached area depends on the distance: starting from the area i, action n reaches the area $(i+1) \bmod N$ with probability 1 while action l reaches the area $(i+5) \bmod N$ with probability 0.7, the areas $(i+4) \bmod N$ and $(i+6) \bmod N$ with probability 0.1 each, and the areas $(i+3) \bmod N$ and $(i+7) \bmod N$ with probability 0.05 each. When the robot faces an area i containing a box j and it is in action mode, it can pick it up ($\mathtt{p}(j,i)$) so that it can later drop ($\mathtt{d}(j,t)$) it in the target area t, if such area is empty. Both operations succeed with probability 0.95. Only one box can be carried at a time. Possible user preferences are that the robot moves quickly at least once, that it drops box 1 somewhere, that a box is never picked up and dropped in the same area, and so on.

Preliminaries

In this section we introduce *Markov Decision Processes* (MDPs) (Bellman 1957) as our model. We fix Σ as the finite set of *atomic propositions*. Each state in the MDP is *labelled* with only the atomic propositions that are satisfied in this state[1]. State evolves to other states as the result of performing actions that probabilistically decide the state to transfer to. The formal definition of MDP is as follows.

Definition 1 *A* Markov Decision Process *is a tuple* $\mathcal{M} = (S, L, Act, \bar{s}, \mathrm{P})$ *where*

- S *is a finite set of states,*
- $L\colon S \to 2^{\Sigma}$ *is a* labeling function*,*
- Act *is a finite set of* actions*,*
- $\bar{s} \in S$ *is the* initial state*, and*
- $\mathrm{P}\colon S \times Act \to Dist(S)$ *is the* transition probability function *where by* $Dist(S)$ *we denote the set of probability distributions over* S*.*

For example, $robotAt(2)$ is the atomic proposition holding whenever the robot is in area 2. To simplify the exposition, we use the special action $\perp$ to model the stopping of the computation; for each state $s \in S$, $\mathrm{P}(s,\perp) = \delta_s$ where δ_s is the Dirac distribution assigning probability 1 to s. We denote by $Act(s)$ the set of possible actions from state s. Note that for each s, $Act(s) \neq \emptyset$ since $\perp \in Act(s)$.

A *path* π in an MDP $\mathcal{M}$ is a finite or infinite sequence of alternating states and actions $s_0 a_1 s_1 \ldots$ starting from a state s_0, also denoted by $first(\pi)$ and, if the sequence is finite, ending with a state denoted by $last(\pi)$, such that for each $i > 0$, $a_i \in Act(s_{i-1})$ and $\mathrm{P}(s_{i-1}, a_i)(s_i) > 0$. We denote the set of all finite paths for a given MDP $\mathcal{M}$ by $Paths^*_{\mathcal{M}}$ and the set of all finite and infinite paths by $Paths_{\mathcal{M}}$. We may drop the subscript $\mathcal{M}$ whenever it is clear from the context.

[1]In (Bienvenu, Fritz, and McIlraith 2011), fluents and non-fluent relational formulas are used for characterizing state properties. The former are of the form $f(s, x_0, x_1, \ldots)$ whose value depends on the assignment of variables x_i in the state s, while the latter do not depend on the state s.

For a given path $\pi = s_0 a_1 s_1 \ldots$, we denote by $|\pi|$ the length of π, i.e., the number of actions occurring in π. If π is infinite, then $|\pi| = \infty$. For $1 \leq i \leq |\pi|$, we denote by $action(\pi, i)$ the action a_i; for $0 \leq i \leq |\pi|$, we denote by $state(\pi, i)$ the state s_i and by $suffix(\pi, i)$ the suffix of π starting from $state(\pi, i)$.

The nondeterminism of the transitions enabled by a state is resolved by a *policy* χ based on the previous history. Formally, a policy for an MDP $\mathcal{M}$ is a function $\chi\colon Paths^* \to Dist(Act)$ such that for each $\pi \in Paths^*$, $\{\, a \in Act \mid \chi(\pi)(a) > 0 \,\} \subseteq Act(last(\pi))$ and if $action(\pi, |\pi|) = \perp$, then $\chi(\pi) = \delta_{\perp}$. The last condition ensures that no action except $\perp$ can be chosen after the stop action $\perp$ has occurred.

