Reachability Analysis for Uncertain SSPs

Olivier Buffet
National ICT Australia & The Australian National University
RSISE - building 115 / The Australian National University / Canberra, ACT 0200 / Australia
firstname.lastname@nicta.com.au

Abstract

Stochastic Shortest Path problems (SSPs) can be efficiently dealt with by the Real-Time Dynamic Programming algorithm (RTDP). Yet, RTDP requires that a goal state is always reachable. This paper presents an algorithm checking for goal reachability, especially in the complex case of an uncertain SSP where only a possible interval is known for each transition probability. This gives an analysis method for determining if SSP algorithms such as RTDP are applicable, even if the exact model is not known. We aim at a symbolic analysis in order to avoid a complete state-space enumeration.

1. Introduction

In decision-theoretic planning, Markov Decision Problems [4] are of major interest when a probabilistic model of the domain is available. A range of algorithms make it possible to find plans (policies) optimizing the expected long-term utility. Yet, optimal policy convergence results all depend on the assumption that the probabilistic model of the domain is accurate.

Unfortunately, a large number of MDP models are based on uncertain probabilities (and rewards). Many rely on statistical models of systems, may they be toy problems such as the mountain-car or the inverted-pendulum, or real problems such as plant control or animal behavior analysis. These statistical models are based on simulations, observations of a real system or human expertise.

Using uncertain models requires answering two closely related questions: 1- how to model this uncertainty, and 2- how to use the resulting model. Existing work shows that uncertainty is sometimes represented as a set of possible models, each assigned a probability [15]. The simplest example is sets of possible models all assumed equally probable [1; 16]. Rather than construct a possibly infinite set of models we represent uncertainty by defining an interval for each transition probability [13; 14].

Uncertain probabilities have been investigated in resource allocation problems [15] — investigating efficient exploration [17] and state aggregation [13] — and policy robustness [1; 14; 16]. We focus on the later, considering a two-player game where the opponent chooses from the possible models to reduce the long-term utility.

Our principal aim is to develop an efficient planner for a common sub-class of MDPs for which optimal policies are guaranteed to eventually terminate in a goal state: Stochastic Shortest Path (SSP) problems. A greedy version of *Real-Time Dynamic Programming algorithm* (RTDP) [2] is particularly suitable for SSPs, as it finds good policies quickly and does not require complete exploration of the state space. Yet, if it can be made robust [11], it also requires that a goal state is reachable from any visited state, which can be checked through a reachability analysis.[1]

This paper shows how to make the reachability analysis for SSPs, focusing on uncertain ones. A logical extension would be to derive a symbolic analysis, what seems essential for the practical use of algorithms such as RTDP. In Section 2 we present SSPs, RTDP and robustness. We then explain the algorithm for reachability analysis. Finally, a practical experiment is presented before a conclusion.[2]

2. Background

2.1. Stochastic Shortest Path Problems

A Stochastic Shortest Path Markov Decision Problem [4] is defined here as a tuple $\langle S, s_0, G, A, T, c\rangle$. It describes a control problem where S is the finite set of **states** of the system considered, $s_0 \in S$ is a starting state, and $G \subseteq S$ is a set of goal states. A is the finite set of possible **actions** a. Actions control transitions from one state s to another state s' according to the system's probabilistic dynamics, described by the **transition function** T defined as

[1] In some planning technics [6], the "reachability analysis" has a somewhat different purpose as it looks for states reachable by some plan. Both meanings have some similarities but should not be mixed up.

[2] This work is presented in more details in [8].

$T(s,a,s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$. The aim is to optimize a performance measure based on the **cost function** $c : S \times A \times S \to \mathbb{R}^+$.[3]

SSPs assume a goal state is reachable from any state in S, at least for the optimal policy, so that one cannot get stuck in a looping subset of states. An algorithm solving an SSP has to find a **policy** that maps states to probability distributions over actions $\pi : S \to \Pi(A)$ which optimizes the chosen performance measure, here the **value** J defined as the expected sum of *costs* to a goal state.

In this paper, we only consider SSPs for planning purposes. In this framework, well-known stochastic dynamic programming algorithms such as *value iteration* (VI) make it possible to find a deterministic policy that corresponds to the minimal expected long-term cost J. *Value iteration* works by computing the value function $J^*(s)$ that gives the expected reward of the optimal policies. It is the unique solution of the fixed point equation[3]:

$$J(s) \quad = \quad \min_{a \in A} \sum_{s' \in S} T(s,a,s') \left[c(s,a,s') + J(s') \right]. \quad (1)$$

Updating J with this formula leads to the optimal value function. For convenience, we also introduce the Q-value: $Q(s,a) = \sum_{s' \in S} T(s,a,s')[c(s,a,s') + J(s')]$.

