Strong Probabilistic Planning

Silvio do Lago Pereira, Leliane Nunes de Barros and Fábio Gagliardi Cozman
Institute of Mathematics and Statistics - University of São Paulo
Rua do Matão, 1010 - São Paulo, Brazil

Abstract

We consider the problem of synthesizing policies, in domains where actions have *probabilistic* effects, that are optimal in the *expected-case* among the optimal *worst-case strong* policies. Thus we combine features from nondeterministic and probabilistic planning in a single framework. We present an algorithm that combines dynamic programming and model checking techniques to find plans satisfying the problem requirements: the strong preimage computation from model checking is used to avoid actions that lead to cycles or dead ends, reducing the model to a Markov Decision Process where all possible policies are strong and worst-case optimal (*i.e.*, successful and minimum length with probability 1). We show that backward induction can then be used to select a policy in this reduced model. The resulting algorithm is presented in two versions (enumerative and symbolic); we show that the latter version allows planning with extended reachability goals.

Introduction

Planning under uncertainty has many facets; in particular, actions can range from completely nondeterministic to probabilistic. The latter situation is typically modelled with Markov Decision Processes (MDPs) (Boutilier, Dean, & Hanks 1999; Bonet & Geffner 2003). However, it is not always possible to model a real problem under the assumptions of additive costs and Markovian transitions demanded by MDPs (Dolgov & Durfee 2005). In this paper we look into situations that go beyond the usual theory of MDPs but that still lead to efficient solution algorithms.

Our proposal is inspired on assumptions adopted in *nondeterministic planning*; that is, planning where actions have uncertain outcomes but are not associated with probabilities (Giunchiglia & Traverso 1999). Plans are then required to offer *strong* guarantees; for example, plans must reach a goal state for sure and through an acyclic path that has minimum length. We embrace these requirements as they seem most appropriate in many real planning problems, independently of whether one has probabilities or not. But suppose that one *does have* probabilities on transitions and actions have uniform costs. We ask, How can one find strong policies that maximize expected rewards? If the reader finds it surprising that a standard MDP may fail to produce such a policy, consider our Example 1.

We use *strong probabilistic planning* to refer to situations where one wishes to produce an optimal worst-case strong policy that minimizes the expected number of steps to goal states. We show that model checking (MC) techniques can be employed to great effect in strong probabilistic planning. We start discussing straightforward reachability goals; *i.e.*, reaching a goal state with certainty while minimizing the number of steps. However, we also present a symbolic version of our planner that is even capable of handling extended reachability goals (Pistore, Bettin, & Traverso 2001; Lago, Pistore, & Traverso 2002). With these goals, we can also impose constraints on states visited during policy execution.

We should note that in principle we are dealing with a *constrained* MDP; that is, an MDP with additional constraints on functionals of the trajectories (Puterman 1994, page 229). There are two difficulties in directly resorting to the theory of constrained MDPs. First, it is not entirely obvious how to model constraints such as "probability zero that a cycle occurs" in a way that leads to efficient search for policies. Second, general constrained MDPs fail the dynamic programming principle, and thus are not solved by backward induction; rather, constrained MDPs may depend on randomized policies obtained by linear programming (Altman 1999). We show that strong probabilistic planning admits a backward reasoning scheme; thus we find that our model-checking strategy is more illuminating and yields better results than a general approach based on constrained MDPs.

The remainder of this paper is organized as follows. The next section briefly points out the differences between nondeterministic and probabilistic actions. We then present a review of algorithms for MC/MDP-based planning, where we contribute with a unifying perspective that seeks to compare them on a common ground. The insights produced by this analysis steer us to our algorithms for strong probabilistic planning. We describe two algorithms, one enumerative and one symbolic, and prove that they return a policy that is expected-case optimal among the strong policies which are worst-case optimal. We also discuss our implementation and then close the paper.

Copyright © 2007, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Actions: nondeterministic and probabilistic

Uncertainty generally interferes with two important aspects of the planning task: *observability* and *determinism*. Due to uncertainty, the current state may not be precisely known. Moreover, even when the current state is known, after executing certain actions, the next state may not be precisely predicted (Ghallab, Nau, & Traverso 2004). In this work, we will deal only with uncertainty about the predictability of next states. To model this kind of uncertainty, one can consider two entities:

- the *agent*, an entity which plans to achieve its goals; and
- the *nature*, an entity whose intentions are unknown and whose actions can interfere with the agent actions.

Suppose that the agent and the nature are in a game: the agent starts by choosing an action α from the set $\mathcal{A}$ and, depending on this action, the nature chooses its action from the set $\mathcal{N}(\alpha)$. For example, consider the scenario depicted in Figure 1: when the agent chooses action *throw-coin*, the nature should in turn choose its action from the set $\mathcal{N}(\textit{throw-coin}) = \{\textit{turn-tail-up}, \textit{turn-head-up}\}$. Since nature actions can interfere with agent actions, the best choice for the agent depends on what information it has about nature behavior. Clearly, if the agent knows exactly how nature chooses its actions, there is no uncertainty about the next state (*i.e., deterministic planning*). Thus, in planning under uncertainty about the predictability of next states, we always assume that agent has not certainty about nature behavior.

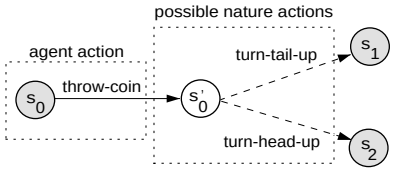

Figure 1: Uncertainty model as a game against nature.

There are two main uncertainty models (LaValle 2006) that are usually considered in planning under uncertainty[1]:

- *nondeterministic*: the agent has no idea about how the nature chooses its action from $\mathcal{N}(\alpha)$;
- *probabilistic*: the agent has observed the nature and gathered statistics about how frequently the actions in $\mathcal{N}(\alpha)$ are chosen by nature.

