Bounded Real-Time Dynamic Programming: RTDP with monotone upper bounds and performance guarantees

H. Brendan McMahan MCMAHAN@CS.CMU.EDU
Maxim Likhachev MAXIM+@CS.CMU.EDU
Geoffrey J. Gordon GGORDON@CS.CMU.EDU
Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh PA 15213 USA

Abstract

MDPs are an attractive formalization for planning, but realistic problems often have intractably large state spaces. When we only need a partial policy to get from a fixed start state to a goal, restricting computation to states relevant to this task can make much larger problems tractable. We introduce a new algorithm, Bounded RTDP, which can produce partial policies with strong performance guarantees while only touching a fraction of the state space, even on problems where other algorithms would have to visit the full state space. To do so, Bounded RTDP maintains both upper and lower bounds on the optimal value function. The performance of Bounded RTDP is greatly aided by the introduction of a new technique to efficiently find suitable upper bounds; this technique can also be used to provide informed initialization to a wide range of other planning algorithms.

1. Introduction

In this paper we consider the problem of finding a policy in a Markov decision process with a fixed start state[1] $\mathtt{s}$, a fixed zero-cost absorbing goal state $\mathtt{g}$, and non-negative costs. Such problems are also commonly known as stochastic shortest path problems (Bertsekas & Tsitsiklis, 1996). Perhaps the simplest algorithm for this problem is value iteration, which solves for an optimal policy on the full state space. In many realistic problems, however, only a small fraction of the the state space is relevant to the problem of reaching $\mathtt{g}$ from $\mathtt{s}$. This fact has inspired the development of a number of algorithms that focus computation on states that seem to be most relevant to finding an optimal policy from $\mathtt{s}$. Such algorithms include Real-Time Dynamic Programming (RTDP) (Barto et al., 1995), Labeled-RTDP (LRTDP) (Bonet & Geffner, 2003b), LAO* (Hansen & Zilberstein, 2001), Heuristic Search/DP (HDP) (Bonet & Geffner, 2003a), Envelope Propagation (EP) (Dean et al., 1995), and Focused Dynamic Programming (FP) (Ferguson & Stentz, 2004).

Many of these algorithms use heuristics (lower bounds on the optimal value function) and/or sampled greedy trajectories to focus computation. In this paper, we introduce Bounded RTDP, or BRTDP, which is based on RTDP and uses both a lower bound and sampled trajectories. Unlike RTDP, however, it also maintains an upper bound on the optimal value function, which allows it to focus on states that are both relevant (frequently reached under the current policy) and poorly understood (large gap between upper and lower bound). Further, acting greedily with respect to an appropriate upper bound allows BRTDP to make anytime performance guarantees.

Finding an appropriate upper bound to initialize BRTDP can greatly impact its performance. One of the contributions of this paper is an efficient algorithm for finding such an upper bound. Nevertheless, our experiments show that BRTDP performs well even when initialized naively.

We evaluate BRTDP on two criteria: *off-line convergence*, the time required to find an approximately optimal partial policy before any actions are taken in

[1] An arbitrary start-state distribution can be accomodated by adding an imaginary start state with a single action which produces the desired state distribution; the algorithms considered in this paper become less effective, however, as the start-state distribution becomes denser.

Appearing in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005. Copyright 2005 by the author(s)/owner(s).

the real world; and *anytime performance*, the ability to produce a reasonable partial policy at any time after computation is started. Our experiments show that when run off-line, BRTDP often converges much more quickly than LRTDP and HDP, both of which are known to have good off-line convergence properties. And, when used as an anytime algorithm, a suitably-initialized BRTDP consistently outperforms a similarly initialized RTDP (which is known to have good anytime properties). Furthermore, given reasonable initialization assumptions, BRTDP will always return a policy with a provable performance bound; we know of no other MDP algorithms with this property.

In fact, the gap in offline performance between BRTDP and competing algorithms can be arbitrarily large because of differences in how they check convergence. HDP, LRTDP, and LAO* (and most other algorithms of which we are aware[2]) have convergence guarantees based on achieving small Bellman residual on *all states* reachable under the current policy, while BRTDP only requires a small residual on states reachable *with significant probability.* Let $f_\pi(y)$ be the expected number of visits to state y given that the agent starts at s and executes policy π. We say an MDP has *dense noise* if all policies have many nonzero entries in f_π. For example, planning problems with action errors have $f_\pi > 0$ for all reachable states. (Action errors mean that, with some small probability, we take a random action rather than the desired one.) Dense noise is fairly common, particularly in domains from robotics. For example, Gaussian errors in movement will make every state have positive probability of being visited. Gaussian motion-error models are widespread, e.g. (Ng et al., 2004). Unpredictable motion of another agent can also cause large numbers of states to have positive visitation probability; an example of this sort of model is described in (Roy et al., 2004).