This kind of problem can easily be viewed as a shortest path problem where choosing a path only probabilistically leads you to the expected destination. SSPs can represent a useful subset of MDPs. They are essentially a finite-horizon MDP with no discount factor.

2.2. RTDP

A first algorithm making use of the structure of SSPs is a version of the *Real-Time Dynamic Programming* algorithm (RTDP) [2]. It uses the fact that the SSP cost function is positive and the additional assumption that every trial will reach a goal state with probability 1. Thus, with a zero initialization of the J, both the J and Q-values monotonically increase during their iterative computation.

The idea behind RTDP (Algorithm 1) is to follow paths from the start state s_0, always greedily choosing actions of low value and updating $Q(s,a)$ as states s are encountered. In other words, the action chosen is the one expected to lead to the lowest future costs, until the iterative computations show that another action may do better.

RTDP has the advantage of quickly avoiding plans that lead to high costs. Thus, the exploration looks mainly at a promising subset of the state space. Because it follows paths by simulating the system's dynamics, common transitions are favored, so that good policies are obtained early.

[3] As the model is not sufficiently known, we do not make the usual assumption $c(s,a) = \mathbb{E}_{s'}[c(s,a,s')]$.

Algorithm 1 RTDP algorithm for SSPs

```
RTDP(s:state) // s = s_0
repeat
   RTDPTRIAL(s)
until // no termination condition
.....
RTDPTRIAL(s:state)
while ¬GOAL(s) do
   a =GREEDYACTION(s)
   J(s) =QVALUE(s, a)
   s =PICKNEXTSTATE(s, a)
.....
GREEDYACTION(s)
return argmin_{a∈A} QVALUE(s, a)
.....
QVALUE(s: state, a: action)
return Σ_{s'∈S} T(s, a, s').[c(s, a, s') + J(s')]
```

Yet, the bad update frequency of rare transitions slows the convergence.

2.3. Robust Value Iteration

We now turn to the problem of taking the model's uncertainty into account when looking for a "best" policy. Here, uncertainty only consists in inaccurate knowledge of the transition function T. The (possibly infinite) set of alternative models is denoted $\mathcal{M}$.

We follow the approach described in [1], that consists of finding a policy that behaves well under the worst possible model. This amounts to a two-player zero-sum game where a player's gain is its opponent's loss. The player chooses a policy while its "disturber" opponent chooses a model. A simple process may be used to compute the value function while looking simultaneously for the worst model. It requires the hypothesis that state-distributions $T(s,a,\cdot)$ are independent from one state-action pair (s,a) to another. Under this assumption, the worst model can be chosen locally when Q is updated for a given state-action pair. If this assumption does not always actually hold, it induces a larger set of possible models, what results in a worst-case assumption in the pessimistic approach.

Problem — We are particularly interested in handling *uncertain SSPs* (USSP), where only intervals of possible transition probabilities are known: $T(s,a,s') \in [Pr^{\min}(s'|s,a), Pr^{\max}(s'|s,a)]$. Yet, to use (robust) RTDP, this theorem is of major interest:

Theorem 1. [4] *If the goal is reachable with positive probability from every state, RTDP unlike the greedy policy cannot be trapped into loops forever and must eventually reach*

the goal in every trial. That is, every RTDP trial terminates in a finite number of steps.

The purpose of this paper is to determine from which states a goal state is still reachable in uncertain SSPs. This could be achieved with a policy choosing all actions with equal probability, and letting the opponent learn how to prevent goal states from being reached. Yet, this problem is no SSP, what would imply coming back from RTDP to *Value Iteration*. Moreover, we prefer performing a graph analysis, as it gives more practical information and would be a first step toward a symbolic analysis avoiding the enumeration of the complete state-space. To our knowledge, this particular problem has not been addressed up to now.

3. Algorithms

When applying algorithms such as RTDP on an SSP having no proper policy, the main problem is to detect if current state s still has a positive probability of reaching the goal set, in which case s is said to be "`reaching`". If s is non-`reaching`, RTDP should stop and a specific process be applied, such as associating this state to an infinite cost.

Non-`reaching` states constitute looping sub-sets of states which we will refer to as "dead-ends". The process just described results in dead-ends avoidance. Yet some states may be `reaching` but also have a positive probability to lead to a dead-end whatever the policy. If non-`reaching` states incur infinite costs, these "`dangerous`" states will necessarily have an infinite long-term cost to the goal. It would thus be of interest to also identify these `dangerous` states.

Note that what to do when in a non-`reaching` state may depend on the user's preferences. But in all cases the first step is to perform a "reachability analysis" through a graph traversal beginning with goal states. Then, if required, a "danger analysis" can be performed through another (simpler) graph traversal beginning with non-`reaching` states. This paper mainly focuses on the "reachability analysis", as this process is necessary and somewhat subtle in the case of USSPs.