If a nondeterministic model is considered then it is only known that the nature will make a choice from $\mathcal{N}(\alpha)$. In this case, a pessimist strategy for agent action selection should be appropriate. On the other hand, if a probabilistic model is considered then, for each agent action $\alpha \in \mathcal{A}$, a probability distribution over $\mathcal{N}(\alpha)$ should be specified as part of the model. In this case, the strategy for agent action selection should be optimal with respect to this information.

[1] In a nondeterministic model, uncertainty is not measured and is also called *Knightian uncertainty*; while in a probabilistic model, uncertainty is measurable and is also called *risk* (Trevizan, Cozman, & de Barros 2007).

Approaches for planning under uncertainty

In this section we offer a unified perspective on planning under uncertainty as we present the necessary background for the latter sections. We would like to stress that a combined presentation of nondeterministic and probabilistic planning has rarely appeared in the literature; thus we devote considerable space to this discussion.

The approaches of interest here are:

- *Model checking* (MC), used for planning under nondeterministic uncertainty models; and
- *Markovian decision processes* (MDP), used for planning under probabilistic uncertainty models.

Both approaches are very attractive, but for different reasons: the main advantage of MC is the *effectiveness* of the solutions; while the main advantage of MDP is the *efficiency* of the solutions. Nondeterministic planning based on MC aims the synthesis of *effective* plans, that are guaranteed to achieve a goal state regardless of nature behavior. On the other hand, probabilistic planning based on MDP leads to *efficient* plans, that yield optimal expected-case performance with respect to the information about nature behavior, if executed numerous times for the same problem.

Nondeterministic planning based on MC

The basic idea underlying nondeterministic planning based on MC is to solve problems model-theoretically (Giunchiglia & Traverso 1999).

Definition 1. *A nondeterministic planning domain is a tuple $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T} \rangle$, where:*

- *$\mathcal{S}$ is a finite nonempty set of states;*
- *$\mathcal{A}$ is a finite nonempty set of actions;*
- *$\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \{0, 1\}$ is a state transition function.* ◆

We assume that $\mathcal{A}$ contains a *trivial* action τ such that, for all $\sigma \in \mathcal{S}$, we have that $\mathcal{T}(\sigma, \tau, \sigma') = 1$ if and only if $\sigma = \sigma'$. So, when agent executes action τ in a certain state, it always remains in this same state. Intuitively, action τ represents the fact that agent can choose do nothing in any state. Given a state σ and an action α, the set of *α-successors* of σ, denoted by $\mathcal{T}(\sigma, \alpha)$, is the set $\{\sigma' : \mathcal{T}(\sigma, \alpha, \sigma') = 1\}$.

A *policy* in a nondeterministic planning domain $\mathcal{D}$ is a *partial* function $\pi : \mathcal{S} \mapsto \mathcal{A}$, that maps states to actions. A *nondeterministic policy* is a partial function $\pi : \mathcal{S} \mapsto 2^{\mathcal{A}} \setminus \emptyset$, that maps states to sets of actions. The set $\mathcal{S}_\pi$ of states reached by a policy π is $\{\sigma : (\sigma, \alpha) \in \pi\} \cup \{\sigma' : (\sigma, \alpha) \in \pi \text{ and } \sigma' \in \mathcal{T}(\sigma, \alpha)\}$. Given a policy π, the corresponding *execution structure* $\mathcal{D}_\pi$ is the subsystem of $\mathcal{D}$ that has $\mathcal{S}_\pi$ as set of states and contains all transitions induced by the actions in policy π.

Definition 2. *A nondeterministic planning problem is a tuple $\mathcal{P} = \langle \mathcal{D}, s_0, \mathcal{G} \rangle$, where:*

- *$\mathcal{D}$ is a nondeterministic planning domain;*
- *$s_0 \in \mathcal{S}$ is an initial state;*
- *$\mathcal{G} \subseteq \mathcal{S}$ is a set of goal states.* ◆

Given a nondeterministic planning problem, we distinguish among three kinds of solutions:

- a *weak solution* is a policy that may achieve the goal, but due to nondeterminism, is not guaranteed to do so. A policy π is a weak solution if some finite path in $\mathcal{D}_\pi$, starting from s_0, reaches a state in $\mathcal{G}$ (Cimatti *et al.* 1997).
- a *strong solution* is a policy that always achieves the goal, in spite of nondeterminism. A policy π is a strong solution if the subsystem $\mathcal{D}_\pi$ is acyclic and all paths starting from s_0 reach a state in $\mathcal{G}$ (Cimatti, Roveri, & Traverso 1998).
- a *strong-cyclic solution* is a policy that always achieves the goal, under the fairness assumption that execution will eventually exit from cycles. A policy π is a strong-cyclic solution if all paths in $\mathcal{D}_\pi$ starting from s_0 reach a state in $\mathcal{G}$ (Daniele, Traverso, & Vardi 1999).