For HDP or LRTDP to converge on problems with dense noise, they must do work that is at least linear in the number of nonzero entries in f_π, even if most of those entries are almost zero. With an appropriate initialization, BRTDP's bounds allow it to make performance guarantees on MDPs with dense noise without touching all reachable states, potentially making it arbitrarily faster than HDP, LRTDP, and LAO*.

[2]After the submission of our paper, it was pointed out that our exploration strategy is similar to that of the HSVI algorithm (Smith & Simmons, 2004); since HSVI is designed for POMDPs rather than MDPs, the forms of the bounds that it maintains are different from ours, and its backup operations are much more expensive.

2. Basic Results

We represent a stochastic shortest path problem with a fixed start state as a tuple $\mathcal{M} = (S, A, P, c, \mathsf{s}, \mathsf{g})$, where S is a finite set of states, $\mathsf{s} \in S$ is the start state, $\mathsf{g} \in S$ is the goal state, A is a finite action set, $c : S \times A \to \mathbb{R}_+$ is a cost function, and P gives the dynamics; we write P^a_{xy} for the probability of reaching state y when executing action a from state x. Since g is a zero-cost absorbing state we have $c(\mathsf{g}, \mathsf{g}) = 0$ and $P^a_{\mathsf{g},\mathsf{g}} = 1$ for all actions a. If $v \in \mathbb{R}^{|S|}$ is an arbitrary assignment of values to states, we define state-action values with respect to v by

$$Q_v(x, a) = c(x, a) + \sum_{y \in S} P^a_{xy} v(y).$$

A stationary policy is a function $\pi : S \to A$. A policy is proper if an agent following it from any state will eventually reach the goal with probability 1. We make the standard assumption that at least one proper policy exists for $\mathcal{M}$, and that all improper policies have infinite expected total cost at some state (Bertsekas & Tsitsiklis, 1996). For a proper policy π, we define the value function of π as the solution to the set of linear equations $v_\pi(x) = c(x, \pi(x)) + \sum_{y \in S} P^{\pi(x)}_{xy} v_\pi(y)$. It is well-known that there exists an optimal value function $v^\star$, and it satisfies the Bellman equations:

$$v^\star(x) = \min_{a \in A} Q_{v^\star}(x, a), \quad v^\star(\mathsf{g}) = 0.$$

For an arbitrary v, we define the (signed) Bellman error of v at x by $\mathrm{be}_v(x) = v(x) - \min_{a \in A} Q_v(x, a)$. The greedy policy with respect to v, greedy(v), is defined by $\mathrm{greedy}(v)(x) = \mathrm{argmin}_{a \in A} Q_v(x, a)$.

We are particularly interested in monotone value functions: we refer to v as monotone optimistic (a monotone lower bound) if $\forall x,\ \mathrm{be}_v(x) \leq 0$. We call it monotone pessimistic (a monotone upper bound) if $\forall x,\ \mathrm{be}_v(x) \geq 0$. We use the following two theorems, which can be proved using techniques from, *e.g.*, (Bertsekas & Tsitsiklis, 1996, Sec. 2.2).

Theorem 1 *If v is monotone pessimistic, then v is an upper bound on $v^\star$. Similarly, if v is monotone optimistic, then v is a lower bound on $v^\star$.*

Theorem 2 *Suppose v_u is a monotone upper bound on $v^\star$. If π is the greedy policy with respect to v_u, then for all x, $v_\pi(x) \leq v_u(x)$.*

No analog to Theorem 2 exists for a greedy policy based on a lower bound v_ℓ, monotone or otherwise: such a policy may be arbitrarily bad. As an example, consider the values v_d found by solving $\mathcal{M}_d$, the

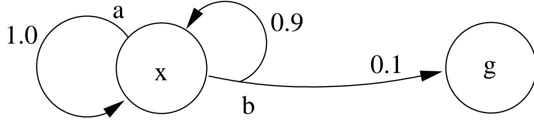

Figure 1. An MDP where the greedy policy with respect to v_d, the values from the deterministic relaxation, is improper. Costs are $c(x,a) = 1$ and $c(x,b) = 10$.

