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.

1 Introduction

Optimal behavior is a very desirable property of autonomous agents and has received much attention in the automated planning community over the last years. Particularly, when agent's actions have probabilistic effects, Markov Decision Processes (MDPs) provide a simple and elegant framework for synthesizing optimal expected-case policies [Puterman 1994]. In this framework, the execution of each action produces a reward and the objective is to maximize the expected value of a sum of exponentially discounted rewards received by the agent. The classical MDP formulation has some important properties: it is subject to the principle of local optimality, according to which the optimal action for a state is independent of the actions chosen for other states, and optimal policies are stationary and deterministic. These properties translate into very efficient dynamic-programming algorithms [Bellman 1957], e.g., *value-iteration* [Puterman 1994]. However, as pointed in [Dolgov and Durfee 2005], there are some situations where the classical MDP formulation proves inadequate, because it can be very difficult (if not impossible) to fold all the relevant feedback from the planning environment into a single reward function.

In this paper, we use *strong probabilistic planning* to refer to situations where one wishes to produce a policy that guaranteedly reaches a goal state, with minimum number of steps in the worst-case and maximum reward in the expected-case. For instance, such situation can happen in a domain where the agent should deliver a mail, minimizing the time spent in the worst-case, while maximizing its expected reward. To get

A. Gelbukh and E.F. Morales (Eds.): MICAI 2008, LNAI 5317, pp. 636–652, 2008.

some intuition on the difficulty of combining these constraints in the classical MDP framework, consider the situations depicted in Figure 1, where the agent's goal is to deliver a mail in room s_2. For both situations, *value-iteration* synthesizes the optimal expected-case policy $\pi = \{(s_0, a)\}$. However, by following π in the first one (Figure 1-a), the agent cannot guarantee that the mail will be delivered (since the execution of action a in state s_0 can lead to the dead-end s_3); and, in the second situation (Figure 1-b), the agent cannot guarantee a minimum number of steps for delivering the mail (since the execution of action a in state s_0 can require an unbounded number of steps before reaching the goal state s_2). Thus, in the first case, the agent incurs the risk of failing to deliver the mail; while in the second case, the agent incurs the risk of delivering the mail when it is no more useful. As we show in this paper, strong probabilistic planning is a novel framework to implement *risk-averse* agents, by combining decision-theoretic concepts [Boutilier, Dean, and Hanks 1999] with model checking (MC) techniques [Muller-Olm, Schimidt, and Steffen 1999].

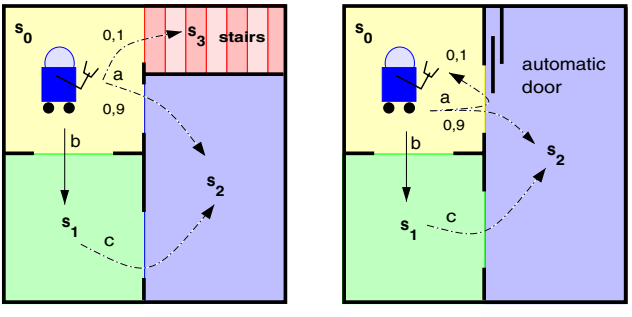

Fig. 1. A delivering agent domain

We should note that, at least in principle, we are dealing with constrained MDPs; that is, MDPs with additional constraints on functionals over trajectories [Puterman 1994, page 229]. However, there are two difficulties in directly resorting to the theory of constrained MDPs [Altman]. First, it is not entirely obvious how to model the constraint that the policy should be optimal in the worst-case. Second, general constrained MDPs fail the dynamic programming principle and, therefore, cannot be solved by backward induction [Altman]. On the other hand, as we show in this paper, strong probabilistic planning admits a backward reasoning scheme and, thus, yields better results than a general approach based on constrained MDPs.

The remainder of this paper is organized as follows. First, we present a review of MC/MDP-based planning, that contributes with a unifying perspective that seeks to compare its algorithms on a common ground. The insights produced by this comparison steer us to our algorithm for strong probabilistic planning. We describe two versions of this algorithm, 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. Finally, we discuss our implementation and present our conclusions.

2 Approaches for Planning under Uncertainty

In this section we give an 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 is rarely found in the literature; thus, we devote considerable space to this issue. The approaches of interest here are:

- *Model Checking* (MC), used for planning under nondeterministic uncertainty models; and
- *Markov Decision Processes* (MDP), used for planning under probabilistic uncertainty models.

Both approaches are very attractive, but for different reasons: the main advantage of MC-based approach is the possibility of exploiting the knowledge about the structure of the planning domain (i.e., the state-transition graph induced by the actions); while the main advantage of MDP-based approach is the possibility of exploiting the knowledge about the probability of the actions' effects.