The strong nondeterministic planning algorithm. The strong nondeterministic planning algorithm, adapted from the work of Cimatti, Roveri, & Traverso (1998), allows us to synthesize plans that are guaranteed to reach a goal state, regardless of nondeterminism. This algorithm is correct, complete and returns an *optimal worst-case* policy π, in the sense that the worst path in the execution structure $\mathcal{D}_\pi$ has minimal length.

```
StrongNondeterministicPlanning(P)
1 π ← ∅
2 π' ← {(σ,τ) : σ ∈ G}
3 while π ≠ π' do
4   S ← StatesCoveredBy(π')
5   if s0 ∈ S then return π'
6   π ← π'
7   π' ← π' ∪ Prune(StrongPreimage(S), S)
8 return failure
```

The basic planning step in this algorithm is performed by function $\textsc{StrongPreimage}(S)$, which returns the set of pairs (σ, α) such that execution of action α in state σ *necessarily* leads to a state in S. This function is defined as:

$$\textsc{StrongPreimage}(S) = \{(\sigma, \alpha) : \varnothing \neq \mathcal{T}(\sigma, \alpha) \subseteq S\}$$

By iterating the strong preimage function, from the set of goal states $\mathcal{G}$, the algorithm builds up a finite backward search tree (Figure 2). Since the set of states is finite and this function is monotonic with respect to set inclusion, *i.e.*, $\mathcal{G} \subseteq \textsc{StrongPreimage}^1(\mathcal{G}) \subseteq \textsc{StrongPreimage}^2(\mathcal{G}) \subseteq \cdots \subseteq \textsc{StrongPreimage}^n(\mathcal{G})$, after a finite number of iterations, a fixpoint is obtained.

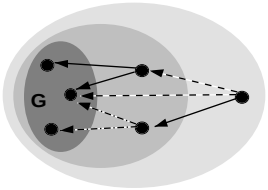

Figure 2: A backward search tree built after three iterations of the strong preimage function.

During this iterative process, the algorithm maps the states in the search tree to actions (or sets of actions) and, therefore, a policy is synthesized as a side effect. At each iteration, the set of states covered by π', the policy under construction, is obtained by the following function:

$$\textsc{StatesCoveredBy}(\pi') = \{\sigma : (\sigma, \alpha) \in \pi'\}$$

If there exists a strong policy to reach a state in $\mathcal{G}$, from the initial state s_0, then in one of the iterations, the condition $s_0 \in \textsc{StatesCoveredBy}(\pi')$ is satisfied and the algorithm returns policy π' as solution to the planning problem.

Finally, to avoid the assignment of new actions to states already covered in previous iterations (*i.e.* to avoid cycles and to guarantee optimal worst-case policies), the algorithm uses the following function:

$$\textsc{Prune}(R, S) = \{(\sigma, \alpha) \in R : \sigma \notin S\}$$

Probabilistic planning based on MDPs

The basic idea underlying probabilistic planning based on MDP is to represent the planning problem as an optmization problem (Boutilier, Dean, & Hanks 1999).

Definition 3. *A probabilistic planning domain is a tuple* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T} \rangle$*, where:*

- $\mathcal{S}$ *is a finite nonempty set of states;*
- $\mathcal{A}$ *is a finite nonempty set of actions;*
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ *is a state transition function.* ◆

Given two states σ, σ' and an action α, the probability of reaching σ' by executing α in σ is $\mathcal{T}(\sigma, \alpha, \sigma')$. Furthermore, for each state $\sigma \in \mathcal{S}$, if there exists α and σ' such that $\mathcal{T}(\sigma, \alpha, \sigma') \neq 0$, then $\sum_{\sigma' \in \mathcal{S}} \mathcal{T}(\sigma, \alpha, \sigma') = 1$. Particularly, for the trivial action τ, we must have:

$$\mathcal{T}(\sigma, \tau, \sigma') = \begin{cases} 0 & \text{iff} \quad \sigma \neq \sigma' \\ 1 & \text{iff} \quad \sigma = \sigma' \end{cases}$$

Given a state σ, the set of *executable actions* in σ, denoted by $\mathcal{A}(\sigma)$, is the set $\{\alpha : \exists \sigma' \in \mathcal{S}$ such that $\mathcal{T}(\sigma, \alpha, \sigma') \neq 0\}$.

A *policy* in a probabilistic planning domain $\mathcal{D}$ is a *total* function $\pi : \mathcal{S} \mapsto \mathcal{A}$, that maps states to actions. Given a policy π, the corresponding *execution structure* $\mathcal{D}_\pi$ is the subsystem of $\mathcal{D}$ that has $\mathcal{S}$ as set of states and contains all transitions induced by the actions in policy π.

Definition 4. *A probabilistic planning problem is a tuple* $\mathcal{P} = \langle \mathcal{D}, \mathcal{G} \rangle$*, where:*

- $\mathcal{D}$ *is a probabilistic planning domain;*
- $\mathcal{G} \subseteq \mathcal{S}$ *is a set of goal states.* ◆

A *reward function* $\mathcal{R} : \mathcal{S} \mapsto \mathbb{R}_+$ is a function that maps states to rewards. Intuitively, when the agent reaches a state σ it receives a reward $\mathcal{R}(\sigma)$. In the case of probabilistic planning for reachability goals, given a set of goal states $\mathcal{G}$, a Boolean reward function can be defined as following:

$$\mathcal{R}(\sigma) = \begin{cases} 0 & \text{iff} \quad \sigma \notin \mathcal{G} \\ 1 & \text{iff} \quad \sigma \in \mathcal{G} \end{cases}$$

A reward is an "incentive" that attracts the agent to goal states. Moreover, to force the agent to prefer shortest paths to goal states, at each executed step, future rewards are discounted by a factor $0 < \gamma < 1$ (The use of such discount factor also guarantees convergence of fixpoint computations (Puterman 1994).). Hence, if the agent reachs a goal state

by following a path with n steps, it receives a reward of γ^n. Since the agent wants to maximize its reward, it should minimize the expected length of paths to goal states.