deterministic relaxation of $\mathcal{M}$. (In $\mathcal{M}_d$, the agent gets to choose any outcome of any action from each state, rather than choosing an action and then facing a stochastic outcome. $\mathcal{M}_d$ is important because RTDP, LRTDP, HDP, and LAO* are often initialized to v_d.) It is easy to show that v_d is a monotone lower bound on $v^\star$. Further, $\mathcal{M}_d$ is deterministic, so we can solve it by A^* or Dijkstra's algorithm. However, greedy(v_d) need not even be proper. Consider the MDP in Figure 1: $v_d(x) = 10$, so $Q_{v_d}(x,a) = 11$ and $Q_{v_d}(x,b) = 19$, and the greedy policy for v_d always selects action a. Since RTDP, LRTDP, HDP, and LAO* select actions greedily, this example shows that these algorithms may initially produce arbitrarily bad policies.[3]

3. Efficient Monotone Bounds

Our planning algorithm, BRTDP, is described below in Section 4. It can be initialized with any upper and lower bounds v_u and v_ℓ on $v^\star$, and provides performance guarantees if v_u and v_ℓ are monotone. So, we need to compute monotone bounds v_u and v_ℓ efficiently. This section describes how to do so assuming we can afford to visit every state a small number of times; Section 5 describes looser bounds which don't require visiting all of S. As noted above, we can initialize v_ℓ to the value of the deterministic relaxation $\mathcal{M}_d$; so, the remainder of this section deals with v_u.

For any proper policy π, the value function v_π is a monotone upper bound. A proper policy can be found reasonably quickly, for example by computing π_p from the deterministic relaxation. Unfortunately, directly solving the linear system to evaluate π requires about $\mathcal{O}(|S|^3)$ time (worst-case). This is the fastest technique we are aware of in the literature. We introduce a new algorithm, called Dijkstra Sweep for Monotone Pessimistic Initialization (DS-MPI), which can compute a monotone upper bound in $\mathcal{O}(|S|\log|S|)$ time.

Suppose we are given a policy π along with $p_g, w \in \mathbb{R}_+^{|S|}$ that satisfy the following property: if we execute π from x until some fixed but arbitrary condition[4] is met, then $w(x)$ is an upper bound on the expected cost of the execution from x until execution is stopped, and $p_g(x)$ is a lower bound on the probability the current state is the goal when execution is stopped. If $p_g(x) > 0$ and $w(x)$ is finite for all x (and if the other conditions of the theorem below are satisfied), then we can use p_g and w to construct a monotone upper bound. We first prove that we can do so, then show an algorithm for constructing such a p_g and w.

Theorem 3 *Suppose p_g and w satisfy the conditions given above for some policy π. Further, suppose for all x, there exists an action a such that either (I) $p_g(x) < \sum_{y\in S} P_{xy}^a p_g(y)$ or (II) $w(x) \geq c(x,a) + \sum_{y\in S} P_{xy}^a w(y)$ and $p_g(x) = \sum_{y\in S} P_{xy}^a p_g(y)$. Define $\lambda(x,a)$ by*

$$\lambda(x,a) = \frac{c(x,a) + \sum_{y\in S} P_{xy}^a w(y) - w(x)}{\sum_{y\in S} P_{xy}^a p_g(y) - p_g(x)}$$

when case (I) applies, let $\lambda(x,a) = 0$ when case (II) applies, and let $\lambda(x,a) = \infty$ otherwise. Then, if we choose $\lambda \geq \max_{x\in S}\min_{a\in A}\lambda(x,a)$, the value function $v_u(x) = w(x) + (1 - p_g(x))\lambda$ is a finite monotone upper bound on $v^\star$.

Proof: It is sufficient to show that all Bellman errors for v_u are positive, that is for all x,

$$v_u(x) - \min_{a\in A}\left[c(x,a) + \sum_{y\in S} P_{xy}^a v_u(y)\right] \geq 0.$$

Plugging in the definition of v_u from above gives

$$w(x) + (1 - p_g(x))\lambda \geq$$
$$\min_{a\in A}\left[c(x,a) + \sum_{y\in S} P_{xy}^a (w(y) + (1 - p_g(y))\lambda)\right]. \quad (1)$$

We need to show that this holds for all x given λ as defined by the Theorem. Consider an a for which one of the conditions (I) or (II) on w and p_g holds. (This need not be the minimizing action in (1).) In case (I) we can solve (1) for λ and arrive at the constraint $\lambda \geq \lambda(x,a)$. In case (II), any λ will satisfy (1), so we pick the constraint $\lambda \geq 0 = \lambda(x,a)$. Since we only need (1) to hold for a single action in state x,

[3] We can, however, always extract a proper policy π_p from v_d. In order for x to get a value $v_d(x)$ there must exist $a \in A$ and $y \in S$ satisfying $P_{xy}^a > 0$, $v_d(x) = c(x,a) + v_d(y)$. We can set $\pi_p(x)$ equal to any such action a; it is natural to pick the a which makes P_{xy}^a as large as possible.

