New Grid-Based Algorithms for Partially Observable Markov Decision Processes: Theory and Practice

Blai Bonet BONET@CS.UCLA.EDU
Cognitive Systems Laboratory, Department of Computer Science,
University of California, Los Angeles, CA 90024 USA

Abstract

We present two new algorithms for Partially Observable Markov Decision Processes (POMDPs). The first algorithm is a general grid-based algorithm for POMDPs with theoretical optimality guarantees. The other algorithm is for the subclass of problems known as Stochastic Shortest-Path problems in belief space. Both algorithms are optimal and robust with respect to a novel robustness criterion that is also in the paper. In the practical side, we test the approach over a number of diverse problems.

1. Introduction

The ability to model and solve general sequential decision tasks is of one of main goals in Artificial Intelligence. Through the years different frameworks had been proposed for achieving this goal under different set of assumptions. For example, under deterministic actions and complete information, we encounter the well-known classical heuristic search setting (Pearl, 1983), in stochastic environments with complete information we find Bayesian networks and influence diagrams (Pearl, 1988; Howard & Matheson, 1984), etc. In the recent past, AI turned attention to the theory of Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs), developed mainly within Operations Research, for three important reasons. First, these theories provide the most general and clean setting for representing real-life problems which have large state spaces and complex transition dynamics and costs structures. Second, they also provide sound mathematical foundations for a diversity of learning algorithms developed in the Reinforcement and Machine Learning communities (Sutton & Barto, 1998; Bertsekas & Tsitsiklis, 1996), and third, general and efficient algorithms for solving MDPs had been developed, the most important being Value Iteration and Policy Iteration.

The Markov Decision theory deals with *dynamical systems* that evolves in discrete time and are controlled by an agent. The system transition dynamics and costs are described by general probabilistic laws that depend in the applied control and the state of the system. This general structure allows the description of complex sequential decision problems into the MDP model. The task of the agent or computer is to find an optimal control strategy, which maps states to controls, that minimizes the total expected cost.

Partially Observable Markov Decision Processes are an extension of MDPs in which the agent does not know the state of the system. This is an important departure from the MDP model since the optimal strategy for the underlying MDP is useless if the state is not known. The POMDP framework also extends the MDP framework by allowing controls to return information about the system, e.g. performing a blood test returns one result among many possibilities. Thus, in the POMDP setting, the agent needs to keep track of the history of the system in order to estimate its current state. Interestingly, it is known that is better for the agent to maintain estimates in the form of *probability distributions* instead of history records. These probability distributions, also known as *belief states*, summarize all the past information and allow the agent to compute exact probabilities about any future evolution of the system. Thus, the POMDP problem is to find an optimal control strategy that maps belief states to controls; a solution form that is obviously much more complex than in the case of an MDP. Indeed, it is known that solving a POMDP is NP-HARD while solving an MDP is tractable.

As the title suggests, the contributions of this paper are both in theory and practice. In the theoretical side, we present two new algorithms for POMDPs that might solve larger problem instances than the current best optimal algorithms. The new algorithms belong to the class of grid-based algorithms and are not only optimal but also robust in a novel sense. More specificly, the first algorithm is a general algorithm for POMDPs while the second algorithm is for the subclass of tasks known as Stochastic Shortest-Path problems in belief space. This is an important subclass that can be used to model diverse tasks as robot navigation and localization problems, stochastic games, medical diagnosis, and general planning problems under uncertainty and partial information.

In the practical side, we performed experiments over a number of benchmark problems. The benchmark include some small problems that had been used before by others and some larger problems from ourselves. The results show that our approach can be succesfully applied to some problems involving large state spaces.

The paper, which is an extension of previous work in (Bonet, 2002) and (Bonet & Geffner, 2001), is organized as follows. In the next Section, we give formal definitions for MDPs and POMDPs, review the best current algorithms and present a novel robustness criterion. In Section 3, we show new mathematical results about POMDPs that allow us to obtain the algorithms. The first algorithm is presented in Section 4 together with its complexity and optimality guarantees. In Section 5, we define the class of Stochastic Shortest-Path problems in belief space and present the second algorithm. The experimental results are shown in Section 6. The paper finishes with a conclusions Section and with an Appendix that contains the description of some of the larger problems.

2. Preliminaries

This section contains a brief review of MDPs, POMDPs and the algorithms for solving them. We use notation and presentation style close to that in (Bertsekas, 1995); the reader is referred there for an excellent exposition of the field.

The MDP model assumes the existence of a stochastic physical system that evolves in discrete time and is controlled by an agent. At every time point, the agent applies a control and incurs in a cost that depends in the state of the system and the control. The task is to find a control strategy (also known as a policy) that minimize the expected total cost over the *infinite horizon* time setting. Formally, an MDP is defined by

(M1) a finite state space $S = \{1, \ldots, n\}$,

(M2) a finite set of controls $U(i)$ for each state $i \in S$, (M3) transition probabilities $p(i,u,j)$ for all $u \in U(i)$ and $i \in S$ equal to the probability of the next state being j after applying control $u \in U(i)$ in state i, and

(M4) a cost $g(i,u)$ associated to control $u \in U(i)$ and state i.

A strategy or policy π is an infinite sequence $(\mu_0, \mu_1, \dots)$ of decision functions where μ_k maps states to controls so that the agent applies control $\mu_k(i)$ in state $x_k = i$ at time k, the only restriction being $\mu_k(i) \in U(i)$ for all $i \in S$. If $\pi = (\mu, \mu, \dots)$ the policy is called *stationary* (i.e. the control does not depend on time) and it is simply denoted by μ. The cost associated to policy π when the system starts at state x_0 is defined as[1]

$$J_\pi(x_0) \overset{\text{def}}{=} \lim_{N\to\infty} E\left\{ \sum_{k=0}^{N-1} \alpha^k g(x_k, \mu_k(x_k)) \right\}$$

where the x_k's are random variables for the state of the system at time k that are distributed according to the measure induced by the transition probabilities and the policy π. The number $\alpha \in [0,1]$, called *discount factor*, is used to discount future costs at a geometric rate.

The MDP *problem* is to find an *optimal policy* π^* satisfying

$$J^*(i) \overset{\text{def}}{=} J_{\pi^*}(i) \le J_\pi(i), \quad i = 1, \dots, n,$$

for every other policy π. Although there could be none or more than one optimal policy, the optimal cost vector J^* is always unique. The existence of π^* and how to compute it are non-trivial mathematical problems. However, when $\alpha < 1$ the optimal policy always exists and, more important, there exists a stationary policy that is optimal. In that case, J^* is the unique solution to the *Bellman Optimality* equations

$$J^*(i) = \min_{u \in U(i)} \left\{ g(i,u) + \alpha \sum_{j=1}^{n} p(i,u,j) J^*(j) \right\}, \quad i = 1, \dots, n. \tag{1}$$

If J^* is a solution for (1) then the *greedy* stationary policy μ^* with respect to J^*, i.e.

$$\mu^*(i) \overset{\text{def}}{=} \operatorname*{argmin}_{u \in U(i)} \left\{ g(i,u) + \alpha \sum_{j=1}^{n} p(i,u,j) J^*(j) \right\},$$

is an optimal policy for the MDP. Therefore, solving the MDP problem is equivalent to solving (1). Such equation can be solved by considering the DP *operators* T_μ and T:

$$(T_\mu J)(i) \overset{\text{def}}{=} g(i, \mu(i)) + \alpha \sum_{j=1}^{n} p(i, \mu(i), j) J(j),$$

$$(TJ)(i) \overset{\text{def}}{=} \min_{u \in U(i)} \left\{ g(i,u) + \alpha \sum_{j=1}^{n} p(i,u,j) J(j) \right\}$$

1. When the limit is not known to exist it is replaced by the inferior or superior limit.

that map n-dimensional vectors to n-dimensional vectors. When $\alpha < 1$, it is not hard to show that such operators are *contraction mappings* with unique fix points J_μ and J^* satisfying[2]

$$\begin{aligned} J_\mu &= T_\mu J_\mu = \lim_{k\to\infty} T_\mu^k J, \\ J^* &= TJ^* = \lim_{k\to\infty} T^k J \end{aligned}$$

where J is any n-dimensional vector.

The Value Iteration algorithm computes J^* iteratively by using (1) as an update rule: starting from any vector J, the algorithm computes a succession of vectors $\{J_k\}_{k\geq 0}$ as

$$\begin{aligned} J_0 &\overset{\text{def}}{=} J, \\ J_{k+1} &\overset{\text{def}}{=} TJ_k. \end{aligned}$$

The algorithm stops when $J_{k+1} = J_k$, or when the residual $\max_{i\in S} |J_{k+1}(i) - J_k(i)|$ is sufficiently small. In the latter case, when $\alpha < 1$, the suboptimality of the resulting policy is bounded by a constant multiplied by the residual.

2.1 Partially Observable Markov Decision Processes

The POMDP framework was first studied in the Operations Research community (Astrom, 1965; Sondik, 1971; Lovejoy, 1991), yet it had attracted a lot of recent attention from the AI community. A good introduction to POMDPs from the AI perspective can be found in (Cassandra, Kaelbling, & Littman, 1994; Kaelbling, Littman, & Cassandra, 1999). A POMDP is characterized by:

(P1) A finite state space $S = \{1, \ldots, n\}$,

(P2) a finite set of controls $U(i)$ for each state $i \in S$,

(P3) transition probabilities $p(i, u, j)$ for all $u \in U(i)$ and $i \in S$ equal to the probability of the next state being j after applying control $u \in U(i)$ in state i,

(P4) a finite set of observations $Z(i, u) \subseteq Z$ that may result after applying control $u \in U(i)$ in state i,

(P5) observation probabilities $p(i, u, o)$ for all $u \in U(i)$ and $o \in Z(i, u)$ equal to the probability of getting observation o in state i after applying control $u \in U(i)$, and

(P6) a cost $g(i, u)$ associated to control $u \in U(i)$ and state i.

It can be shown that finding an optimal strategy for the system specified in (P1)–(P6) is equivalent to finding an optimal strategy for the associated MDP problem in *belief space*, the so-called belief-MDP, whose basic elements are:

(B1) A belief space B of probability distributions over S,

2. An operator T is a contraction mapping if there is $0 \leq \alpha < 1$ such that $\|TJ - TJ'\| \leq \alpha\|J - J'\|$ for all J, J' in the domain of T.

(B2) a set of controls $U(x) \overset{\text{def}}{=} \{u : \forall i[x(i) > 0 \Rightarrow u \in U(i)]\}$ for each belief $x \in B$, and

(B3) a cost $g(x,u) \overset{\text{def}}{=} \sum_{i=1}^{n} g(i,u)\, x(i)$ for each control $u \in U(x)$ and belief state x.

The definition (B2) for the allowable controls in a belief state is motivated from the existing work in the planning community in which controls are defined in terms of *preconditions* and *effects*. Under this view, $U(x)$ is the set of all controls whose preconditions are *known* to be satisfied in the current state of the system. This use of preconditions allows the engineer to easily specify real-life problems at different levels of granularity. For example, if connecting a 120 Volts device to a 240 Volts outlet has effects that the engineer wants to leave out of the model, then such situations are precluded via a simple precondition.[3]

The transition probabilities for the belief-MDP are determined by the abilities of the agent. For example, it is well-known, that a full capable and rational agent ought perform Bayesian updates of belief states in order to behave optimally. However, in the most general and realistic case, the agent might not be able to do so.[4] Thus, we will assume that after applying a control u in belief state x, the agent next belief is in a set $A \subseteq B$ with probability $\nu_u(x, A)$. Here, $\nu_u(x,\cdot)$ is a measure over the space of belief states that is called the *transition measure* associated with u and x. Using this notation, the DP operators for the belief-MDP become

$$\begin{aligned}(T_\mu J)(x) &\overset{\text{def}}{=} g(x,\mu(x)) + \alpha \int J(z)\, \nu_{\mu(x)}(x, dz),\\ (TJ)(x) &\overset{\text{def}}{=} \min_{u \in U(x)} \left\{ g(x,u) + \alpha \int J(z)\, \nu_u(x,dz) \right\}\end{aligned}$$

where μ is a stationary policy in belief space, $J : B \to \mathbb{R}$ is a real function over B, and $\alpha \in [0,1]$ is the discount factor.[5] As before, when $\alpha < 1$, the DP operators are contraction mappings with unique fix points. This fact guarantees the existence of optimal policies and that there is an optimal policy that is stationary.

From now on, we will assume that the agent always performs Bayesian updating whenever it applies a control and receives an observation. In this case, the transition measures are discrete measures defined as[6]

$$\nu_u(x, \{z\}) \overset{\text{def}}{=} \sum_{o \in Z(x,u)} p(x,u,o)\, \mathbf{1}_{\{x_u^o\}}(z) \tag{2}$$

3. Other more crude interpretations of $U(x)$ are possible, e.g. it can be defined as the set of controls u such that $|g(x,u)| < \infty$. Thus, the main results of the paper are valid under weaker assumptions than it is standard.
4. For example, some real-world approaches to robotics do approximate Bayesian updating by means of different sampling techniques as Monte-Carlo sampling and particle filters (Thrun, 2000).
5. To be mathematically correct, $\nu_u(\cdot,\cdot)$ must satisfy two conditions:

   (*i*) $\nu_u(x,\cdot)$ is a measure for each $x \in B$, and

   (*ii*) $\nu_u(\cdot, A)$ is a measurable function for each measurable set $A \subseteq B$.

   The first condition is required by the definition while the second is a technical one that guarantees all mathematical objects are well-defined (Fristedt & Gray, 1997, Ch.26).
6. $\mathbf{1}_A$ refers to the indicator function of the set A.

where $Z(x,u)$ is the set of possible observations after applying control u in belief state x, $p(x,u,o)$ is the probability of receiving observation o after applying u in x, and x_u^o is the Bayesian update of belief x after applying u and receiving o, i.e.

$$\begin{aligned} x_u^o(i) &\overset{\text{def}}{=} \frac{x_u(i)\,p(i,u,o)}{p(x,u,o)}, \\ x_u(i) &\overset{\text{def}}{=} \sum_{j=1}^{n} x(j)\,p(j,u,i), \\ p(x,u,o) &\overset{\text{def}}{=} \sum_{i=1}^{n} x_u(i)\,p(i,u,o), \\ Z(x,u) &\overset{\text{def}}{=} \{o \in Z : p(x,u,o) > 0\}. \end{aligned}$$

The DP operators associated with the measure in (2) become:

$$\begin{aligned} (T_\mu J)(x) &\overset{\text{def}}{=} g(x,\mu(x)) + \alpha \sum_{o \in Z(x,\mu(x))} p(x,\mu(x),o) J(x^o_{\mu(x)}) \\ (TJ)(x) &\overset{\text{def}}{=} \min_{u \in U(x)} \left\{ g(x,u) + \alpha \sum_{o \in Z(x,u)} p(x,u,o) J(x_u^o) \right\}. \end{aligned}$$

Unfortunately, the Value Iteration method is no longer feasible since each DP update has to be over an uncountable number of belief states (similarly for Policy Iteration). Thus, the question of how to compute an optimal stationary policy, or an approximation to it, is a major problem in the field.