Given an MDP $\mathcal{M}$, a policy χ and a state s induce a probability distribution over paths as follows. The basic measurable events are the cylinder sets of finite paths, where the cylinder set of π, denoted by C_π, is the set $\{\, \pi' \in Paths \mid \pi \leqslant \pi' \,\}$ where $\pi \leqslant \pi'$ means that the sequence π is a prefix of the sequence π'. The probability $\mu_{\chi,s}$ of a cylinder set C_π is defined recursively as follows:

$$\mu_{\chi,s}(C_\pi) = \begin{cases} 0 & \text{if } \pi = t \text{ for a state } t \neq s, \\ 1 & \text{if } \pi = s, \\ \mu_{\chi,s}(C_{\pi'}) \cdot \chi(\pi')(a) \cdot \mathrm{P}(last(\pi'), a)(t) & \\ & \text{if } \pi = \pi' a t. \end{cases}$$

Standard measure theoretical arguments ensure that $\mu_{\chi,s}$ extends uniquely to the σ-field generated by cylinder sets. Given a finite path π, we define $\mu_{\chi,s}(\pi)$ as $\mu_{\chi,s}(\pi) = \mu_{\chi,s}(C_\pi) \cdot \chi(\pi)(\perp)$, where $\chi(\pi)(\perp)$ is the probability of terminating the computation after π has occurred. The definition could be extended to a set of finite paths: given a set of finite paths $\mathcal{B}$, we define $\mu_{\chi,s}(\mathcal{B})$ as $\mu_{\chi,s}(\mathcal{B}) = \sum_{\pi \in \mathcal{B}} \mu_{\chi,s}(\pi)$.

As an example of path and its probability, consider the path $\pi = s_0 \mathtt{m} s_1 \mathtt{l} s_2 \mathtt{a} s_3 \mathtt{p}(2,4) s_4$ relative to the robot, where each action is chosen by the policy with probability 1 and the states have the following meaning:

State	Robot at	Mode	CarryBox	Box 2 at
s_0	0	c	-	4
s_1	0	m	-	4
s_2	4	c	-	4
s_3	4	a	-	4
s_4	4	c	2	-

The probability of π is $1 \cdot 1 \cdot 1 \cdot 1 \cdot 0.1 \cdot 1 \cdot 1 \cdot 1 \cdot 0.95 = 0.095$.

Probabilistic Preference Planning Problem

In this section we present the Probabilistic Preference Planning Problem (P4). This is the extension of the classical preference-based planning problem to probabilistic settings. In P4, preference formulas and goal formulas are described with probability, and both are based on the concept of property formula.

Definition 2 *The syntax of a* property formula φ *is defined as follows:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{final}(\varphi) \mid \mathbf{occ}(a) \mid \mathbf{X}(\varphi) \mid \mathbf{U}(\varphi, \varphi) \mid \forall x(\varphi)$$

where $p \in \Sigma$ *is an atomic proposition and* $a \in Act$*.* x *is a variable that can appear only in the terms* p *and* a *of* $\mathbf{occ}(a)$*.*

Throughout the paper, we use standard derived operators such as $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$, $\mathbf{True} = p \vee \neg p$, $\mathbf{False} = \neg\mathbf{True}$, $\mathbf{F}(\varphi) = \mathbf{U}(\mathbf{True}, \varphi)$, $\mathbf{G}(\varphi) = \neg\mathbf{F}(\neg\varphi)$, $\varphi \Rightarrow \psi = \neg\varphi \vee \psi$, and $\exists x(\varphi) = \neg\forall x(\neg\varphi)$.

Remark 1 *We allow the property formula to express meta-description of atomic propositions with variables and their domains. Take the sorting robot as an example, if we have the atomic propositions $carryBox(1)$, $carryBox(2)$, ..., $carryBox(B) \in \Sigma$ to represent the fact that the robot is carrying a specific box, we can express the property "the robot is carrying nothing" as $\forall x(\neg carryBox(x))$, and the domain of x is $\mathrm{dom}(x) = \{1, \ldots, B\}$.*

Property formulas are defined on finite paths. Given an MDP $\mathcal{M}$, a path π and a formula φ, we use $\pi \models_{\mathcal{M}} \varphi$ (or $\pi \models \varphi$ if $\mathcal{M}$ is clear from the context) to represent that the path π satisfies the property φ. The formal semantics of property formulas is defined inductively as follows:

$\pi \models p$	if $p \in L(\mathit{first}(\pi))$
$\pi \models \neg\varphi$	if $\pi \not\models \varphi$
$\pi \models \varphi \wedge \psi$	if $\pi \models \varphi$ and $\pi \models \psi$
$\pi \models \mathbf{final}(\varphi)$	if $\mathit{last}(\pi) \models \varphi$
$\pi \models \mathbf{X}(\varphi)$	if $\mathit{suffix}(\pi, 1) \models \varphi$
$\pi \models \mathbf{U}(\varphi, \psi)$	if $\pi \models \psi$ or $\pi \models \varphi \wedge \mathbf{X}(\mathbf{U}(\varphi, \psi))$
$\pi \models \mathbf{occ}(a)$	if $\mathit{action}(\pi, 1) = a$
$\pi \models \forall x(\varphi)$	if $\pi \models \bigwedge_{v \in \mathrm{dom}(x)} \varphi[x \to v]$

where $\mathrm{dom}(x)$ is the domain of x and $\varphi[x \to v]$ represents the formula where all free occurrences of x in φ have been replaced by $v \in \mathrm{dom}(x)$.

The formula $\mathbf{occ}(a)$ holds on π if the *first* action of π is a. Temporal operators $\mathbf{X}(\varphi)$ and $\mathbf{U}(\varphi, \psi)$ are defined in the standard way (Vardi 1995). The formula $\forall x(\varphi)$ holds if φ holds on π for all assignments of x in $\mathrm{dom}(x)$, where x is a variable appearing in φ.