[4] For example, we might execute π for t steps, or execute π until we reach a state in some subset of S. Formally, π can be an arbitrary (history-dependent) policy, and the stopping condition can be an arbitrary function from histories to $\{\textbf{stop}, \textbf{don't stop}\}$ which ensures that all trajectories eventually stop.

Initialization:
$\forall(x,a),\ \hat{p}_g(x,a) \leftarrow 0;\quad \forall a,\ \hat{p}_g(\mathsf{g},a) \leftarrow 1$
$\hat{w}(x,a) \leftarrow c(x,a)$
p_g, w initialized arbitrarily
$\forall x,\ \ \pi(x) \leftarrow$ undefined; $\pi(\mathsf{g}) \leftarrow$ (arbitrary action)
$\forall x,\ \ \mathrm{pri}(x) = \infty$, $\mathrm{fin}(x) =$ false

Sweep:
queue.enqueue(goal, 0)
while not queue.empty() **do**
  $x \leftarrow$ queue.pop()
  $\mathrm{fin}(x) =$ true
  $w(x) \leftarrow \hat{w}(x, \pi(x))$
  $p_g(x) \leftarrow p_g(x, \pi(x))$
  for all (y, a) s.t.$(P^a_{yx} > 0)$ and (not $\mathrm{fin}(y)$) **do**
    $\hat{w}(y,a) \leftarrow \hat{w}(y,a) + P^a_{yx} w(x)$
    $\hat{p}_g(y,a) \leftarrow \hat{p}_g(y,a) + P^a_{yx} p_g(x)$
    $\mathrm{pri} \leftarrow \langle 1 - \hat{p}_g(y,a), \hat{w}(y,a)\rangle$
    if $\mathrm{pri} < \mathrm{pri}(y)$ **then**
      $\mathrm{pri}(y) \leftarrow \mathrm{pri}$
      $\pi(y) \leftarrow a$
      queue.decreaseKey$(y, \mathrm{pri}(y))$

Algorithm 1: DS-MPI

choosing the action with the loosest constraint yields $\lambda \geq \min_{a \in A} \lambda(x,a)$; intersecting these constraints for all states gives the λ defined by the Theorem. Since we assumed that every state has an action that satisfies either (I) or (II), this λ is finite and so v_u is finite. □

Now we will show how to construct the necessary w, p_g, and corresponding π. The idea is simple: suppose state x_1 has an action a such that $P^a_{x_1\mathsf{g}} > 0$. Then we can set $w(x_1) = c(x_1, a)$ and $p_g(x_1) = P^a_{x_1\mathsf{g}}$. Now, pick some x_2 and a_2 such that $P^{a_2}_{x_2\mathsf{g}} + P^{a_2}_{x_2x_1} > 0$. We can set $p_g(x_2)$ equal to $P^{a_2}_{x_2\mathsf{g}} + P^{a_2}_{x_2x_1} p_g(x_1)$ and $w(x_2) = c(x_2, a_2) + P^{a_2}_{x_2\mathsf{g}} 0 + P^{a_2}_{x_2x_1} w(x_1)$. We can now select x_3 to be any state with an action that has positive probability of reaching g, x_1, or x_2, and we will be able to assign it a positive p_g. The policy π corresponding to p_g and w is given by $\pi(x_i) = a_i$, and the stopping condition ends a trajectory whenever a transition from x_i to x_j occurs with $j \geq i$. The p_g and w values we compute are exact values, not bounds, for this policy and stopping condition.