2.2 Algorithms based on Sondik's Representation

These algorithms are based on the fact that $T^k J$ can be represented as

$$(T^k J)(x) = \min_{\gamma \in \Gamma_k} \sum_{i=1}^{n} x(i)\,\gamma(i)$$

where Γ_k is a *finite collection* of n-dimensional vectors. This result, due to Edward Sondik, is known as Sondik's piecewise linear and convex representation of the Value Function (Sondik, 1971). The algorithms based on Sondik's representation work in stages by computing the set Γ_{k+1} from Γ_k and stopping when k is sufficiently large to guarantee a given bound. Sadly, the sets Γ_k grow in size double exponentially in k and, although different techniques had been proposed to remove redundant vectors from Γ_k, the worst-case growth is always exponential. Therefore, even with fixed dimension $|S|$, all known optimal algorithms that work with Sondik's representation are exponential. The original Sondik algorithms, for the finite and infinite horizon cases, were published in (Smallwood & Sondik, 1973; Sondik, 1978). Important improvements over the basic algorithm appeared from AI 20 years after Sondik's work (Littman, 1996; Cassandra, Littman, & Zhang, 1997; Zhang & Liu, 1997; Zhang & Lee, 1997; Cassandra, 1998; Kaelbling et al., 1999).

The improvements in the algorithms and computer technology allowed the development of the first automated tools for solving POMDPs problems. Yet, almost immediately it was obvious that new algorithms were needed in order to solve more interesting problems. Of special interest are grid-based algorithms that work with explicitly-stored value functions.

2.3 Grid-Based Algorithms

The basic idea in a grid-based algorithm is to solve find the value of the value function for a finite number of belief states and then use such values to approximate the value for all other belief states.

We begin with some definitions in order to formalize this basic idea. A *grid* G over the belief space B is a *finite collection* of beliefs together with a *projection* $\eta : B \to G$. A *grid-based approximation* to J^* is a real-valued vector $\tilde{J} : G \to \mathbb{R}$ that is used to approximate $J^*(x)$ with $\tilde{J}(\eta(x))$. For our purposes, we will define a *grid-based algorithm* as an algorithm that on input G, and possibly other parameters, outputs a grid-based approximation $\tilde{J}$. We want to remark that other more general definitions for grid-based algorithms had been given, see (Hauskrecht, 2000) for a survey of exact and approximated POMDP algorithms.

Several grid-based algorithms had been proposed for finding approximate solutions to POMDPs but, although some algorithms had shown good performance over benchmark problems, none of them offer bounds on the quality of the results. That is, a bound over the error

$$\|J^* - \tilde{J} \circ \eta\| \overset{\text{def}}{=} \sup_{x \in B} |J^*(x) - \tilde{J}(\eta(x))|.$$

Quite often the projection η is defined in terms of a distance function (or metric) σ in belief space such that $\eta(x)$ is a nearest grid-point to x under σ. In such case, we will say that G is a *metric* grid and we define the *mesh* of the grid as the maximum separation between a grid-point and its surrogated points, i.e.

$$\text{mesh}(G, \eta, \sigma) \overset{\text{def}}{=} \sup_{x \in G} \sup \{\sigma(x, y) : y \in \eta^{-1}(x)\}.$$

We finish this section with a novel criterion for robustness of grid-based algorithms based on distance metrics. We say that a grid-based method is *robust in the strong sense* if, independently of the position of the grid-points, the approximation error goes to zero as the mesh goes to zero. Formally,

Definition 1 (Strong Robustness) *Let G be a metric grid on B, η the projection induced by G and $\tilde{J}_G : G \to \mathbb{R}$ a grid-based approximation to J^*. We say that $\tilde{J}_G$ is* robust in the strong sense *if*

$$\lim_{\epsilon \searrow 0} \sup_{G} \|J^* - \tilde{J}_G \circ \eta\| = 0$$

where the sup is over all grids with mesh at most ϵ.

Our goal is to obtain a grid-based algorithm for POMDPs that is robust in the strong sense.

3. Basic Mathematical Results

It should be clear that for achieving strong robustness some kind of continuity of J^* is necessary: if $J^*(x)$ is approximated by $J^*(y)$, then $|J^*(x) - J^*(y)|$ should go to zero as x

"approaches" y. As it is usual, the notion of continuity depends in a notion of distance over the belief space B. A very popular norm that is often used is the supremum or max norm which is defined as

$$\sigma(x,y) \overset{\text{def}}{=} \sup_{i\in S} |x(i)-y(i)|.$$

Some authors replace the max norm by other *topologically equivalent* norms like the L_1 or L_2 norms.[7] In this paper, we propose to use a new metric in belief space that allow us to obtain the algorithm. The new metric, denoted by ρ, is defined as

$$\rho(x,y) \overset{\text{def}}{=} \begin{cases} 2 & \text{if } x \text{ and } y \text{ have different support,} \\ \sum_{i=1}^{n} |x(i)-y(i)| & \text{otherwise.} \end{cases}$$

for all belief states $x, y \in B$. As it is standard, we say that x and y have different support if there exists an $i \in S$ such that $x(i) > 0, y(i) = 0$ or $x(i) = 0, y(i) > 0$. The following result establishes that ρ is a metric in the formal sense and give some of its properties.

Theorem 1 *ρ is a metric over the set of belief states. For all $x, y \in B$, $u \in U(x)$ and $o \in Z(x,u)$, if $\rho(x,y) < 2$, then*

(i) $\rho(x,y) = 2$ *if and only if* x, y *have different support,*

(ii) $U(x) = U(y)$ *and* $Z(x,u) = Z(y,u)$,

(iii) $\rho(x_u, y_u) \le \rho(x,y)$,

(iv) $|p(x,u,o) - p(y,u,o)| \le \sum_{i=1}^{n} p(i,u,o)\,|x_u(i) - y_u(i)| \le \rho(x_u, y_u)$, *and*

(v) $\rho(x_u^o, y_u^o) \le \frac{2}{p\wedge q} \sum_{i=1}^{n} p(i,u,o)\,|x_u(i) - y_u(i)|$

where $p = p(x,u,o)$, $q = p(y,u,o)$, *and* $a \wedge b \overset{\text{def}}{=} \min\{a,b\}$.

Proof: The facts that ρ is a metric and (i) are easy to show so they are left to the reader. Let $x, y \in B$ be such that $\rho(x,y) < 2$. Since x and y have identical support, it is obvious that $U(x) = U(y)$ and $Z(x,u) = Z(y,u)$. For (iii) and (iv),

$$\begin{aligned} \rho(x_u, y_u) &= \sum_{i=1}^{n} |x_u(i) - y_u(i)| = \sum_{i=1}^{n} \left| \sum_{j=1}^{n} p(j,u,i)\,(x(j) - y(j)) \right| \\ &\le \sum_{1\le i,j\le n} p(j,u,i)\,|x(j) - y(j)| = \rho(x,y), \end{aligned}$$

$$\begin{aligned} |p(x,u,o) - p(y,u,o)| &= \left| \sum_{i=1}^{n} x_u(i) p(i,u,o) - y_u(i) p(i,u,o) \right| \\ &\le \sum_{i=1}^{n} p(i,u,o)\,|x_u(i) - y_u(i)| \le \rho(x_u, y_u). \end{aligned}$$

7. Two norms are topologically equivalent if they generate the same collection of open sets.

For (v), let $p = p(x,u,o)$ and $q = p(y,u,o)$ and assume, without loss of generality, that $p > q$. Then,

$$\begin{aligned}
\rho(x_u^o, y_u^o) &= \sum_{i=1}^{n} |x_u^o(i) - y_u^o(i)| \\
&= \sum_{i=1}^{n} p(i,u,o) \left| \frac{x_u(i)}{p(x,u,o)} - \frac{y_u(i)}{p(y,u,o)} \right| \\
&= \frac{1}{pq} \sum_{i=1}^{n} p(i,u,o) \left| q x_u(i) - p y_u(i) \right| \\
&\leq \frac{1}{pq} \sum_{i=1}^{n} p(i,u,o) \big(p\,|x_u(i) - y_u(i)| + |p-q|\, x_u(i) \big) \\
&= \frac{1}{pq} \left(p \sum_{i=1}^{n} p(i,u,o)|x_u(i) - y_u(i)| + |p-q| \sum_{i=1}^{n} p(i,u,o) x_u(i) \right) \\
&= \frac{1}{pq} \left(p \sum_{i=1}^{n} p(i,u,o)\, |x_u(i) - y_u(i)| + |p-q| p \right) \\
&\leq \frac{2}{p \wedge q} \sum_{i=1}^{n} p(i,u,o)\, |x_u(i) - y_u(i)|.
\end{aligned}$$

$\square$

An interesting fact about the metric ρ is that its associated topology, denoted by ρ-topology, has isolated points. Indeed, the "deterministic" belief states are the isolated points of the topology. The following is the main result of the paper. It justifies the use of ρ and will allow us to obtain a strong-robust algorithm.

First, choose reals $\gamma < 1$ and $\epsilon > 0$ such that $\xi + \alpha\,(2\xi|Z|)^\gamma - \xi^\gamma \;\leq\; 0$ for all $0 \leq \xi \leq \epsilon$. Below we will show that this is always possible when $\alpha < 1$.

Theorem 2 *Define* $\|g\| \overset{def}{=} \sup_{i \in S} \sup_{u \in U(i)} |g(i,u)|$ *and suppose that* $\alpha \in [0,1)$ *and* $\|g\| < \infty$. *Then, the optimal cost function* J^* *is a* uniform continuous *map from* B *to* $\mathbb{R}$. *Indeed, if* γ *and* ϵ *are chosen as above, then*

$$|J^*(x) - J^*(y)| \;\leq\; \frac{\|g\|}{1-\alpha}\, \rho(x,y)^\gamma$$

for all $x, y \in B$ *such that* $\rho(x,y) \leq \epsilon$.

Remarks: The continuity of J^* with respect to the ρ-topology follows easily from the fact that J^* is the limit of a uniform convergent sequence of continuous functions. The fact that it is uniform continuous is more interesting since the metric space (B,ρ) is not compact neither complete. The theorem goes one step further by giving a bound on the modulus of continuity.

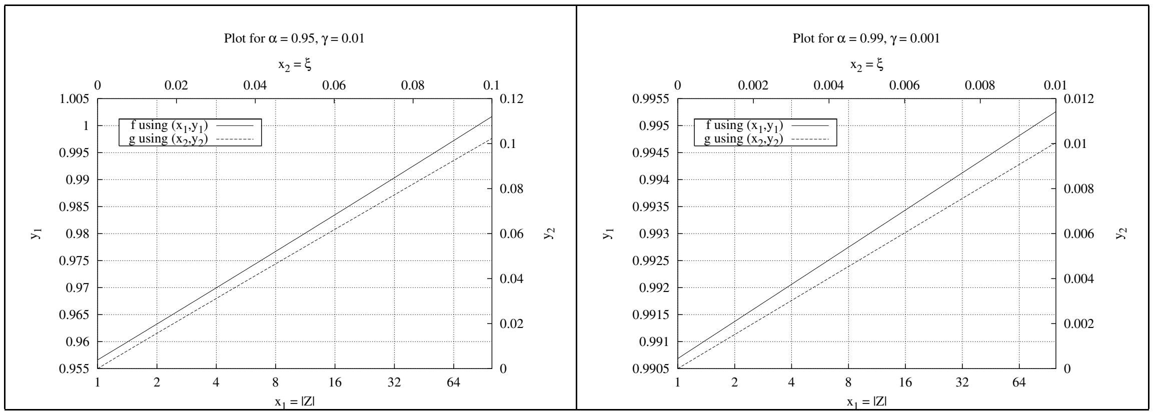

Figure 1: plot of the functions $f(\alpha, |Z|, \gamma) = \alpha(2|Z|)^\gamma$ and $g(\xi, \gamma) = \xi^{1-\gamma}$ for different values of the parameters. The function f is plotted with respect to the axis x_1 (bottom) and y_1 (left), and g is plotted with respect to the axis x_2 (top) and y_2 (right).

To see how γ and ϵ can be chosen, define the functions $f(\alpha, |Z|, \gamma) \overset{\text{def}}{=} \alpha(2|Z|)^\gamma$ and $g(\xi, \gamma) \overset{\text{def}}{=} \xi^{1-\gamma}$. Then,

$$\begin{aligned}
\xi + \alpha(2\xi|Z|)^\gamma - \xi^\gamma &\leq 0 \qquad \text{if and only if} \\
\xi^\gamma[\xi^{1-\gamma} + \alpha(2|Z|)^\gamma - 1] &\leq 0 \qquad \text{if and only if} \\
\xi^{1-\gamma} + \alpha(2|Z|)^\gamma &\leq 1 \qquad \text{if and only if} \\
f(\alpha, |Z|, \gamma) + g(\xi, \gamma) &\leq 1.
\end{aligned}$$

Figure 1 shows two plots of f and g: the first for $\alpha = 0.95$ and $\gamma = 0.01$ and different values for $|Z|$ and ξ, and the second for $\alpha = 0.99$ and $\gamma = 0.001$. Thus, for example, for a problem with 8 observations and $\alpha = 0.95$ it is enough to take $\gamma = 0.01$ and $\epsilon = 0.02$ since $f(.95, 8, .01) = .97670813$ and $g(.02, .01) = .02079791$.

