Robust LRTDP: Reachability Analysis

Olivier Buffet
National ICT Australia &
The Australian National University
Canberra, Australia
olivier.buffet@nicta.com.au

December 21, 2004 - Revised April 4, 2005

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, including the more 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.

Contents

1 Introduction

In decision-theoretic planning, Markov Decision Problems [Bertsekas and Tsitsiklis, 1996] 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 physical or natural 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 (themselves being mathematical models), observations of a real system or human expertise.

Working with uncertain models first 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 model probability [Munos, 2001]. The simplest example is sets of possible models that are assumed equally probable [Bagnell *et al.*, 2001; Nilim and Ghaoui, 2004]. Rather than construct a possibly infinite set of models we represent model uncertainty by allowing each probability in a single model to lie in an interval [Givan *et al.*, 2000; Hosaka *et al.*, 2001].

Uncertain probabilities have been investigated in

- resource allocation problems [Munos, 2001], such as efficient exploration [Strehl and Littman, 2004] and state aggregation [Givan *et al.*, 2000], and
- policy robustness [Bagnell *et al.*, 2001; Hosaka *et al.*, 2001; Nilim and Ghaoui, 2004].

We focus on the later, considering a two-player game where the opponent chooses one of 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 all 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) [Barto *et al.*, 1995] 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, it also requires that a goal state is reachable from any visited state, which can be checked through a reachability analysis.

This paper shows how to make the reachability analysis for SSPs, including uncertain ones. 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 Background

2.1 Stochastic Shortest Path Problems

A Stochastic Shortest Path Markov Decision Problem [Bertsekas and Tsitsiklis, 1996] 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 $T(s, a, s') = Pr(s_{t+1} = s' | s_t =$

$s, a_t = a)$. The aim is to optimize a performance measure based on the **cost function** $c : S \times A \times S \to \mathbb{R}^+$.[1]

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** V defined as the expected sum of *costs* to a goal state.

In this paper, we only consider SSPs for planning purposes, with full knowledge of the tuple defining the problem: $\langle S, s_0, G, A, T, c \rangle$. 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 V. *Value iteration* works by computing the value function $V^*(s)$ that gives the expected reward of the optimal policies. It is the unique solution of the fixed point equation [Bellman, 1957]:

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

Updating V with this formula leads to the optimal value function. For convenience, we also introduce the Q-value:

$$Q(s, a) = \sum_{s' \in S} T(s, a, s')[c(s, a, s') + V(s')].$$

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) [Barto *et al.*, 1995]. 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 value function, 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. 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. The (possibly infinite) set of alternative models is denoted $\mathcal{M}$.

[1] 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 $\neg$GOAL(s) **do**
  // *pick best action and update hash*
  a =GREEDYACTION(s)
  VALUE(s)=QVALUE(s, a)
  // *stochastically simulate next state*
  s =PICKNEXTSTATE(s, a)
end while

We follow the approach described in [Bagnell *et al.*, 2001], that consists of finding a policy that behaves well under the worst possible model. This amounts to considering 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 simultaneously chooses a model. A simple process may be used to compute the value function while looking simultaneously for the worst model. It requires that the model's parameters ($T(s, a, s')$ and $c(s, a, s')$ for all s, a and s') are independent, meaning that changing parameters relative to a given transition has no effect on another transition's parameters. Under this assumption, the worst model can be chosen locally when Q is updated for a given state-action pair. As can be seen from Algorithm 2, the worst local model may change while Q-values evolve. Previous updates of reachable states' values may change their relative ordering, changing which outcomes are considered to be worst.

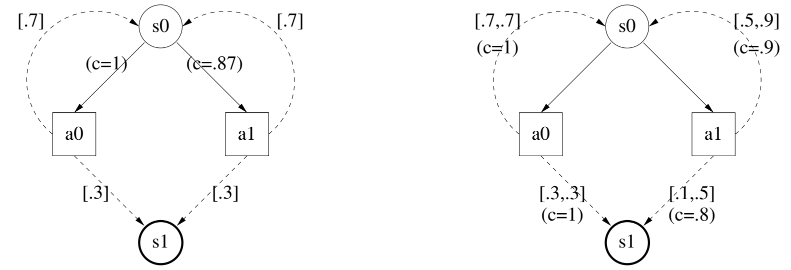
Algorithm 2 Robust Value Iteration (for an SSP)

