Solving Large POMDPs using Real Time Dynamic Programming

Héctor Geffner and Blai Bonet
Departamento de Computación
Universidad Simón Bolívar
Aptdo. 89000, Caracas 1080-A, Venezuela
{hector,bonet}@usb.ve

Abstract

Partially Observable Markov Decision Processes (POMDPs) are general models of sequential decision problems in which both actions and observations may be noisy. Many problems of interest can be formulated as POMDPs yet the use of POMDPs has been limited by the lack of effective algorithms: optimal algorithms don't scale up and heuristic algorithms often do poorly. In this paper, a new POMDP algorithm is introduced that combines the benefits of optimal and heuristic procedures producing good solutions quickly even in problems that are large. Like optimal procedures, the procedure RTDP-BEL attempts to solve the *information* MDP, yet like heuristic procedures, it makes decisions in real time following a suitable heuristic function. RTDP-BEL is a Real Time Dynamic Programming algorithm [1], namely, a greedy search algorithm that learns to solve MDPs by repeatedly updating the heuristic values of the states that are visited. As shown by Barto *et al.* such updates eventually deliver an optimal behavior provided that the state space is finite and the initial heuristic values are admissible. Since information MDPs have infinite state spaces we discretize probabilities and combine them with heuristic values obtained from the underlying MDP. Although the resulting algorithm is not guaranteed to be optimal, experiments over a number of benchmarks suggest that large POMDPs are quickly and consistently solved, and that solutions, if not optimal, tend to be very good.

1 Introduction

Partially Observable Markov Decision Processes (POMDPs) are general models of sequential decision problems in which both actions and observations may be probabilistic [17, 12, 6]. Many problems of interest can be formulated as POMDPs yet the use of POMDPs has been limited by the lack of effective algorithms [7]. In this paper, we introduce a new POMDP algorithm that combines the benefits of exact and heuristic algorithms producing good solutions in a short time. The algorithm is a Real Time Dynamic Programming (RTDP) procedure [1] and can be seen both as a learning real-time search algorithm [10] or as an asynchronous form of value iteration [2]. While RTDP is guaranteed to yield optimal solutions for a large class of finite-state MDPs, it can only produce approximate solutions to POMDPs. Nonetheless, the results obtained over a number of benchmark problems suggest that these solutions are good, improve with time, and can be computed reasonably fast even in large problems.

The paper is mostly self-contained and briefly reviews Learning Real Time Search (Section 2), Markov Decision Problems (Section 3), Real Time Dynamic Programming (Section 4), and Partially Observable MDPs (Section 5). It then presents the POMDP algorithm (Section 6) an empirical evaluation (Section 7), and a summary of the main ideas and current work (Section 8).

2 Learning Real Time Search

Real time search algorithms are algorithms that, like chess playing programs, perform a limited amount of local search before committing and executing moves [10]. For example, a real time search algorithm for solving the 8-puzzle may *evaluate* each of the states that can be obtained in a single move and then *move* to the state with lowest value. If the 8-puzzle is simulated in the computer, the resulting algorithm is a form of heuristic hill-climbing; otherwise it's a form of closed-loop control: the state is observed, an action is selected and executed, a new state is observed, and so on.

In a real time search framework, the agent does not solve the problem before acting; it just performs a limited lookahead, make what appears to be the best move, and repeats this process until the goal is reached. When there is sufficient time, the lookahead may be designed to seek global optimal moves (by making use of algorithms such as A^*), while when time is scarce, it may just consider the value of neighbor states only. In between, many alternatives are possible like considering the value of the states that can be reached in a fixed number of moves [10].

In the simplest case, when there is *no lookahead*, the basic loop of the heuristic real time search algorithm (RTS) is given in Fig. 1, where $c(a, s)$ stands for the positive cost of taking action a in state s, s_a stands for the state predicted after doing action a in s, and h is the heuristic function.

1. **Evaluate** each action a applicable in current state s as
$$Q(a, s) = c(a, s) + h(s_a)$$
2. **Apply** action $\mathbf{a}$ that minimizes $Q(\mathbf{a}, s)$, breaking ties randomly
3. **Observe** resulting state s'
4. **Exit** if s' is a goal state, else set s to s' and go to 1

Figure 1: Real Time Search (RTS)

RTS evaluates the actions that are applicable in the current state s, applies the best action $\mathbf{a}$, and repeats the same process until reaching a goal. If the action model is completely accurate, Step 3 can be omitted and s' can be replaced by the predicted state $s_{\mathbf{a}}$. Otherwise, the use of the observed state s' as opposed to the predicted state s_a, makes RTS a closed-loop algorithm that is more robust to the possible presence of perturbations in the real or simulated system.

RTS, unlike systematic heuristic algorithms such as A^* [14], does not yield *optimal* (i.e., minimum-cost) solutions in general even when the heuristic function h is *admissible* (i.e., underestimates the cost to the goal). Actually, like most hill climbing algorithms, RTS can be trapped into loops and miss the goal state completely. A simple variation due to Korf, however, avoids these problems while retaining the greedy character of RTS.