Example 1 *Requiring that the robot never drops a box in the same position it has picked it up can be expressed as $\forall b(\forall z(\mathbf{G}(\mathbf{occ}(\mathtt{p}(b, z)) \Rightarrow \neg\mathbf{F}(\mathbf{occ}(\mathtt{d}(b, z))))))$. The LTL formula $\forall p(\mathbf{G}(robotAt(p) \wedge \mathbf{occ}(\mathtt{l}) \Rightarrow \mathbf{X}(\neg robotAt(p))))$ means that the robot never goes to its starting area while doing a quick movement.*

Along with the definition of property formulas, we denote by $\mathcal{P}_J(\varphi)$ the *probabilistic property*, where $J \subseteq [0, 1]$ is a closed interval. Given an MDP $\mathcal{M}$, $s \in S$, and a policy χ, $s \models^{\chi}_{\mathcal{M}} \mathcal{P}_J(\varphi)$ ($s \models \mathcal{P}_J(\varphi)$ for short if χ and $\mathcal{M}$ are clear from the context) if $\mu_{\chi,s}(\{\, \pi \in \mathit{Paths}^*_{\mathcal{M}} \mid \pi \models \varphi \,\}) \in J$.

For defining the preference formula, we assume a total order $\ll$ on the subset of probabilistic properties. For two probabilistic properties ψ_1 and ψ_2, we say that ψ_1 is *more preferred* than ψ_2 if and only if $\psi_1 \ll \psi_2$. We now define preference formula using $\mathcal{P}_J(\varphi)$ and $\ll$.

Definition 3 *A* preference formula Φ_P *is of the form $\psi_1 \ll \psi_2 \ll \cdots \ll \psi_k$, where each ψ_i ($0 < i \leq k$) is a probabilistic property, $\psi_k = \mathcal{P}_{[1,1]}(\mathbf{True})$, and for each $m, n \leq k$, $m < n$ implies $\psi_m \ll \psi_n$.*

Definition 3 assigns the order of probabilistic properties that corresponds to the appearance order of them. Earlier a formula appears, the more preferred it is. The formula $\mathcal{P}_{[1,1]}(\mathbf{True})$ means no preference and it is the least preferred one. With this notion we introduce the preference planning problem for the probabilistic setting:

Definition 4 *A* Probabilistic Preference Planning Problem *(P4) is a tuple $(\mathcal{M}, \Phi_G, \Phi_P)$ where $\mathcal{M}$ is an MDP, Φ_G is a* goal formula *which is a probabilistic property of the form $\Phi_G = \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$ where γ is a property formula, and $\Phi_P = \psi_1 \ll \psi_2 \ll \cdots \ll \psi_k$ is a preference formula.*

A solution *to a P4 problem is a policy χ such that $\bar{s} \models^{\chi}_{\mathcal{M}} \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$. An* optimal solution *to a P4 is a solution χ^* such that there exists $i \in \{1, \ldots, k\}$ such that $\bar{s} \models^{\chi^*}_{\mathcal{M}} \psi_i$ and for each $0 < j < i$ and each solution χ, $\bar{s} \not\models^{\chi}_{\mathcal{M}} \psi_j$.*

Since we assume that $\psi_k = \mathcal{P}_{[1,1]}(\mathbf{True})$, we have that an optimal solution for $(\mathcal{M}, \Phi_G, \Phi_P)$ exists whenever there is a policy χ such that $\bar{s} \models^{\chi}_{\mathcal{M}} \mathcal{P}_{J_g}(\mathbf{final}(\gamma))$. We remark that different optimal solutions may exist.

We can extend P4 to General Preference Formulas (Bienvenu, Fritz, and McIlraith 2011) by decorating preference formulas with values taken from a totally ordered set $(V, \trianglelefteq)$ and by defining $\ll$ according to $\trianglelefteq$.

Planning Algorithm for P4

In this section we introduce a planner P4Solver to solve P4. To compute the optimal solution to a P4 problem, we start with the most preferred formula ψ_1. If we find a solution, then it is an optimal solution to this P4. If we fail to reach a solution, we continue with the next preferred formula, and after we have enumerated all the property formulas, we conclude that P4 has no solution. As said before, this happens only when the goal formula cannot be satisfied. The scheme is shown in Algorithm 1.

Algorithm 1 P4Solver

Input: MDP model $\mathcal{M}$, preference formula Φ_P, goal formula Φ_G
Output: An optimal policy χ^*

```
Parse Φ_P to the form Φ_P = ψ_1 ≪ ψ_2 ≪ ··· ≪ ψ_k
χ* = ∅
for i = 1 to k do
    χ* = PolFinder(ψ_i, Φ_G, M)
    if χ* ≠ ∅ then
        return (χ*, i)
return unsatisfiable
```

The procedure $\mathit{PolFinder}$ is the core of the algorithm. It translates P4 to a quadratic programming problem based on the concept of progression of property formula.