The *optimal expected-value* of a state σ can be computed as the fixpoint of the following equation (Bellman 1957):

$$v_n(\sigma) = \begin{cases} \mathcal{R}(\sigma) & \text{iff} \quad n = 0 \\ \max\limits_{\alpha \in \mathcal{A}(\sigma)} \{\, g_n(\sigma, \alpha) \,\} & \text{iff} \quad n > 0, \end{cases}$$

where the *expected gain* in state σ when action α is executed, denoted by $g(\sigma, \alpha)$, is defined as:

$$g_n(\sigma, \alpha) = \gamma \times \sum_{\sigma' \in \mathcal{S}} \mathcal{T}(\sigma, \alpha, \sigma') \times v_{n-1}(\sigma')$$

By selecting an action α that produces the optimal value for a state σ, for each $\sigma \in \mathcal{S}$, we can build an optimal policy:

$$\pi^\star(\sigma) = \arg \max_{\alpha \in \mathcal{A}(\sigma)} \{g_n(\sigma, \alpha)\}$$

A policy π is a *solution* for a probabilistic planning problem $\mathcal{P}$ if and only if π is an optimal policy for $\mathcal{P}$ (Ghallab, Nau, & Traverso 2004). According to this definition, any probabilistic planning problem has a "solution", since it is always possible to find optimal policies. Note, however, that this does not mean that such solution allows the agent to reach a goal state: *an optimal policy is independent of the initial state of the agent.*

The probabilistic planning algorithm. The probabilistic planning algorithm, based on the value-iteration method (Bellman 1957), allows us to synthesize *optimal expected-case* policies for probabilistic planning problems.

PROBABILISTICPLANNING($\mathcal{P}$)
1 `foreach` $\sigma \in \mathcal{S}$ `do` $v_0(\sigma) \leftarrow \mathcal{R}(\sigma)$
2 $n \leftarrow 0$
3 `loop`
4 $n \leftarrow n + 1$
5 `foreach` $\sigma \in \mathcal{S}$ `do`
6 `foreach` $\alpha \in \mathcal{A}(\sigma)$ `do`
7 $g_n(\sigma, \alpha) \leftarrow \gamma \times \sum_{\sigma' \in \mathcal{S}} (\mathcal{T}(\sigma, \alpha, \sigma') \times v_{n-1}(\sigma'))$
8 $v_n(\sigma) \leftarrow \max_{\alpha \in \mathcal{A}(\sigma)} \{g_n(\sigma, \alpha)\}$
9 $\pi_n(s) \leftarrow \arg\max_{\alpha \in \mathcal{A}(\sigma)} \{g_n(\sigma, \alpha)\}$
10 `if` $\max\limits_{\sigma \in \mathcal{S}} |v_n(\sigma) - v_{n-1}(\sigma)| < \epsilon$ `then return` π_n

The probabilistic planning algorithm starts by assigning value $\mathcal{R}(\sigma)$ to each state $\sigma \in \mathcal{S}$. Then, it iteratively refines this value by selecting an action that maximizes the expected gain. At each iteration n, and for each state σ, the value $v_n(\sigma)$ is computed from the value $v_{n-1}(\sigma)$, that was computed at the previous iteration. It can be shown that there exists a maximum number of iterations needed to guarantee that this algorithm returns an optimal policy (Ghallab, Nau, & Traverso 2004). However, in practical applications, the condition used to stop iteration is the following:

$$\max_{\sigma \in \mathcal{S}} |v_n(\sigma) - v_{n-1}(\sigma)| < \epsilon$$

With this condition, the algorithm guarantees that the returned policy is an ϵ-optimal policy, *i.e.*, for each state $\sigma \in \mathcal{S}$, the expected value $v(\sigma)$ does not differ from the optimum value $v^\star(\sigma)$ by more than an arbitrarily small fixed error ϵ.

Comparison between the approaches

In this section, we present a brief comparison between the algorithms for probabilistic planning and for nondeterministic planning based on a planning domain (Figure 3). By analyzing the solutions that these two algorithms find for similar planning problems, we intend to indicate the advantages of each one and move toward a third alternative, which combines both of them (the resulting algorithm is presented in the next section).

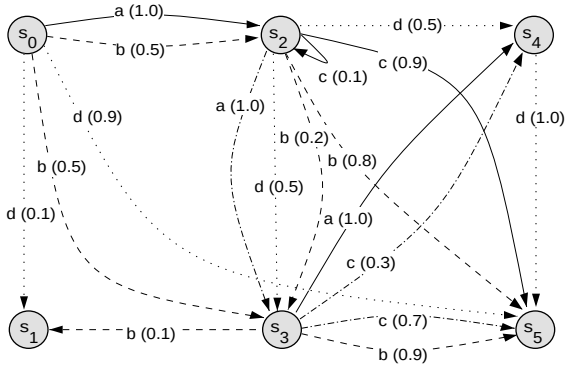

Figure 3: A domain where actions have uncertain effects.

The next example shows the frailties of probabilistic planning when strong policies are required.

Example 1. *Consider $\mathcal{D}$ the planning domain depicted in Figure 3 and $\mathcal{G} = \{s_5\}$ is the set of goal states. For this problem, the algorithm* PROBABILISTICPLANNING($\langle \mathcal{D}, \mathcal{G} \rangle$) *returns the following policy (with $\gamma = 0.9$):*

$\pi(s_0) = d$
$\pi(s_1) = \tau$
$\pi(s_2) = c$
$\pi(s_3) = c$
$\pi(s_4) = d$
$\pi(s_5) = \tau$

This policy is an optimal expected-case solution, i.e., it has shortest execution in the expected-case. By executing action d in state s_0, we expect that in 90% of the executions the goal state can be reached with only one step. This is very efficient and, in some applications, this could be advantageous, even if 10% of the executions fail to reach the goal state. However, there are many other practical applications where failures are unacceptable. In such applications, a plan that may lead to longer executions, but necessarily reaches the goal, is preferable to a plan that in the optimistic case may reach the goal earlier, but in the pessimist case may no longer reach the goal. Clearly, the policy returned by the probabilistic algorithm is weak for state s_0, because it cannot guarantee that the goal state will be reached from this state. Therefore, if an application does not permit failures, a weak policy is inappropriate. On the other hand, if s_2 is considered as the initial state, the returned policy is a strong-cyclic solution (a better solution, because it guarantees to reach the goal state from s_2). However, due to cycles,

the number of steps that a strong-cyclic policy need to reach a goal state is unbounded (e.g., in Figure 3, too many steps c could be needed until agent could leave state s_2). Therefore, if an application is critical in terms of time, a strong-cyclic policy is inappropriate. ◆

The next example illustrate the danger of excessive freedom in nondeterministic planning.