Proof of Theorem 2: Let $0 < \epsilon, \gamma < 1$ be such that $\epsilon + \alpha(2\epsilon|Z|)^\gamma - \epsilon^\gamma \leq 0$, and $x, y \in B$ so that $\rho(x, y) < \epsilon$. Thus, $U(x) = U(y)$ and $Z(x, u) = Z(y, u)$. First, we show using induction that

$$|(T^k J_0)(x) - (T^k J_0)(y)| \leq \frac{\|g\|}{1-\gamma}\rho(x, y)^\gamma.$$

For the base case,

$$\begin{aligned}
|(TJ_0)(x) - (TJ_0)(y)| = &\left| \min_{u \in U(x)} \left\{ g(x, u) + \alpha \int J_0(z)\nu_u(x, dz) \right\} - \right. \\
&\left. \qquad \min_{u \in U(y)} \left\{ g(y, u) + \alpha \int J_0(z)\nu_u(y, dz) \right\} \right| \\
&\leq |g(x, u) - g(y, u)| \\
&\leq \|g\|\rho(x, y) \leq \frac{\|g\|}{1-\alpha}\rho(x, y)^\gamma
\end{aligned}$$

since $\xi \leq \xi^\gamma$ for $0 \leq \xi \leq 1$. The control $u \in U(x)$ in the first inequality is the control that minimizes the second term, and we have assumed without loss of generality that the first term is larger than the second. The inductive step is

$$
\begin{aligned}
&|(T^{k+1}J_0)(x) - (T^{k+1}J_0)(y)| \\
&\quad = \left| \min_u \left\{ g(x,u) + \alpha \int (T^{k-1}J_0)(z)\nu_u(x,dz) \right\} - \right. \\
&\qquad\qquad\qquad \left. \min_u \left\{ g(y,u) + \alpha \int (T^{k-1}J_0)(z)\nu_u(y,dz) \right\} \right| \\
&\quad \leq |g(x,u) - g(y,u)| \ + \ \alpha \left| \sum_{o \in Z(x,u)} p(x,u,o)(T^kJ_0)(x_u^o) - p(y,u,o)(T^kJ_0)(y_u^o) \right| \\
&\quad \leq \|g\|\rho(x,y) \ + \ \alpha \sum_{o \in Z(x,u)} \left| p(x,u,o)(T^{k-1}J_0)(x_u^o) - p(y,u,o)(T^{k-1}J_0)(y_u^o) \right| \\
&\quad \leq \|g\|\rho(x,y) \ + \ \alpha \sum_{o \in Z(x,u)} \left[(p \wedge q)\left|(T^kJ_0)(x_u^o) - (T^kJ_0)(y_u^o)\right| \ + \ \|T^kJ_0\||p-q| \right] \\
&\quad \leq \|g\|\rho(x,y) \ + \ \frac{\alpha\|g\|\rho(x,y)}{1-\alpha} \ + \ \frac{\alpha\|g\|}{1-\alpha} \sum_{o \in Z(x,u)} (p \wedge q)\rho(x_u^o, y_u^o)^\gamma \\
&\quad = \frac{\|g\|\rho(x,y)}{1-\alpha} \ + \ \frac{\alpha\|g\|}{1-\alpha} \sum_{o \in Z(x,u)} (p \wedge q)\, \rho(x_u^o, y_u^o)^\gamma \\
&\quad \leq \frac{\|g\|\rho(x,y)}{1-\alpha} \ + \ \frac{\alpha\|g\|}{1-\alpha} (2\rho(x,y))^\gamma \sum_{o \in Z(x,u)} (p \wedge q)^{1-\gamma} \\
&\quad \leq \frac{\|g\|\rho(x,y)}{1-\alpha} \ + \ \frac{\alpha\|g\|}{1-\alpha} (2\rho(x,y))^\gamma \, |Z|^\gamma \\
&\quad = \frac{\|g\|}{1-\alpha} \left[\rho(x,y) + \alpha(2\rho(x,y)|Z|)^\gamma \right] \\
&\quad \leq \frac{\|g\|}{1-\alpha} \rho(x,y)^\gamma
\end{aligned}
$$

by the choice of γ and where $p = p(x,u,o)$ and $q = p(y,u,o)$ for some $u \in U(x)$ as in the base case. The bound $\sum_{o \in Z(x,u)} (p \wedge q)^{1-\gamma} \leq |Z(x,u)|/|Z(x,u)|^{1-\gamma}$ comes from the fact that the expression is maximized when all $p = q = 1/|Z(x,u)|$. Hence,

$$
\sum_{o \in Z(x,u)} (p \wedge q)^{1-\gamma} \ \leq \ |Z(x,u)|^\gamma \ \leq \ |Z|^\gamma.
$$

In the third inequality, we have used

$$
\begin{aligned}
\sum_{o \in Z(x,u)} \|T^kJ_0\| \, |p-q| \ &\leq \ \frac{\|g\|}{1-\alpha} \sum_{o \in Z(x,u)} \sum_{i=1}^n p(i,u,o)|x_u(i) - y_u(i)| \\
&= \ \frac{\|g\|}{1-\alpha} \rho(x_u, y_u) \ \leq \ \frac{\|g\|}{1-\alpha} \rho(x,y).
\end{aligned}
$$

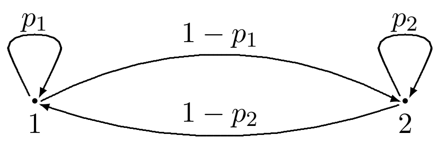

Figure 2: Transition probabilities for u_1 in the Example.

Therefore,

$$|J^*(x) - J^*(y)| \;=\; \lim_{k\to\infty} \Big|(T^k J_0)(x) - (T^k J_0)(y)\Big| \;\le\; \frac{\|g\|}{1-\alpha}\rho(x,y)^\gamma .$$

□

Corollary 3 *Suppose $\alpha \in [0, 1/2)$ and $\|g\| < \infty$. Then,*

$$|J^*(x) - J^*(y)| \;\le\; \frac{\|g\|\rho(x,y)}{(1-\alpha)(1-2\alpha)}$$

for all beliefs $x, y \in B$ such that $\rho(x,y) < 2$.

Proof: By a similar proof using $\rho(x^o_u, y^o_u) \le \frac{2}{p \wedge q}\sum_{i=1}^n p(i,u,o)|x_u(i) - y_u(i)|$. □

It is important to remark that a result as Theorem 2 is not possible for the sup metric. In fact, the following example shows a POMDP in which J^* is not continuous in the σ-topology.

Example: Let $S = \{1, 2\}$ and consider two controls u_1, u_2 such that $U(1) = \{u_1\}$, $U(2) = \{u_1, u_2\}$, $g(1, u_1) = 1$, $g(2, u_1) = g(2, u_2) = 0$, $p(2, u_2, 2) = 1$ and the transition probabilities for u_1 are given by two parameters p_1, p_2 as shown in Fig. 2. Also, there is just one observation that is always received with probability 1. Each belief state in this problem is of the form $(p, 1-p)$ so it can be represented by $p \in [0, 1]$. When $\alpha < 1$, the corresponding POMDP is guaranteed to have a solution, and it is easy to check that J^* becomes

$$J^*(p) \;=\; \begin{cases} p + \alpha J^*(pp_1 + (1-p)(1-p_2)) & \text{if } p > 0, \\ 0 & \text{if } p = 0. \end{cases}$$

Note that $pp_1 + (1-p)(1-p_2) = 1 + p(p_1 + p_2 - 1) - p_2$. Thus, if $p_1 + p_2 = 1$, then $J^*(1-p_2) = 1 - p_2 + \alpha J^*(1-p_2)$. Hence, if $p_1 + p_2 = 1$,

$$J^*(p) \;=\; \begin{cases} p + \alpha(1-p_2)/(1-\alpha) & \text{if } p > 0, \\ 0 & \text{if } p = 0. \end{cases}$$

Clearly, J^* is discontinuous at $p = 0$ with respect to the σ-topology. On the other hand, $p = 0$ is an isolated point in the ρ-topology, so J^* is continuous with respect to the latter. Also note that the jump can be arbitrary large. □

Thus, in the most general case, any grid-based method based on the sup metric is not robust in the strong sense. More precisely, the example proves the following theorem. On the other hand, Theorem 5 shows that this result does not hold in the case when all set $U(i)$ are identical.

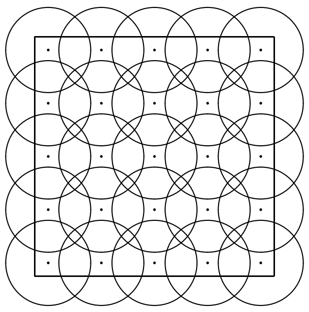

Figure 3: An ϵ-cover of the unit square.

Theorem 4 *Let σ' be a metric in belief space such that the σ'-topology is identical to the σ-topology, e.g. σ' = Euclidean distance. Then, any grid-based method such that η maps belief states into their nearest grid-points (under σ') is not robust in the strong sense for general* POMDP*s.*

Theorem 5 *Assume that $U(i) = U(j)$ for all states $i, j \in S$. Then, the optimal value function is uniform continuous in the σ-topology.*

Proof: We use standard results from mathematical analysis that can be found in any good text, for example (Royden, 1968). Each function $T^k J$ is a continuous map in the σ-topology. Hence, the compactness of (B, σ) implies that $T^k J$ is uniform continuous. Now, use the fact that J^* is the uniform limit of a sequence of uniform continuous functions to conclude that J^* is a continuous function. The proof is finished using another compactness argument. $\square$

An immediate consequence of Theorem 2 is that any "reasonable" topological grid-based algorithm based upon ρ is robust in the strong sense. In the next section we present a very simple algorithm.

4. An Optimal Grid-Based Algorithm

A finite collection $\{B_1, \ldots, B_m\}$ of closed (or open) balls with respect to ρ and centered at $\{x_1, \ldots, x_m\}$ is called an *ϵ-cover* if each ball is of radius ϵ and they cover B. That is, if $B_i = \{x \in B : \rho(x_i, x) \leq \epsilon\}$ and $B \subseteq \cup_i^m B_i$. For example, Fig. 3 shows an ϵ-cover of the unit square. It is not hard to see that an ϵ-cover always exists for every $\epsilon > 0$ and that the x_i can be chosen so that $x_i(j)$ are rational for $1 \leq i \leq m, 1 \leq j \leq |S|$. Indeed, the fact that a finite cover always exists for positive ϵ is known as that the space (B, ρ) is totally bounded, and the fact that all centers can be chosen with rational coordinates follows from separability (Royden, 1968). We will say that an ϵ-cover is a rational cover if all centers have rational coordinates and ϵ is also rational.

Each such cover induces a projection $\eta : B \to \{x_1, \ldots, x_m\}$ by $\eta(x) = x_i$ if and only if i is minimum such that $x \in B_i$. By Theorem 2,

$$|J^*(x) - J^*(\eta(x))| \;\leq\; \frac{\|g\|}{1-\alpha}\rho(x, \eta(x))^\gamma \;\leq\; \frac{\|g\|\epsilon^\gamma}{1-\alpha}.$$

Therefore, in order to obtain a robust algorithm we need to construct a "good" approximation to $J^* \circ \eta$. Consider the sequence of vectors $\tilde{J}_k : \{x_1, \ldots, x_m\} \to \mathbb{R}$ defined as

$$\tilde{J}_0(x_i) \overset{\text{def}}{=} 0, \tag{3}$$

$$\tilde{J}_{k+1}(x_i) \overset{\text{def}}{=} \min_{u \in U(x_i)} \left\{ g(x_i, u) + \alpha \sum_{o \in Z(x_i,u)} p(x_i, u, o) \tilde{J}_k(\eta((x_i)_u^o)) \right\}. \tag{4}$$

This is the Value Iteration algorithm applied to an MDP with state space $\{x_1, \ldots, x_m\}$ and transition probabilities

$$p(x_i, u, x_j) \overset{\text{def}}{=} \sum_{o : \eta((x_i)_u^o) = x_j} p(x_i, u, o).$$

It is easy to see that the associated DP operators are identical to the restriction of $T_\mu(J \circ \eta)$ and $T(J \circ \eta)$ to the set $\{x_1, \ldots, x_m\}$. Since η^{-1} partitions the belief space into a finite number of pieces, (4) defines a grid-based algorithm for POMDPs. The following results bound the approximation error and the loss incurred by the resulting greedy policy when the Value Iteration method $\langle \tilde{J}_k \circ \eta \rangle_{k \geq 0}$ is stopped at an appropriate iteration.

Theorem 6 *Suppose $\alpha \in [0, 1)$, $\|g\| < \infty$ and γ is chosen as in Theorem 2. Let $\tilde{J}_k$ be defined by* (4)*, then*

$$\begin{aligned} \|J^* - \tilde{J}_k \circ \eta\| &\leq \frac{\|g\|\epsilon^\gamma}{(1-\alpha)^2} + \frac{\alpha^k \|g\|}{1-\alpha}, \\ \lim_{k \to \infty} \|J^* - \tilde{J}_k \circ \eta\| &\leq \frac{\|g\|\epsilon^\gamma}{(1-\alpha)^2}. \end{aligned}$$

Proof: First note that

$$\begin{aligned} |J^*(x) - \tilde{J}_k(\eta(x))| &= |J^*(x) - J^*(\eta(x)) + J^*(\eta(x)) - \tilde{J}_k(\eta(x))| \\ &\leq \frac{\|g\|\epsilon^\gamma}{1-\alpha} + |J^*(\eta(x)) - \tilde{J}_k(\eta(x))|. \end{aligned}$$

Now, use induction on k to find a bound on $|J^*(x) - \tilde{J}_k(x)|$ for $x \in \{x_1, \ldots, x_m\}$ as follows

$$\begin{aligned} |J^*(x) - \tilde{J}_1(x)| &\leq \alpha \sum_{o \in Z(x,u)} p(x, u, o) \left| J^*(x_u^o) - \tilde{J}_0(\eta(x_u^o)) \right| \leq \alpha \|J^*\|, \\ |J^*(x) - \tilde{J}_2(x)| &\leq \alpha \sum_{o \in Z(x,u)} p(x, u, o) \left| J^*(x_u^o) - \tilde{J}_1(\eta(x_u^o)) \right| \\ &\leq \alpha \sum_{o \in Z(x,u)} p(x, u, o) \left(\frac{\|g\|\epsilon^\gamma}{1-\alpha} + \alpha \|J^*\| \right) \\ &= \frac{\|g\|\epsilon^\gamma}{1-\alpha} \alpha + \alpha^2 \|J^*\| \end{aligned}$$

where $u \in U(x)$ (see the proof of Theorem 2). Then,

$$|J^*(x) - \tilde{J}_{k+1}(x)| \leq \alpha \sum_{o \in Z(x,u)} p(x, u, o) \left| J^*(x_u^o) - \tilde{J}_k(\eta(x_u^o)) \right|$$

$$\begin{aligned}
&\leq \alpha \left(\frac{\|g\|\epsilon^\gamma}{1-\alpha} + \frac{\|g\|\epsilon^\gamma}{1-\alpha} \sum_{j=1}^{k-1} \alpha^j + \alpha^k \|J^*\| \right) \\
&= \frac{\|g\|\epsilon^\gamma}{1-\alpha} \sum_{j=1}^{k} \alpha^j + \alpha^{k+1} \|J^*\|.
\end{aligned}$$

Therefore,

$$\sup_{x \in B} |J^*(x) - \tilde{J}_k(\eta(x))| \leq \frac{\|g\|\epsilon^\gamma}{1-\alpha} + \frac{\|g\|\epsilon^\gamma}{1-\alpha} \sum_{j=1}^{k-1} \alpha^j + \alpha^k \|J^*\| \leq \frac{\|g\|\epsilon^\gamma}{(1-\alpha)^2} + \frac{\alpha^k \|g\|}{1-\alpha}.$$

$\square$

Corollary 7 *Suppose $\alpha \in [0, 1)$ and $\|g\| < \infty$. Let μ^k be the greedy policy in belief space with respect to $\tilde{J}_k \circ \eta$, i.e. $T_{\mu^k}(\tilde{J}_k \circ \eta) = T(\tilde{J}_k \circ \eta)$. Then,*

$$\|J^* - J_{\mu^k}\| \leq \frac{2\alpha(1+\alpha)\|g\|\epsilon^\gamma}{(1-\alpha)^3}$$

for every $k \geq k_0$ where k_0 is such that $\alpha^{k_0} \leq \epsilon^\gamma/(1-\alpha)$.