Definition 5 *Given a property formula φ, a state s, and an action a, the* progression $\rho^s_a(\varphi)$ *of φ from s through a is defined as follows.*

$$\rho^s_a(p) = \begin{cases} \mathbf{True} & \text{if } s \models p \\ \mathbf{False} & \text{otherwise} \end{cases}$$

$$\rho^s_a(\neg\varphi) = \neg\rho^s_a(\varphi)$$

$$\rho^s_a(\varphi_1 \vee \varphi_2) = \rho^s_a(\varphi_1) \vee \rho^s_a(\varphi_2)$$

$$\rho_a^s(\mathbf{final}(\varphi)) = \begin{cases} \mathbf{final}(\varphi) & \text{if } a \neq \bot \\ \mathbf{True} & \text{if } a = \bot \text{ and } s \models \rho_\bot^s(\varphi) \\ \mathbf{False} & \text{otherwise} \end{cases}$$

$$\rho_a^s(\mathbf{X}(\varphi)) = \begin{cases} \varphi & \text{if } a \neq \bot \\ \mathbf{False} & \text{otherwise} \end{cases}$$

$$\rho_a^s(\mathbf{U}(\varphi_1, \varphi_2)) = \begin{cases} \rho_a^s(\varphi_2) \vee (\rho_a^s(\varphi_1) \wedge \mathbf{U}(\varphi_1, \varphi_2)) & \\ & \text{if } a \neq \bot \\ \rho_\bot^s(\varphi_2) & \text{otherwise} \end{cases}$$

$$\rho_a^s(\mathbf{occ}(a')) = \begin{cases} \mathbf{True} & \text{if } a' = a \\ \mathbf{False} & \text{otherwise} \end{cases}$$

$$\rho_a^s(\forall x(\varphi)) = \bigwedge_{v \in \mathrm{dom}(x)} \rho_a^s(\varphi[x \to v])$$

Note that we can simplify the resulting progressed formula with the basic rules of Boolean logic: if $\rho_a^s(\varphi_2)$ reduces to $\mathbf{True}$, then we can simplify $\rho_a^s(\mathbf{U}(\varphi_1, \varphi_2)) = \rho_a^s(\varphi_2) \vee (\rho_a^s(\varphi_1) \wedge \mathbf{U}(\varphi_1, \varphi_2))$ to $\mathbf{True}$ as well.

Intuitively, the progression of a formula φ from s through a is a new property formula that says which parts of φ are still to be satisfied after reaching s and performing a. When we are in a state s along a path π, the sequence of progressed formulas encodes what it has been already satisfied while reaching s and what the remainder of π has to satisfy from s, so that $\pi \models \varphi$. Once we reach the end of the path, the last progressed formula can be definitely decided, and this is exactly the satisfaction of π of the original formula φ.

Theorem 1 *Let π be a path, then*

$$\pi \models \varphi \Leftrightarrow \begin{cases} \pi \models \rho_\bot^{first(\pi)}(\varphi) & \text{if } |\pi| = 0, \\ suffix(\pi, 1) \models \rho_{action(\pi,1)}^{first(\pi)}(\varphi) & \text{otherwise.} \end{cases}$$

The proof is a minor adaptation of the one in (Bienvenu, Fritz, and McIlraith 2011). Note that the value of $\rho_a^s(\varphi)$ may remain unknown until we reach the last state of the path.

Example 2 *Take the rail robot example and assume that we want to check whether the robot moves quickly at least once on the path $\pi = s_0\mathtt{p}(1,0)s_1\mathtt{n}s_2\mathtt{l}s_3$, represented by the property formula $\mathbf{F}(\mathbf{occ}(\mathtt{l}))$. At first we cannot decide whether $\pi \models \mathbf{F}(\mathbf{occ}(\mathtt{l}))$ since $\mathbf{F}(\mathbf{occ}(\mathtt{l})) \neq \mathbf{True}$. So we use the information of s_0 and action $\mathtt{p}(1,0)$ and get $\rho_{\mathtt{p}(1,0)}^{s_0}(\mathbf{F}(\mathbf{occ}(\mathtt{l}))) = \rho_{\mathtt{p}(1,0)}^{s_0}(\mathbf{occ}(\mathtt{l})) \vee \mathbf{F}(\mathbf{occ}(\mathtt{l})) = \mathbf{False} \vee \mathbf{F}(\mathbf{occ}(\mathtt{l})) = \mathbf{F}(\mathbf{occ}(\mathtt{l}))$, since $\mathtt{p}(1,0) \neq \mathtt{l}$. Now we need to check whether $suffix(\pi, 1) \models \rho_{\mathtt{p}_{1,0}}^{s_0}(\mathbf{F}(\mathbf{occ}(\mathtt{l})))$, i.e., $s_1\mathtt{n}s_2\mathtt{l}s_3 \models \mathbf{F}(\mathbf{occ}(\mathtt{l}))$. In a similar way we can iteratively attempt to match all actions of π to $\mathtt{l}$, and finally get $\pi \models \mathbf{F}(\mathbf{occ}(\mathtt{l}))$ since the last second action is $\mathtt{l}$ (note that the last action is $\bot$ without explicit notion on a finite path).*