3.1. Certain SSP

In a certain SSP, if s' is `reaching`, any state s such that $T(s,a,s') > 0$ for some action a is also reaching. This results in a straightforward analysis by making a graph traversal starting with goal states.

Let $Parents(s)$ be the set of states s' for which there exists a transition $(s',a) \rightarrow s$: $Parents(s) = \{s' \in S \text{ s.t. } \exists a \in A\ Pr(s|s',a) > 0\}$. Alg. 2 uses this information to perform the reachability analysis. Then states which have not been marked as `reaching` are dead-ends, and a second graph traversal starting with these states will identify `dangerous` states (see Alg. 3).[4]

Algorithm 2 PROPAGATEREACHABILITYSSP ($Parents$)

```
PUSHALL(G, st) {st: stack of goal states}
while st ≠ ∅ do
  POP(s, st)
  if ¬reaching(s) then
    MARK(s, reaching)
    PUSHALL(Parents(s), st)
```

Algorithm 3 PROPAGATEDANGER($Parents$)

```
for all s ∈ S s.t. ¬reaching(s) do
  PUSH(s, st)
while st ≠ ∅ do
  POP(s, st)
  if ¬dangerous(s) and ∀ a ∈ A :
  ∃ s' ∈ S s.t. Pr(s'|s,a) > 0 & dangerous(s') then
    MARK(s, dangerous)
    PUSHALL(Parents(s), st)
```

3.2. Uncertain SSP: overview

In an uncertain SSP, the reachability analysis depends on the opponent being able to forbid a transition $(s,a) \rightarrow s'$ if $Pr^{\min}(s'|s,a) = 0$. A difficulty is that $Pr^{\min}(s'_1|s,a) = 0$ and $Pr^{\min}(s'_2|s,a) = 0$ are not sufficient to tell if s'_1 and s'_2 may be forbidden simultaneously in some possible model. Fig. 1 shows an example where the 3 potentially reachable states cannot be forbidden simultaneously (there is no possible model s.t. $\forall j \in \{1,2,3\}\ T(s_o,a_0,s'_j) = 0$). With upper probabilities of 1, any 2 states could be forbidden.

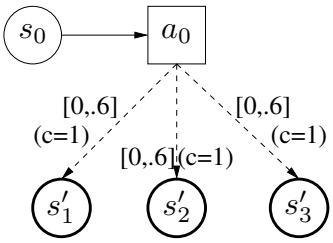


Figure 1. USSP where only 1 of the 3 reachable states can be forbidden (goal states in bold circles).

Let us define the set of all lists of states which cannot be

[4] Alg. 3 can be implemented efficiently by remembering which state-action pairs are known to be dangerous.

forbidden simultaneously (from (s,a)):[5]

$$L^{\oslash}_{(s,a)} = \left\{ \begin{array}{l} l \subseteq S \text{ s.t. } \left(s' \in l \Rightarrow Pr^{\max}_{(s'|s,a)} > 0\right), \text{ and} \\ \left[\begin{array}{l} \left(\exists s' \in l \text{ s.t. } Pr^{\min}_{(s'|s,a)} > 0\right) \\ \text{or } \left(\sum_{s' \in S \setminus l} Pr^{\max}_{(s'|s,a)} < 1\right) \end{array} \right] \end{array} \right\}.$$

To know if a given action a can lead to a goal state from current state s, one has to find at least one such list where all states are `reaching`. In this case, the opponent cannot prevent the planner having some chance of terminating. The reachability analysis only needs to work with the subset of minimal lists defined as:

$$L^{\min \oslash}_{(s,a)} = \left\{ \begin{array}{l} l \in L^{\oslash}_{(s,a)} \text{ s.t. } \forall l' \in L^{\oslash}_{(s,a)} : \\ (l \cap l' = l) \text{ or } \left((l \cap l') \notin L^{\oslash}_{(s,a)}\right) \end{array} \right\}.$$

In other words, removing any state of such a list makes it possible for the opponent to forbid all states in the list. On Fig. 1: $L^{\min \oslash}_{(s_o,a_o)} = \{\{s'_1, s'_2\}, \{s'_1, s'_3\}, \{s'_2, s'_3\}\}$.

From this basic idea, two problems arise:

- How to perform the reachability analysis ? (Sec. 3.3)
- How to obtain these lists ? (Sec. 3.4)

The remainder of this section gives a brief idea of the answers to these two questions before going in more details through Sections 3.3 and 3.4.

Performing the Reachability Analysis – The minimal lists we have just described are defined with respect to a given state-action pair (s,a). They are used to obtain a new set $L^{\min \oslash}_{(s)}$ of minimal lists relative to the state s, since the precise action chosen is of no interest when just checking whether a state could reach the goal or not.