Initialize V to 0.
repeat
  for all s: state **do**
    for all a: action **do**
      $Q_{\max}(s, a) \leftarrow \max_{m_s^a \in \mathcal{M}_s^a} \sum_{s' \in S} T_{m_s^a}(s, a, s') \left[J(s') + c_{m_s^a}(s, a, s') \right]$
    end for
    $V(i) \leftarrow \min_{a \in A} Q_{\max}(s, a)$
  end for
until V converges

Problem — We are particularly interested in handling *uncertain SSPs* (USSP), where only intervals of possible transition probabilities are known: $T(s, a, s') \in [Pr^{\min}(s'|s, a), Pr^{\max}(s'|s, a)]$.Figure 1 is an example.

For a given state-action pair (s, a), there is a list $R = (s'_1, \cdots, s'_k)$ of reachable states, and for each of them: $T(s, a, s'_i) \in I_i = [p_i^{\min}, p_i^{\max}]$. Thus, possible models are the ones that comply with these interval constraints while ensuring $\sum_i T(s, a, s'_i) = 1$. Figure 2 illustrates this with three reachable states.

Yet, to use RTDP (or a robust RTDP), the following result is of major interest:



a) certain SSP b) uncertain SSP

Figure 1: Two views of one SSP, depending on whether model uncertainty is taken into account (costs in parenthesis). In the uncertain SSP, action a_0 will be prefered as it quickly reaches the goal s_1.

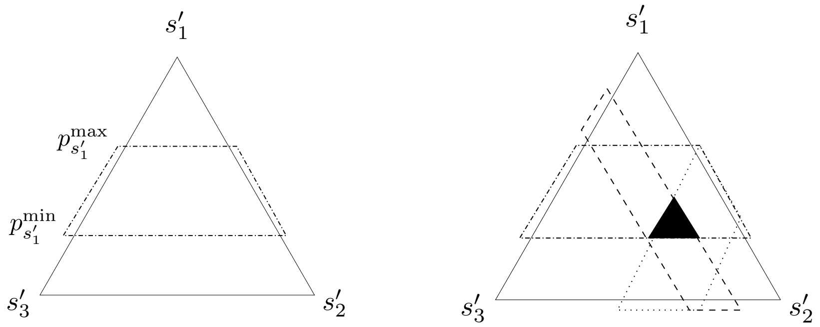


Figure 2: Here, a triangle is a probability simplex representing all possible probability distributions with three different outcomes ($Pr(s'_i) = 1$ at vertex s'_i). On the left triangle is the trapezium showing the interval constraint for s'_1. The right triangle shows possible models at the intersection of the three interval constraints.

Theorem 1. [Bertsekas and Tsitsiklis, 1996] *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 analyse if a goal state is always reachable, in the case of certain as well as uncertain SSPs.

3 Algorithms

Analysing the reachability of a (U)SSP, one may learn if each state can lead:

- to a goal state (noted "`reaching`") or
- to a dead-end (noted "`dangerous`"), i.e. to a subset of non-`reaching` states in which RTDP can be trapped.

Both characteristics may be true at the same time, probabilistic outcomes leading either to termination of the plan or to endless evolution. As dead-ends are known when the search for `reaching` states has been done, the analysis has to be done in two consecutive steps: a "reachability analysis" precedes a "danger analysis". Moreover, these characteristics induce opposite behaviors from the opponent, as it will attempt to prevent the planner from reaching the goal while trying to lead it to a dead-end. This paper mainly focuses on the "reachability analysis", as only this process is specific to USSPs and surprisingly subtle.

3.1 Basic Problem

The reachability analysis depends on the fact that the opponent can 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. 3 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 simultaneously.

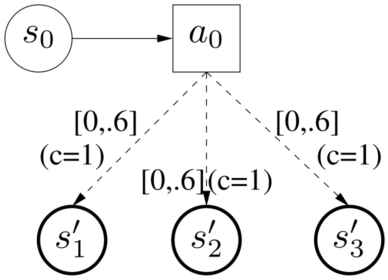


Figure 3: USSP where only 1 of the 3 reachable states can be forbidden (goal states in bold circles).