Learning Real Time A^*, abbreviated LRTA* [10], is a slight modification of RTS in which the heuristic function $h(s)$ is used to provide the *initial* cost estimates only. Every time an action a is selected and applied in s, such estimates, symbolized as $V(s)$, are updated as:

$$V(s) := c(a, s) + V(s_a) \tag{1}$$

where $V(s_a)$ is the estimate of the predicted state s_a. These updates aim to enforce the relation that has to hold between $V(s)$ and $V(s_a)$ when a is an optimal action in s, which as noted in [1] is the key idea in dynamic programming.

For the implementation of LRTA*, the estimates $V(s)$ are stored in a hash table that initially contains the heuristic value of the starting state only. Then, when the value $V(s)$ of a state s that is not in the table is needed, a new entry with $V(s)$ set to $h(s)$ is allocated. These entries are updated following (1) when a move from s is performed.

The loop of the LRTA* algorithm extends the RTS loop with a single additional *update* step as indicated Fig. 2.

1. **Evaluate** each action a applicable in s as

$$Q(a, s) = c(a, s) + V(s_a)$$

   initializing $V(s_a)$ to $h(s_a)$ when s_a is not in the table
2. **Apply** action $\mathbf{a}$ with minimum $Q(\mathbf{a}, s)$ value, breaking ties randomly
3. **Update** $V(s)$ to $Q(\mathbf{a}, s)$
4. **Observe** resulting state s'
5. **Exit:** if s' is a goal, else set s to s' and go to 1

Figure 2: Learning Real Time Search (LRTA*)

A basic property of LRTA* is that it does not get trapped into loops and always reaches the goal if the goal is reachable. More interestingly, when the heuristic function $h(s)$ is *admissible*, consecutive trials of LRTA* eventually yield an optimal behavior [10].[1] As a result, if LRTA* is given the non-informative but admissible heuristic $h(s) = 0$ in the 8-puzzle, it will eventually find the optimal heuristic values and the optimal moves. The rate at which LRTA* finds these moves depends in general on the quality of h. A better heuristic function yields a more focused search, a smaller hash table, and hence, a higher ratio of updates on the relevant states and faster convergence.

[1] Both results presume that the action model is correct, the state space is finite, and goals are reachable from every state [10].

3 MDPs

The type of MDPs that we consider are a simple generalization of the standard search model used in AI in which actions can have *probabilistic* effects and future costs may be discounted. Goal MDPs, as we call them, are characterized by:

- a state space S
- initial and goal situations given by sets of states
- sets $A(s) \subseteq A$ of actions applicable in each state s
- positive costs $c(a, s)$ of performing action a in s,
- transition probabilities $P_a(s'|s)$ of ending up in state s' after doing action $a \in A(s)$ in s, and
- a discount factor $0 \leq \gamma \leq 1$

Since the effects of actions are no longer predictable, the solution of an MDP is not an action sequence but a function mapping states s into actions $a \in A(s)$. Such a function is called a *policy* and its effect is to assign a probability to each state trajectory. For convenience, actions in the goal states are assumed to have zero costs and no effects (i.e., goal are absorbing).

The *expected cost* of a policy from a given state is the (possibly discounted) weighted average of the costs of all trajectories starting in that state times their probability. A policy is *optimal* when it's expected cost from any state is minimized. General conditions for the existence of such policies and algorithms for computing them can be found in [15, 2]. The key fact underlying such algorithms is that a necessary and sufficient condition for a policy π^* to be optimal is that the expected costs $V^*(s)$ that result from starting in state s and then acting according to π^* must satisfy Bellman's equation:

$$V^*(s) = \min_{a \in A(s)} [c(a, s) + \gamma \sum_{s' \in S} P_a(s'|s) V^*(s')] \tag{2}$$

Indeed, it is common to solve MDPs by obtaining the values $V^*(s)$ from this equation and plugging the results into:

$$\pi^*(s) = \overset{-1}{\min_{a \in A(s)}} [c(a, s) + \gamma \sum_{s' \in S} P_a(s'|s) V^*(s')] \tag{3}$$

The *value iteration method* [15] solves (2) by successive approximations in which at each time point i, estimates $V_i(s')$ of $V^*(s')$ are plugged into the right hand side of (2) to yield better estimates $V_{i+i}(s)$ on the left hand side. Such an operation is called an *update,* and for a large class of MDPs, repeated updates eventually yield the optimal values.

4 Real Time Dynamic Programming

Classical techniques for solving MDPs like value iteration have wide applications in Engineering and can be used to solve problems with million of states [15]. Yet many problems of interest, in particular in AI, have state spaces that are significantly larger. For such problems, even a single iteration of value iteration is unfeasible. Part of the problem is that the standard DP methods are designed to look for optimal policies over *all* possible states. In most cases this is not needed. Often when the initial and goal states are given, only a small fraction of the states are relevant for solving the problem and in such cases most of the effort that goes into updating *all* states is wasted. Real Time Dynamic Programming is a family of methods for solving MDPs that tries to avoid such wasteful updates by performing the updates concurrently with the problem solving [1]. The idea, like in Korf's LRTA*, is to update while searching, focusing only on the states that are encountered in the search.