From there, determining which states can reach a goal state is again done through a propagation starting from these goal states. This "back"-propagation takes place in an `AND-OR` graph where nodes are states and their minimal lists, as illustrated by Fig. 2. This is an `AND-OR` graph because a list is "`reaching`" if *all* its children states are `reaching` (`AND`), and a state is `reaching` if *one* of its children lists is `reaching` (`OR`).

After this reachability analysis for uncertain SSPs, the danger analysis from Alg. 3 can be performed with no modification, using the most probable model for example. Indeed in this second phase the opponent has no need to prevent some transitions from happening (by assigning them a zero probability mass). On the contrary, its aim should be to allow all possible transitions in a view to give more ways of getting to a dead-end.

[5] $\oslash \sim$ "states **cannot** be forbidden simultaneously"

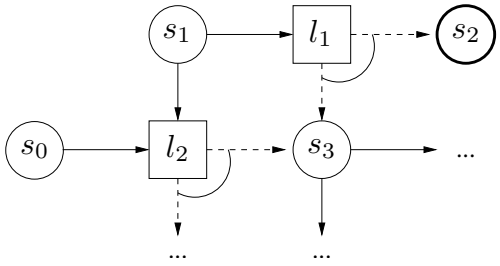


Figure 2. Example of `AND-OR` graph in which the reachability analysis is done (starting with goal states as s_2 here). If s_3 is `reaching`, then so is l_1 (the opponent cannot forbid s_2 and s_3), and therefore s_1.

How to Obtain the Lists — Previous section has shown how to use minimal lists of states which cannot be forbidden simultaneously so as to perform the reachability analysis. An essential question that we still have to answer is how to obtain these lists. This is an indirect process as it consists in 1- looking for *maximal* lists of states which *can* be forbidden simultaneously, then in 2- adding a state to turn them into *minimal* lists of states which *cannot* be forbidden simultaneously.

As we have defined the notion of "list of states which *cannot* be forbidden simultaneously", we define the opposite notion of "list of states which *can* be forbidden simultaneously":

$$L^{\odot}_{(s,a)} = \left\{ \begin{array}{l} l \subseteq S \text{ s.t. } \left(\sum_{s' \in S \setminus l} Pr^{\max}_{(s'|s,a)} \geq 1\right) \text{ and} \\ s' \in l \Rightarrow \left(Pr^{\min}_{(s'|s,a)} = 0 \text{ and } Pr^{\max}_{(s'|s,a)} > 0\right) \end{array} \right\}.$$

But we only need to consider the subset of these lists which are "maximal":

$$L^{\max \odot}_{(s,a)} = \left\{ \begin{array}{l} l \in L^{\odot}_{(s,a)} \text{ s.t. } \forall l' \in L^{\odot}_{(s,a)} : \\ (l \cup l' = l) \text{ or } \left(l \cup l' \notin L^{\odot}_{(s,a)}\right) \end{array} \right\}.$$

Indeed, adding any reachable state to such a list turns it into a list from $L^{\oslash}_{(s,a)}$. Obtaining the minimal lists required for the reachability analysis requires then two algorithms:

- one to create $L^{\max \odot}_{(s,a)}$ ($\forall (s,a) \in S \times A$), and
- one to turn any set $L^{\max \odot}_{(s,a)}$ in the corresponding set $L^{\min \oslash}_{(s,a)}$.

These two algorithms are the one called inside the double "for" loop of Alg. 4. This general algorithm sums up the complete process followed to analyse the reachability of a USSP as discussed up to now. Next two sections give more details on these various steps.

Algorithm 4
REACHABILITYANALYSIS ($\langle S, s_0, G, A, T, c\rangle$: USSP)

Ensure: Mark states as `reaching` if they may reach a goal state, or `dangerous` if they may lead to a dead-end.
for all $s \in S$ s.t. $\neg(s \in G)$ **do**
  for all $a \in A(s)$ **do**
    $L^{\max\odot}_{(s,a)} \leftarrow$ FORBIDDENFROMSA $(s, a, Children(s, a))$
    $L^{\min\oslash}_{(s,a)} \leftarrow$ NOTFORBIDDENFROMSA $(s, a, Children(s, a), L^{\max\odot}_{(s,a)})$
  $L^{\min\oslash}_{(s)} \leftarrow$ NOTFORBIDDENFROMS$(s, L^{\min\oslash}_{(s,\cdot)})$
$(L^{\min\oslash}, L^{parents}_{(\cdot)}) \leftarrow$ ALLMINLISTS$(L^{\min\oslash}_{(\cdot)})$
PROPAGATEREACHABILITYUSSP$(L^{\min\oslash}, L^{parents}_{(\cdot)})$
PROPAGATEDANGER()