2.1 Nondeterministic Planning Based on MC

The basic idea underlying nondeterministic planning based on MC is to solve problems model-theoretically [Giunchiglia and 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 an agent executes action τ in a certain state, it always remains in this same state. Intuitively, action τ represents the fact that the 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 $\pi : \mathcal{S} \mapsto 2^{\mathcal{A}} \setminus \varnothing$, 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 subgraph of $\mathcal{D}$ that has $\mathcal{S}_\pi$ as set of states and that 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 the initial state;*
- $\mathcal{G} \subseteq \mathcal{S}$ *is a set of goal states.* ◆

Given a nondeterministic planning problem, we distinguish 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 path in $\mathcal{D}_\pi$, starting from s_0, reaches a state in $\mathcal{G}$ [Cimatti et al. 1997].
- a *strong-cyclic solution* is a policy that always achieves the goal, under the fairness assumption that execution will eventually exit from all 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, and Vardi 1999].
- a *strong solution* is a policy that always achieves the goal, in spite of nondeterminism. A policy π is a strong solution if the subgraph $\mathcal{D}_\pi$ is acyclic and all paths starting from s_0 reach a state in $\mathcal{G}$ [Cimatti, Roveri, and Traverso 1998].

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

```
StrongNondeterministicPlanning(P)
1 π ← ∅
2 π' ← {(σ, τ) : σ ∈ G}
3 while π ≠ π' do
4   S ← StatesCoveredBy(π')
5   if s_0 ∈ S then return π'
6   π ← π'
7   π' ← π' ∪ Prune(StrongPreimage(S), S)
8 return failure
```

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

```
StrongPreimage(S)
1 return {(σ, α) : σ ∈ S, α ∈ A and ∅ ≠ T(σ, α) ⊆ S}
```

By iterating the strong preimage function from the set of goal states $\mathcal{G}$, the planning algorithm builds up a finite backward search tree (Figure 2). Because the set of states $\mathcal{S}$ is finite and this function is monotonic w.r.t. set inclusion, i.e., $\mathcal{G} \subseteq$ StrongPreimage$^1(\mathcal{G}) \subseteq$ StrongPreimage$^2(\mathcal{G}) \subseteq \cdots \subseteq$ StrongPreimage$^n(\mathcal{G})$, after a finite number of iterations, a fixpoint is obtained. During this iterative process, the planning algorithm maps the states in the search tree to actions (or sets of actions) and, therefore, a policy is synthesized as a side effect. Furthermore, at each iteration, the set of states covered by the policy under construction π' is obtained by the following function:

```
StatesCoveredBy(π')
1 return {σ : (σ, α) ∈ π'}
```

Thus, if there exists a strong policy to reach a state in $\mathcal{G}$, from the initial state s_0, then in one of these iterations, the condition $s_0 \in \text{STATESCOVEREDBY}(\pi')$ is satisfied and the algorithm returns the policy π' as a 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 planning algorithm uses the following pruning function:

PRUNE(R, S)
1 `return` $\{(\sigma, \alpha) \in R : \sigma \in S\}$

As we can see, this function prunes from the set R (the strong preimage) every pair (σ', α) such that the state σ' is already in the set S (the covering of the current policy).

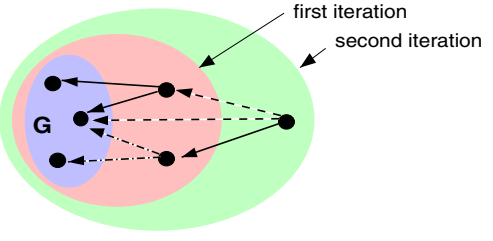

Fig. 2. A backward search tree built after two iterations of the strong preimage function

2.2 Probabilistic Planning Based on MDPs

The basic idea underlying probabilistic planning based on MDPs is to represent the planning problem as an optimization problem [Boutilier, Dean, and 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:

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

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 subgraph of $\mathcal{D}$ that has $\mathcal{S}$ as set of states and that 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:

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

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 reaches 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]:

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

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, and Traverso 2004]. According to this definition, any probabilistic planning problem has a "solution", since it is always possible to find optimal policies for MDPs. 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(P)
1 foreach σ ∈ S do v_0(σ) ← R(σ)
2 n ← 0
3 loop
4    n ← n + 1
5    foreach σ ∈ 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 $\max_{\sigma \in \mathcal{S}} |v_n(\sigma) - v_{n-1}(\sigma)| < \epsilon$ 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 these values by selecting actions that maximize the expected gains. 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, and 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.

2.3 Comparison between the Approaches

In this section, we present a brief comparison between the algorithms for probabilistic planning and for nondeterministic planning, by considering the planning domain depicted in 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).