More precisely, while an iteration of value iteration uses Bellman's equation to update the values of *all* states s in the form:

$$V^*_{t+1}(s) := \min_{a \in A(s)} [c(a,s) + \gamma \sum_{s' \in S} p_a(s'|s) V^*_t(s')] \tag{4}$$

an iteration of RTDP uses the same expression to update the value of the *current* state only. Moreover, after such an update, the best action according to the current estimates is applied and the same process is repeated on the state that results. The basic loop of the RTDP algorithm is shown in Figure 3.

By comparing the algorithms in Figures 2 and 3, it's clear that the main difference between LRTA* and RTDP is the evaluation of the 'promise' $Q(a,s)$ of action a in s. Indeed as noted in [1], RTDP is nothing else but the stochastic generalization of LRTA* and collapses to LRTA* when there is no discounting and actions are deterministic (i.e., all probabilities are either 0 or 1).

1. **Evaluate** each action a applicable in s as:

$$Q(a,s) = c(a,s) + \gamma \sum_{s' \in S} p_a(s'|s)V(s')$$

   initializing $V(s')$ to $h(s')$ when s' is not in the table

2. **Apply** action $\mathbf{a}$ with minimum $Q(\mathbf{a}, s)$ value, breaking ties randomly
3. **Update** $V(s)$ to $Q(\mathbf{a}, s)$
4. **Observe** resulting state s'
5. **Exit** if s' is a goal, else set s to s' and go to 1

Figure 3: Real Time Dynamic Programming (RTDP)

The heuristic values $h(s)$ used to initialize $V(s)$ are crucial in both RTDP and LRTA*: better heuristics mean a more focused search for the goal and a more focused search means more updates on the states that matter. Likewise, if the heuristic function is admissible, RTDP eventually converges to an optimal solution [1].

Good heuristic functions can make LRTA* and RTDP suitable for problems that have huge state spaces. For example, [4] presents a LRTA* algorithm for Strips planning that handles problems with more than 10^{20} states. Here we apply RTDP to MDPs with very large spaces as well. Heuristic functions indeed are an alternative to the function approximation methods used in model-free RTDP methods [18, 19] for dealing with large problems.

5 Partially Observable MDPs

Partially Observable MDPs generalize MDPs allowing agents to have incomplete information about the state of the environment [17, 12]. Besides the sets of actions and states, and the probability and cost functions, a POMDP involves prior beliefs in the form of a probability distribution $P(s)$ and an *observation model* in the form of a set O of possible observations and probabilities $P_a(o|s)$ of observing $o \in O$ in state s after having done action a.

The techniques considered above are not directly applicable to POMDPs because while they do not presume that the agent can *predict* the next state, they do assume that he can *recognize* the next state once he gets there.

The most common way to solve POMDPs is by formulating them as completely observable MDPs over the *belief states* of the agent [17, 6], where belief states are probability distributions over the real states s. Indeed while the effects of actions on states cannot be predicted, the effects of actions on *belief states* can. More precisely, the belief state b_a that results from having done action a in the belief state b, and the belief state b_a^o that results from having observed o after having done a in b, are given by the equations [6]:

$$b_a(s) = \sum_{s' \in S} P_a(s|s')b(s') \quad (5)$$

$$b_a(o) = \sum_{s \in S} P_a(o|s)b_a(s) \quad (6)$$

$$b_a^o(s) = P_a(o|s)b_a(s)/b_a(o) \quad \text{when } b_a(o) \neq 0 \quad (7)$$

As a result, the *incompletely observable* problem of going from an initial state to a goal state can be transformed into the *completely observable* problem of going from one *initial belief state* to a *final belief state.* The Bellman Equation for the resulting *belief* MDP is

$$V^*(b) = \min_{a \in A(b)} [c(a,b) + \gamma \sum_{o \in O} b_a(o)V^*(b_a^o)] \quad (8)$$

where $c(a, b)$ is the average cost of doing action a in b

$$c(a,b) = \sum_{s \in S} c(a,s)b(s) \quad (9)$$

and $A(b)$ is the set of actions a that are applicable in every state s possible with respect to b, i.e., with $b(s) > 0$. Note that a belief MDP is nothing else but an MDP with probability distributions as states.