We denote by p_{sa} the probability to choose action a from state s, and by μ_φ^s the probability of state s to satisfy a property formula φ. Formally, $\mu_{\mathbf{True}}^s = 1$, $\mu_{\mathbf{False}}^s = 0$, and

$$\mu_\varphi^s = \sum_{a \in Act(s)} p_{sa} \cdot \sum_{s' \in S} \mathrm{P}(s, a)(s') \cdot \mu_{\rho_a^s(\varphi)}^{s'}. \qquad (1)$$

Note that when $a = \bot$, $p_{sa} \cdot \sum_{s' \in S} \mathrm{P}(s, a)(s') \cdot \mu_{\rho_a^s(\varphi)}^{s'}$ reduces to $p_{s\bot} \cdot \mu_{\rho_\bot^s(\varphi)}^s$ since $\mathrm{P}(s, \bot) = \delta_s$.

When solving a P4, we construct the equations iteratively for each μ_φ^s with the form above until the progression reaches $\mathbf{True}$ or $\mathbf{False}$ so we can definitely know the probability of the formula to be 1 or 0, respectively. If there exists a solution to the corresponding set of formulas to the P4, then the values p_{sa} induce the solution of P4. So P4 can be converted into a set of non-linear equations (actually, quadratic equations) whose solution, if any, is the solution for P4. This conversion is managed by $PolFinder$.

$PolFinder$, from the initial state of $\mathcal{M}$, explores the state space of $\mathcal{M}$ and for each newly reached pair of state and progressed formula, generates the corresponding instance of the Equation 1. In the bounded version of $PolFinder$, we also consider the number of past actions in the definition of Equation 1. Obviously, the correctness of P4Solver relies on the termination of $PolFinder$. Since we consider only finite MDPs, $PolFinder$ terminates for sure. As complexity, for an MDP $\mathcal{M}$ and a property formula φ, the time complexity of $PolFinder$ is linear in the size of $\mathcal{M}$ and double exponential in the size of φ.

Theorem 2 *For a P4 $(\mathcal{M}, \Phi_G, \Phi_P)$, it has an optimal policy if and only if P4Solver returns a policy χ^* and the index i of the satisfied preference formula.*

Implementation

We implemented the P4Solver algorithm in Scala and delegated to Z3 (de Moura and Bjørner 2008) the evaluation of the generated quadratic programming problem. We represent each state of the MDP by the atomic propositions it is labelled with; for instance, the state of the MDP depicted in Figure 1 is represented by the atomic propositions $robotAt(1)$, $carryBox(1)$, $boxAt(2,6)$, $boxAt(3,21)$, and $mode(\mathtt{c})$, given that the robot is in control mode.

An action a is symbolically encoded as a possibly empty list of variables, a precondition, and the effects. The variables bind the condition and the effects to the state s enabling the action. The precondition is simply a property formula that has to be satisfied by the state s (or, more precisely, by the path $\pi = s$) in order to have a enabled in s. The effect of a is a probability distribution mapping each target state occurring with non-zero probability to the corresponding probability value. The target states are encoded by a pair (R, A) of sets of atomic propositions: those not holding anymore (R) and those now holding (A). As a concrete example, the action $\mathtt{d}(b, p)$ models the drop of box b at position p: the precondition is $mode(\mathtt{a}) \wedge robotAt(p) \wedge carryBox(b) \wedge \neg\exists b'(boxAt(b', p))$; the effects are $(\{carryBox(b), mode(\mathtt{a})\}, \{mode(\mathtt{c}), boxAt(b, p)\})$ with probability 0.95 and $(\{mode(\mathtt{a})\}, \{mode(\mathtt{c})\})$ with probability 0.05.

The preference and goal formulas are encoded according to the grammars in Definitions 2 and 3, and the probability intervals by the corresponding bounds.

Each state, action, and formula is uniquely identified by a number that is used to generate the variables for the SMT solver: for instance, the scheduler's choice of action 4 in state 37 is represented by the variable `sched_s37_a4` while `sched_s37_stop` stands for the probability of

N	B	Property	Nodes	t_g	Vars	t_e	Res
5	2	1	279	0.06	333	0.06	sat
5	2	2	329	0.05	348	0.08	sat
5	2	3	1332	0.10	1764	0.73	sat
5	2	4	1690	0.09	1870	0.83	sat
6	2	1	376	0.06	453	0.11	sat
6	2	2	436	0.05	471	0.11	sat
6	2	3	2173	0.15	2884	1.90	sat
6	2	4	2707	0.13	3042	1.89	sat
7	2	1	487	0.07	591	0.15	sat
7	2	2	557	0.06	612	0.15	sat
7	2	3	3304	0.28	4368	10.21	sat
7	2	4	4048	0.20	4606	5.08	sat