3.3. Performing the Reachability Analysis

From (s, a) to s — We know that deciding whether a state s may reach a goal according to its children does not require considering all actions separately. Indeed, when sets $L^{\min\oslash}_{(s,a)}$ have been determined for all a, they can be merged in a single set of minimal lists: $\bigcup_{a\in A(s)} L^{\min\oslash}_{(s,a)}$. From this new set, lists including other lists have to be removed since: if $l \subseteq l'$, then l' is not minimal in our new set of lists. This process is detailed in Alg. 5.[6]

Algorithm 5 NOTFORBIDDENFROMS (s: state, $L^{\min\oslash}_{(s,\cdot)}$: minimal lists of states which cannot be forbidden simultaneously from s and an action)

$L \leftarrow \emptyset$
{a- Put all minimal lists obtained in $L^{\min\oslash}_{(s)}$.}
for all $a \in A(s), l \in L^{\min\oslash}_{(s,a)}$ **do**
  if $\forall l' \in L, l' \neq l$ **then**
    $L \leftarrow L \bigcup \{l\}$
    $parents(l) \leftarrow \{s\}$
{ b- Remove lists subsuming other lists.}
for all $l, l' \in L$ s.t. is_not(l, l') **do**
  if $l \subset l'$ **then**
    $L \leftarrow L \backslash \{l'\}$
return $L^{\min\oslash}_{(s)} = L$

Building a Graph — We have already seen that determining which states can reach a goal state will be done through a back-propagation in an `AND-OR` graph, as illustrated by Fig. 2. To help the graph traversal, we benefi t from Alg. 5 to record which are the parent-states of each of these lists. Yet a list may have several parents, and the traversal also requires knowing for each state in which minimal lists it appears (its parent-lists). To that end, Alg. 6 builds a set of all minimal lists $L^{\min\oslash} = \bigcup_{s\in S} L^{\min\oslash}_{(s)}$, computing at the same time the sets of parents of each list: $parents(l)$ and of each state: $L^{parents}_{(s)}$ in the `AND-OR` graph.

Algorithm 6 ALLMINLISTS ($L^{\min\oslash}_{(\cdot)}$: minimal lists from a given state)

$L^{\min\oslash} \leftarrow \emptyset$
for all $s \in S, l \in L^{\min\oslash}_{(s)}$ **do**
  if $\exists l' \in L^{\min\oslash}$ s.t. $l' = l$ **then**
    $parents(l') \leftarrow parents(l') \bigcup parents(l)$
  else
    $L^{\min\oslash} \leftarrow L^{\min\oslash} \bigcup \{l\}$
    for all $s'' \in parents(l)$ **do**
      $L^{parents}_{(s'')} \leftarrow L^{parents}_{(s'')} \bigcup \{l\}$
return $L^{\min\oslash}, L^{parents}_{(\cdot)}$

Graph Traversal — We now have a complete description of the graph in which to propagate the reachability. As mentionned earlier, this propagation starts from goal states. Alg. 7 shows an implementation of this process using a stack of states to visit. In this algorithm, when a state is recognized as `reaching`, it is removed from all its parent-lists. Then, when such a list is empty (i.e. is `reaching`), all its parent-states can be marked as `reaching`.

Algorithm 7 PROPAGATEREACHABILITYUSSP ($L^{\min\oslash}$, $L^{parents}_{(\cdot)}$)

PUSHALL(G, st) {st: stack of goal states}
while $st \neq \emptyset$ **do**
  POP(s, st)
  if $\neg reaching(s)$ **then**
    MARK$(s,$ `reaching`$)$
    for all $l \in L^{parents}_{(s)}$ **do**
      $l \leftarrow l \backslash \{s\}$
      if $l = \emptyset$ **then**
        PUSHALL$(parents(l), st)$

Alg. 3 can still be used for the danger analysis. Let us just recall that danger analysis differs from reachability analysis as:

- it starts from non-`reaching` states (identifi ed through the reachability analysis), and
- the graph used is not the same: a state s is `dangerous` if, for all action $a \in A(s)$, there exists a dangerous state in $children(s, a)$.

[6] As 2 lists may be the same object **or** may contain the same elements, we use two notations: is(l, l') and $l = l'$.

3.4. How to Obtain the Lists

Knowing how to perform the reachability analysis using the required minimal lists, we will now see more precisely how to obtain them through the indirect process: 1- looking for *maximal* lists of states which *can* be forbidden simultaneously, then 2- adding a state to turn them into *minimal* lists of states which *cannot* be forbidden simultaneously.

To that end, we have defined the notion of "list of states which *can* be forbidden simultaneously" $L^{\odot}_{(s,a)}$ and the subset of these lists which are "maximal" $L^{\max\odot}_{(s,a)}$. We now describe how to obtain $L^{\max\odot}_{(s,a)}$ and then deduce $L^{\min\oslash}_{(s,a)}$.