We define the *goal belief states* as the probability distributions b_G such that $b_G(s) = 0$ for all non-goal states s, i.e., $s \notin G$. While in goal MDPs we are interested in policies over S that minimize the expected cost to the goals $G \subseteq S$, in goal POMDPs we are interested in policies over $Bel(S)$ – the probability distributions over S – that minimize the expected cost to the beliefs states b_G. Finding such policies however is not simple. Belief MDPs involve state spaces that are infinite and continuous. For this reason, exact algorithms exploit a finite representation of the value function $V^*(b)$ due to Sondik [17], who showed that the optimal value function V_t that results when the next t time points are considered (the so-called optimal t-horizon value function) is piecewise linear and convex, and can be represented by a *finite* set of real vectors A_t as:

$$V_t(b) = \max_{\alpha \in A_t} \alpha \cdot b \tag{10}$$

The optimal *infinite horizon* value function, $V^* = \lim_{t \to \infty} V_t$, can be approximated arbitrarily close by taking t sufficiently large. The Witness algorithm [6] approximates V^* in this way and is based on an efficient method for finding the set of vectors A_t in terms of the set of vectors A_{t-1}.

POMDP algorithms based on Sondik's representation have been successfully used to solve small problems but have difficulties for scaling up. This is because the vector sets A_t can grow exponentially in size [7]. These difficulties have led to a number of fast approximation methods that often perform well when the degree of uncertainty is low but tend to do poorly in other cases. One such method, is the Q_{MDP} method [7] that is based on the *simplifying assumption* that all future states will be completely observable. In such case, the value $Q(a, b)$ of an action a given a belief state b can be computed from

$$Q(a, b) = \sum_{s \in S} b(s)\, Q^*_{\text{MDP}}(s, a) \tag{11}$$

where $Q^*_{\text{MDP}}(s, a)$ is the optimal value of action a in s in the underlying MDP. Since the simplifying assumption is usually false, the $Q(a, b)$ values are just *heuristic estimates* of the optimal $Q^*(b, a)$ values. Moreover, the estimates are *admissible* as they represent the optimal cost of a simpler problem in which the agent has more information that it actually has. Indeed, the Q_{MDP} method is a *real-time search algorithm* (Fig. 1) operating in belief space with an heuristic function h_{MDP}

$$h_{\text{MDP}}(b) \stackrel{\text{def}}{=} \sum_{s \in S} b(s)\, V^*_{\text{MDP}}(s) \tag{12}$$

where $V^*_{\text{MDP}}(s) = \min_a Q^*_{\text{MDP}}(a, s)$. Given this correspondence, the limitations of Q_{MDP} [7] — loops and poor performance in large problems — are not surprising. Following Korf's [10], these problems can be eliminated by converting the Q_{MDP} algorithm into a *learning* algorithm in the style of LRTA* (Figure 2). This only requires cost estimates $V(b)$ initialized to $h_{\text{MDP}}(b)$ and updated to

$$V(b) := c(a, b) + V(b_a) \tag{13}$$

every time action a is performed in b. The resulting LRTA* algorithm is as greedy as the Q_{MDP} method but avoids cycles and improves with time.

1. **Evaluate** each action a applicable in b as
$$Q(a,b) = c(a,b) + \sum_{o \in O} b_a(o) V(b_a^o)$$
initializing $V(b_a^o)$ to $h(b_a^o)$ when b_a^o not in table
2. **Apply** action $\mathbf{a}$ that minimizes $Q(a,b)$ breaking ties randomly
3. **Update** $V(b)$ to $Q(\mathbf{a}, b)$
4. **Observe** o
5. **Compute** b_a^o
6. **Exit** if b_a^o is a goal (belief) state, else set b to b_a^o and go to 1

Figure 4: RTDP-BEL: the RTDP algorithm for POMDPs

Still neither the $Q_{\rm MDP}$ method nor its LRTA* version deal with *observations.* In the presence of observations the result of an action a is no longer a unique belief state b_a but a collection of belief states b_a^o each with probability $b_a(o)$. As a result, the update captured by expression (13) needs to be modified into:
$$V(b) := c(a,b) + \sum_{o \in O} b_a(o) V(b_a^o) \tag{14}$$

6 Solving POMDPs by RTDP

The algorithm that results from extending the $Q_{\rm MDP}$ method with observations and updates is the RTDP algorithm shown in Fig. 4 that we call RTDP-BEL.

The main computational effort in the inner loop of RTDP-BEL lies in the computation of $Q(a,b)$ in Step 1 whose complexity is $|S|^2 \times |O|$ in the worst case (computing $b_a(o)$ and b_a^o from Equations 6 and 7 is $|S|^2$). The duration of each trial and the number of trials needed until the performance of RTDP-BEL stabilizes depends on the quality of the heuristic function $h_{\rm MDP}$ for the POMDP at hand. In most of the examples reported in the literature [7] (see below), the $h_{\rm MDP}$ heuristic is good and delivers good solutions after few trials. The same is not true for non-informative heuristics such as $h = 0$ (see Section 7.1) or for information-gathering problems (see Section 7.4). Yet before analyzing the performance of RTDP-BEL, we introduce a simple approximation scheme that is needed for handling large and noisy POMDPs.