To know if a given action can lead to a goal state, one has to find such a list of states which cannot be forbidden simultaneously, where all states can lead to a goal state. In this case, the opponent cannot prevent the planner having some chance of terminating. The reachability analysis will therefore be based on the set $L^{\min}_{sa}(s, a)$ of minimal lists of states which cannot be forbidden simultaneously, where "minimal" means that removing any state makes it possible to forbid all states in the list. On Fig. 3: $L^{\min}_{sa}(s_o, a_o) = \{\{s'_1, s'_2\}, \{s'_1, s'_3\}, \{s'_2, s'_3\}\}$.

From this basic idea, two problems have to be solved:

- How to perform the reachability analysis ? (Sec. 3.2)
- How to obtain these lists ? (Sec. 3.3)

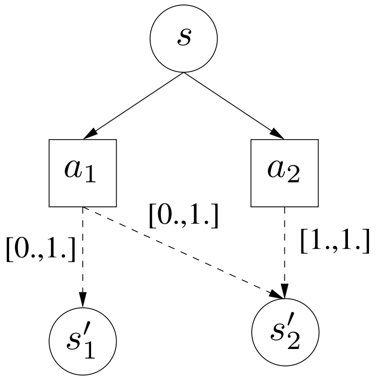
3.2 Performing the Reachability Analysis

From (s, a) to s — Deciding whether a state s may reach a goal according to its children does not require considering all actions separately. Indeed, if there exists an action a and a list $l \in L^{\min}_{sa}(s, a)$ such that all states in l are "`reaching`", then s is `reaching`. So, from sets of minimal lists defined for each state-action pair, we have to create a set of minimal lists for each state (see Fig. 4). This is done in Alg. 3 by simply merging sets: $\bigcup_{a \in A(s)} L^{\min}_{sa}(s, a)$ and removing lists including other lists: if $l \subseteq l'$, then l' is not minimal in our new set of lists.[2]

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

Algorithm 3 NOTFORBIDDENFROMS (s: state, $L_a^{\min}$: minimal lists of states which cannot be forbidden simultaneously from s and an action)

```
L ← ∅
{a- Put all minimal lists obtained in L_s^min.}
for all a ∈ A(s), l ∈ L_a^min(a) do
  if ∀l′ ∈ L, l′ ≠ l then
    L ← L ⋃ {l}
    parents(l) ← {s}
  end if
end for
{ b- Remove lists subsuming other lists.}
for all l, l′ ∈ L s.t. is_not(l, l′) do
  if l ⊂ l′ then
    L ← L\{l′}
  end if
end for
return L
```



In this situation, two actions a_1 and a_2 are possible from state s, each having a single minimal list of states which cannot be forbidden simultaneously ($l_1 = \{s'_1, s'_2\}$ and $l_2 = \{s'_2\}$). As $l_2 \subseteq l_1$, checking if the states in l_2 are `reaching` is sufficient to know if s is `reaching`, since choosing action a_2 will ensure that there is some chance of reaching a goal state.

Figure 4: An example of list of states which cannot be forbidden from a given state.

Building a Graph — Determining which states can reach a goal state will be done through a propagation starting from these goal states. This "back"-propagation takes place in an `AND-OR` graph where nodes are states and the minimal lists, as illustrated by Fig. 5. This is an `AND-OR` graph because a list is "`reaching`" if *all* its child states are `reaching` (`AND`), and a state is `reaching` if *one* of its children lists is `reaching` (`OR`).

To help the graph traversal, we also benefit from Alg. 3 to record that s is a parent state 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. To that end, Alg. 4 builds a set of all minimal lists $L^{\min} = \bigcup_{s \in S} L_s^{\min}(s)$, computing at the same time the sets of parents of each list: $parents(l)$ and of each state: $L_s^{parents}(s)$ in the `AND-OR` graph.

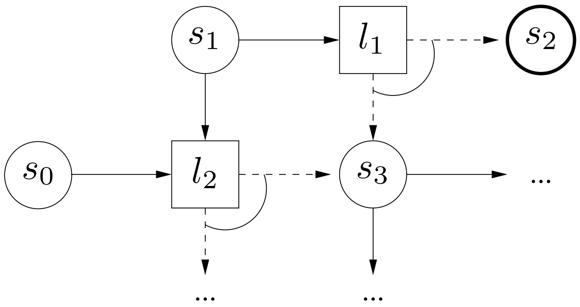


Figure 5: Example of AND-OR graph in which the reachability analysis is done.