The proof of the corollary is based in the following results about the suboptimality of greedy policies. The proofs, taken from the reference, are included here for the purpose of completeness.

Lemma 8 (Bertsekas, 1995) *Consider a discounted MDP in an arbitrary state space S. For every value function J, state $x \in S$ and k,*

$$\begin{aligned}
(T^k J)(x) + \underline{c}_k &\leq (T^{k+1} J)(x) + \underline{c}_{k+1} \\
&\leq J^* \\
&\leq (T^{k+1} J)(x) + \overline{c}_{k+1} \leq (T^k J)(x) + \overline{c}_k
\end{aligned}$$

where

$$\begin{aligned}
\underline{c}_k &\overset{def}{=} \frac{\alpha}{1-\alpha} \inf_{x \in S} \Big[(T^k J)(x) - (T^{k-1} J)(x) \Big] \\
\overline{c}_k &\overset{def}{=} \frac{\alpha}{1-\alpha} \sup_{x \in S} \Big[(T^k J)(x) - (T^{k-1} J)(x) \Big].
\end{aligned}$$

Similar inequalities are also valid if T and J^* are replaced by T_μ and J_μ for any stationary policy μ. In this case, the corresponding constants are denoted by $\underline{c}_k^\mu$ and $\overline{c}_k^\mu$.

Proof: Denote $\underline{\lambda} = \inf_{x \in S}[(TJ)(x) - J(x)]$ and fix $x \in S$. Then, $J(x) + \underline{\lambda} \leq (TJ)(x)$. The monotonicity[8] of T implies $(TJ)(x) + \underline{\lambda} \leq (T^2 J)(x)$, so

$$J(x) + (1+\alpha)\underline{\lambda} \leq (TJ)(x) + \alpha\underline{\lambda} \leq (T^2 J)(x).$$

8. An operator T is monotonic if $J \leq J'$ implies $TJ \leq TJ'$.

After k applications of T, we obtain

$$J(x) + \underline{\lambda}\sum_{j=0}^{k}\alpha^j \;\le\; (TJ)(x) + \underline{\lambda}\sum_{j=1}^{k}\alpha^j \;\le\; \dots \;\le\; (T^{k+1}J)(x).$$

Taking the limit as $k \to \infty$ and using $\underline{c}_1 = \alpha\underline{\lambda}/(1-\alpha)$, we obtain

$$J(x) + \left(\frac{\underline{c}_1}{\alpha}\right) \;\le\; (TJ)(x) + \underline{c}_1 \;\le\; (T^2J)(x) + \alpha\underline{c}_1 \;\le\; J^*(x). \tag{5}$$

Replacing J by T^kJ, we get $(T^{k+1}J)(x) + \underline{c}_{k+1} \le J^*(x)$ that is the second inequality in the Lemma. For the first inequality, note that $\alpha\underline{\gamma} \le \inf_{x\in S}[(T^2J)(x) - (TJ)(x)]$. By definition of $\underline{c}_2$ and $\underline{c}_1 = \alpha\underline{\lambda}/(1-\alpha)$, then $\alpha\underline{c}_1 \le \underline{c}_2$. Using this result in (5), we get $(TJ)(x) + \underline{c}_1 \le (T^2J)(x) + \underline{c}_2$. Now substitute J by $T^{k-1}J$ to get the first inequality in the Lemma. Similar arguments show the validity of the two other inequalities. $\square$

Corollary 9 (Bertsekas, 1995) *Consider a discounted* MDP *in an arbitrary state space S and an arbitrary value function J. Let μ be the greedy policy with respect to J, i.e. $T_\mu J = TJ$. Then, the following bound on the suboptimality of μ holds:*

$$\sup_{x\in S}\big[J_\mu(x) - J^*(x)\big] \;\le\; \frac{\alpha}{1-\alpha}\left(\sup_{x\in S}\big[(TJ)(x) - J(x)\big] \;-\; \inf_{x\in S}\big[(TJ)(x) - J(x)\big]\right).$$

Proof: Fix $x \in S$ and apply above Lemma with $k = 1$ for T and T_μ to get

$$\underline{c}_1 \;\le\; J^*(x) - (TJ)(x) \;\le\; \bar{c}_1, \qquad \underline{c}_1^\mu \;\le\; J_\mu(x) - (T_\mu J)(x) \;\le\; \bar{c}_1^\mu$$

for all $x \in S$. Subtracting both equations,

$$\begin{aligned} J_\mu(x) - J^*(x) \;&=\; J_\mu(x) - (T_\mu J)(x) - J^*(x) + (TJ)(x) \\ &\le\; \frac{\alpha}{1-\alpha}\left(\sup_{x\in S}\big[(T_\mu J)(x) - J(x)\big] \;-\; \inf_{x\in S}\big[(TJ)(x) - J(x)\big]\right) \\ &\le\; \frac{\alpha}{1-\alpha}\left(\sup_{x\in S}\big[(TJ)(x) - J(x)\big] \;-\; \inf_{x\in S}\big[(TJ)(x) - J(x)\big]\right) \end{aligned}$$

since μ is greedy with respect to J. The result follows by taking the sup in both sides. $\square$

Proof of Corollary 7: Note that

$$\begin{aligned} &|(T(\tilde{J}_k \circ \eta))(x) - \tilde{J}_k(\eta(x))| \\ &\quad\le\; |(T(\tilde{J}_k \circ \eta))(x) - (TJ^*)(x)| \;+\; |J^*(x) - \tilde{J}_k(\eta(x))| \\ &\quad\le\; \Bigg|\min_u\Big\{g(x,u) + \alpha\sum_{o\in Z(x,u)} p(x,u,o)\,\tilde{J}_k(\eta(x_u^o))\Big\} - \\ &\qquad\quad \min_u\Big\{g(x,u) + \alpha\sum_{o\in Z(x,u)} p(x,u,o)\,J^*(x_u^o)\Big\}\Bigg| \;+\; |J^*(x) - \tilde{J}_k(\eta(x))| \end{aligned}$$

Algorithm 1: An ϵ-optimal grid-based algorithm for POMDPs.

```
Input: A number $\epsilon > 0$ and a finite collection $\{x_1, \ldots, x_m\}$ so that the open/closed
       balls radius $\epsilon$ and centered at $x_i$'s form a rational cover for $B$.
Result : An $m$-dimensional real vector $\tilde{J}$ such that
       $\|J^* - \tilde{J} \circ \eta\| \le \frac{2\|g\|\epsilon^\gamma}{(1-\alpha)^2}$
       $\|J^* - J_{\tilde{\mu}}\| \le \frac{2\alpha(1+\alpha)\|g\|\epsilon^\gamma}{(1-\alpha)^3}$
       where $\tilde{\mu}$ is the greedy policy with respect to $\tilde{J} \circ \eta$.
begin
    Set $\tilde{J}_0(x_i) = 0$ for $i = 1, \ldots, m$;
    $k \leftarrow 0$;
    while $k \le (\gamma \log \epsilon - 2 \log(1-\alpha)) / \log \alpha$ do
        $\tilde{J}_{k+1} \leftarrow T(\tilde{J}_k \circ \eta)$;
        $k \leftarrow k + 1$;
    end
end
```

$$
\begin{aligned}
&\le \alpha \sum_{o \in Z(x,u)} p(x,u,o)\, |\tilde{J}_k(\eta(x_u^o)) - J^*(x_u^o)| \;+\; |J^*(x) - \tilde{J}_k(\eta(x))| \\
&\le (1+\alpha)\|J^* - \tilde{J}_k \circ \eta\| \\
&\le (1+\alpha)\left(\frac{\|g\|\epsilon^\gamma}{(1-\alpha)^2} + \frac{\alpha^k\|g\|}{1-\alpha}\right) \\
&\le (1+\alpha)\frac{2\|g\|\epsilon^\gamma}{(1-\alpha)^2}
\end{aligned}
$$

for all $k \ge k_0$. Now, use the bound given by Corollary 9 on the suboptimality of μ^k:

$$
J_{\mu^k}(x) - J^*(x) \;\le\; \frac{\alpha}{1-\alpha}\left(\sup_{x \in B} |(T(\tilde{J}_k \circ \eta))(x) - \tilde{J}_k(\eta(x))|\right) \;\le\; \frac{\alpha(1+\alpha)}{1-\alpha}\frac{2\|g\|\epsilon^\gamma}{(1-\alpha)^2}
$$

for all $k \ge k_0$. □

These results show the correctness of the grid-based algorithm shown in Algorithm 1. The algorithm is robust in the strong sense, tractable in the parameters ϵ^{-1}, α, $|Z|$ and $\|g\|$, and only exponential in the dimension $|S|$. The exponentiality in $|S|$ cannot be removed since it is known that solving general POMDPs, either exactly or approximately, is NP-HARD.

In the next section, we define the important subclass of Stochastic Shortest-Path problems in belief space and then give an algorithm that is also based on the metric ρ.

5. Stochastic Shortest-Path Problems and Real-Time DP

A Stochastic Shortest-Path problem is an MDP in which the state space $S = \{1, \ldots, n, t\}$ is such that t is a goal (target) state that is absorbing (i.e. $p(t,u,t) = 1$ and $g(t,u) = 0$ for all $u \in U(t)$) and the discount factor $\alpha = 1$. In this case, the existence of optimal policies (and optimal stationary policies) is a major mathematical problem. However, the existence is guaranteed under the following reasonable conditions:

(A1) there exists a stationary policy that achieves the goal with probability 1 from any start state,

(A2) all costs are positive except $g(t, \cdot\,) \equiv 0$.

The first assumption just expresses the fact that the problem admits a well-behaved solution. Such policies are known as *proper* policies. The second assumption, in the other hand, guarantees that all improper policies incur in infinite cost for at least one state. Thus, both assumptions preclude cases where the optimal solution might "wander" around without ever getting to the goal. For example, a problem having a zero-cost cycle in state space violates the second assumption. Under above assumptions, we have

Theorem 10 (Bertsekas, 1995) *Suppose A1 and A2 hold.*[9] *Then, there exists an optimal policy μ^* that is stationary. Also, if J is a n-dimensional vector J such that $J(t) = 0$, then*

$$\lim_{k\to\infty} T^k_{\mu^*} J \;=\; J_{\mu^*} \;=\; J^* \;=\; \lim_{k\to\infty} T^k J.$$

That is, the Value Iteration algorithm solves SSPs problems that satisfy assumptions A1 and A2. From now on, we will only consider SSP problems and the corresponding version in belief space. To be more precise, we say that a POMDP is a Stochastic Shortest-Path in belief space if it is defined by:

(S1) A finite state space $S = \{1, \ldots, n, t\}$,

(S2) a finite set of controls $U(i)$ for each state $i \in S$,

(S3) transition probabilities $p(i, u, j)$ for all $u \in U(i)$ such that $p(t, u, t) = 1$ for all $u \in U(t)$,

(S4) a finite set of observations $Z(i, u) \subseteq Z$ for all $u \in U(i)$ and $i \in S$,

(S5) observation probabilities $p(i, u, o)$ for all $u \in U(i)$ and $o \in Z(i, u)$, and

(S6) a cost $g(i, u)$ for each $u \in U(i)$ and $i \in S$ such that $g(t, u) = 0$ for all $u \in U(t)$.

Stochastic Shortest-Path problems in belief space arise frequently in AI from diverse tasks as planning with uncertainty and incomplete information, robot navigation and localization, game playing, diagnosis and prescription, etc. Moreover, it is often the case that we are only interested in knowing how to go from a fixed initial belief state, denoted by x_i, to the goal belief state x_t where $x_t(t) = 1$. The optimal solution in this case is an *partial* optimal policy μ such that $\mu(x) = \mu^*(x)$ for all states x that are *reachable* from x_i when using μ^*, the so-called *relevant states* from x_i. Finding a partial optimal policy can be considerably simpler, the extreme case when the set of relevant states is finite and the complete state space is infinite. Thus, the question of how to find partial optimal policies is of great relevance. Two recent algorithms for finding partial optimal policies in MDPs

9. The assumptions can be weakened while preserving the validity of the theorem (Bertsekas, 1995).

are Real-Time Dynamic Programming (RTDP) and LAO* (Barto, Bradtke, & Singh, 1995; Hansen & Zilberstein, 2001).

Below, we will show how to use the ideas in previous sections together with a modification of the RTDP algorithm for finding near optimal partial policies for SSPs in belief space. More precisely, we will focus in the class of problems that satisfy some conditions of "regularity". Formally, we say that a POMDP is a Stochastic Shortest-Path problem in belief space if it satisfies:

(A1′) There exists a stationary policy that reaches x_t in finite time with probability one from any initial belief state,

(A2′) all costs are positive, except $g(t,u) = 0$ for all $u \in U(t)$.

These are analogous to assumptions A1 and A2 in the partially observable setting. These two conditions guarantee that suitable discretizations of the belief space generate finite-state-space SSPs. Thus, the idea is to discretize and then use RTDP to compute an optimal partial policy. This was the approach taken in (Bonet & Geffner, 2000) where a number of non-trivial problems were solved.

One well-known problem of RTDPis that it only converges asymptotically as opposed to convergence in finite time. This makes RTDP useless for solving problems in an off-line setting. To get around of this problem, we have defined a modification of RTDP that terminates in a finite number of steps. This modification that makes RTDP into a genuine off-line algorithm was first reported in (Bonet & Geffner, 2001).

The rest of this section is as follows. First, we present the standard RTDP algorithm for solving SSP problems and some of its properties. Then, we show a modification of RTDP that terminates in a finite number of steps and then show some of its properties. At the end, we put everything together to obtain a new algorithm for SSPs in belief space.

For the rest of the section, we will denote the state space of the MDP as $\{1, \ldots, n, t\}$ where 1 is the initial state and t is the goal state.