To complete the algorithm, it remains to give a method for determining what state to select next when there are multiple possible states. We propose the greedy maximization of $p_g(x_k)$: having fixed $x_1, \ldots, x_{k-1}$, select (x_k, a_k) to maximize $\sum_{i<k} P^{a_k}_{x_kx_i} p_g(x_i)$. If there are multiple states that achieve the same $p_g(x_k)$, we choose the one that minimizes $\sum_{i<k} P^{a_k}_{x_kx_i} w(x_i)$. Algorithm 1 gives the pseudocode for calculating p_g and w; the queue is a min priority queue (with priorities in $\mathbb{R}^2$ which are compared according to lexical order), and $\hat{p}_g$ and $\hat{w}$ are analogous to the Q values for v. After applying the sweep procedure, one can apply Theorem 3 to construct v_u.

In fact, condition (I) or (II) will always hold for action a_k, and so it is sufficient to set $\lambda = \max_{x_i \in S} \lambda(x_i, a_i)$. To see this, consider the (x_k, a_k) selected by DS-MPI after $x_1, \ldots, x_{k-1}$ have already been popped (i.e., $\mathrm{fin}(x_i) = $ true, $i < k$). Then $p_g(x_k) = \sum_{i<k} P^{a_k}_{x_kx_i} p_g(x_i)$. At convergence all states x have $p_g(x) > 0$, and so the only way $p_g(x_k)$ can equal $\sum_{y \in S} P^{a_k}_{x_ky} p_g(y)$ is if all outcomes of (x_k, a_k) were already finished when $p_g(x_k)$ was determined. This implies that $\sum_{i<k} P^{a_k}_{x_kx_i} w(x_i) = \sum_{y \in S} P^{a_k}_{x_ky} w(y)$, and so $w(y) = c(x,a) + \sum_{y \in S} P^{x_k}_{x_ky} w(y)$ and condition (II) holds. Otherwise, condition (I) must hold for (x_k, a_k). Additional backups of w and p_g will preserve these properties, so if extra computation time is available, additional sweeps will tighten our upper bound.

If the dynamics are deterministic, we can always pick (x_k, a_k) so $p_g(x_k) = 1$, and so our scheduling corresponds to Dijkstra's algorithm. This sweep is similar to the policy improvement sweeps in the Prioritized Policy Iteration (PPI) algorithm described in (McMahan & Gordon, 2005). The primary differences are that the PPI version assumes it is already initialized to some upper bound and performs full Q updates, while this version performs incremental updates.

The running time of DS-MPI is $\mathcal{O}(|S| \log |S|)$ (assuming a constant branching factor) if a standard binary heap is used to implement the queue. However, an unscheduled version of the algorithm will still produce a finite (though possibly much looser) upper bound, so this technique can be run in $\mathcal{O}(|S|)$ time. If no additional information is available, then this performance is the best possible for arbitrary MDPs: in general it is impossible to produce an upper bound on $v^\star$ without doing $\mathcal{O}(|S|)$ work, since we must consider the cost at each reachable state.

4. Bounded RTDP

The pseudocode for Bounded RTDP is given in Algorithm 2. BRTDP has four primary differences from RTDP: first, it maintains upper and lower bounds v_u and v_ℓ on $v^\star$, rather than just a lower bound. When a policy is requested (before or after convergence), we return greedy(v_u); v_ℓ helps guide exploration in simulation. Second, when trajectories are sampled in simulation, the outcome distribution is biased to prefer transitions to states with a large gap

```
Main loop:
  while (v_u(s) − v_ℓ(s)) > α do
    runSampleTrial()

runSampleTrial:
  x ← s
  traj ← (empty stack)
  while true do
    traj.push(x)
    v_u(x) ← min_a Q_{v_u}(x, a)
    a ← argmin_a Q_{v_ℓ}(x, a)
    v_ℓ(x) ← Q_{v_ℓ}(x, a)
    ∀y,  b(y) ← P^a_{xy}(v_u(y) − v_ℓ(y))
    B ← Σ_y b(y)
    if (B < (v_u(s) − v_ℓ(s))/τ) then break
    x ← sample from distribution b(y)/B
  while not traj.empty() do
    x ← traj.pop()
    v_u(x) ← min_a Q_{v_u}(x, a)
    v_ℓ(x) ← min_a Q_{v_ℓ}(x, a)
```

Algorithm 2: Bounded RTDP

$(v_u(x) - v_\ell(x))$. Third, BRTDP maintains a list of the states on the current trajectory, and when the trajectory terminates, it does backups in reverse order along the stored trajectory. Fourth, simulated trajectories terminate when they reach a state that has a "well-known" value, rather than when they reach the goal. We assume that BRTDP is initialized so that v_u is an upper bound and v_ℓ is a lower bound.