Algorithm 4 ALLMINLISTS ($L_s^{\min}$: minimal lists from a given state)

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

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. 5 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 lists it belongs to. When a list is empty, all its parents states can be marked as `reaching`.

Note: In fact, once the AND-OR graph is known, checking if a goal state is reachable from a given state s amounts to a non-deterministic planning problem where:

- a list is considered as an action: when in a state s_i, a list has to be chosen, and
- the outcome of an action is a non-deterministic transition to one of the states in the list.

In our particular case, checking one state at a time if it can reach a goal state is not interesting, since the dangerosity analysis can be performed only if the reachability analysis has been accomplished on all states (or at least all states reachable from s_0).

Danger analysis differs from reachability analysis in that:

- it starts from non-`reaching` states, 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 $children(s, a)$.

Algorithm 5 PROPAGATEREACHABILITY ($L^{\min}$, $L_s^{parents}$)

```
pushAll(G, st) {st: stack of goal states}
while st ≠ ∅ do
   pop(s, st)
   if ¬reaching(s) then
      Mark(s, reaching)
      for all l ∈ L_s^parents(s) do
         l ← l\{s}
         if l = ∅ then
            pushAll(parents(l), st)
         end if
      end for
   end if
end while
```

Moreover, here Alg. 6 is recursive while Alg. 5 is iterative.

Algorithm 6 PROPAGATEDANGER()

```
for all s ∈ S s.t. ¬reaching(s) do
   Mark(s, dangerous)
   for all s'' ∈ Parents(s) do
      FindDangerous(s'')
   end for
end for
.....................................................................
FindDangerous(s: state)
if reaching(s) then
   if ∀a ∈ A, ∃s' ∈ Children(s) s.t. dangerous(s')  then
      Mark(s, dangerous)
      for all s'' ∈ Parents(s) do
         FindDangerous(s'')
      end for
   end if
end if
```

3.3 How to Obtain the Lists

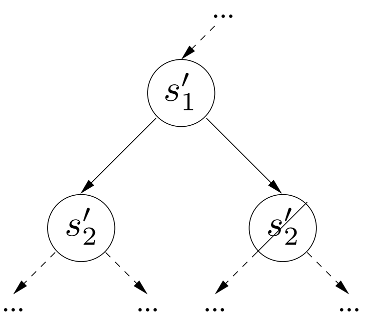
Previous section has detailed how to use the minimal lists mentioned in the introduction of Sec. 3 to perform the reachability analysis. An essential question that we still have to answer is how to obtain these lists. The process adopted is indirect, as it consists in 1- looking for *maximal* lists of 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.

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. 7.

Then, the recursive function FFSA tries all minimal sets of reachable states in which the remaining probability mass p can be placed. Minimality is guaranteed because states are sorted in decreasing order of $Pr^{\max}(\cdot|s,a)$, and added sequentially to "reachable states" lists following this order. This recursive selection process is illustrated by Fig. 6. When the total probability mass 1 has been distributed, FFSA can stop and gather all unused states as a new *maximal list*.

Algorithm 7 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$(S', Pr^{\max}(\cdot|s,a))$
  $p \leftarrow \sum_{s' \in S} Pr^{\min}(s'|s,a)$
  for all $s' \in S'$ s.t. $(Pr^{\min}(s'|s,a) \neq 0) \vee (\text{ is}(s',s))$ **do**
    $p \leftarrow p + Pr^{\max}(s'|s,a) - Pr^{\min}(s'|s,a)$
    $S' \leftarrow S' \backslash \{s'\}$
  end for
  return FFSA$(0, s, a, S', p)$
  . . . .
  FFSA(j: integer,s: state, a: action, S': set of states, p: probability)
  $L^{\max}_{sa} \leftarrow \emptyset$
  if $p^{\max} \geq 1$ **then**
    $L^{\max}_{sa} \leftarrow \{S'\}$
  else
    for $i = j$ to $|S|$ s.t. $s'_i \in S'$ **do**
      $L^{\max}_{sa} \leftarrow L^{\max}_{sa} \bigcup$ FFSA$(i+1, S' \backslash \{s'_i\},$
        $p + Pr^{\max}(s'_i|s,a) - Pr^{\min}(s'_i|s,a))$
    end for
  end if
  return $L^{\max}_{sa}$



$Pr^{\max}(s'_1|s,a) \geq Pr^{\max}(s'_2|s,a) \geq Pr^{\max}(s'_3|s,a) \geq \cdots$
Note: here we start with state s'_1, assuming that all reachable states can be forbidden.