5.1 Real-Time Dynamic Programming

The RTDP algorithm is the stochastic generalization of Korf's Learning Real-Time A* (LRTA) for deterministic heuristic search (Korf, 1990). RTDP is a randomized learning algorithm that computes a partial optimal policy by performing successive walks, also called trials, over the state space. Each trial starts at the initial state $x_0 = 1$ and finishes at the goal state t. At all times k, the RTDP algorithm maintains an approximation J_k to J^* that is used to *greedly select* a control u_k to apply in the current state i_k. Initially, J_0 is implicitly stored as an heuristic function $h(\cdot)$. Then, every time a control u_k is selected in state i_k, a new approximation J_{k+1} is computed by

$$J_0(i) \overset{\text{def}}{=} h(i), \tag{6}$$

$$J_{k+1}(i) \overset{\text{def}}{=} \begin{cases} J_k(i) & \text{if } i \neq i_k, \\ g(i_k, u_k) + \sum_{j=1}^{n} p(i_k, u_k, j) J_k(j) & \text{if } i = i_k. \end{cases} \tag{7}$$

Since J_k differs from J_0 at most in k states, J_k can be stored efficiently into a hash table H. Initially, H is empty and the value $H(i)$ is given by the heuristic $h(i)$. Thereafter, every

Algorithm 2: A trial of the RTDP algorithm.

```
begin
    $i \leftarrow 1$   (set initial state);
    while $i$ is not a goal state do
        Evaluate each control $u \in U(i)$ as
            $Q(i,u) = g(i,u) + \sum_{j=1}^{n} p(i,u,j)J(j)$
        where $J(j) = H(j)$ (resp. $= h(j)$) if $j \in H$ (resp. $j \notin H$);
        Choose control $u^*$ that minimizes $Q(i,u)$ breaking ties randomly (or in a systematic way);
        Update the hash table $H(i) \leftarrow Q(i,u^*)$;
        Generate next state $j$ with probability $p(i,u^*,j)$;
        Set $i \leftarrow j$;
    end
end
```

time a control u_k is selected an update of the form of (7) is applied to H such that J_k can be computed from H and h. The Algorithm 2 shows a description of an RTDP trial.

It is known that under assumptions A1 and A2, the RTDP trials *eventually* transverse minimum-cost paths from the initial state to the goal state if the heuristic function is *admissible*, i.e. if $0 \leq h(i) \leq J^*(i)$ for all $i \in S$ (Barto et al., 1995; Bertsekas & Tsitsiklis, 1996). Formally,

Theorem 11 (Barto et al., 1995) *Suppose that assumptions A1 and A2 hold. Assume that an infinite number of* RTDP *trials are performed each one starting at 1 and that the hash table is preserved between trials. If h satisfies $0 \leq h \leq J^*$, then with probability 1:*

(i) the sequence of vectors J_k generated by RTDP *converges to (some) J_∞,*

(ii) $J_\infty(i) = J^*(i)$ for all relevant states i.

The J_∞ vector in the theorem is a random vector that corresponds to the final hash table denoted by H_∞. Both objects J_∞ and H_∞ are random variables in the formal sense and the relation between them is that $J_\infty(i)$ is equal to $H_\infty(i)$ (resp. $h(i)$) if $i \in H_\infty$ (resp. if not). Part of the claim in the theorem is that each sample path of the algorithm corresponds to an *asynchronous* DP over the SSP problem defined by the states that appear infinitely often in the path. That is, each sample path corresponds to an asynchronous DP over a random MDP.

As the theorem asserts, the convergence is only asymptotically and there is no obvious way of stopping the RTDP algorithm while offering some guarantees. In the next subsection, we present a novel method for stopping RTDP when certain guarantees had been achieved.

5.2 An Off-line RTDP Algorithm

We present a modification of the RTDP algorithm that terminates the trials when a given precision had been achieved. The method is named the *stopping rule* and uses a single parameter $\delta > 0$ to control the precision. In order to introduce the idea, we offer two definitions for the method: the first is in terms of a global condition that is easy to understand but difficult to implement and the second is a condition that is more difficult to understand but suggests an easy implementation.

Plainly, the idea is to stop the trials when the value for all relevant states are off from satisfying the Bellman equations by at most δ, i.e. when

$$\left| J_k(i) - \min_{u \in U(i)} \left\{ g(i,u) + \sum_{j=1}^{n} p(i,u,j) J_k(j) \right\} \right| \;<\; \delta \tag{8}$$

is satisfied for all relevant states i. The difficulty in testing (8) lies in computing the set of relevant states. To identify such convergence, we consider a *recursive labeling*. The idea is to label states into $\{\text{solved}, \text{unsolved}\}$ such that a state is solved when it satisfies (8), and to stop the RTDP algorithm when the initial state is labeled as solved.

Initially, the goal state is labeled as solved and all other states are labeled as unsolved. Fix an RTDP trial $(i_0 = 1, \ldots, i_m = t)$ and consider the moment just after the end of the trial and beginning of the next trial. Let J and μ be the current approximation function (stored in the hash table) and the greedy policy with respect to J, and define the set of states $\{K_k \subseteq S : k = 0, \ldots, m\}$ such that $i_k \in K_k$ and all states reachable from i_k using μ are contained in K_k. That is, K_k is a minimal set of states such that $i_k \in K_k$ and $(\forall i \in K_k)\,(\forall j \in S)(p(i,\mu(i),j) > 0 \implies j \in K_k)$; by definition $K_0 \supseteq K_1 \supseteq \cdots \supseteq K_m = \{t\}$. Then, the method labels all states in the sets K_k as solved in descending order from $k = m-1$ to $k = 0$ until one of the following conditions fails:

(i) all states in sets K_{k+1} are solved, or

(ii) for all $i \in K_k$,

$$\left| J(i) - g(i,\mu(i)) - \sum_{j=1}^{n} p(i,\mu(i),j) J(j) \right| \;<\; \delta. \tag{9}$$

This is done every time at the end of the trials until the initial state is labeled as solved. Below we show that the procedure always finishes and, more important, that when δ is sufficiently small the resulting partial policy is optimal.

It is important to note that the RTDP trials do not have to go all the way up to the goal, they can be ended as soon as a solved state is visited. From now on, when we speak of using the stopping rule we refer to the RTDP algorithm with the stopping rule and termination of trials at solved states.

Theorem 12 *For all $\delta > 0$ the* RTDP *algorithm with the stopping rule labels the initial state as solved in a finite amount of time.*

Proof: For each relevant state i_k consider the set K_k of its reachable states when using the optimal policy μ^*. The collection $\mathcal{K} = \{K_j\}$ is partially ordered by set inclusion. Since the

RTDP algorithm converges to the optimal policy and cost vector over the relevant states, then the stopping rule will label as solved all states in a minimal element of $\mathcal{K}$ after some finite time. After that, the RTDP algorithm with stopping rule will be applying standard RTDP in a reduced MDP consisting of all unsolved states and with terminal costs given by the cost of the solved states. Then, again, the RTDP algorithm converges in this MDP so after some finite time it will label another non-empty set of states as solved. Since the number of states is finite and the initial state is visited infinitely often, the algorithm will label the initial state in finite time. $\square$

Theorem 13 *There exists $\delta_0 > 0$ such that if the* RTDP *algorithm with the stopping rule is applied with $0 < \delta < \delta_0$, then the resulting approximation J satisfies $|J(i) - J^*(i)| < \delta$ for all relevant states i, and the resulting partial policy is optimal. Here, $J(i)$ refers to the value for i in the hash table after termination or $h(i)$ if there is no entry for i.*

Proof: It is enough to let $\delta_0 = \inf\{\|J_\mu - J^*\| : \mu \text{ non-optimal stationary policy}\}$. $\square$

This labeling procedure can be implemented by different methods as a time-oriented recursion, as a space-oriented iteration, etc. The following describes a space-oriented iteration.

5.2.1 Implementation

To keep track of the visited states during a trial $(i_0, \ldots, i_m)$, we modified RTDP to push the states into a stack as they are visited. At the end of the trial the states are processed in reverse visit order as they are popped out from the stack. For each such state i_k, RTDP calls the function `check-solved(`i_k`)` that returns true or false whether conditions (i)–(ii) above are satisfied or not. In the affirmative case, `check-solved(`i_k`)` also returns the set K_k of states that need to be labeled. Otherwise, the stack is cleaned and the process is terminated.

The function `check-solved(`i_k`)` works by applying a breadth-first search maintaining two queues called `open` and `closed`. At the beginning, `open` contains only i_k and `closed` is empty. The function then iterates by removing the front state i from `open`, inserting the possible $\mu(i)$ successors of i back into `open` and pushing i into `closed`. Two exceptions are considered:

(a) if state i is already solved it is ignored and a new state is removed from `open`, and

(b) if i had never been visited (i.e. it is not in the hash table) or (9) does not hold for i, then a hash table update of the form of (7) is done for state i.

The procedure is applied until `open` becomes empty or an exception of type (b) occurs. In the former case, `check-solved(`i_k`)` returns true and the set K_k is the set of states in `closed`. In the latter case, the function returns false. Note that the updates in (b) are only for speeding up the convergence of the algorithm and so they are not necessary for the correctness of the procedure.

5.3 Real-Time Dynamic Programming for SSPs in Belief Space

To apply the RTDP algorithm to the SSP in belief space, we need to discretize the belief states before accessing the hash table. Any such discretization needs to satisfy two conditions: it has to be efficient and it has to preserve the support of the belief states.[10] We will consider a discretization scheme that is parametrized by a positive integer L, for levels, denoted by $\tilde{x}$ for belief state x, and defined as

$$\tilde{x}(i) \stackrel{\text{def}}{=} K(x)[\![1 + L \cdot x(i)]\!]$$

where $[\![\xi]\!]$ denotes the closest integer to real ξ. The constant $K(x)$ is chosen so that $\sum_i \tilde{x}(i) = 1$, and the addition of 1 guarantees that x and $\tilde{x}$ have the same support. It is easy to verify that the discretization becomes finer as $L \to \infty$ and, more important, that it generates a *non-uniform covering* of the belief space. For this reason, it is hard to give tight bounds on the suboptimality of the partial policies returned by RTDP. Nonetheless, we have tested the resulting algorithm in a number of benchmark problems involving different number of states, observations and varying degrees of stochasticity.

6. Experimental Results

We have implemented the RTDP algorithm with the stopping rule and above discretization into the General Planning Tool (GPT) (Bonet & Geffner, 2001). GPT is a *domain-independent* planner for problems involving uncertainty and partial information. Planning problems can be specified either by using Tony Cassandra's POMDP format[11] or the GPT language. GPT's language is based on the PDDL language for classical planning (McDermott, 1998b) that had become the standard in planning: it is the bi-annual planning competition (McDermott, 1998a; Bacchus, 2000) and for interchanging problems in the community.

We tested GPT over two group of problems. The first group, which are specified in Cassandra's format, contains the problems `tiger`, `cheese`, `4x4`, `4x3.CO`, `4x3` and `info`. The first five problems, taken from Cassandra's page[12], are well-known and had been used by others to measure the quality and performance of different POMDP algorithms. Some of these problems do not correspond to SSP problems in belief space so we modified them to satisfy assumptions A1$'$ and A2$'$. The `info` problem is due to Sebastian Thrun and consist of a navigation problem involving map; it has also been used by us and others (Thrun, Langford, & Fox, 1999; Bonet & Geffner, 2000; Aberdeen & Baxter, 2002). Table 1 shows basic information about the problems as the number of states ($|S|$), number of observations ($|Z|$), number of controls ($|U|$), size of the initial belief state ($|x_i|$), discount factor (α) and precision used for the stopping rule (δ).

We ran GPT using the different discretization levels $L = 5, 20, 50, \infty$. The value $L = \infty$ corresponds to no discretization and is only applicable when the number of relevant states is finite. All the experiments were run in a Sun Ultra 10 with 384MB of Ram and a CPU of 440MHz. The results are shown in Table 2. We report five quantities per experiment:

10. Standard discretizations using regular meshes like the Freudenthal triangulation (Lovejoy, 1991) do not satisfy the second condition.
11. `http://www.cs.brown.edu/research/ai/pomdp/examples/pomdp-file-spec.html`
12. `http://www.cs.brown.edu/research/ai/pomdp`

data	`tiger`	`cheese`	`4x4`	`4x3.CO`	`4x3`	`info`
$\|S\|$	3	11	16	11	11	20
$\|Z\|$	2	7	2	11	6	4
$\|U\|$	3	4	4	4	4	4
$\|x_i\|$	2	10	15	9	9	2
α	0.95	0.95	0.95	0.95	0.95	1.0
δ	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}

Table 1: information for small examples in Cassandra's POMDP format. Some of these problems had been modified to satisfy assumptions A1$'$ and A2$'$.

the total time to solve the problem, the size of the hash table at termination, the size of the resulting policy, the hash value for the initial belief state at termination ($J(x_i)$), and an estimate of the quality of the resulting policy ($\hat{J(x_i)}$). The size of the policy refers to the number of belief states in the final policy (see below). The value $J(x_i)$ is always a lower bound for the optimal value $J^*(x_i)$ *over the discretized problem*, i.e. the MDP problem given in equations (3) and (4). The estimate $\hat{J(x_i)}$ is computed by averaging the discounted accumulated cost of 10,000 trials of operation of the system under the resulting policy. As it can be seen in `tiger` problem, for example, the value $J(x_i)$ decreases and the quality of the policy increases as L increases. The small gap $|J(x_i) - \hat{J(x_i)}$ for $L = \infty$ tells us that the resulting policy is near optimal. We also note that solving the problem with $L = \infty$ generates simpler policies than with discretization. This can be explained by the fact that the problem has a small number of relevant belief states but not the discretized problem.

The Fig. 4 contains a graphical representation of the policies found for the `tiger` problem. Each policy is shown using a labeled directed graph in which the nodes represent belief states and the arcs represent transitions due to actions and observations. The labels on the nodes indicate the action to apply in each belief state while the labels on the arcs indicate the observation associated with the transition. The graph contains one initial node (in medium gray) that corresponds to the initial belief state and one or more sinks (in light gray) that correspond to the goal belief states. Thus, if μ denotes the policy found by GPT, then the label of a node x is the control $\mu(x)$, and if there is an edge labeled o between x and y, then $p(x, \mu(x), o) > 0$, $y = \eta(x^o_{\mu(x)})$ and η is the projection due to the discretization.

In the problems `cheese`, `4x4` and `4x3.CO`, GPT is able to find near optimal policies in less than a second for all discretizations; a fact that says the problems are relatively simple. In the `4x3` problem, the number of relevant states is infinite so GPT cannot solve it with $L = \infty$; note the explosion in the size of the hash table and resulting policy as L increases. Finally, GPT is unable to find a proper policy for `info` for $L = 5$, i.e. the resulting policy is not able to get to the goal in finite time. This fact is due to the loss of information introduced in the discretization. The policies for the `info` problem are shown in Fig. 5.