Example 2. *Consider $\mathcal{D}$ the planning domain depicted in Figure 3, s_0 the initial state and $\mathcal{G} = \{s_5\}$ the set of goal states. For this problem, the algorithm* StrongNondeterministicPlanning$(\langle \mathcal{D}, s_0, \mathcal{G} \rangle)$ *returns the following nondeterministic policy:*

$\pi(s_0) = a$
$\pi(s_2) = \{a, b, d\}$
$\pi(s_3) = \{a, c\}$
$\pi(s_4) = d$
$\pi(s_5) = \tau$

This policy is an optimal worst-case strong solution, i.e., it necessarily reaches to reach the goal state with a bounded number of steps (that is minimal in the worst-case). Because in the nondeterministic model there is no preference among actions, any one of the six policies corresponding to this nondeterministic solution can be selected for execution:

$\pi_1 = \{(s_0, a), (s_2, a), (s_3, a), (s_4, d), (s_5, \tau)\}$
$\pi_2 = \{(s_0, a), (s_2, a), (s_3, c), (s_4, d), (s_5, \tau)\}$
$\pi_3 = \{(s_0, a), (s_2, b), (s_3, a), (s_4, d), (s_5, \tau)\}$
$\pi_4 = \{(s_0, a), (s_2, b), (s_3, c), (s_4, d), (s_5, \tau)\}$
$\pi_5 = \{(s_0, a), (s_2, d), (s_3, a), (s_4, d), (s_5, \tau)\}$
$\pi_6 = \{(s_0, a), (s_2, d), (s_3, c), (s_4, d), (s_5, \tau)\}$

Although an agent would prefer to select the policy π_4, which has the possibility of reaching the goal with two steps, it can even select the worst of them (π_1), which always needs exactly four steps to reach the goal state. Therefore, if an application needs an efficient strong policy, a nondeterministic strong policy is inappropriate. ◆

Remark. As we have seen, the probabilistic planning algorithm cannot guarantee to find policies that avoid failures and cycles (*i.e.* strong policies); conversely, the nondeterministic planning algorithm cannot guarantee to select the best strong policy. Thus, we propose a third algorithm, named *strong probabilistic planning*, that can guarantee to find an *optimal expected-case policy among those policies which are optimal in the worst-case*.

Strong probabilistic planning

The strong probabilistic planning combines two common approaches for planning under uncertainty. In this framework, the MC approach is used to guarantee that only optimal worst-case strong solutions can be synthesized during the planning task, while the MDP approach is used to guarantee that an optimal expected-case policy, among those that are optimal in the worst-case, is returned by the planning algorithm.

We present two versions of the algorithm for strong probabilistic planning: an *enumerative* version, where states are explicitly represented and manipulated by standard set operations, and a *symbolic* version, where states are implicitly represented by propositional formulas and can be manipulated by efficient operations on MTBDD's (Bryant 1986).

Enumerative strong probabilistic planning

Given a planning problem $\mathcal{P} = \langle \mathcal{D}, s_0, \mathcal{G} \rangle$, where $\mathcal{D}$ is a probabilistic planning domain, the strong probabilistic planning algorithm starts by constructing an initial policy that maps each goal state $\sigma \in \mathcal{G}$ to the trivial action τ, and by assigning optimal expected-value 1 for each one of them. After this, in each subsequent iteration, the algorithm alternates strong preimage (Mller-Olm, Schimidt, & Steffen 1999) and optimal expected-value computations. By using the strong preimage computation, it guarantees that the synthesized policy will necessarily reach a goal state (without possibility of failure and with a bounded number of steps); and, by using the optimal expected-value computation, it guarantees that, whenever a state is mapped to more than one action by the strong preimage computation, only an optimal action will be chosen in that state. Example 3 gives some intuition about how the the strong probabilistic planning algorithm works.

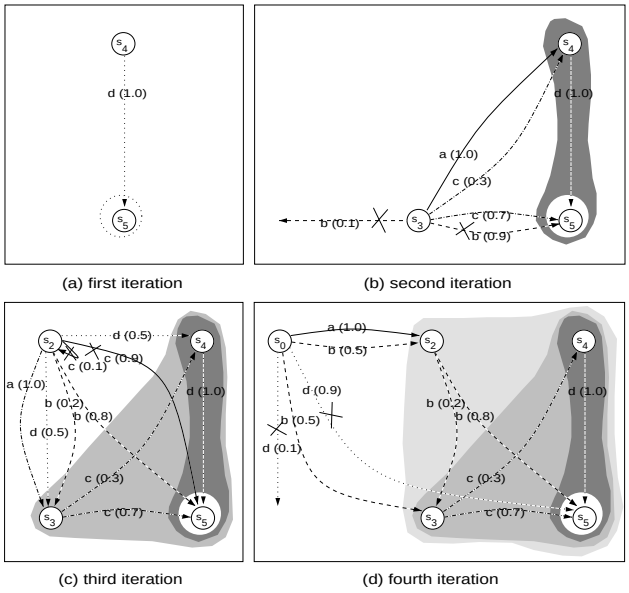

Figure 4: Strong probabilistic planning algorithm execution.