Figure 6: Tree view of the process selecting unforbidden states in function FFSA (Alg. 7).

Alg. 7 is in fact generating all minimal lists of states in which the complete probability mass 1 can be placed, the remaining states being one of the maximal lists we are looking for.

With sorted reachable states, if $l_2 = \{s'_2\}$ is not a sufficient list to contain the 1 probability mass, we should not try to add s'_1 to l_2 as $l_1 = \{s'_1\}$ alone may be one of the minimal lists. On the contrary, s'_2 could be added to l_1 if it needs more states.

From Maximal to Minimal Lists — A first set of minimal lists 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. 8.

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. The solution lies in the fact 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. 8.

Example: Let us suppose we have 5 reachable states s'_1 to s'_5, such that $\forall i Pr^{\min}(s'_i|s,a) = 0$ and $Pr^{\max}(s'_1|s,a) \geq \cdots \geq Pr^{\max}(s'_5|s,a)$. If $l = \{s'_2, s'_4\}$ is a maximal list of states which can be forbidden simultaneously, then:

- Because $\sum_{i\in\{1,3,5\}} Pr^{\max}(s'_i|s,a) \leq \sum_{i\in\{1,3,4\}} Pr^{\max}(s'_i|s,a)$, then $\{s'_2, s'_5\}$ is necessarily another maximal list. A similar reasoning also identifies $\{s'_3, s'_4\}$ and $\{s'_4, s'_5\}$ as maximal lists.
- Following the rule defined, we get:
  - s'_1 is not added to $\{s'_2, s'_4\}$, as s'_1 may be a maximal list all alone, resulting in $\{s'_1, s'_2\}$ being a minimal list (of states which cannot be forbidden simultaneously). There is no guarantee that $\{s'_1, s'_2, s'_4\}$ is minimal.
  - s'_3 is not added to $\{s'_2, s'_4\}$, the same minimal list being obtained by adding s'_2 to $\{s'_3, s'_4\}$.
  - s'_5 is added to $\{s'_2, s'_4\}$ as, in a symetric way, s'_2 (resp. s'_4) will not be added to $\{s'_4, s'_5\}$ (resp. $\{s'_2, s'_5\}$).

Alg. 9 sums up the process followed to analyse the reachability of a USSP.

4 Application

4.1 Mountain-Car

Problem — We use here the mountain-car problem as defined in [Sutton and Barto, 1998]: starting from the bottom of a valley, a car has to get enough momentum to reach the top of a mountain (see Fig. 7). The same dynamics as described in the mountain car software[3] 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(see also [Buffet and Aberdeen, 2004]). The objective is to minimize the number of time steps to reach goal.

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

Algorithm 8 NOTFORBIDDENFROMSA (s: state, a: action, S': states reachable from (s, a), $L^{\max}$: maximal lists of states which can be forbidden simultaneously)

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

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(cf. [Buffet and Aberdeen, 2004] Appendix B.1).

Results — In practice, we use LRTDP [Bonet and Geffner, 2003] 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 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

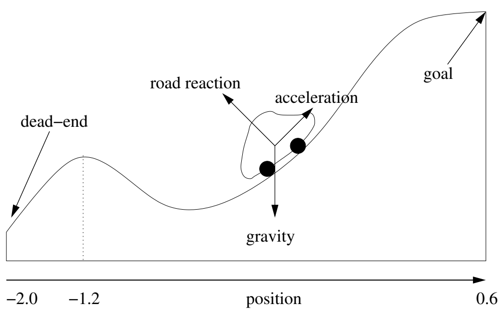


Figure 7: The mountain-car problem with a dead-end.