The problems in the second group were taken from the GPT distribution. They are planning problems involving uncertainty and partial information that have large number of states, actions and observations but reduced stochasticity. Indeed, except for the omelette problem, all the uncertainty in these problems comes from the initial situation. The problems

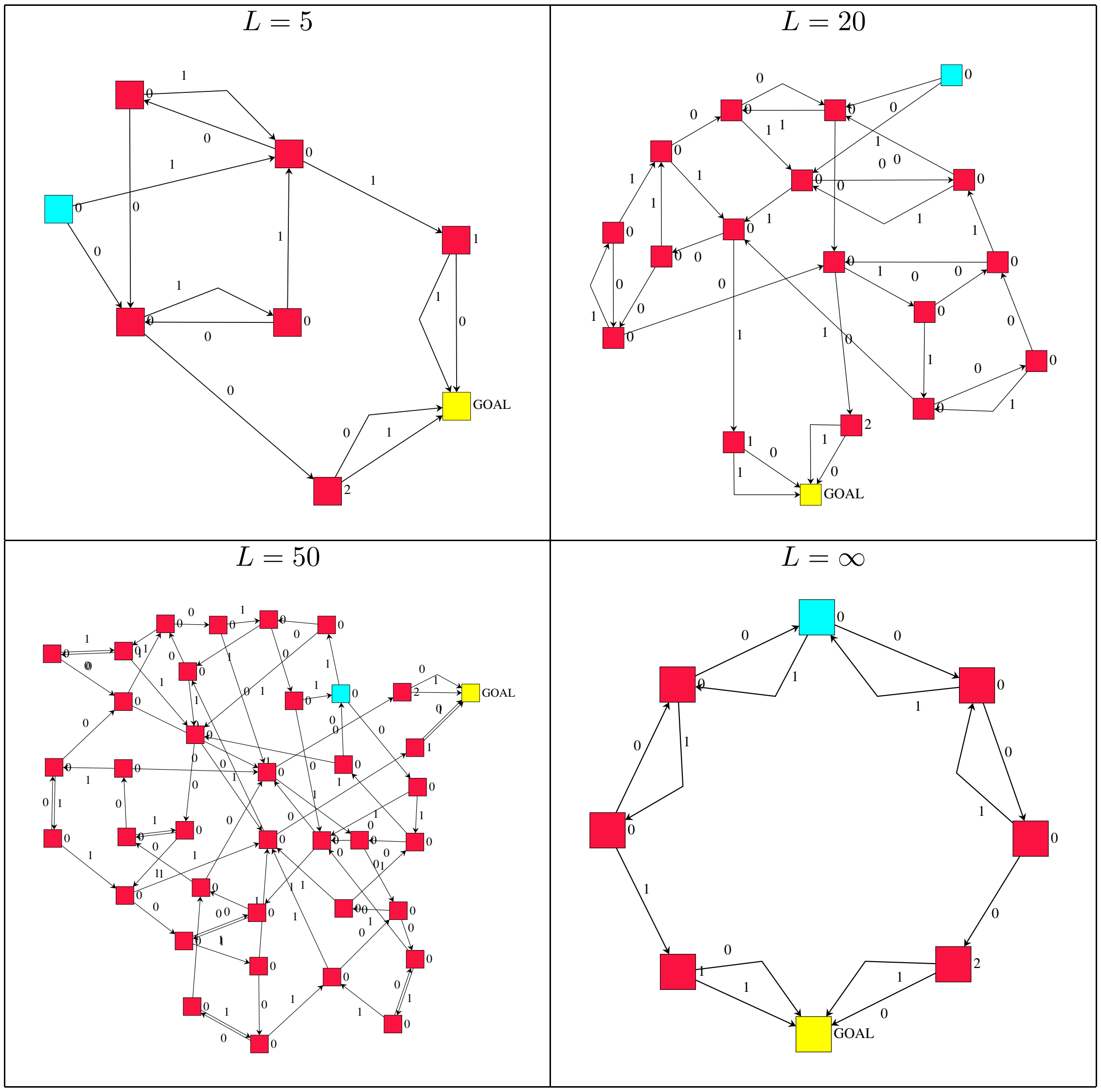

Figure 4: policies for the `tiger` problem for different discretization levels.

L	data	`tiger`	`cheese`	`4x4`	`4x3.CO`	`4x3`	`info`
5	time	0.30	0.21	0.23	0.22	6.38	–
	|hash|	8	14	42	12	142	–
	|policy|	8	10	7	12	34	–
	$J(x_i)$	15.10010	4.34068	3.82099	4.58631	5.50610	–
	$\widehat{J(x_i)}$	5.45038	4.34220	3.81235	4.59743	5.28302	–
20	time	0.36	0.21	0.23	0.22	24.79	0.26
	|hash|	22	14	56	12	449	40
	|policy|	18	10	7	12	90	22
	$J(x_i)$	7.80507	4.34068	3.83384	4.58631	5.28917	18.11919
	$\widehat{J(x_i)}$	4.21244	4.32422	3.77473	4.57941	5.01361	12.32140
50	time	1.34	0.21	0.24	0.22	86.97	0.31
	|hash|	38	14	67	12	985	103
	|policy|	38	10	7	12	202	24
	$J(x_i)$	6.71017	4.34068	3.79861	4.58631	5.18142	15.07960
	$\widehat{J(x_i)}$	4.89798	4.33555	3.78746	4.59628	5.02720	11.77360
∞	time	0.21	0.21	0.22	0.22	–	0.25
	|hash|	10	14	37	12	–	60
	|policy|	8	10	7	12	–	23
	$J(x_i)$	4.27105	4.34068	3.76669	4.58631	–	11.57031
	$\widehat{J(x_i)}$	4.20818	4.34555	3.76910	4.56710	–	11.58910

Table 2: results for small examples in Cassandra's POMDP format. The times are in seconds. A dash means that RTDP was unable to obtain a proper policy.

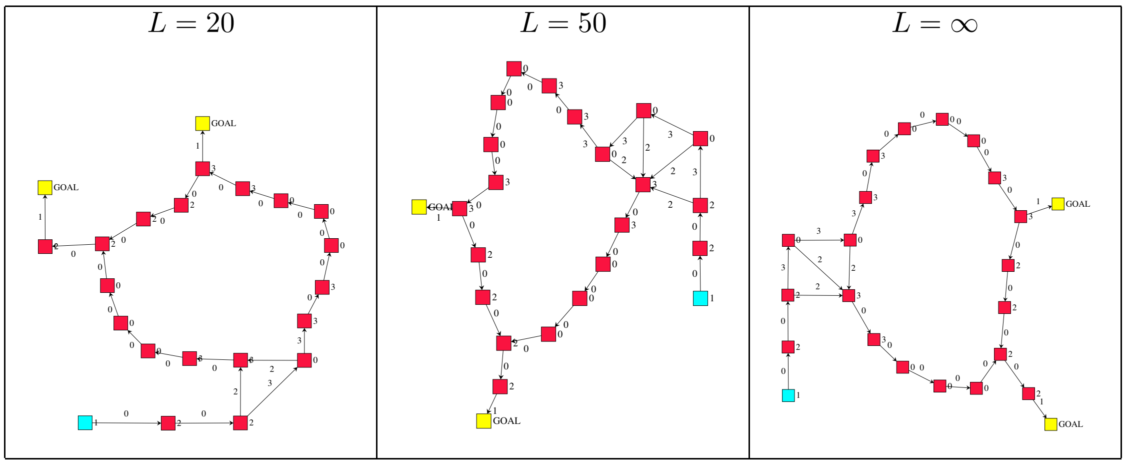

Figure 5: policies for the `info` problem for different discretization levels.

data	`omelette`	`mm-1`	`mm-2`	`sort`	`wmp-1`	`wmp-2`	`wmp-3`	`psr`
$\|S\|$	300	27	81	24	3,894	11,682	62,070	5,728
$\|Z\|$	3	11	16	3	6	8	14	329
$\|U\|$	11	27	81	32	7	7	7	10
$\|x_i\|$	1	27	81	24	56	56	336	128
α	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
δ	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}

Table 3: information for problems from the GPT distribution.

L	data	`omelette`	`mm-1`	`mm-2`	`sort`	`wmp-1`	`wmp-2`	`wmp-3`	`psr`
5	time	3.46	1.59	1,178.64	0.96	8.58	20.71	835.85	56.77
	|hash|	903	347	15,442	219	2,344	7,554	25,729	2,267
	|policy|	19	35	109	47	282	382	1,485	114
	$J(x_i)$	23.00000	1.88888	2.51851	6.58333	35.12500	12.57142	42.25007	29.14062
	$\widehat{J(x_i)}$	23.22280	1.89250	2.51000	6.59210	34.82890	12.52980	42.17350	29.01060
20	time	6.45	1.55	1,187.79	1.01	8.02	19.46	828.39	57.64
	|hash|	1,900	347	15,442	219	2,281	7,438	25,789	2,298
	|policy|	19	35	109	47	282	382	1,485	114
	$J(x_i)$	23.00000	1.88888	2.51851	6.58333	35.12500	12.57142	42.25007	29.14062
	$\widehat{J(x_i)}$	22.81840	1.88760	2.51380	6.58430	34.92340	12.54250	42.02960	29.26340
50	time	20.87	1.59	1,296.27	0.98	8.67	20.64	809.07	57.07
	|hash|	5,689	347	15,442	219	2,397	7,690	26,339	2,251
	|policy|	19	35	109	47	282	382	1,485	114
	$J(x_i)$	23.00000	1.88888	2.51851	6.58333	35.12500	12.57142	42.25007	29.14062
	$\widehat{J(x_i)}$	23.00400	1.89390	2.51230	6.57410	35.26110	12.62550	42.64560	28.96510
∞	time	4.23	1.57	1,332.86	0.98	8.84	19.43	850.90	57.24
	|hash|	1,199	347	17,730	219	2,352	7,520	26,387	2,267
	|policy|	19	35	109	47	282	382	1,485	114
	$J(x_i)$	23.00000	1.88888	2.51851	6.58333	35.12500	12.57142	42.25008	29.14062
	$\widehat{J(x_i)}$	22.95400	1.89140	2.52010	6.58180	35.05310	12.59500	42.89100	28.99590

Table 4: results for problems from the GPT distribution. The times are in seconds. A dash means that RTDP was unable to obtain a proper policy.

in this group are `omelette`, `mm-1`, `mm-2`, `sort`, `wmp-1`, `wmp-2`, `wmp-3` and `psr`. The general information about these problems in summarized in Table 3, and the results of the experiments in Table 4.

The `omelette` problem, due originally to Hector Levesque, had been used by us previously (Bonet & Geffner, 2000). This problem involves an agent that has a large supply of eggs and whose goal is to get three good eggs and no bad ones into one of two bowls. The eggs can be either good or bad, and the agent can perform tasks as breaking a egg into a bowl, pouring the content of a bowl into another and cleaning a bowl. The agent can also sense if a bowl contains a bad egg by smelling it. This problem has 300 states and 11 actions and GPT is able to solve it in a few seconds. The policy found by GPT is optimal for all discretizations as it can be verified analytically. The optimal policy for the `omelette` problem is shown in Fig. 6.

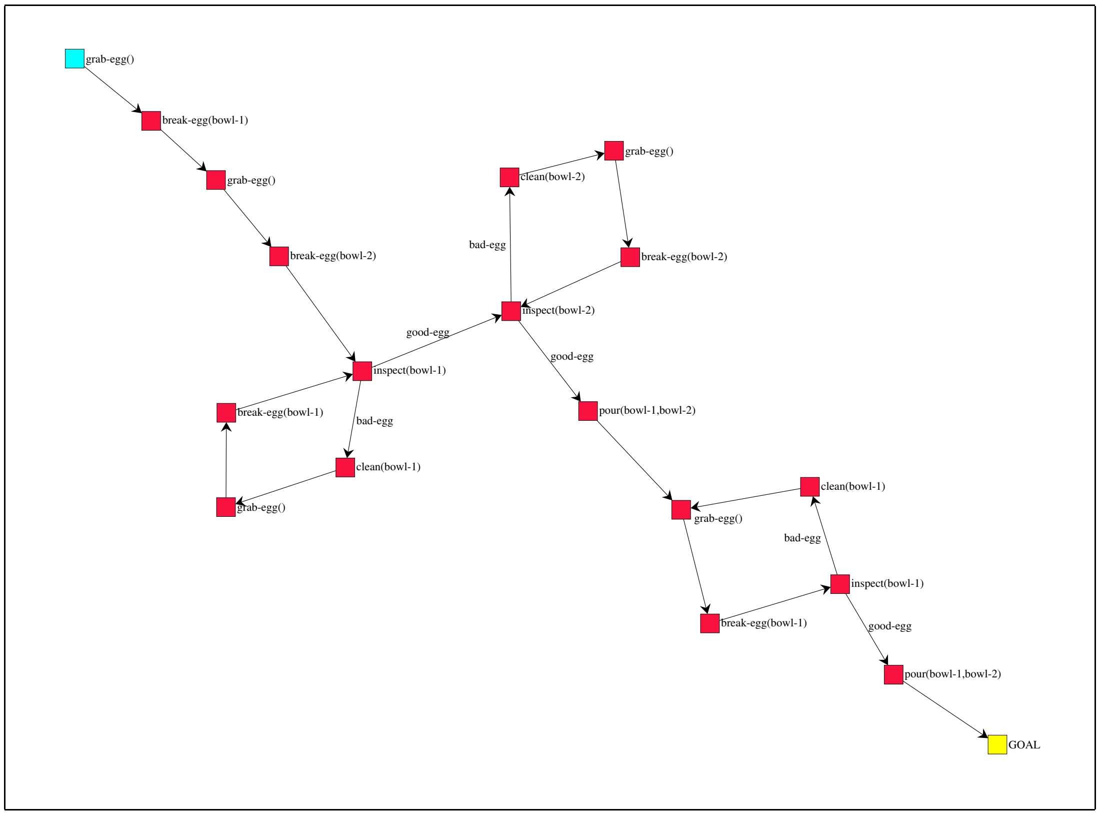

Figure 6: optimal policy for the `omelette` problem.

The `mm-1` and `mm-2` corresponds to the well-known Master Mind game. These problems are state identification problems in which the agent task is to correctly guess a hidden number by performing sensing actions. Each sense action is a guess of the hidden number. The result of the action is a pair containing the number of digits with correct value and position and the number of digits with correct value. The two versions `mm-1` and `mm-2` correspond respectively to a hidden number of 3 and 4 digits from $\{0, 1, 2\}$. Thus, `mm-1` has 27 states and `mm-2` has 81. GPT is able to find an optimal policy for this problems independently of the number of discretizations. The first instance is solved in a second and a half while the second instance takes about 20 minutes. The optimal policy for `mm-1` is shown in Fig. 7.

The `sort` problem corresponds to a sorting problem for 4 digits. The initial configuration is an unknown permutation of the 4 digits and the agent can apply comparisons and swaps between elements. The task is to apply a minimum number of actions so to take the initial permutation to the state 1234. Thus, the result is an optimal sorting circuit for 4 elements. GPT is able to solve this problem optimally and independently of the discretizations in less than a second.