Example 3. *Let $\gamma = 0.9$ and consider the planning problem $\mathcal{P} = \langle \mathcal{D}, s_0, \{s_5\} \rangle$, where $\mathcal{D}$ is the planning domain depicted in Figure 3. Initially, we have $\pi = \{(s_5, \tau)\}$ and $v(s_5) = 1$:*

- *In the first iteration (Figure 4-a), the pruned strong preimage of $\{s_5\}$ is $\{(s_4, d)\}$ and the expected gain for executing action d in state s_4 is $g(s_4, d) = \gamma \times 1.0 \times v(s_5) = 0.9$. Thus, we let $v(s_4) = 0.9$ and $\pi = \{(s_4, d), (s_5, \tau)\}$.*
- *In the second iteration (Figure 4-b), the pruned strong preimage of $\{s_4, s_5\}$ is $\{(s_3, a), (s_3, c)\}$. With this strong preimage computation, we can avoid action b, which*

could cause a failure (i.e., going from s_3 to s_1 leads the agent to a dead end). The expected gain for the remaining actions are:

$g(s_3, a) = \gamma \times 1.0 \times v(s_4) = 0.81$
$g(s_3, c) = \gamma \times (0.3 \times v(s_4) + 0.7 \times v(s_5)) = 0.87$

With this optimal expected-case value computation, we can give preference to action c. Now, we let $v(s_3) = 0.87$ and $\pi = \{(s_3, c), (s_4, d), (s_5, \tau)\}$. Thus, when we have to select among actions that certainly lead to the goal, we choose the one that produces the maximum expected gain.

- *In the third iteration (Figure 4-c), the pruned strong preimage of $\{s_3, s_4, s_5\}$ is $\{(s_2, a), (s_2, b), (s_2, d)\}$. Now, the strong preimage computation avoids action c, which could cause cycle. The expected gains for the other actions are:*

  $g(s_2, a) = \gamma \times (1.0 \times v(s_3)) = 0.79$
  $g(s_2, b) = \gamma \times (0.2 \times v(s_3) + 0.8 \times v(s_5)) = 0.88$
  $g(s_2, d) = \gamma \times (0.5 \times v(s_3) + 0.5 \times v(s_4)) = 0.80$

  Being action b the best choice in state s_2. Thus, we let $v(s_2) = 0.88$ and $\pi = \{(s_2, b), (s_3, c), (s_4, d), (s_5, \tau)\}$
- *Finally, in the last iteration (Figure 4-d), the pruned strong preimage of $\{s_2, s_3, s_4, s_5\}$ is $\{(s_0, a), (s_0, b)\}$. The action d, which could cause failure, is eliminated. The expected gains are:*

  $g(s_0, a) = \gamma \times (1.0 \times v(s_2)) = 0.789$
  $g(s_0, b) = \gamma \times (0.5 \times v(s_3) + 0.5 \times v(s_2)) = 0.787$

  Now, action a is the best choice. Thus, we let $v(s_0) = 0.80$ and $\pi = \{(s_0, a), (s_2, b), (s_3, c), (s_4, d), (s_5, \tau)\}$. Because the initial state s_0 is covered by this policy, the strong probabilistic planning stops and returns π as solution (which corresponds to policy π_4 in the comparison section). ◆

The enumerative version. The enumerative version of the strong probabilistic planning algorithm is composed of two functions: the StrongProbabilisticPlanning function, that performs the strong preimage computation, and the Choose function[2], that performs the optimal expected-value computation.

```
StrongProbabilisticPlanning(P)
1 foreach σ ∈ G do v(σ) ← 1
2 π ← ∅
3 π' ← {(σ, τ) : σ ∈ G}
4 while π ≠ π' do
5   S ← StatesCoveredBy(π')
6   if s0 ∈ S then return π'
7   π ← π'
8   π' ← π' ∪ Choose(Prune(StrongPreimage(S), S))
9 return failure

Choose(R)
1 π ← ∅
2 foreach σ ∈ StatesCoveredBy(R) do
3   A ← {α : (σ, α) ∈ R}
4   foreach α ∈ A do
5     g(σ, α) ← γ × Σ_{σ'∈T(σ,α)} (T(σ, α, σ') × v(σ'))
6   v(σ) ← max_{α∈A} g(σ, α)
7   π ← π ∪ {(σ, arg max_{α∈A} g(σ, α))}
8 return π
```

[2]Because all paths in a strong policy have a bounded number of steps (finite horizon), a discount factor is no longer necessary to guarantee convergence; however, it is still necessary to force the agent to give preference to shortest paths.

The following theorems are the main results because they prove that backward induction works for our model.

Theorem 1. *If a probabilistic planning problem $\mathcal{P}$ has a strong solution, the algorithm* StrongProbabilisticPlanning *returns an optimal worst-case strong policy for $\mathcal{P}$.*