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

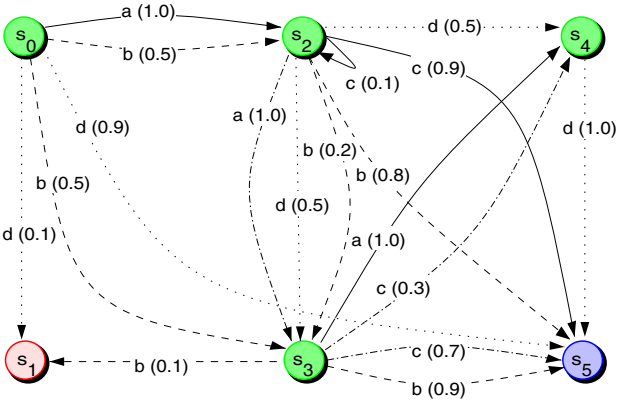

Fig. 3. A domain where actions have uncertain effects

Example 1. *Consider the planning domain $\mathcal{D}$, depicted in Figure 3, and let $\mathcal{G} = \{s_5\}$ be 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 allow 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 the planning domain $\mathcal{D}$, depicted in Figure 3, and let s_0 be the initial state and $\mathcal{G} = \{s_5\}$ be 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 the goal state after 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*.

3 Strong Probabilistic Planning

The strong probabilistic planning combines two widely used 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 that can be manipulated by efficient operations on MTBDD's [Bryant 1986].

3.1 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 [Muller-Olm, Schimidt, and 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.

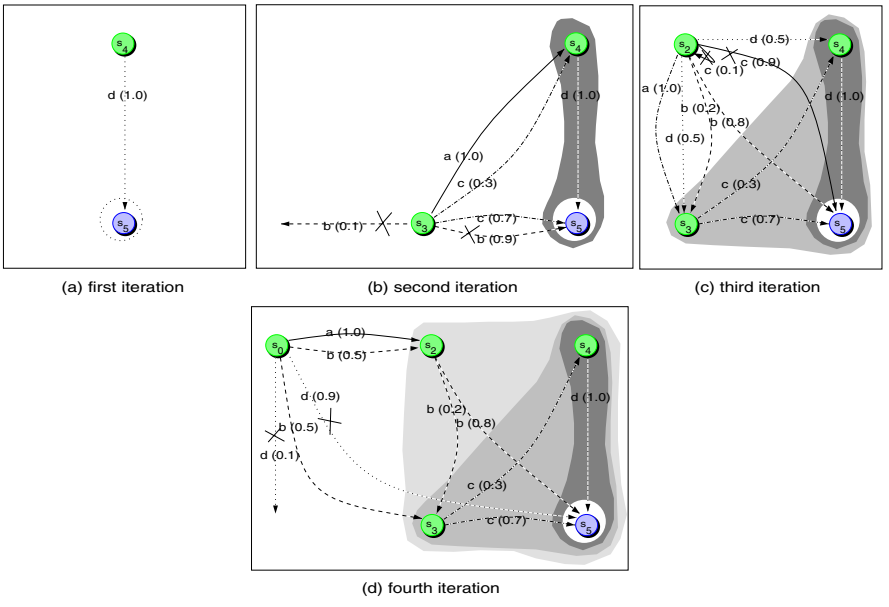

Fig. 4. Strong probabilistic planning algorithm execution

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 $\{s_4, s_5\}$, the covering set of the current policy, 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. 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 StrongPreimage function, that performs the strong preimage computation, and the Choose function[1], that performs the optimal expected-value computation.

Given a planning problem $\mathcal{P}$, the enumerative version of the strong probabilistic planning algorithm starts by assigning reward 1 to goal states and by building a policy that maps each one of these states to the trivial action τ. Afterwards, alternately, the algorithm computes strong preimages and optimal expected values. By using the function StrongPreimage, it guarantees that the synthesized policy necessarily reaches a goal state (in spite of nondeterminism); and by using the function Choose, it guarantees that whenever a state is mapped to more than one action (by the strong preimage function), only an action with optimal expected value can be chosen in that state.

```
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'))
```

[1] 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.

```
6    v(σ) ← max_{α∈A} g(σ, α)
7    π ← π ∪ {(σ, arg max_{α∈A} g(σ, α))}
8 return π
```

The function CHOOSE implements the optimality principle through backward induction. This is possible because the combined use of the functions STRONGPREIMAGE and PRUNE avoids cycles, allowing a topological sorting of the states covered by the policy (Figure 5). By processing states in reverse topological order, when the value of a state is computed, the value of each one of its possible successors is already known. This can reduce the number of state updates.

