Fast Reachability Analysis for Uncertain SSPs

Olivier Buffet
National ICT Australia &
The Australian National University
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, what can be checked easily for a certain SSP, and with a more complex algorithm for an *uncertain* SSP, i.e. where only a possible interval is known for each transition probability. This paper makes a simplified description of these two processes, and demonstrates how the time consuming uncertain analysis can be dramatically speeded up. The main improvement still needed is to turn to a symbolic analysis in order to avoid a complete state-space enumeration.

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] — investigating 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 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) [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 [Buffet and Aberdeen, 2004; 2005], it also requires that a goal state is reachable from any visited state, which can be checked through a reachability analysis.

This paper makes a simple description of the reachability analysis for certain and uncertain SSPs [Buffet, 2004], and shows how the time consuming uncertain analysis can be dramatically speeded up. In Section 2 we present SSPs, RTDP and robustness. We then explain the algorithms for reachability analysis in the certain and uncertain case. Finally, the fast uncertain reachability analysis is depicted and practical experiments are 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]

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

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 $\pi : S \rightarrow \Pi(A)$ 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 only inaccurate knowledge of the transition function T. 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]:

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

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 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.

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
  $a$ =GREEDYACTION($s$)
  $J(s)$ =QVALUE($s, a$)
  $s$ =PICKNEXTSTATE($s, a$)
end while
```

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 $\mathcal{M}$.

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 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. [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 determine from which states a goal state is still reachable in SSPs. The uncertain case could be brought back to the certain case by finding an appropriate pessimistic model. To that end, our policy should be fixed to one that chooses all actions with equal probability and the opponent could then learn a model to prevent goal states from being reached. Yet, the opponent's 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.

3 Reachability Analyses

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 $(s',a) \rightarrow s$: $Parents(s) = \{s' \in S \text{ s.t. } \exists a \in APr(s|s',a) > 0\}$. 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).[2]

Algorithm 2 PROPAGATEREACHABILITYSSP ($Parents$)

```
pushAll($G, st$) {$st$: stack of goal states}
while $st \neq \emptyset$ do
  pop($s, st$)
  if $\neg reaching(s)$ then
    Mark($s$, reaching)
    pushAll($Parents(s), st$)
  end if
end while
```

Algorithm 3 PROPAGATEDANGER($Parents$)