6.1 Belief Discretization

The approximation scheme is a standard state aggregation technique that reduces space requirements and allows some form of generalization by representing the value of many states by a single entry in the hash table. Basically, given an integer resolution parameter $r > 0$, the probabilities $b(s)$ are *discretized* into r discrete levels before accessing the table. More precisely, for reading and writing the table, and only for this, the vector b of probabilities $b(s)$ for all s is replaced by the vector of discretized probabilities $b_r(s)$, where $b_r(s)$ is the element in the set 0, $1/r$, $2/r$, $\ldots$, 1 closest to $b(s)$. This is computed by setting $b_r(s)$ to $round(p * r)/r$. The result of this approximation is that a single value $V(b_r)$ approximates the infinite values $V(b)$ of all belief states that discretize into b_r. The consequence is that the hash table grows smaller, the values in the table are updated more often, and RTDP-BEL learns faster. Best results have been obtained for values of r in the interval $[10, 100]$. Higher values often generate too many entries in the table, while lower values often collapse the values of belief states that should be treated differently. It should be noted that while the number of discretized belief states b_r is finite it is still quite large.

6.2 Implementation

We mention two points about the implementation of RTDP-BEL that are important. First, we use a cutoff parameter MAX, set to 250, such that any trial with more than MAX steps is terminated. Such cutoff parameter is not strictly needed in finite state stochastic-shortest path MDPs as in such problems RTDP is bound to reach the goal in every trial after a finite number of steps [1, 2]. Yet even in such cases, and with more reason in belief MDPs, the cutoff parameter is useful as long searches for the goal usually produce updates in areas of the state space that are not relevant. In addition, we use an sparse representation for the belief states b so that the complexity of computing the next belief state b_a^o from b grows with $|S|^2$ only in states of complete uncertainty but is significantly faster in other cases.

7 Experimental Results

7.1 Small Problems

For the first set of experiments with RTDP-BEL we considered the three problems from [11, 16, 13] called 'Cheese', 4x3 and 4x4. These are all small navigation problems involving in the order of $11 - 16$ states, $2 - 7$ observations, and 4 actions, formulated here as goal POMDPs with positive action costs. For reasons of space we omit the details about the formulations (which follow the above references) and the resulting performance curves, and focus on the most relevant results.

The 'Cheese' and '4x4' problems can both be solved optimally with value iteration as the set of belief states b that are reachable from the initial belief state is finite and small (this is due to the absence of noise in the actions and observations). The *average costs to the goal*[2] for the resulting optimal policies are 4.813 and 4.154 respectively, which are quite close to the measures 4.799 and 4.150 obtained by RTDP-BEL for the resolution values considered in the interval $10 \ldots 100$. The curves showing the average cost to the goal as a function of the number of trials is almost flat meaning that the behavior of RTDP-BEL is nearly optimal from the the first trial. The average cost to the goal obtained for the '4x3' problem was 5.9 but this result cannot be validated by value iteration as the set of reachable belief states for this problem is infinite. In all cases the *average time* per trial (i.e., time to reach the goal or the cutoff) is in the order of a few thousandths of second, [3] and the size of the hash table is below 69, 97, and 112 respectively for the resolution values $r = 20$, $r = 100$ and $r = \infty$.

The comparison with other POMDP methods over these problems is subtle because they use a different cost/reward structure. In [7] the optimal *average reward per step* reported for the 'Cheese' and '4x3' problems is 0.186 and 0.192 respectively. Due to the simple structure of these problems, the average reward per step should correspond to $1/(1 + \text{ACG})$, where ACG is the average cost to the goal. Plugging the values 4.799 and 4.150 into this equation, we obtain that the average reward per step obtained by RTDP-BEL for these two problems would be 0.172 and 0.194 respectively. While the second value is

[2]The average cost to the goal in all these problems is the average number of steps to either reach the goal or exceed the cutoff.

[3]The experiments were run on a Pentium Pro running at 200 Mhz. The code is written in C.

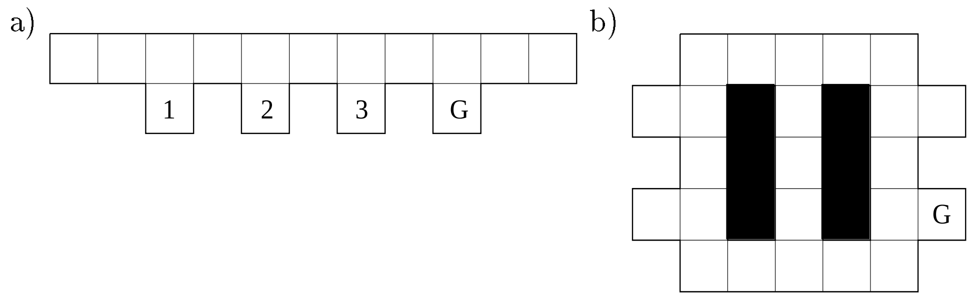

Figure 5: The grids corresponding to the problems Brown57 and Brown89

almost identical to the one reported in [7], the first one is not. We haven't been able to figure out the reason for this difference.

We also used these problems to evaluate the importance of the heuristic function h_{MDP} used by RTDP-BEL by setting it to 0. In such case, RTDP-BEL eventually produces the same behavior but the convergence takes longer and the size of the hash table grows larger (many more belief states are visited). For the larger problems considered below, the size of the table grows so large that the use of heuristic $h = 0$ is not even feasible.