Proof. We denote by π_i the policy built in the i-th iteration of the algorithm. By definition, a state $\sigma \in \mathcal{S}$ is covered by π_0 if and only if σ is a goal state; thus, π_0 covers all states from which, in the worst case, there is a path of length 0 to a goal state. In the first iteration, if the initial state s_0 is covered by π_0, clearly, the algorithm returns an optimal worst-case policy for $\mathcal{P}$. Otherwise, the pruned strong preimage of the set S_0 of states covered by π_0 is computed. For each pair $(\sigma, \alpha) \in$ Prune(StrongPreimage$(S_0), S_0)$), all α-successors of σ are goal states, independently of the chosen actions; thus, the policy $\pi_1 := \pi_0 \cup$ Choose(Prune(StrongPreimage$(S_0), S_0)$)) covers all states from which, in the worst case, there is a path of length 1 to a goal state. By the inductive hypotesis, for $j < i$, policy π_j covers all states from which, in the worst case, there exists a path of length j to a goal state. Therefore, in the i-th iteration, if the initial state s_0 is covered by π_{i-1}, the algorithm returns an optimal worst-case policy for $\mathcal{P}$. Otherwise, the pruned strong preimage of the set S_{i-1} of states covered by π_i is computed. If $(\sigma, \alpha) \in$ Prune(StrongPreimage$(S_{i-1}), S_{i-1})$), then at least one α-successor of σ takes, in the worst case, $i - 1$ steps to reach a goal state (otherwise the state σ would have been covered by policy π_{i-1} and, thus, been pruned). Therefore, independently of the chosen actions, the policy $\pi_i := \pi_{i-1} \cup$ Choose(Prune(StrongPreimage$(S_{i-1}), S_{i-1})$)) covers all states from which, in the worst case, there is a path of (optimal) length i to a goal state. ◆

Theorem 2. *The optimal worst-case strong policy returned by algorithm* StrongProbabilisticPlanning *is optimal in the expected-case.*

The expected-case optimality of the policy returned by the algorithm StrongProbabilisticPlanning is derived from the fact that function Choose uses the optimality principle (Bellman 1957) to choose the best action for each state covered by this policy.

Symbolic strong probabilistic planning

The basic idea underlying the symbolic version of the strong probabilistic planning algorithm is to represent states as sets of propositions and to consistently work with propositional formulas that characterize sets of states. In order to do this, a new definition of planning domain is needed:

Definition 5. *A symbolic probabilistic planning domain is a tuple $\mathcal{D} = \langle \mathbb{P}, \mathcal{S}, \mathcal{A}, \mathcal{L}, \mathcal{T} \rangle$, where:*

- $\mathbb{P}$ *is a finite nonempty set of atomic propositions;*
- $\mathcal{S}$ *is a finite nonempty set of states;*
- $\mathcal{A}$ *is a finite nonempty set of actions;*
- $\mathcal{L} : \mathcal{S} \mapsto 2^{\mathbb{P}}$ *is a state labeling function;*
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ *is a state transition function.* ◆

Each atomic proposition $p \in \mathbb{P}$ denotes a state property. The set of atomic propositions which are satisfied in a state $\sigma \in \mathcal{S}$ is denoted by $\mathcal{L}(\sigma)$. The *intension* of a propositional formula φ in $\mathcal{D}$, denoted by $[\![\varphi]\!]_{\mathcal{D}}$, is the set of states in $\mathcal{D}$ which satisfies φ. Formally, we have[3]:

- $[\![\varphi]\!] = \{\sigma \in \mathcal{S} : \varphi \in \mathcal{L}(\sigma)\}$ if $\varphi \in \mathbb{P}$
- $[\![\neg\varphi]\!] = \mathcal{S} \setminus [\![\varphi]\!]$
- $[\![\varphi \wedge \varphi']\!] = [\![\varphi]\!] \cap [\![\varphi']\!]$
- $[\![\varphi \vee \varphi']\!] = [\![\varphi]\!] \cup [\![\varphi']\!]$

Furthermore, we assume that $\top \in \mathcal{L}(\sigma)$, for all state $\sigma \in \mathcal{S}$. Therefore, it follows that $[\![\top]\!] = \mathcal{S}$.

The trivial action $\tau \in \mathcal{A}$ and the transition function $\mathcal{T}$ are defined as in the pure probabilistic case.

Definition 6. *A symbolic probabilistic planning problem is a tuple* $\mathcal{P} = \langle \mathcal{D}, s_0, (\varphi, \varphi') \rangle$*, where:*

- $\mathcal{D}$ *is a symbolic probabilistic planning domain;*
- $s_0 \in \mathcal{S}$ *is an initial state;*
- (φ, φ') *is an extended reachability goal.* ◆

An *extended reachability goal* is a pair of propositional formulas (φ, φ'): the *preservation condition* φ specifies a property that should be satisfied in each state visited through the path to a goal state (excepting the goal state); and the *achievement condition* φ' specifies a property that should be satisfied in all goal states, *i.e.*, $\mathcal{G} = [\![\varphi']\!]_{\mathcal{D}}$.

Extended goals (Pistore, Bettin, & Traverso 2001; Lago, Pistore, & Traverso 2002) represent an improvement on the expressiveness of the reachability planning framework. By using such goals, besides defining acceptable final states, we can also establish preference among possible intermediate states. Note that a reward function has the same expressiveness of extended goals; however, extended goals are high level specifications.

An example of a symbolic probabilistic domain is depicted in Figure 5. The shadowed states are the ones that can be covered by a policy for the extended reachability goal $(\neg q, p \wedge q \wedge r)$, which specify that the agent should preserve property $\neg q$ (equivalently, avoid property q), until reaching a state where the three properties p, q and r can be satisfied. Other examples of useful extended reachability goals are:

- $(\top, r)$: to achieve property r;
- (p, r): to achieve property r, by preserving property p;
- $(\neg q, r)$: to achieve property r, by avoiding property q;
- $(p \wedge \neg q, r)$: to achieve r, by preserving p and avoiding q.

[3]For the sake of simplicity, we omit subscript $\mathcal{D}$ in $[\![.]\!]$.

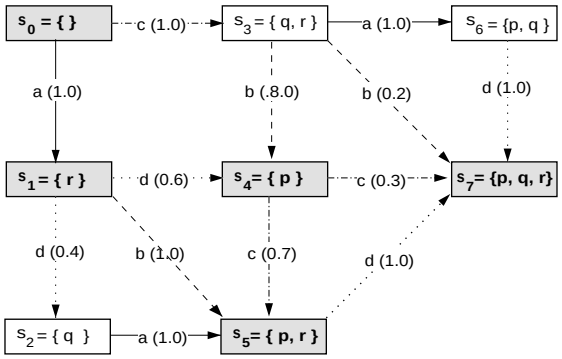

Figure 5: A symbolic probabilistic planning domain.