```
for all $s \in S$ s.t. $\neg reaching(s)$ do
  push($s, st$)
end for
while $st \neq \emptyset$ do
  pop($s, st$)
  if $\neg dangerous(s)$ and $\forall\, a \in A$ :
  $\exists\, s' \in S$ s.t. $Pr(s'|s,a) > 0$ & $dangerous(s')$ then
    Mark($s$, dangerous)
    pushAll($Parents(s), st$)
  end if
end while
```

3.2 Uncertain SSP

In an uncertain SSP, 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. 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.

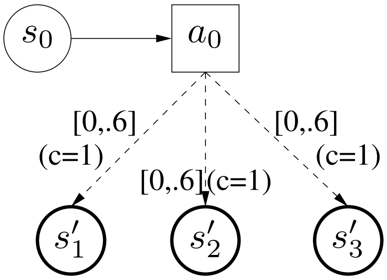

Figure 1: USSP where only 1 of the 3 reachable states can be forbidden (goal states in bold circles).

Let us define the set of all lists of states which cannot be forbidden simultaneously (from (s,a)):[3]

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

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:

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

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 ?
- How to obtain these lists ?

We now just give a brief idea of the answers to these two questions (details in Sections 3.2 and 3.3 of [Buffet, 2004]).

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

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

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

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).

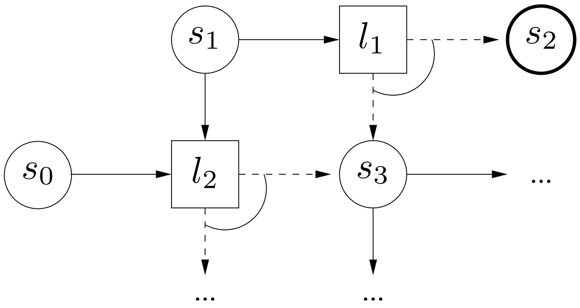

Figure 2: Example of AND-OR graph in which the reachability analysis is done (starting with goal states as s_2 here). If s_3 is reaching, then so is l_1 (the opponent cannot forbid s_2 and s_3), and therefore s_1.

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.

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":

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

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

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

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 $L^{\min\oslash}_{(s,a)}$.

Experiments — The various algorithms developped to perform the reachability and danger analyses have been developped and tested on several problems (see [Buffet, 2004]). The three main remarks coming from these experiments are the following:

1. In some problems, $Pr(s'|s,a) = \epsilon$ can be sufficient to consider that transition $(s,a) \rightarrow s'$ can be forbidden (because of "attracting" parts of the state space which nearly behave like dead-ends).
2. The analyses require enumerating the whole state-space, whereas this is often not feasible. This is all the more unfortunate that one of (L)RTDP's main advantage is to avoid visiting the complete state-space.
3. The reachability analysis for uncertain SSPs can be very time consuming.

The second point is a major subject for future work, with the idea that we should turn our algorithms into a symbolic analysis. Next section shows how to easily address the third point through a simple preprocessing phase.

4 Improved Reachability Analysis

The improved algorithm we propose here is based on the idea that, if the reachability analysis for uncertain SSPs is time consuming, in many cases only a small part of the model requires a special treatment. A lot of information can already be obtained through analyses performed on chosen certain SSPs.

More precisely, we apply the certain reachability and danger analyses on an optimistic and a pessimistic model first, to quickly classify most states. Then, the uncertain algorithms only need to be run on states which remain unclassified. 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.

4.1 Upper- and Lower-Bounding Reachability Graphs

The precomputation phases work on two reachability 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 exists 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 exists 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.

4.2 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 danger analysis and get states which *may be* `dangerous`.~~ (useless step)

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 explains 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.

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

4.3 Algorithms' Complexities

Here is a list of the most important parameters with respect to the algorithmic complexities 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,
- b_p: maximum "reverse" branching-factor for a state (i.e. maximum number of parents for a state).

With this, we have the following worst-case complexities:

- constructing $Parents(\cdot)$ for a certain SSP: $O(|S|.|A|.b_a)$,
- certain reachability analysis (Alg. 2): $O(|S|.b_p)$, and
- danger analysis (Alg. 3): $O(|S|.b_p)$.

While these three algorithms 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. Due to the number of independent steps in the uncertain reachability analysis, it is a difficult task to give its algorithmic complexity through a formula. A good intuition can be obtained by computing the complexity of the various steps of this complex algorithm, as done in [Buffet, 2004], Appendix A.

The preprocessing quickly determines for most states if they are `reaching` or `dangerous`. This results in largely reducing the number of unidentified states which require an uncertain reachability analysis, therefore cutting down the complexity of this last algorithm.

5 Experiments

Problems — Experiments have been conducted on two different problems:
• One is 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. 3). The same dynamics as described in the mountain car software[4] 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.

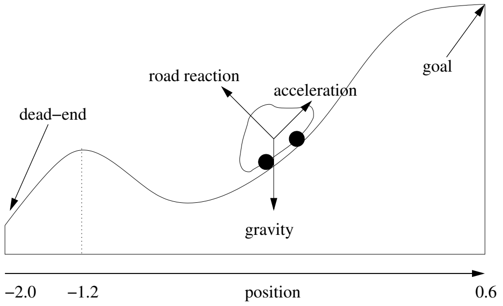

Figure 3: The mountain-car problem with a dead-end.

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 (cf. [Buffet and Aberdeen, 2004] Appendix B.1).
• The other is a **sailing** problem sharing some similarities with the mountain-car task. It's complete description can be found in [Vanderbei, 1996]. Here, the space is discretized to a 10×10 grid, $\times 8$ wind angles and $\times 8$ possible headings. The system's stochasticity is due to the random changes in the wind's direction. If there is here no true dead-end, rLRTDP is easily trapped in some parts of the state-space, forcing us to consider that a transition with probability $Pr^{\min}(s'|s,a) < 0.01$ can be forbidden. The uncertain model is also learned by drawing 1000 samples for each state-action pair, using the same $\alpha = 0.05$.

Results — As we have just seen, branching factors play 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 1, 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,

[4] `http://www.cs.ualberta.ca/˜sutton/··· MountainCar/MountainCar.html`

most of the state space is effectively handled through the certain analyses, only a small part depending on "uncertain" dynamics.

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

Table 1: Average performance (duration in seconds) obtained with 100 executions for the 3 phases: 1- model Initialization (including the statistical modeling), 2- Reachability 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.

6 Conclusion

The goal reachability checked through the algorithm presented here is an essential tool for robust RTDP [Buffet and Aberdeen, 2004; 2005]. This paper briefly describes how to perform reachability and danger analyses in certain and uncertain SSPs, and explains how the analyses for uncertain SSPs can be speeded up through a simple preprocessing phase.

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.

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 [Coudert *et al.*, 1990] by using BDDs [Bryant, 1985]. The major problem should be the algorithm producing the minimal lists in $L^{\min \emptyset}_{(s,a)}$, what would enable a symbolic characterization of the AND-OR graph.

Finally, it is important to notice that the core of the algorithms presented in this document is not specific to decision-making, but rather to certain and uncertain Markov chains (with end 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

[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 U., 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 U. Press, Princeton, New-Jersey, 1957.

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

[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, 2004.

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

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

[Coudert *et al.*, 1990] 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.

[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.

[Strehl and Littman, 2004] 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.

[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. U. of Princeton, http://www.sor.princeton.edu/˜rvdb/sail/sail.html.