Maximal Lists — Considering a state-action pair (s, a), the opponent's work consists in distributing a total probability mass of 1 so as to respect the uncertain model and to forbid as many reachable states as possible. A first action is then to put as much probability mass as possible on states which cannot be forbidden and on s itself (if it is reachable), so that as little probability mass as possible remains for other reachable states. This operation is accomplished in the first "`for`" loop of Alg. 8. The remaining states in S' are the states s' which can be forbidden: $Pr^{\min}(s'|s, a) = 0$.

Then, the recursive function FFSA tries all "minimal lists of reachable states in which the remaining probability mass p can be placed".[7] When one such list is found, the remaining states constitute one of the maximal lists we are looking for. This recursive selection process is illustrated by Fig. 3. When the total probability mass 1 has been distributed, FFSA can stop and gather all unused states as a new *maximal list*.

"Minimality" is here guaranteed because states are sorted in decreasing order of $Pr^{\max}(\cdot|s, a)$, and added sequentially to "reachable states" lists following this order. Without this condition, and if the remaining probability mass p were bounded by the maximal probabilities of the two remaining states to consider (s'_1 and s'_2): $Pr^{\max}(s'_1|s, a) < p < Pr^{\max}(s'_2|s, a)$, then trying to distribute p through s'_1 first would lead to also assign $p - Pr^{\max}(s'_1|s, a) > 0$ to s'_2, what would authorize access to both states (whereas s'_1 could be forbidden).

From Maximal to Minimal Lists — A first set of minimal lists from $L^{\min\oslash}_{(s,a)}$ consists of singletons whose state cannot be forbidden ($Pr^{\min}(s'|s, a) > 0$), or $s' = s$. They are built in the first `for` loop of Alg. 9.

Then, all other minimal lists are created by adding a state (that can be forbidden) to a maximal list obtained previously. The main difficulty is to ensure that the lists created are minimal, i.e. do not subsume another generated list. It helps again that reachable states are sorted by decreasing $Pr^{\max}(\cdot|s, a)$. Indeed, for a maximal list $l_f = \{s'_{f(1)}, \cdots, s'_{f(k)}\}$, corresponding minimal lists have to be obtained only by adding states s'_i such that $f(1) < \cdots < f(k) < i$, as done in the second part of Alg. 9.

This condition is necessary and sufficient as, if there exists i such that $i < f(k)$ and $l_f \cup \{s'_i\}$ is a minimal list, then this minimal list will be obtained by adding $s'_{f(k)}$ to $l_f \cup \{s'_i\} \setminus \{s'_{f(k)}\}$, which is one of the maximal lists obtained through Alg. 8.

[7] These minimal lists should not be confused with the ones from $L^{\min\oslash}_{(s,a)}$.

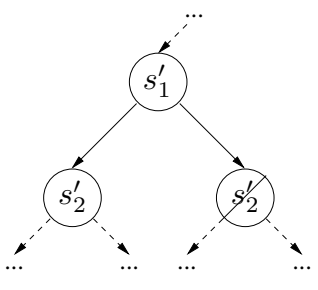
Algorithm 8 FORBIDDENFROMSA (s: state, a: action, $S' = \{s'_1, \ldots, s'_{|S'|}\}$: states reachable from (s, a))

Ensure: Method building the list of maximal sets of states which can be forbidden simultaneously (for a given state-action pair).

$S' \leftarrow$ SORTDECREASING$(S', Pr^{\max}(\cdot|s, a))$
$p \leftarrow 0$
for all $s' \in S'$ s.t. $Pr^{\min}(s'|s, a) \neq 0 \ \vee$ is(s', s) **do**
  $p \leftarrow p + Pr^{\max}(s'|s, a)$
  $S' \leftarrow S' \backslash \{s'\}$
return FFSA$(0, s, a, S', p)$

. . . . . .

FFSA(j: integer,s: state, a: action, S': set of states, p: probability)
$L^{\max\odot}_{(s,a)} \leftarrow \emptyset$
if $p \geq 1$ **then**
  $L^{\max\odot}_{(s,a)} \leftarrow \{S'\}$
else
  for $i = j$ to $|S|$ s.t. $s'_i \in S'$ **do**
    $L^{\max\odot}_{(s,a)} \leftarrow L^{\max\odot}_{(s,a)} \bigcup$ FFSA$(i + 1,\ S' \backslash \{s'_i\},\ p + Pr^{\max}(s'_i|s, a))$
return $L^{\max\odot}_{(s,a)}$



$Pr^{\max}(s'_1|s, a) \geq Pr^{\max}(s'_2|s, a) \geq Pr^{\max}(s'_3|s, a) \geq \cdots$
Note: here we start with state s'_1, assuming that all reachable states can be forbidden.
At each node of this tree is decided whether the following state is kept (s'_2) or not ($\not{s}'_2$).