Algorithm 9 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**
    $L^{\max}_{sa}(s, a) \leftarrow$ FORBIDDENFROMSA $(s, a, Children(s, a))$
    $L^{\min}_{sa}(s, a) \leftarrow$ NOTFORBIDDENFROMSA $(s, a, Children(s, a), L^{\max}_{sa}(s, a))$
  end for
  $L^{\min}_{s}(s) \leftarrow$ NOTFORBIDDENFROMS$(s, L^{\min}_{sa}(s, \cdot))$
end for
$(L^{\min}, L^{parents}_{s}) \leftarrow$ ALLMINLISTS$(L^{\min}_{s})$
PROPAGATEREACHABILITY$(L^{\min}, L^{parents}_{s})$
PROPAGATEDANGER$()$

practically impossible to leave the valley. We have therefore decided to consider that such transitions can also be forbidden.

With this new criterion, all states are here `dangerous`. We turn non-`reaching` ones into new goal states with a high cost of $10,000$. With this new problem, LRTDP produces the value function from Fig. 8, where a 0-valued corner indicates that this part of the state space was not visited.

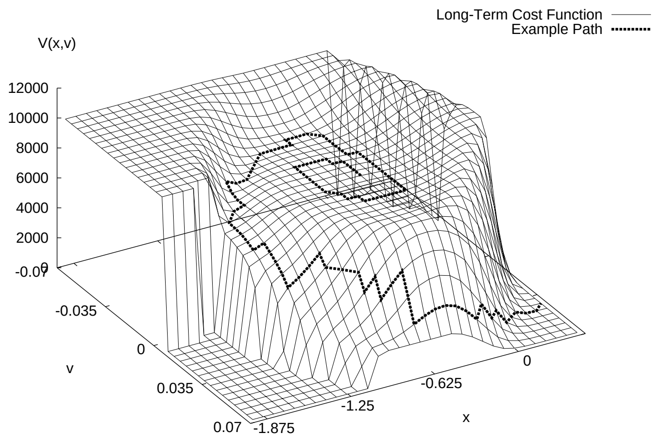


Figure 8: Value function obtained with the help of the reachability analysis.

The reachability analysis is nearly instantaneous in this case. Most of the computation time is used by LRTDP itself.

4.2 Sailing

Problem — The sailing problem used here shares similarities with the mountain-car task. It's complete description can be found in [Vanderbei, 1996], and another use in [Peret and Garcia, 2004]. Here, the space is discretized to a 10×10 grid, $\times 8$ wind angles and $\times 8$ possible headings. The uncertainty of the system is due to the stochastic changes in the wind's direction. The uncertain model is also learned by drawing 1000 samples for each state-action pair, using the same $\alpha = 0.05$.

Results — Here, the analysis is much more expensive (whereas LRTDP is very fast to converge). This is clearly due to the fact that this second problem not only has more states, but also more actions and more reachable states per state-action pair. Table 2 gives for each problem: number of states, number of actions, time to create the model (including the statistical computation of interval-probabilities), time to perform the reachability analysis, and time for rLRTDP to converge. All times are in seconds (on a 2.8GHz P4 with 512Mo of memory).

	mountain-car	sailing
$\#_S$	1024	6400
$\#_A$	2	8
Init	0.7780	5.8647
Reach$^{\text{ability}}$	0.2810	167.7658
rLRTDP	10.6862	1.4320

Table 1: Performance (speed) of the various algorithms. "Init" is the phase where the model is built (including the statistical modeling).

5 Conclusion

The goal reachability checked through the algorithm presented here is an essential tool for the robust version of RTDP we present in [Buffet and Aberdeen, 2004]. 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.

How to use reachability and danger analyses amounts to a risk management issue: Do we want to necessarily avoid dead-ends ? What compromise could be done ? If even the start state is `dangerous`, it may be infeasible to completely avoid dead-ends. Here are some options:

- If in a non-`dangerous` state, simply optimize while forbidding actions that could lead to a dangerous state.
- If in a `dangerous` (but still `reaching`) state, try to reach a non-dangerous state with the highest probability.
- If there is very few non-`dangerous` states (as in our experiment), turn dead-ends into goal states with some very high cost.
- Introduce a non-classical preference criterion making a compromise between danger-avoidance and low expected cost to the goal.

Yet, the main issue is how to avoid enumerating the complete state-space. In a structured domain, it would be of major interest to conduct a symbolic analysis, maybe with BDDs [Bryant, 1985], what first requires symbolically characterizing the AND-OR graph used. But practical use will require examining the complexity of the resulting algorithm.