The `wmp-1`, `wmp-2`, `wmp-3` are different versions of Russell and Norvig's Wumpus World problem (Russell & Norvig, 1994). They have respectively 1 gold and 1 pit in unknown position, 1 gold and 1 wumpus in unknown position, and 1 gold, 1 wumpus and 1 pit in unknown position. In the second and third case, the agent has 1 arrow that can shoot to kill

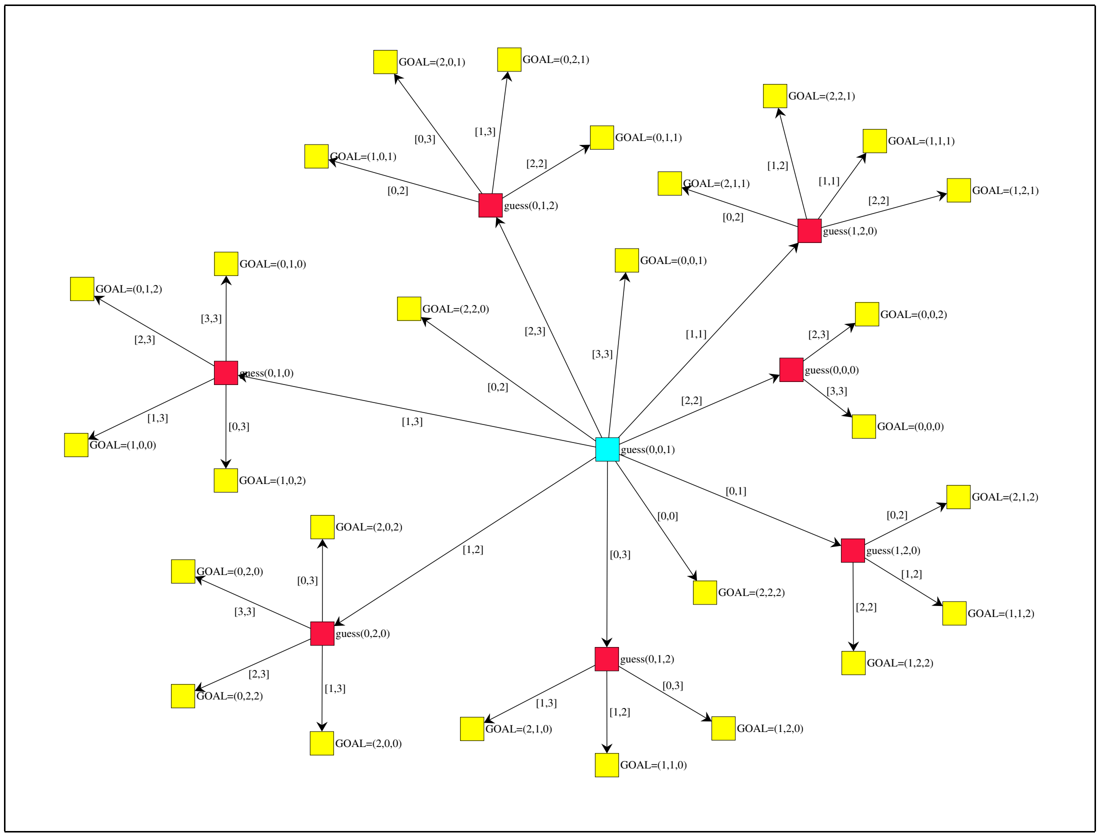

Figure 7: optimal policy for the `mm-1` problem.

the wumpus. These relatively simple problems generate large state spaces with 3,894, 11,682 and 62,070 states respectively. GPT finds optimal solutions to these problems in about 9 seconds, 20 seconds and 820 seconds respectively and independently of the discretization levels. The Wumpus World is an interesting problem for two reasons. First, it is not easy to make an specification of it neither by means of standard matrix representation or factored representations (Boutilier, Dearden, & Goldszmidt, 2000), and second, it is not easy to solve. We have included a description of this problem in the Appendix that shows the flexibility of the GPT language.

The last problem, `psr`, is an instance of a power supply restoration task in a distribution network. The problem involves a number of switches and detectors that need to be opened or closed in order to isolate an unknown number of faults in the network while maximizing the restoration of power elsewhere. The PSR domain is due to Sylvie Thiébaux and some sub-optimal solutions had been given by other planners for larger instances (Thiébaux & Cordier, 2001; Bertoli, Cimatti, Slaney, & Thiébaux, 2002). This instance has 5,728 states and is solved optimally by GPT in about a minute. A full description of the problem also appears in the Appendix.

7. Conclusions

This paper presents new results for Partially Observably Markov Decision Processes on three aspects: formal mathematical results, design of algorithms, and experimental results. In the mathematical side, we were able to prove that the optimal value function satisfies a Lipschitz condition in a novel metric space. This is an important result on its own since it implies that is possible to design optimal and robust grid-based algorithms for POMDPs. Other theoretical contributions include the new metric in belief space, the novel robustness criterion and the analysis of the algorithms.

In the algorithmic side, we presented two new grid-based algorithms for POMDPs. The first one is a general algorithm that is based on a simple and regular grid over the belief space. In this case, we were able to prove strong mathematical guarantees in both the approximation error and in the suboptimality of the resulting policies. To the best of our knowledge, this is the first optimal grid-based algorithm for general POMDPs. Indeed, all other optimal algorithms that we know are based on Sondik's representation of the value function.

The second algorithm that we presented is only for the subclass of problems known as Stochastic Shortest-Path problems in belief space. This is an important subclass since it includes diverse tasks as robot navigation and localization problems, stochastic game playing, medical diagnosis, and general planning problems under uncertainty and partial information. One distinctive characteristic of this class is that quite often we are only interested in going from a given initial state to the goal state. The second algorithm, which is based in Real-Time Dynamic Programming and a novel discretization of the belief space, is designed to compute such partial solutions. One important feature of this algorithm is that opposed to the standard RTDP, this one is guarantee to finish in finite time instead of asymptotically. Although it can also be shown that the second algorithm becomes optimal as the discretization gets finer, we have not been able to provide bounds on the approximation error. The main reason for this failure is that the algorithm uses a non-uniform grid for which is harder to provide useful bounds.

In the experimental side, we implemented the second algorithm into the General Planning Tool. GPT is an integrated system for planning under uncertainty and partial information. We tested GPT over a number of problems from different domains with different characteristics and obtained interesting results. The problems range from very simple ones as the well-known tiger problem to more complex ones like the game of Master Mind and the Wumpus World. All the problems and the GPT tool are available on the author's web page.

Very important issues remain open. In the theoretical side, any improvement in the bound for the modulus of continuity for the optimal value function would have direct impact on the complexity of the algorithms. In the particular case when all $U(i)$ are identical, it was shown that the optimal value function is uniform continuous in the standard sup topology. This means that, in such case, any reasonable grid-based algorithm is robust in the strong sense. Yet, a bound on the modulus of continuity is needed for offering optimality guarantees.

Another important area of research is in algorithms with non-uniform grids. In this case, very little is known and we need results in both theory and algorithms. We think that some effort should be directed towards the design of adaptive algorithms. These are algorithms that dynamically partition the belief space into areas of different granularity in order to minimize the number of grid points while preserving the guarantees.

Finally, a more formal comparison between the advantages and disadvantages of optimal grid-based algorithm and algorithms based on Sondik's representation is needed. We believe that there are some classes of problems in which each class of algorithms perform better.

Acknowledgements

This research was supported in part by grants from NSF, ONR, AFOSR, and DoD MURI program. The author is also supported by a USB/CONICIT fellowship from Venezuela.

References

Aberdeen, D., & Baxter, J. (2002). Scalable internal-state policy-gradient methods for POMDPs. In Sammut, C., & Hoffmann, A. (Eds.), *Proc. 19th International Conf. on Machine Learning*, pp. 3–10, University of New South Wales, Sydney. Morgan Kaufmann.

Astrom, K. (1965). Optimal control of Markov Decision Processes with incomplete state estimation. *J. Math. Anal. Appl.*, *10*, 174–205.

Bacchus, F. (2000). AIPS-00 Planning Competition. See `http://www.cs.toronto.edu/aips2000`.

Barto, A., Bradtke, S., & Singh, S. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*, 81–138.

Bertoli, P., Cimatti, A., Slaney, J., & Thiébaux, S. (2002). Solving power supply restoration problems with planning via symbolic model-checking. In Harmelen, F. V. (Ed.), *Proc. 15th European Conf. on Artificial Intelligence*, Lyon, France. IOS Press.

Bertsekas, D. (1995). *Dynamic Programming and Optimal Control, (2 Vols).* Athena Scientific.

Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-Dynamic Programming.* Athena Scientific.

Bonet, B. (2002). An ϵ-optimal grid-based algorithm for Partially Observable Markov Decision Processes. In Sammut, C., & Hoffmann, A. (Eds.), *Proc. 19th International Conf. on Machine Learning*, pp. 51–58, University of New South Wales, Sydney. Morgan Kaufmann.

Bonet, B., & Geffner, H. (2000). Planning with incomplete information as heuristic search in belief space. In Chien, S., Kambhampati, S., & Knoblock, C. (Eds.), *Proc. 5th International Conf. on Artificial Intelligence Planning and Scheduling*, pp. 52–61, Breckenridge, CO. AAAI Press.

Bonet, B., & Geffner, H. (2001). GPT: a tool for planning with uncertainty and partial information. In Cimatti, A., Geffner, H., Giunchiglia, E., & Rintanen, J. (Eds.), *Proc. IJCAI-01 Workshop on Planning with Uncertainty and Partial Information*, pp. 82–87, Seattle, WA.

Boutilier, C., Dearden, R., & Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, *121*(1–2), 49–107.

Cassandra, A., Kaelbling, L., & Littman, M. (1994). Acting optimally in partially observable stochastic domains. In Hayes-Roth, B., & Korf, R. (Eds.), *Proc. 12th National Conf. on Artificial Intelligence*, pp. 1023–1028, Seattle, WA. AAAI Press / MIT Press.

Cassandra, A., Littman, M., & Zhang, N. (1997). Incremental pruning: A simple, fast, exact algorithm for Partially Observable Markov Decision Processes. In Geiger, D., & Shenoy, P. (Eds.), *Proc. 13th Conf. on Uncertainty in Artificial Intelligence*, pp. 54–61. Morgan Kaufmann.

Cassandra, A. R. (1998). *Exact and approximate algorithms for partially observable Markov Decision Processes.* Ph.D. thesis, Brown University.

Fristedt, B., & Gray, L. (1997). *A Modern Approach to Probability Theory.* Birkhauser.

Hansen, E., & Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, *129*, 35–62.

Hauskrecht, M. (2000). Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, *13*, 33–94.

Howard, R., & Matheson, J. (1984). Influence diagrams. In Howard, R., & Matheson, J. (Eds.), *The principles and applications of decision analysis.* Strategic Decision Group, Menlo Park, CA.

Kaelbling, L., Littman, M., & Cassandra, A. (1999). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*, 99–134.

Korf, R. (1990). Real-time heuristic search. *Artificial Intelligence*, *42*, 189–211.

Littman, M. (1996). *Algorithms for Sequential Decision Making.* Ph.D. thesis, Brown University.

Lovejoy, W. (1991). A survey of algorithmic techniques for partially observed Markov decision processes. *Annals of Operations Research*, *28*, 47–66.

McDermott, D. (1998a). Official results for the first planning competition held at AIPS–98. See `http://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html`.

McDermott, D. (1998b). Planning Domain Definition Language. See `http://ftp.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz`.

Pearl, J. (1983). *Heuristics.* Morgan Kaufmann.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann.

Royden, H. L. (1968). *Real Analysis* (Second edition). Collier MacMillan.

Russell, S., & Norvig, P. (1994). *Artificial Intelligence: A Modern Approach.* Prentice Hall.

Smallwood, R., & Sondik, E. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, *21*, 1071–1088.

Sondik, E. (1971). *The Optimal Control of Partially Observable Markov Decision Processes.* Ph.D. thesis, Stanford University.

Sondik, E. (1978). The optimal control of partially observable Markov decision processes over the infinite horizon: discounted costs. *Operations Research*, *26*(2).

Sutton, R., & Barto, A. (1998). *Reinforcement Learning: An Introduction.* MIT Press.

Thiébaux, S., & Cordier, M. (2001). Supply restoration in power distribution systems – a benchmark for planning under uncertainty. In *Proc.6th European Conf. on Planning*, pp. 85–95, Toledo, Spain. Springer.

Thrun, S. (2000). Probabilistic algorithms in robotics. *AI Magazine*, *21*(4), 93–109.

Thrun, S., Langford, J. C., & Fox, D. (1999). Monte Carlo hidden Markov models: Learning non-parametric models of partially observable stochastic processes. In *Proc. 16th International Conf. on Machine Learning*, pp. 415–424. Morgan Kaufmann.

Zhang, N., & Lee, S. (1997). Planning with Partially Observable Markov Decision Processes: Advances in exact solution methods. In Geiger, D., & Shenoy, P. (Eds.), *Proc. 13th Conf. on Uncertainty in Artificial Intelligence*, pp. 523–530. Morgan Kaufmann.

Zhang, N., & Liu, W. (1997). A model approximation scheme for planning in partially observable stochastic domains. *Journal of Artificial Intelligence Research*, *7*, 199–230.