Table 1: Performance of P4Solver: Rail Robot

Type	Bound	Property	Nodes	t_g	Vars	t_e	Res
np	2	1	28	0.01	41	0.01	sat
np	4	2	533	0.08	268	1.58	sat
np	4	3	533	0.09	270	1.53	sat
np	4	4	533	0.06	265	1.55	sat
np	4	5	533	0.08	290	0.06	sat
np	∞	1	265	0.04	458	0.06	sat
np	∞	2	529	0.08	522	0.09	sat
np	∞	3	—time-out—				
np	∞	4	534	0.06	523	0.08	sat
np	∞	5	265	0.05	458	0.07	sat
pr	2	1	87	0.01	41	0.01	unsat
pr	4	2	6058	0.19	271	—time-out—	
pr	4	3	6058	0.21	284	—time-out—	
pr	4	4	6058	0.15	272	—time-out—	
pr	4	5	6058	0.08	290	0.09	sat
pr	∞	1	497	0.05	458	0.06	sat
pr	∞	2	993	0.09	522	0.08	sat
pr	∞	3	—time-out—				
pr	∞	4	1002	0.08	523	0.08	sat
pr	∞	5	497	0.04	458	0.06	sat

Table 2: Performance of P4Solver: Dinner Domain

stopping in state 37. We encode the SMT problem by the SMT-LIB format (Barrett, Stump, and Tinelli 2010), thus we can replace Z3 with any other solver supporting such format. Moreover, it is easy to change the output format to support solvers like Redlog and Mathematica.

We have run our prototype on a single core of a laptop running openSUSE 13.1 on a Intel® Core™ i5-4200M CPU and 8Gb of RAM. The results for the rail robot and an adaptation of the dinner domain (Bienvenu, Fritz, and McIlraith 2011) are shown in Tables 1 and 2, respectively.

For the robot example, we consider $N = 5, 6, 7$ positions and $B = 2$ boxes and four preference formulas that require to eventually pickup or drop the boxes, placed in different areas to be sort according to the goal positions (each box i in position i). As preferences, we have $\mathbf{F}(\exists B(\exists P(\mathbf{occ}(\mathtt{p}(B, P)))))$ as preference 1 and 3 (with different initial states), $\mathbf{F}(\exists B(\exists P(\mathbf{occ}(\mathtt{d}(B, P)))))$ as preference 2, and $\mathbf{F}(\exists P(\mathbf{occ}(\mathtt{d}(1, P))))$ as preference 4. The initial state relative to preference 4 has $boxAt(1, 1)$, so the preference requires to drop the box even if this action is not needed to reach the goal (asking for $boxAt(1, 1)$ as well). The goal has to be satisfied with probability in $[0.75, 1]$ and the preference with probability in $[1, 1]$. The remaining columns in the table show the number of nodes (state, formula) of the MDP we generated, the time t_g spent by Scala to generate the SMT problem, the number of variables in the problem, the time t_e used by Z3 to solve the problem, and whether the problem is satisfiable.

For the dinner domain example, we consider two types of a simplified version, where we have removed some of the foods available for dinner: "np" for the non-probabilistic version and "pr" for the probabilistic one. In the former all actions reach a single state, in the latter an action may have no effect with non-zero probability. In the column "Bound" we show the bound on the length of the paths we used to construct the model: in some cases, the resulting model is so large to take too much time to be generated (30 seconds) or solved (30 minutes). We marked these cases with "—time-out—". We are at home with the kitchen clean as initial state, and our goal state is to be at home, sated, and with the kitchen clean. As preferences, let ε be the formula $\exists L(\mathbf{F}(\mathbf{occ}(eat)(pizza, L))$ and ι be $\mathbf{occ}(drive(home, italianRest))$. ε is preference 1, $\varepsilon \wedge \mathbf{F}(\iota)$ is preference 2, $\mathbf{F}(\varepsilon \wedge \iota)$ is preference 3, $\iota \wedge \varepsilon$ is preference 4, and $\forall L(\mathbf{occ}(drive(home, L)) \Rightarrow \mathbf{X}(at(italianRest)))$ is preference 5. Probabilities for goal and preferences are again $[0.75, 1]$ and $[1, 1]$, respectively.

It is interesting to observe the "unsat" result for property 1 of the probabilistic bounded version: this is caused by the fact that the bound prevents to satisfy with enough probability the preference, while this always happens for the unbounded version since it is enough to try repeatedly the action until we reach the desired state.