7.2 Medium Sized Problems

We consider now the two medium sized navigation problems depicted Fig. 5 and taken from [7], where a robot starting at a random non-goal location (with a uniform belief state over all such locations) has to find its way to the goal (marked as G). The first problem involves 57 states (14 rooms with 4 possible orientations plus a goal), 21 observations (all combinations of walls in each of the four directions, a 'star' and three visible landmarks when the robot faces south in three particular locations), and 5 actions (stay in place, more forward, turn right, turn left, turn around), and both observations and actions are noisy (accuracies in the order of 70%).

In this problem, RTDP-BEL yields an average cost to the goal of 15.94 for the two resolutions $r = 20$ and $r = 100$ after a few trials, and even the very first trial yields good results; namely 16.59 and 17.10 respectively (Figure 6). The resolution $r = 5$ is too coarse for this problem, producing an inferior ACG value close to 19.84.

RTDP-BEL solves this problem in real-time with an average time per move that is less than 0.025 seconds. The average time for the first trial is nearly

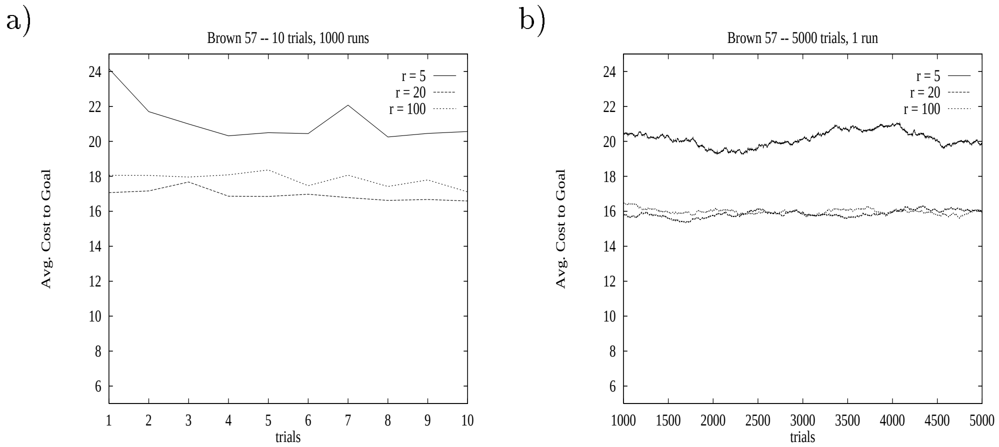

Figure 6: Average cost to goal in Brown57. (a) Short term behavior: Average over 1000 runs shown. (b) Long term behavior: Average over last 1000 trials of single run shown

0.56 seconds which decreases slightly for successive trials. The average size of the hash table after a single run of 5000 trials is 1434, 8399 and 19123 for the resolutions $r = 5$, $r = 20$ and $r = 100$ respectively.

The comparison with the results reported in the literature is not simple at they are focused on the *median* number of steps to the goal rather than the *average*. The *median* has a tendency to ignore the size of 'peaks' and therefore tends to be below the average in these type of problems. The best algorithm in [7] is reported a median of 14 steps with a *success rate* of 99.2%.[4] The median of RTDP-BEL for this problem is 18, 14 and 14 for the resolutions $r = 5$, $r = 20$ and $r = 100$ with a 100% success rate in all cases, from the first trial.

The second problem, Brown89, is larger involving 89 states (4 orientations in 22 rooms plus a goal), 17 observations (all combinations of walls plus 'star') and the same five actions (Fig. 5.b). Convergence for this problem takes longer (Fig. 7), and indeed after 10000 trials and approximately 19105 seconds for $r = 20$ (16926 seconds for $r = 5$) is not entirely clear RTDP-BEL has converged. At that point the average cost to the goal is 32.360 (34.214 for $r = 5$); roughly a 52% improvement over the ACG that results after the

[4]The success rate is the percentage of runs that the algorithm reaches the goal state in less than 251 steps.

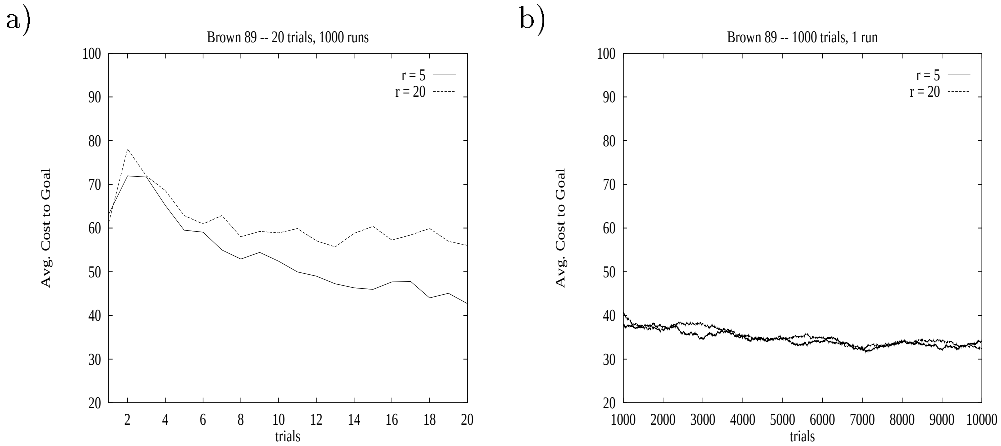

Figure 7: Average cost per trial in Brown89. (a) Short term behavior: Average over 1000 runs shown. (b) Long term behavior: Average over last 1000 trials of single run shown