As mentionned in a previous "box" in Section 3.2, finding if a goal state is reachable from a state s in the AND-OR graph amounts to a non-deterministic planning problem. Thus, if this graph is represented in a symbolic way (such as BDDs), model-checking could be used to make the reachability analysis of a given state.

The main problem is then, starting with our uncertain graph, to produce the symbolic 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.

References

[Bagnell *et al.*, 2001] J.A. Bagnell, A. Y. Ng, and J. Schneider. Solving uncertain markov decision problems. Technical Report CMU-RI-TR-01-25, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2001.

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

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

[Bertsekas and Tsitsiklis, 1996] D.P. Bertsekas and J.N. Tsitsiklis. *Neurodynamic Programming*. Athena Scientific, 1996.

[Bonet and Geffner, 2003] B. Bonet and H. Geffner. Labeled rtdp: Improving the convergence of real time dynamic programming. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS'03)*, June 2003.

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

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

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

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

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

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

[Peret and Garcia, 2004] L. Peret and F. Garcia. On-line search for solving markov decision processes via heuristic sampling. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'2004)*, 2004.

[Strehl and Littman, 2004] A. L. Strehl and M. L. Littman. An empirical evaluation of interval estimation for markov decision processes. In *Proceedings of the Sixteenth International Conference on Tools with Artificial Intelligence (ICTAI'04)*, 2004.

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

[Vanderbei, 1996] Robert J. Vanderbei. Optimal sailing strategies, statistics and operations research program, 1996. University of Princeton, http://www.sor.princeton.edu/˜rvdb/sail/sail.html.

A Improved Reachability Analysis (in progress)

Applying the algorithms (reachability and danger analyses) designed for uncertain SSPs is time consuming whereas in many cases, only a small part of the model requires a special treatment.

What we propose here is to apply the certain counterpart of these algorithms on an optimistic and a pessimistic model first, to quickly obtain a classification of most states. Then, the uncertain algorithms only need to be run on unclassified (uncertain ?) states. As detailed below, this process can be viewed as lower- and upper-bouding a solution with simple technics before using an exact –but costly– computation.

A.1 Upper- and Lower-Bounding Reachability Graphs

The precomputation phases work on two reachabilitu graphs obtained from the original uncertain SSP:

- **the lower-bounding reachability graph G_{lo}:** in which s' is reachable from s if and only if there exist an action a such that $Pr^{\min}(s'|s,a) > 0$, and
- **the upper-bounding reachability graph G^{up}:** in which s' is reachable from s if and only if there exist an action a such that $Pr^{\max}(s'|s,a) > 0$.

G_{lo} represents all transitions which are certainly valid, and G^{up} represents all transitions which could be valid. Yet, these graphs should not be seen as an "optimistic" and a "pessimistic" graph, as the point of view may differ depending on which analysis is being performed.

A.2 Principle

The optimistic, pessimistic and exact-computation phases are the following:

1. **optimistic:**
   (a) use G^{up} to perform a certain reachability analysis and get states which *may be* `reaching` (and subsequently those certainly not-`reaching`), and
   (b) use G_{lo} to perform a danger analysis and get states which *are certainly* `dangerous`.
2. **pessimistic:**
   (a) use G_{lo} to perform a certain reachability analysis and get states which *are certainly* `reaching`, and
   (b) use G^{up} to perform a certain danger analysis and get states which *may be* `dangerous`.

3. **exact-computation:** To complete the analyses, two graphs must be designed which embed states not yet certainly `reaching` (or `dangerous`) and their direct children. Then can be performed:
   (a) the construction of the required AND-OR graph,
   (b) the reachability analysis (starting with states known to potentially reach a goal), and
   (c) the danger analysis (starting with states known to be trapped).

Here, one could say that the planner is optimistic when the opponent is pessimistic (and conversely), what the explain the inverted use of G_{lo} and G^{up} with the reachability and danger analyses. The former tells whether the planner has some hope to reach a goal state, and the later tells if the opponent has some hope to definitely avoid a goal state.

Some Implementation Details — In the various steps described above, the danger analysis in the lower-bouding phase 2.(b) is in fact useless: uncertain information on the danger of a state has no interest.

A second remark is that this complete process requires a three-state logic telling if a `property` is true, false or unknown.