Related Work and Discussion

Related Work. The Planning Domain Definition Language (PDDL) was first proposed in (McDermott et al. 2000) and evolves with the International Planning Competition (IPC) to PDDL3 (Gerevini and Long 2005) and PPDDL (Younes et al. 2005; Younes and Littman 2004). PDDL3 is an extension of PDDL that includes temporally extended preferences described by a subset of LTL formulas, but it does not consider probabilistic aspects, so it is not feasible for modelling MDPs. PDDL3 is a quantitative preference language evaluating plans by optimizing an associated numeric objective function, while our preference language expresses the preference of the plans by defining an order on the probabilistic properties the plan has to satisfy. This order can be easily extended to quantitative preferences by associating each probabilistic property with a unique element taken from a totally ordered set, as in (Bienvenu, Fritz, and McIlraith 2011). Another difference between PDDL3 and our P4 language is that PDDL3 does not allow us to express preferences on the occurrence of actions, only a subset of temporal operators is available, and the nesting of LTL formulas is limited, while our P4 supports all them. On the other hand, PDDL3 has some features that are missing in P4: preference formulas in PDDL3 allow us to make a comparison between plans in the degree they violate the preference,

by means of the value optimizing the associated numeric objective function. This quantitative information can be convenient in defining some user's preferences.

PPDDL extends PDDL with features for modelling probabilistic effects and rewards, so it can be used to model MDPs. Being based on PDDL2.1 (Fox and Long 2003), the above considerations about PDDL3 and P4 apply also to PPDDL since the differences between PDDL2.1 and PDDL3 are only on derived predicates and constraints.

PP and LPP are noteworthy preference languages related to our work. PP was proposed in (Son and Pontelli 2004) with an answer set programming implementation. LPP was proposed in (Bienvenu, Fritz, and McIlraith 2011) together with a bounded best-first search planner PPLAN. The property formula in this paper and Trajectory Property Formula (TPF) of LPP inherit all features of Basic Desire Formulas (BDF) of PP: considering the preferences over states and actions with static properties, and over the trajectories with temporal properties. Both also extend PP to support first order preferences. LPP inherits General Preference of PP and extends structure of preferences with Aggregated Preference Formulas (APF) to support more complex comparison and comprehension of different preferences.

Several planners have been proposed for solving the preference planning problem in non-probabilistic systems. HPLAN-P (Baier, Bacchus, and McIlraith 2009) is a bounded planner that computes optimal plans performing at most a given number of actions. It can deal with temporally extended preferences (TEPs) by compiling them to parametrized finite automata by using the TLPlan's ability to deal with numerical functions and quantifications (Bacchus and Kabanza 2000). HPLAN-P is equipped with heuristics to prune the search space during the computation. SGPLAN$_5$ (Hsu et al. 2007) is another search-based planner with the ability to plan with TEPs. It uses a constraint partitioning approach to split the problem of planning in sub-problems, by taking advantage of local optimal values. This permits to reduce in several cases the makespan of the plan. MIPS-XXL (Edelkamp, Jabbar, and Nazih 2006) and MIPS-BDD (Edelkamp 2006) are two well performing planners that compile the given TEPs into Büchi automata that are then synchronized with the model under consideration. MIPS-XXL targets very large systems and it combines internal weighted best-first search and external cost-optimal breadth-first search to control the pace of planning. MIPS-BDD does symbolic bread-first search on state space encoded to binary decision diagram (BDD). We refer to (Baier and McIlraith 2009) and (Bienvenu, Fritz, and McIlraith 2011) for a more comprehensive comparison.

For MDPs, there are some related works on planning. In (Boutilier, Dean, and Hanks 1999), the authors introduce many MDP-related methods and show how to unify some planning problems in such framework in AI. It presents a quantitative preference based on reward structure on MDPs and shows how to compute a plan with the means in Decision-Theoretic Planning, such as value iteration and policy iteration. To specify uncontrollable behaviours of systems, it introduces exogenous events in the MDP. As mentioned above, the preference formula defined in our work extends probabilistic LTL. It is intuitively possible to take great advantage of the algorithms in decision-theoretic planning. Another similar work in this line is (Younes 2003), which extends PDDL for modelling stochastic decision processes by using PCTL and CSL for specifying Probabilistic Temporally Extended Goals (PTEG). It encodes many stochastic effects in PDDL and discusses the expressiveness of those effects in different kinds of MDP models. The essential difference of the above work to our framework is on the preference language. Our preference language is qualitative which could specify the properties along the sequences of behaviors, whereas the quantitative preference in (Boutilier, Dean, and Hanks 1999) can not capture those properties. And even though in (Younes 2003) PTEG is able to deal with temporal properties on real time, it lacks flexibility to express various features and can not handle preferences over actions.

Discussion. There are some other possible approaches to P4 based on model checking methods. One is to use the current planners on deterministic models: planners like MIPS-XXL, MIPS-BDD and HPLAN-P first translate the LTL formula to an automaton and then find the plan in the product. Whereas it works nicely for models without probability, this approach cannot be adapted in a straightforward way to MDPs. The main bottleneck is the fact that a "deterministic" automaton may be needed. (In the model checking community, this classical automaton approach for MDPs always involves determination, see for details (Baier and Katoen 2008).) Unfortunately, a deterministic automaton for an LTL formula is double exponential, as shown in (Courcoubetis and Yannakakis 1995). In addition, this approach faces some other challenges, such as (i) the preference formula has to be satisfied before the goal formula, (ii) the matching of the desired probability intervals for both preference and goal formulas under the same plan, and (iii) the plan is not required to maximize/minimize the probability of the formulas. Besides, our method to build the equations is an on-the-fly construction of the product of LTL and MDP, wherein we avoid explicit representation of the whole product.