Figure 3. Tree view of the process selecting unforbidden states in function FFSA **(Alg. 8).**

Algorithm 9 NOTFORBIDDENFROMSA (s: state, a: action, S': states reachable from (s, a), $L^{\max\odot}$: maximal lists of states which can be forbidden simultaneously)

$L \leftarrow \emptyset$
{a- Remove states which cannot be forbidden all alone, and put them as singletons in our resulting meta-list.}
$S'' \leftarrow S'$ {S' ordered as in previous algorithm.}
for all $s' \in S'$ s.t. $Pr^{\min}(s'|s,a) > 0 \ \vee$ is(s', s) **do**
  if is_not(s', s) **then**
    $L \leftarrow L \bigcup \{\{s'\}\}$
  $S'' \leftarrow S'' \backslash \{s'\}$
{b- Loop through the maximal lists of states which can be forbidden simultaneously.}
for all $l \in L^{\max\odot}$ **do**
  for all $s' \in S'$ up to $s' \in l$ **do**
    if $Pr^{\min}(s'|s,a) = 0 \ \wedge$ is_not(s, s') **then**
      $L \leftarrow L \bigcup \{l \cup \{s'\}\}$
return L

4. Application

4.1. Test Problems

During its implementation, the different parts of the complete process have been tested on various test problems. Fig. 4 shows one of the interesting cases, which are often rather difficult to represent. Here, actions a_0 and a_1 have common parent states and lead to similar states, but with different probabilities. Another remark is that s_2 is a dead-end.

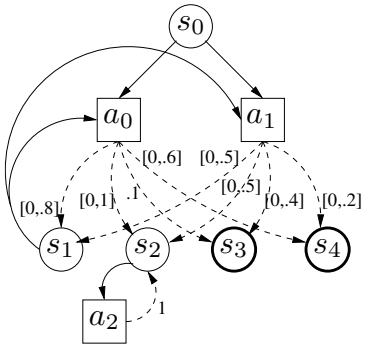


Figure 4. One of the test problems used to check the algorithms. s_3 and s_4 are goal states. Transition costs are not represented.

4.2. Mountain-Car

Problem — We use here the mountain-car problem as defined in[18]: starting from the bottom of a valley, a car has to get enough momentum to reach the top of a mountain (see Fig. 5). The same dynamics as described in the mountain car software[8] have been employed, with the only difference that the left boundary has been moved from -1.2 to -2.0, creating a valley in which the car can be trapped. The objective is to minimize the number of time steps to reach goal.

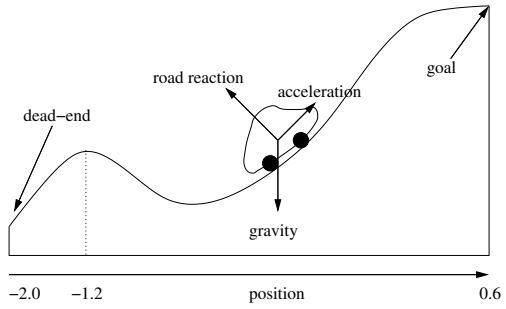


Figure 5. The mountain-car problem with a dead-end.

The continuous state-space is discretized (32×32 grid) and the corresponding uncertain model of transitions is obtained by sampling 1000 transitions from each state-action pair (s, a). For each transition, we computed intervals in which the true model lies with 95% confidence.

Results — In practice, we use LRTDP [5] as an improved version of RTDP with a convergence criterion. With no prior reachability analysis, the algorithm is unable to stop, being stuck in the new valley. Yet, a first reachability analysis does not show any non-`reaching` state. The difficulty comes in fact from a few transition probabilities of very low value ($Pr^{\min}(s'|s,a) < 0.01$) which make it practically impossible to leave the valley. We have therefore decided to consider that such transitions can also be forbidden.

With this new criterion, a second reachability analysis finds a subset of non-`reaching` states, all other states being here `dangerous` (so that no policy can avoid danger). In such a case, there is no way to definitely avoid a dead-end. We can only turn non-`reaching` states into new goal states with a high cost of $10,000$. With this new problem, LRTDP produces the value function from Fig. 6, where a 0-valued corner indicates that this part of the state space was not visited.