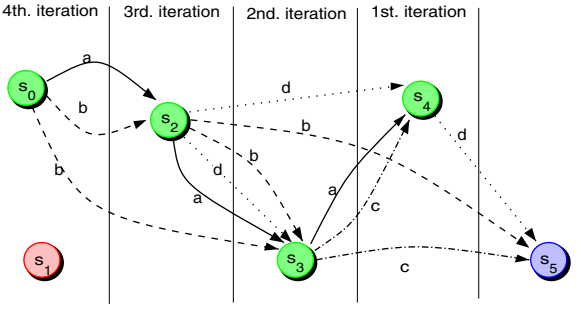

Fig. 5. Topological sorting of the domain's states

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 hypothesis, 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.

3.2 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[2]:

- $[\![\varphi]\!] = \{\sigma \in \mathcal{S} : \varphi \in \mathcal{L}(\sigma)\}$ 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.* ◆

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

An *extended reachability goal* [Pereira and Barros 2008] is a pair of logical formulas (φ, φ'): the *preservation condition* φ specifies a property that should hold 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 hold in all goal states, *i.e.*, $\mathcal{G} = [\![\varphi']\!]_{\mathcal{D}}$.

Extended goals (Pistore, Bettin, and Traverso 2001; Lago, Pistore, and Traverso 2002; Pereira and Barros 2008) 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 can have the same expressiveness of extended goals; however, extended goals are high level specifications.

An example of a symbolic probabilistic domain is depicted in Figure 6. 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.

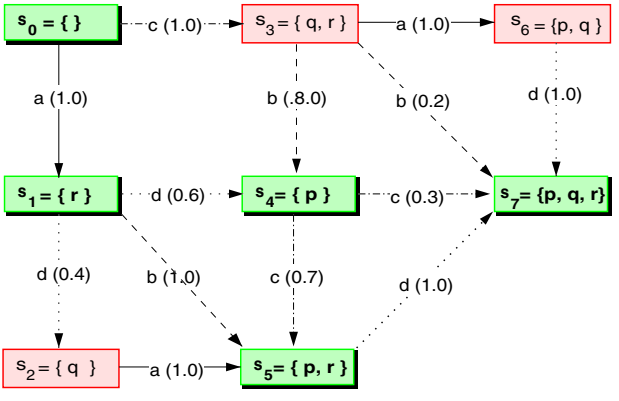

Fig. 6. A symbolic probabilistic planning domain

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:

PRUNE(R, S, φ)
1 `return` $\{(\sigma, \alpha) \in R : \sigma \in [\![\varphi]\!]_{\mathcal{D}}$ and $\sigma \in S\}$

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 s_0 ∈ 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 pruning 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.

4 A Note about Implementation

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

5 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 planning goals mix worst-case and expected-case 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 efficiently by using backward induction (with minimum number of state updates), 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 second section of this paper; we hope that the resulting blend improves understanding of this multifaceted area.

We should still emphasize that we can found in the literature algorithms that are capable of synthesizing weak solutions (e.g., *value-iteration* [Puterman 1994]), as well as strong-solutions (e.g. RTDP [Bertsekas and Tsitsiklis 1991]), for probabilistic planning problems modeled as MDPs. However, to the best of our knowledge, the hybrid framework proposed in this paper is the first one that is capable of guarantee the synthesis of strong solutions for such kind of probabilistic planning problems (the work in [Mausam, Bertoli, and Weld2007] does not offer this guarantee).

References

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

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

Bertsekas, D.P., Tsitsiklis, J.N.: An analysis of stochastic shortest path problems. Math. Oper. Res. 16(3), 580–595 (1991)

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

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

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

Cimatti, A., Roveri, M., Traverso, P.: Strong planning in non-deterministic domains via model checking. In: Artificial Intelligence Planning Systems, pp. 36–43 (1998)

Daniele, M., Traverso, P., Vardi, M.Y.: Strong cyclic planning revisited. In: Biundo, S., Fox, M. (eds.) ECP 1999. LNCS, vol. 1809, pp. 35–48. Springer, Heidelberg (2000)

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

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

Giunchiglia, F., Traverso, P.: Planning as model checking. In: ECP, pp. 1–20 (1999)

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

Mausam Bertoli, P., Weld, D.S.: A hybridized planner for stochastic domains. In: Veloso, M.M. (ed.) IJCAI, pp. 1972–1978 (2007)

Muller-Olm, M., Schimidt, D., Steffen, B.: Model checking: A tutorial introduction. In: Cortesi, A., Filé, G. (eds.) SAS 1999. LNCS, vol. 1694, pp. 330–354. Springer, Heidelberg (1999)

Pereira, S.L., Barros, L.N.: A logic-based agent that plans for extended reachability goals. Autonomous Agents and Multi-Agent Systems 9034, 327–344 (2008)

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

Puterman, M.L.: Markov Decision Processes—Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., Chichester (1994)