A.3 Algorithms' Complexities

Overview — Due to the number of independent algorithms in the uncertain reachability analysis, it is a difficult task to give its algorithmic complexity with some confidence. We list below the main problem's parameters which appear to be important for this algorithmic complexity.

Here are some notations used to compute the complexity of the various algorithms:

- $|S|$: number of states,
- $|A|$: maximum number of actions ($\max_{s \in S} |A(s)|$),
- b_a: maximum branching-factor for a state-action pair (i.e. maximum number of reachable states from any state-action pair),
- b: maximum branching-factor for a state (i.e. maximum number of reachable states from a state, considering all actions),
- b_p: maximum "reverse" branching-factor for a state (i.e. maximum number of parents for a state).

Sketched Computations of Algorithmic Complexities — With these notations, we have the following worst-case complexities for algorithms used with certain SSPs:

- constructing $Parents(\cdot)$ for a certain SSP: $O(|S|.|A|.b_a)$,
- certain reachability analysis (Alg. **??**): $O(|S|.b_p)$, and
- danger analysis (Alg. 6): $O(|S|.b_p)$.

And here are worst-case complexities for the specific reachability analysis for uncertain SSPs (constants are noted k_i):

- ForbiddenFromSA: $O(k_1.b_a.\log(b_a) + k_2.b_a + k_3.2^{b_a})$
  The number of lists obtained is upper-bounded by $L_1 = \frac{b_a!}{[b_a/2]!.[b_a/2]!}$.

- NotForbiddenFromSA: $O(k_1.b_a^2 + k_2.L_1.b_a)$
  The number of lists obtained is upper-bounded by $L_2 = L_1.b_a$.
- NotForbiddenFromS: $O(k_1.|A|.L_2 + k_2.{L_2}^3)$
  The number of lists obtained is upper-bounded by $L_3 = |A|.L_2$.
- AllMinLists: $O(k_1.|S|.{L_3}^2.b_p)$ The number of lists obtained is upper-bounded by $L_4 = |S|.L_3$.

The first two algorithms for uncertain SSPs are called $|S|.|A|$ times, the third one $|S|$ times, and the last one once. Afterwards are executed both the reachability and dangerosity analyses, on graphs of size $|S| + L_4$ and $|S|$. The preprocessing only has an effect on $|S|$, as it aims at quickly determining (with analyses in certain cases) for most states if they are `reaching` or `dangerous`.

While the three algorithms for certain SSPs remain reasonable, the "uncertain" reachability analysis creates many lists of states (often singletons) and performs various manipulations on them. This easily leads to a high increase in complexity as can be observed through the complexities we have just sketched. Yet, due to the number of independent steps in the uncertain reachability analysis, it is a difficult task to give its algorithmic complexity through a single formula.

A.4 Experiments

As we have just seen, the various branching factors play here a noticeable role. This, and the important number of available actions, may explain the dramatic increase in observed computation time in the sailing problem, as shown on Table 2, column "sailing"-"raw". Yet, the preprocessing obviously helps quickly determining for most states if they are `reaching` or not, hence the huge speed-up observed for each problem's reachability analysis (columns "help"). In both problems, most of the state space is effectively handled through the certain analyses, only a small part depending on "uncertain" dynamics.

	mountain-car		sailing	
$\lvert S\rvert$	1024		6400	
$\lvert A\rvert$	2		8	
	raw	help	raw	help
Init	0.7780	0.7801	5.8647	5.8670
Reach^ability	0.2810	0.0277	167.7658	0.4468
rLRTDP	10.6862	10.6447	1.4320	0.5442

Table 2: Average performance (duration in seconds) obtained with 100 executions for the 3 phases: 1- model `Init`ialization (including the statistical modeling), 2- `Reach`ability analysis and 3- `rLRTDP` itself.
("raw"= "no preprocessing", "help"= "with preprocessing")

A surprising observation is that rLRTDP is much faster on the sailing problem when a preprocessing phase is used. This may be linked to the fact that the computer has no problem handling memory in this case, what may slow down rLRTDP if used after the expensive reachability analysis on a complete uncertain graph. The same experiment on a lake of 4×4 instead of 10×10 shows little difference between both cases: without (0.0161s) and with (0.0197s) preprocessing.