Another approach tries to solve P4 with current probabilistic planners capable to make plans with soft preferences. These planners support the Relational Dynamical Influence Diagram Language (RDDL) (Sanner 2010) used for the last ICAPS'11 and ICAPS'14 International Probabilistic Planner Competitions. RDDL is a uniform language to model factored MDPs and corresponding reward structures which could be regarded as quantitative preferences. It is influenced by PPDDL and PDDL3 and it is designed to be a language that is simple and uniform where the expressive power comes from the composition of simple constructs. In particular, it aims at representing problems that are difficult to model in PPDDL and PDDL. In its current specification, RDDL does not support temporal state/action goal or preferences. A direct encoding of our preference formulas in the reward structure is not obvious which we leave as future work.

Conclusion

In this paper we have proposed a framework for probabilistic preference-based planning problem on MDPs. Our language can express rich properties, and we have designed a planning algorithm. Our algorithm is via reduction to a quadratic programming problem. We presented two case studies.

As future work we plan to enhance the algorithm for improving the efficiency. We would also like to investigate the possibility of combining model checking methods with P4. Finding k-optimal path-based policy with approximation is also a promising way to reduce the complexity.

Acknowledgments

Zhikun She is supported by the National Natural Science Foundation of China under grants 11422111 and 11371047. Lijun Zhang (Corresponding Author) is supported by the National Natural Science Foundation of China under grants 61472473, 61428208, and 61361136002, and by the CAS/SAFEA International Partnership Program for Creative Research Teams.

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 16:123–191.

Baier, C., and Katoen, J.-P. 2008. *Principles of Model Checking*. The M.I.T. Press.

Baier, J. A., and McIlraith, S. A. 2009. Planning with preferences. *Artificial Intelligence* 173:593–618.

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence* 173:593–618.

Barrett, C.; Stump, A.; and Tinelli, C. 2010. The SMT-LIB standard: Version 2.0. In *SMT*.

Bellman, R. 1957. A Markovian decision process. *Indiana University Mathematics Journal* 6:679–684.

Bienvenu, M.; Fritz, C.; and McIlraith, S. A. 2011. Specifying and computing preferred plans. *Artificial Intelligence* 175:1308–1345.

Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H. H.; and Poole, D. 2004. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research* 21:135–191.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.

Coste-Marquis, S.; Lang, J.; Liberatore, P.; and Marquis, P. 2004. Expressive power and succinctness of propositional languages for preference representation. In *Proceedings of the 9th International Conference on Knowledge Representation and Reasoning*, 203–121.

Courcoubetis, C., and Yannakakis, M. 1995. The complexity of probabilistic verification. *Journal of the ACM* 42(4):857–907.

de Moura, L. M., and Bjørner, N. 2008. Z3: An efficient SMT solver. In *TACAS*, volume 4963 of *LNCS*, 337–340.

Delgrande, J. P.; Schaub, T.; and Tompits, H. 2004. Domain-specific preferences for causual reasoning and planning. In *Proceedings of the 9th International Conference on Knowledge Representation and Reasoning*, 673–682.

Edelkamp, S.; Jabbar, S.; and Nazih, M. 2006. Large-scale optimal PDDL3 planning with MIPS-XXL. In *5th International Planning Competition Booklet (IPC-2006)*.

Edelkamp, S. 2006. Optimal symbolic PDDL3 planning with MIPS-BDD. In *5th International Planning Competition Booklet (IPC-2006)*.

Fox, M., and Long, D. 2003. PDDL 2.1: An extension to PDDL for expressing temporal planning problems. *Journal of Artificial Intelligence Research* 20:61–124.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3: The language of the fifth international planning competition. Technical report, University of Brescia.

Hansson, H., and Jonsson, B. 1994. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6(5):512–535.

Hsu, C.; Wah, B. W.; Huang, R.; and Chen, Y. 2007. Constraint partitioning for solving planning problems with trajectory constraints and goal preferences. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 1924–1929.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 2000. PDDL—the planning domain definition language. Technical report, Yale Center for Computational Vision and Control.

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

Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2009. HTN planning with preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 1790–1797.

Son, T. C., and Pontelli, E. 2004. Planning with preferences using logic programming. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, 247–260.

Tu, P.; Son, T.; and Pontelli, E. 2007. CPP: A constraint logic programming based planner with preferences. In *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, 290–296.

Vardi, M. Y. 1995. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop)*, volume 1043 of *LNCS*, 238–266.

Younes, H. L. S., and Littman, M. L. 2004. PPDDL 1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University.

Younes, H. L. S.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research* 24:851–887.

Younes, H. 2003. Extending PDDL to model stochastic decision processes. In *Proceedings of the ICAPS-03 Workshop on PDDL, Trento, Italy*, 95–103.