first trial: 61.157 for $r = 20$ (63.146 for $r = 5$). The average time per move in the first trial is 0.057 seconds (0.052 seconds for $r = 5$), while the average time to reach the goal is 3.673 seconds (3.293 seconds for $r = 5$).

In large, noisy problems such as this, RTDP-BEL behaves like an 'anytime algorithm' [8]: more deliberation time usually means a better execution. Yet it's worth noting that even after only 20 trials, that take less than a minute on average, RTDP-BEL yields an average cost of 42.7 (for $r = 5$), which is a 32% improvement over the average cost in the first trial, and 30% off from the cost after 10000 trials.

For this problem, the *median* number of steps for the best heuristic algorithm reported in [7] is 33 with a 83.7% success rate, where the median corresponding to a 'human expert' is reported as 29 with a 100% success rate. The median for RTDP-BEL was found to be 30 for both $r = 5$ and $r = 20$, and in both cases the success rate was 100% from the first trial. The size of the table after a single run of 10000 trials was found to be 5220 and 52090 respectively. Interestingly, [5] recently reports a median of 24 steps for this problem with a success rate of 98%.

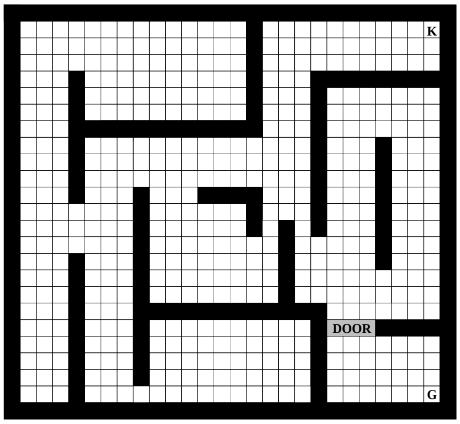

Figure 8: Grid with 989 states from Wiering and Schmidhuber

7.3 A Large Problem

The largest problem considered is the maze shown in Fig. 8 from [20]. The problem involves navigating a 23x26 grid to reach a goal destination marked by the letter G in the figure. As in the problems above, the agent is initially at a random non-goal location with uniform probability. In order to reach the goal the agent must get through a door, and for that he needs the key which is initially located at the position marked K. The total number of reachable states in the problem is 989, corresponding to the free positions in the grid, the boolean feature corresponding to whether the robot is carrying the key or not, and the three door cells (which can be occupied only if the agent is carrying the key). There are four actions corresponding to the four directions and they all have deterministic effects. Observations are also noiseless and reveal the occupancy of the nearby cells. The agent does not know its position nor whether it's carrying a key.

The average cost to the goal obtained in the *first* trial for this problem was 68.44 with a 100% success rate (average taken over 1000 runs). The first trial takes 4.8 seconds on average. The resolution parameter used was $r = 20$ and the discount factor $\gamma = 0.95$. The average cost to the goal does not improve much after thousands of trials, and the average value after $10,000$ trials (taken over the preceding 2000 trials) was around 67.

7.4 Information Gathering Problem

The last problem is a navigation problem over the grid showed in Figure 9(a) suggested by Sebastian Thrun. The agent starts in position 6 and has to reach the *goal* which may be either at position 0 or 4. Moreover, one of these two positions — the one that is not the goal — is a high penalty state with cost 50. Finally, at position 9 there is a 'map' that reports the true position of the goal (i.e., either 0 or 4) with probability p and the other possible position with probability $1-p$. The current position of the robot is always observable.

The resulting curves for different values for p are shown in Figs. 9(b). The optimal solution is to go to position 9, inspect the map for a number of intervals that depend on p, and head up for the position believed to be the goal. This is what the policies obtained by RTDP-BEL do after a few trials. When p is one, completely accurate information about the goal is obtained from the first reading of the 'map', and then the agent heads up for the goal. When p is lower, e.g., $p = 0.75$, more observations about the map are needed before heading for the goal and learning that takes more trials. Note that this model presumes that the noisy observations about the map are independent of each other given the state of the agent. This assumption can be relaxed if so desired by including an additional variable in the model (e.g., 'sensor status').

It is interesting that the heuristic obtained from the underlying MDP is very misleading in this problem as it initially leads the agent in the wrong direction. Then when the agent gets sufficiently close to the possible locations of the goal he 'notices' the high-risk involved in getting closer and backs up. After some *blind* exploration it finds the map and heads back to the goal. After a few trials the algorithm learns to go directly to the map and remain there until it's sufficiently certain of the location of the goal.