5. Conclusion

The goal reachability checked through the algorithm presented here is an essential tool for the robust version of

[8] `http://www.cs.ualberta.ca/~sutton/...MountainCar/MountainCar.html`

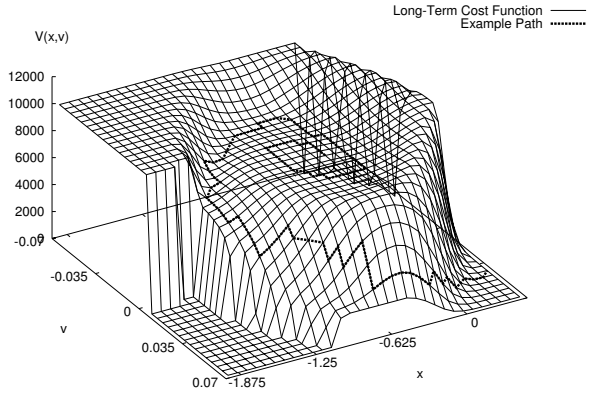


Figure 6. Value function obtained with the help of the reachability analysis.

RTDP we present in [10]. An open question is how to use the information obtained through the reachability analysis. If one does not want to forbid states which are `reaching` and `dangerous`, the cost function is not sufficient for decision-making and a new (non-classical ?) preference criterion has to be introduced.

A first drawback of our approach is the high computation cost. Yet, the uncertain analysis can be preceded by very efficient certain analyses that often decide for most states if they are `reaching` or `dangerous`. This process is detailed in [9] along with experimental results showing a dramatic speed-up.

The main remaining issue is then how to avoid enumerating the complete state-space. In a structured domain, as in temporal planning, it would be of great interest to conduct a symbolic analysis, as it has been done for other purposes for Finite State Automata [12] by using BDDs [7]. The major problem should be the algorithm producing the minimal lists in $L^{\min \oslash}_{(s,a)}$, what would enable a symbolic characterization of the AND-OR graph.

Finally, it is important to notice that the core of the algorithm presented in this document is not specific to decision-making, but rather to uncertain Markov chains (with goal states). It would be simple to rewrite the various procedures to that end, as Markov chains could be described as SSPs with no costs and a single available action per state.

Acknowledgments

National ICT Australia is funded by the Australian Government. This work was also supported by the Australian Defence Science and Technology Organisation.

References

[1] J. Bagnell, A. Y. Ng, and J. Schneider. Solving uncertain markov decision problems. Technical Report CMU-RI-TR-01-25, Robotics Institute, Carnegie Mellon U., 2001.

[2] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72, 1995.

[3] R. Bellman. *Dynamic Programming*. Princeton U. Press, Princeton, New-Jersey, 1957.

[4] D. Bertsekas and J. Tsitsiklis. *Neurodynamic Programming*. Athena Scientifi c, 1996.

[5] B. Bonet and H. Geffner. Labeled rtdp: Improving the convergence of real time dynamic programming. In *Proc. of the 13th Int. Conf. on Automated Planning and Scheduling (ICAPS'03)*, 2003.

[6] C. Boutilier, R. I. Brafman, and C. Geib. Structured reachability analysis for markov decision processes. In *Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence (UAI-98)*, 1998.

[7] R. Bryant. Symbolic manipulation of boolean functions using a graphical representation. In *ACM/IEEE Design Automation*, pages 688–694, 1985.

[8] O. Buffet. Robust (l)rtdp: Reachability analysis. Technical report, National ICT Australia, 2004.

[9] O. Buffet. Fast reachability analysis for uncertain ssps. In *Proc. of the IJCAI 2005 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains*, 2005.

[10] O. Buffet and D. Aberdeen. Planning with robust (l)rtdp. Technical report, National ICT Australia, 2004.

[11] O. Buffet and D. Aberdeen. Robust planning with (l)rtdp. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05)*, 2005.

[12] O. Coudert, J.-C. Madre, and C. Berthet. Verifying temporal properties of sequential machines without building their state diagrams. In *Proc. of the Workshop on Computer-Aided Verification*, 1990.

[13] R. Givan, S. Leach, and T. Dean. Bounded parameter markov decision processes. *Artificial Intelligence*, 122(1-2):71–109, 2000.

[14] M. Hosaka, M. Horiguchi, and M. Kurano. Controlled markov set-chains under average criteria. *Applied Mathematics and Computation*, 120(1-3):195–209, 2001.

[15] R. Munos. Efficient resources allocation for markov decision processes. In *Advances in Neural Information Processing Systems 13 (NIPS'01)*, 2001.

[16] A. Nilim and L. E. Ghaoui. Robustness in markov decision problems with uncertain transition matrices. In *Advances in Neural Information Processing Systems 16 (NIPS'03)*, 2004.

[17] A. L. Strehl and M. L. Littman. An empirical evaluation of interval estimation for markov decision processes. In *Proc. of the 16th Int. Conf. on Tools with Artificial Intelligence (ICTAI'04)*, 2004.

[18] R. Sutton and G. Barto. *Reinforcement Learning: an introduction*. Bradford Book, MIT Press, Cambridge, MA, 1998.