.

The symbolic version. The symbolic version for the algorithm for extended reachability goals is very similar to the enumerative one. The main difference is on the "intensional" representation of set of states and on the definition of the prune function, which is defined as following:

$$\textsc{Prune}(R, S, \varphi) = \{(\sigma, \alpha) \in R : \sigma \in [\![\varphi]\!]_{\mathcal{D}} \text{ and } \sigma \notin S\}$$

Given the strong preimage R of a set of states S, as well as a preserving condition φ, the function PRUNE selects from R all pairs (σ, α), such that state σ has property φ and it was not yet mapped to another action in a previous iteration. By proceeding in this way, the prune function avoids all intermediate states which does not satisfy the preserving condition φ.

The remainder of the planning algorithm is as following:

```
StrongProbabilisticPlanning(P)
1 foreach σ ∈ [[φ']]_D do v(σ) ← 1
2 π ← ∅
3 π' ← {(σ, τ) : σ ∈ [[φ']]_D}
4 while π ≠ π' do
5   S ← StatesCoveredBy(π')
6   if s0 ∈ S then return π'
7   π ← π'
8   π' ← π' ∪ Choose(Prune(StrongPreimage(S), S))
9 return failure

Choose(R)
1 π ← ∅
2 foreach σ ∈ StatesCoveredBy(R) do
3   A ← {α : (σ, α) ∈ R}
4   foreach α ∈ A do
5     g(σ, α) ← γ × Σ_{σ'∈T(σ,α)} (T(σ, α, σ') × v(σ'))
6   v(σ) ← max_{α∈A} g(σ, α)
7   π ← π ∪ {(σ, arg max_{α∈A} g(σ, α))}
8 return π
```

The following theorems prove that backward induction also works for the symbolic version of our model.

Theorem 3. *If a symbolic probabilistic planning problem* $\mathcal{P}$ *has a strong solution, the symbolic version of algorithm* StrongProbabilisticPlanning *returns an optimal worst-case strong policy for* $\mathcal{P}$.

Theorem 4. *The optimal worst-case strong policy returned by the symbolic version of algorithm* StrongProbabilisticPlanning *is optimal in the expected-case.*

The proofs to these theorems are straightforward from proof of Theorem 1. Noticing that, besides prunning the states already covered by the policy under construction, the symbolic version of the function Prune also prunes states that do not satisfy the extended reachability goal.

Implementation

All policies for the examples in this paper were synthesized by programs which we have implemented. The algorithm ProbabilisticPlanning was implemented in Java, while the other two – StrongNondeterministicPlanning and StrongProbabilisticPlanning – were implemented in Prolog. As the comparison of techniques does not take into account efficiency issues, the use of different programming languages for implementations does not affect our analysis.

The code is available from the first author.

Conclusion

In this paper we have identified, and solved, the problem of *strong probabilistic planning*. In essence, this is a situation with features of nondeterministic and probabilistic planning: requirements on the goals mix worst-case and expected analysis, and actions with (uniform) costs and (Markovian) probabilities associated with them.

Our main contribution is to show that the resulting problem can be tackled by backward induction, thus producing the enumerative and symbolic versions of the StrongProbabilisticPlanner algorithm. While Theorems 1 and 2 deal with straightforward reachability goals, Theorems 3 and 4 show that our techniques can be applied in much greater generality to extended reachability goals. With such goals we can also impose constraints on states visited during policy execution. Hence the symbolic framework is more expressive than the enumerative one. As expressiveness increases planner usability, the symbolic framework seems to be more appropriate for practical planning applications.

The desire to combine features of nondeterministic and probabilistic planning have led us to develop a perspective for planning problems that integrates these features coherently, as we feel that current literature treats these varieties of planning as too isolated islands. We have tried to convey some of this perspective in the third section of this paper; we hope that the resulting blend improves understanding of this multifaceted area.

References

Altman, E. 1999. *Constrained Markov Decision Processes*. Florida: Chapman & Hall / CRC.

Bellman, R. E. 1957. *Dynamic Programming*. USA: Princeton University Press.

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the convergence of real-time dynamic programming.

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

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Cimatti, A.; Giunchiglia, F.; Giunchiglia, E.; and Traverso, P. 1997. Planning via model checking: A decision procedure for $\mathcal{AR}$. In *ECP*, 130–142.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Strong planning in non-deterministic domains via model checking. In *Artificial Intelligence Planning Systems*, 36–43.

Daniele, M.; Traverso, P.; and Vardi, M. Y. 1999. Strong cyclic planning revisited. In *ECP*, 35–48.

Dolgov, D. A., and Durfee, E. H. 2005. Stationary deterministic policies for constrained MDPs with multiple rewards, costs, and discount factors. In *IJCAI*, 1326–1331.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. USA: Morgan Kaufmann Publishers Inc.

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In *ECP*, 1–20.

Lago, U. D.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *Eighteenth national conference on Artificial intelligence*, 447–454. Menlo Park, CA, USA: American Association for Artificial Intelligence.

LaValle, S. M. 2006. *Planning Algorithms*. USA: Cambridge University Press.

Mller-Olm, M.; Schimidt, D.; and Steffen, B. 1999. Model checking: A tutorial introduction. In *SAS'99*, *LNCS 1694*, 330–354.

Pistore, M.; Bettin, R.; and Traverso, P. 2001. Symbolic techniques for planning with extended goals in non-deterministic domains.

Puterman, M. L. 1994. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

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