It's worth emphasizing that the search for the map is *blind,* namely unlike the search for the goal it is not guided by any heuristic. This means that in problems involving large state spaces in which the agent has to get information before heading for the goal, the first trials of RTDP-BEL may involve large numbers of steps, and hence the convergence of RTDP-BEL will be slow. For such type of problems, heuristic functions that take the value of information into account would be needed.

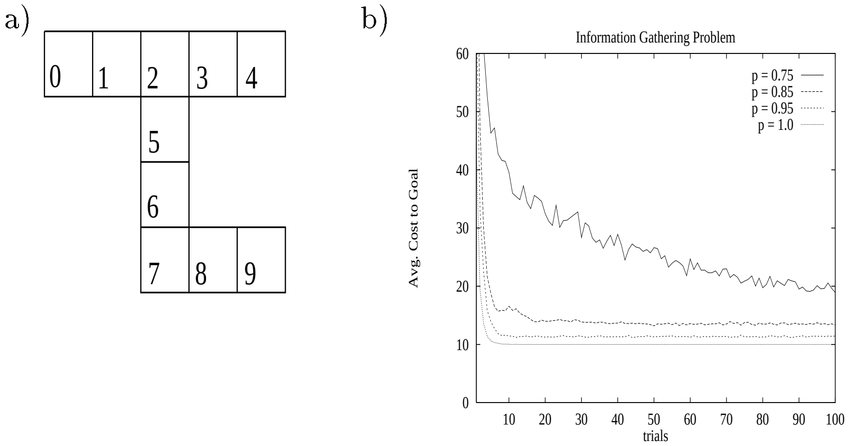

Figure 9: Information gathering problem: average cost over 1000 runs shown.

8 Conclusions

Real-time dynamic programming appears to provide a principled and practical way for solving large POMDPs. The resulting solutions are not necessarily optimal due to two reasons: first, it's not possible to *prove* that RTDP-BEL has converged by just looking at the average cost curves or estimate values; and second, limited resolution values r may introduce errors. In practice, this limitation does not appear to be problematic, and recently we have obtained positive results for a number of high-level planning problems that were translated into POMDPs and solved by RTDP-BEL. Details can be found in [9, 3]. RTDP-BEL is related to methods such as [13] and [5] that also provide approximate solutions of the information MDP, even if the nature of the approximations are different in the three cases.

As we mentioned in Section 7.4 we expect RTDP-BEL *not* to do well in tasks involving *very large spaces* in which the *search for information* must precede the search for the goal. In such cases, the heuristic used to initialize RTDP-BEL may provide no guidance and RTDP-BEL may end up visiting too many states. We are currently exploring the use of POMDPs and the RTDP-BEL algorithm for two problems of this type. One is sorting a vector of numbers; the other is inferring decision trees from data. We have formulated both problems as POMDPs and have obtained results using different heuristic functions and different representations of the belief states. We expect to report these results elsewhere.

9 Acknowledgements

We thank Tony Cassandra for making available the POMDP problems and the parser, and Cassandra, Kaebling and Littman for introducing POMDPs in AI.

References

[1] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[2] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[3] B. Bonet and H. Geffner. Planning and control with incomplete information using POMDPs: Experimental results. Submitted, 1998.

[4] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97.* MIT Press, 1997.

[5] R. Brafman. A heuristic variable grid solution for POMDP's. In *Proceedings AAAI-97*, pages 727–733, 1997.

[6] A. Cassandra, L. Kaebling, and M. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings AAAI94*, pages 1023–1028, 1994.

[7] A. Cassandra, L. Kaebling, and M. Littman. Learning policies for partially observable environments: Scaling up. In *Proc. of the 12th Int. Conf. on Machine Learning*, 1995.

[8] T. Dean and M. Boddy. An analysis of time dependent planning. In *Proceedings AAAI-88*, pages 49–54, 1988.

[9] H. Geffner and B. Bonet. High-level plannnig and control with incomplete information using POMDP's. Submitted, 1998.

[10] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

[11] A. McCallum. Overcoming incomplete perception with utile distinction memory. In *Proceedings Tenth Int. Conf. on Machine Learning*, pages 190–196, 1993.

[12] G. Monahan. A survey of partially observable markov decision processes: Theory, models and algorithms. *Management Science*, 28(1):1–16, 1983.

[13] R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings IJCAI-95*, 1995.

[14] J. Pearl. *Heuristics*. Morgan Kaufmann, 1983.

[15] M. Puterman. *Markov Decision Processes – Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., 1994.

[16] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1994.

[17] E. Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford University, 1971.

[18] R. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

[19] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.

[20] M. Wiering and J. Schmidhuber. Hq-learning: Discovering markovian subgoals for non-markovian reinforcement learning. Technical Report IDSIA-95-96, IDSIA, Switzerland, 1996. http://www.idsia.ch.