Appendix: Encodings for the Wumpus World and PSR

```
;; Wumpus World [Russell & Norvig book, p. 153]
;; ==================================
;;
;; small grid world:
;;
;;    (0,2)  (1,2)  (2,2)
;;    (0,1)  (1,1)  (2,1)
;;    (0,0)  (1,0)  (2,0)
;;
;; Variables:
;; ==========
;;
;;   facing = { 0=north, 1=east, 2=south, 3=west }
;;   arrows = number of remaining arrows
;;   num_wumpus = number of remaining wumpusses (?)
;;   x, y = grid position
;;   ok = true if we're still alive
;;   incave = true if we're still in the cave

(define (domain wumpus)
  (:model (:dynamics :probabilistic) (:feedback :partial))
  (:types PIT WUMPUS GOLD)
  (:functions (pxpos PIT :integer[0,2])
              (pypos PIT :integer[0,2])
              (wxpos WUMPUS :integer[0,2])
              (wypos WUMPUS :integer[0,2])
              (gxpos GOLD :integer[0,2])
              (gypos GOLD :integer[0,2]))
  (:predicates (taken GOLD) (alive WUMPUS))
  (:objects arrows - :integer[0,10]
            num_wumpus - :integer[0,10]
            facing - :integer[0,3]
            x y - :integer[0,2]
            ok - :boolean
            incave - :boolean)

  ;; useful predicates
  (:predicate (glitter ?x ?y - :integer[0,2])
     (:exists ?g - GOLD (:and (:not (taken ?g)) (= (gxpos ?g) ?x) (= (gypos ?g) ?y)))
  )

  (:predicate (breeze ?x ?y - :integer[0,2])
     (:exists ?p - PIT
        (:or (:and (= (pxpos ?p) ?x) (= (pypos ?p) (+ ?y 1)))
             (:and (= (pxpos ?p) ?x) (= (pypos ?p) (- ?y 1)))
             (:and (= (pypos ?p) ?y) (= (pxpos ?p) (+ ?x 1)))
             (:and (= (pypos ?p) ?y) (= (pxpos ?p) (- ?x 1))))
     )
  )

  (:predicate (stench ?x ?y - :integer[0,2])
     (:exists ?w - WUMPUS
        (:or (:and (= (wxpos ?w) ?x) (= (wypos ?w) (+ ?y 1)))
             (:and (= (wxpos ?w) ?x) (= (wypos ?w) (- ?y 1)))
             (:and (= (wypos ?w) ?y) (= (wxpos ?w) (+ ?x 1)))
             (:and (= (wypos ?w) ?y) (= (wxpos ?w) (- ?x 1))))
     )
  )
```

```
;; shoot an arrow in straight direction killing at most one wumpus in its path
(:action shoot
   :precondition (:and (= incave true) (= ok true) (> arrows 0))
   :effect
     (:set arrows (- arrows 1))
     (:forall ?w - WUMPUS
        (:when (:and (= num_wumpus *num_wumpus) ;; while haven't killed
                     (:or (:and (= facing 0) (= x (wxpos ?w)) (< y (wypos ?w)))
                          (:and (= facing 2) (= x (wxpos ?w)) (> y (wypos ?w)))
                          (:and (= facing 1) (= y (wypos ?w)) (< x (wxpos ?w)))
                          (:and (= facing 3) (= y (wypos ?w)) (> x (wxpos ?w)))
                     ))
              (:set (alive ?w) false)
              (:set num_wumpus (- num_wumpus 1))
        )
     )
   :observation num_wumpus
)

;; move one step in facing direction
(:action move
   :precondition (:and (= incave true) (= ok true))

   :effect
     (:when (= facing 0) (:set y (:if (= y 2) 2 (+ y 1))))
     (:when (= facing 2) (:set y (:if (= y 0) 0 (- y 1))))
     (:when (= facing 1) (:set x (:if (= x 2) 2 (+ x 1))))
     (:when (= facing 3) (:set x (:if (= x 0) 0 (- x 1))))

     ;; dead conditions
     (:when (:exists ?w - WUMPUS (:and (= *x (wxpos ?w)) (= *y (wypos ?w))))
            (:set ok false))
     (:when (:exists ?p - PIT (:and (= *x (pxpos ?p)) (= *y (pypos ?p))))
            (:set ok false))

   :observation ok (glitter x y) (breeze x y) (stench x y)
)

(:action rot_right
   :precondition (:and (= incave true) (= ok true))
   :effect (:set facing (+<4> facing 1))                  ;; addition mod 4
   :observation ok (glitter x y) (breeze x y) (stench x y)
)

(:action rot_left
   :precondition (:and (= incave true) (= ok true))
   :effect (:set facing (-<4> facing 1))                  ;; substraction mod 4
   :observation ok (glitter x y) (breeze x y) (stench x y)
)

(:action grab
   :precondition (:and (= incave true) (= ok true))
   :effect
     (:forall ?g - GOLD
        (:when (:and (= x (gxpos ?g)) (= y (gypos ?g)) (:not (taken ?g)))
              (:set (taken ?g) true))
     )
)
```

```
  ;; the following actions reset all variables in order to reduce state-space
  (:action climb
     :precondition (:and (= incave true) (= ok true) (= x 0) (= y 0))
     :effect (:set incave false)
             (:set facing 0)
             (:forall ?g - GOLD
                (:set (gxpos ?g) 0)
                (:set (gypos ?g) 0))
             (:forall ?p - PIT
                (:set (pxpos ?p) 0)
                (:set (pypos ?p) 0))
             (:forall ?w - WUMPUS
                (:set (wxpos ?w) 0)
                (:set (wypos ?w) 0)
                (:set (alive ?w) false))
     :cost (:sum ?g - GOLD (:if (:not (taken ?g)) 100 0))
  )

  ;; the finish action is used only when we are dead
  (:action finish
     :precondition (:and (= incave true) (= ok false))
     :effect (:set incave false)
             (:set facing 0)
             (:set ok true)
             (:set x 0)
             (:set y 0)
             (:forall ?g - GOLD
                (:set (gxpos ?g) 0)
                (:set (gypos ?g) 0))
             (:forall ?p - PIT
                (:set (pxpos ?p) 0)
                (:set (pypos ?p) 0))
             (:forall ?w - WUMPUS
                (:set (wxpos ?w) 0)
                (:set (wypos ?w) 0)
                (:set (alive ?w) false))
     :cost 1000
  )
)

;; PROBLEM: 1 gold, 0 wumpus, 1 pit, 0 arrows
(define (problem p1)
  (:domain wumpus)
  (:objects g1 - GOLD p1 - PIT)
  (:init (:set x 0)
         (:set y 0)
         (:set facing 0)
         (:set ok true)
         (:set incave true)
         (:set arrows 0)
         (:set num_wumpus 0)

         (:set (gxpos g1) :in { 0 1 2 })
         (:set (gypos g1) :in { 0 1 2 })
         (:set (taken g1) false)
         (:not (:and (= (gxpos g1) 0) (= (gypos g1) 0)))

         (:set (pxpos p1) :in { 0 1 2 })
         (:set (pypos p1) :in { 0 1 2 })
         (:not (:and (= (pxpos p1) 0) (= (pypos p1) 0)))
         (:not (:and (= (pxpos p1) (gxpos g1)) (= (pypos p1) (gypos g1))))
  )
  (:goal (= incave false))
)

;; PROBLEM: 1 gold, 1 wumpus, 0 pit, 1 arrows
(define (problem p2)
  (:domain wumpus)
  (:objects g1 - GOLD w1 - WUMPUS)
  (:init (:set x 0)
         (:set y 0)
         (:set facing 0)
         (:set ok true)
         (:set incave true)
         (:set arrows 1)
         (:set num_wumpus 1)

         (:set (gxpos g1) :in { 0 1 2 })
         (:set (gypos g1) :in { 0 1 2 })
         (:set (taken g1) false)
         (:not (:and (= (gxpos g1) 0) (= (gypos g1) 0)))

         (:set (wxpos w1) :in { 0 1 2 })
         (:set (wypos w1) :in { 0 1 2 })
         (:not (:and (= (wxpos w1) 0) (= (wypos w1) 0)))
         (:not (:and (= (wxpos w1) (gxpos g1)) (= (wypos w1) (gypos g1))))
  )
  (:goal (= incave false))
)

;; PROBLEM: 1 gold, 1 wumpus, 1 pit, 1 arrows
(define (problem p3)
  (:domain wumpus)
  (:objects g1 - GOLD w1 - WUMPUS p1 - PIT)
  (:init (:set x 0)
         (:set y 0)
         (:set facing 0)
         (:set ok true)
         (:set incave true)
         (:set arrows 1)
         (:set num_wumpus 1)

         (:set (gxpos g1) :in { 0 1 2 })
         (:set (gypos g1) :in { 0 1 2 })
         (:set (taken g1) false)
         (:not (:and (= (gxpos g1) 0) (= (gypos g1) 0)))

         (:set (wxpos w1) :in { 0 1 2 })
         (:set (wypos w1) :in { 0 1 2 })
         (:not (:and (= (wxpos w1) 0) (= (wypos w1) 0)))
         (:not (:and (= (wxpos w1) (gxpos g1)) (= (wypos w1) (gypos g1))))

         (:set (pxpos p1) :in { 0 1 2 })
         (:set (pypos p1) :in { 0 1 2 })
         (:not (:and (= (pxpos p1) 0) (= (pypos p1) 0)))
         (:not (:and (= (pxpos p1) (gxpos g1)) (= (pypos p1) (gypos g1))))
         (:not (:and (= (pxpos p1) (wxpos w1)) (= (pypos p1) (wypos w1))))
  )
  (:goal (= incave false))
)
```

```
;; Power System Restoration (PSR) due to Sylvie Thiebaux
;; ======================================================

(define (domain power_supply)
  (:model (:dynamics :probabilistic) (:feedback :partial))
  (:types SWITCH SIDE LINE MODE)
  (:functions
    (ac_mode SWITCH MODE)   ; actuator mode; can be ok, out-of-order, or liar
    (fd_mode SWITCH MODE)   ; fault detector mode; can be ok, out-of-order, or liar
    (opposite SIDE SIDE)    ;opposite side of a side
  )
  (:predicates
    (ext LINE SWITCH SIDE)  ; ext(l,x,sx): line l is connected to side sx of x
    (breaker SWITCH)        ; breaker(x): switch x is a circuit-breaker
    (closed SWITCH)         ; closed(x): switch x is closed
    (faulty LINE)           ; faulty(l): line l is faulty
    (fd SWITCH)             ; fd(x): fault detector reading for switch x
    (pd_ok SWITCH)          ; pd_ok(x): true iff the position detector for x is ok
  )
  (:objects side1 side2 - SIDE
            ok out liar - MODE
            done - :boolean
  )

  ; PREDICATES

  (:predicate (con ?x - SWITCH ?sx - SIDE ?y - SWITCH ?sy - SIDE)
     (:and (:or (:not (= ?x ?y))
                (:not (= ?sx ?sy)))
          (:exists ?l - LINE
             (:and (ext ?l ?x ?sx) (ext ?l ?y ?sy))))
  )

  (:predicate (upstream ?x - SWITCH ?sx - SIDE ?y - SWITCH ?sy - SIDE)
    (:and (closed ?x)
          (:or (:and (breaker ?x) (con ?x (opposite ?sx) ?y ?sy))
               (:and (breaker ?x)
                     (:exists ?z - SWITCH
                        (:and (closed ?z)
                              (:exists ?sz - SIDE
                                 (:and (con ?z (opposite ?sz) ?y ?sy)
                                       (upstream ?x ?sx ?z ?sz))))))
               (:and (:not (breaker ?x))
                     (:or (:and (con ?x (opposite ?sx) ?y ?sy)
                                (:exists ?z - SWITCH
                                   (:and (breaker ?z)
                                         (:exists ?sz - SIDE (upstream ?z ?sz ?x ?sx)))))
                          (:exists ?u - SWITCH
                             (:and (closed ?u)
                                   (:exists ?su - SIDE
                                      (:and (con ?u (opposite ?su) ?y ?sy)
                                            (upstream ?x ?sx ?u ?su))))))))
  )

  (:predicate (affected ?x - SWITCH)
    (:exists ?l - LINE
       (:and (faulty ?l)
             (:exists ?y - SWITCH
                (:exists ?sy - SIDE
                   (:and (ext ?l ?y ?sy)
                         (:exists ?sx - SIDE (upstream ?x ?sx ?y ?sy)))))))
  )
```

```
  (:predicate (fed ?x - SWITCH)
    (:or (:and (breaker ?x) (closed ?x))
         (:exists ?y - SWITCH
            (:and (breaker ?y)
                  (:exists ?sy - SIDE
                     (:exists ?sx - SIDE (upstream ?y ?sy ?x ?sx))))))
  )

  ; RAMIFICATIONS: the order is important, don't change!

  (:ramification fd_ramification
    :parameters ?x - SWITCH
    :effect (:when (= (fd_mode ?x) out) (:set (fd ?x) false))
            (:when (:and (:not (= (fd_mode ?x) out)) (fed ?x))
               (:set (fd ?x) (:if (= (fd_mode ?x) ok)
                                  (:formula (affected ?x))
                                  (:formula (:not (affected ?x))))))
  )

  (:ramification open_ramification
    :parameters ?x - SWITCH
    :effect (:when (:and (breaker ?x) (affected ?x)) (:set (closed ?x) false))
  )

  ; ACTIONS

  (:action open
    :parameters   ?x - SWITCH
    :effect       (:when (= (ac_mode ?x) ok) (:set (closed ?x) false))
    :observation
      (= (ac_mode ?x) out)
      (:vector ?y - SWITCH (:if (pd_ok ?y) (:formula (closed ?y)) false))
      (:vector ?y - SWITCH (:formula (fd ?y)))
  )

  (:action close
    :parameters   ?x - SWITCH
    :effect       (:when (= (ac_mode ?x) ok) (:set (closed ?x) true))
    :observation
      (= (ac_mode ?x) out)
      (:vector ?y - SWITCH (:if (pd_ok ?y) (:formula (closed ?y)) false))
      (:vector ?y - SWITCH (:formula (fd ?y)))
  )

  (:action sense
    :observation
      (:vector ?y - SWITCH (:if (pd_ok ?y) (:formula (closed ?y)) false))
      (:vector ?y - SWITCH (:formula (fd ?y)))
  )

  (:action finish
    :effect (:set done true)
    :cost (:sum ?x - SWITCH (:if (fed ?x) 0 10))
  )
)
```

```
(define (problem dom)
        (:domain power_supply)
        (:objects cb1 sd1 sd2 sd3 - SWITCH
                  l1 l2 l3 l4 - LINE)
        (:init  (:set (breaker cb1) true)
                (:set (opposite side1) side2)
                (:set (opposite side2) side1)
                (:set (ext l1 cb1 side2) true)
                (:set (ext l1 sd1 side2) true)
                (:set (ext l2 sd1 side1) true)
                (:set (ext l2 sd2 side2) true)
                (:set (ext l3 sd2 side1) true)
                (:set (ext l3 sd3 side2) true)
                (:set (ext l4 sd3 side1) true)
                (:set (closed cb1) true)
                (:set (closed sd1) true)
                (:set (closed sd2) true)
                (:set (closed sd3) true)
                (:set (fd_mode cb1) :in {ok liar out})
                (:set (fd_mode sd1) :in {ok liar out})
                (:set (fd_mode sd2) :in {ok liar out})
                (:set (fd_mode sd3) :in {ok liar out})
                (= (:sum ?s - SWITCH (:if (= (fd_mode ?s) ok) 0 1)) 1)
                (:set (pd_ok cb1) true)
                (:set (pd_ok sd1) true)
                (:set (pd_ok sd2) true)
                (:set (pd_ok sd3) true)
                (:set (ac_mode cb1) ok)
                (:set (ac_mode sd1) ok)
                (:set (ac_mode sd2) ok)
                (:set (ac_mode sd3) ok)
                (:set (faulty l1) :in {true false})
                (:set (faulty l2) :in {true false})
                (:set (faulty l3) :in {true false})
                (:set (faulty l4) :in {true false})
                ;(= (:sum ?l - LINE (:if (faulty ?l) 1 0)) 3)
                (:set done false)
        )
        (:goal (= done true))
)
```