Like RTDP, BRTDP performs backups along sampled trajectories that begin from s, picking greedy actions with respect to v_ℓ. Unlike RTDP, it biases action outcomes towards less-well-understood states: it picks an outcome y in proportion to $P^a_{xy}(v_u(y) - v_\ell(y))$.

The value of the goal state is known to be zero, and so we assume $v_u(\mathsf{g}) = v_\ell(\mathsf{g}) = 0$ initially (and hence always). This implies that $b(\mathsf{g}) = 0$, and so our trajectories will never actually reach the goal (or any other state whose value is completely known). So, we end trajectories when we reach a state whose successors, on average, have well-known values; more precisely, we look at the expected gap between v_u and v_ℓ for states reached under the greedy action. The normalizing constant B has exactly this interpretation, so we terminate the trajectory when B is small. We experimented with various definitions of "small," and found that they have relatively minor impacts on performance. The adaptive criterion given in Algorithm 2 was as good as anything; $\tau > 1$ is a constant (we used τ between 10 and 100 in our experiments).

The convergence proof for BRTDP is very different from that for RTDP. Proving convergence of RTDP relies on the assumption that all states reachable under the greedy policy are backed up infinitely often in the limit (Bertsekas & Tsitsiklis, 1996). In the face of dense noise, this implies that convergence will require visiting the full state space. We take convergence to mean $v_u(\mathsf{s}) - v_\ell(\mathsf{s}) \leq \alpha$ for some error tolerance α, and BRTDP can achieve this (given a good initialization) without visiting the whole state space even in the face of dense noise. Space constraints preclude a detailed proof, but the result is based on establishing three claims: (1) v_u and v_ℓ remain upper and lower bounds on $v^\star$, (2) trajectories have finite expected lengths, and (3) every trajectory has a positive probability of increasing v_ℓ or decreasing v_u.

5. Initialization Assumptions and Performance Guarantees

We assume that at the beginning of planning, the algorithm is given $\mathcal{M}$, including s. As mentioned in Section 3, if this is the only information available, then on arbitrary problems it may be necessary to consider the whole state space to prove any performance guarantee.

LRTDP, HDP, and LAO* can converge on some problems without visiting the whole state space. This is possible if there exists an $E \subset S$ such that some approximately optimal policy π has $f_\pi(y) > 0$ only for $y \in E$, and further, a tight lower bound on s can be proved by only considering states inside E and possibly a lower bound provided at initialization. While some realistic problems have this property, many do not, including those with dense noise. The question, then, is what is the minimal amount of additional information that might allow convergence guarantees while only visiting a small fraction of S on arbitrary problems. We propose that the appropriate assumption is that an achievable upper bound (v_u^0, π^0) is known; here v_u^0 is some upper bound (it need not be monotone) on v_{π^0} (and hence $v^\star$), where π^0 is known. Such a pair is almost always available trivially, for example, by letting $v_u^0(x) \leftarrow Z$ where Z is some worst-case cost of system failure, and letting $\pi^0(x)$ be the sit-and-wait-for-help action, or something similar. Even such trivial information may be enough to allow convergence while visiting a small fraction of the state space.

It is easiest to see how to use (v_u^0, π^0) via a transformation. Consider $M' = (S, A \cup \{\phi\}, \tilde{P}, \tilde{c}, \mathsf{s}, \mathsf{g})$, where ϕ is a new action that corresponds to switching to π^0 and following it indefinitely. This action has $\tilde{P}^\phi_{x\mathsf{g}} = 1.0$ and costs $\tilde{c}(x, \phi) = v_u^0(x)$; for all other actions, $\tilde{P} = P$ and $\tilde{c} = c$. We know $v_u^0 \geq v_{\pi^0} \geq v^\star$,

and so adding the action ϕ does not change the optimal values, so solving $\mathcal{M}'$ is equivalent to solving $\mathcal{M}$. Suppose we run BRTDP on $\mathcal{M}'$, but extract the current upper bound v_u^t before convergence; then, v_u^t need not be monotone for $\mathcal{M}$, though it will be for $\mathcal{M}'$. We show how to construct a policy for $\mathcal{M}$ using only v_u^t that achieves the values v_u^t. At a state where $v_u^t(x) \geq \min_{a \in A} Q_{v_u^t}(x, a)$, we play the greedy action, and the performance guarantee follows from the standard monotonicity argument. Suppose, however, we reach a state x where $v_u^t(x) < \min_{a \in A} Q_{v_u^t}(x, a)$. Then, it is not immediately clear how to achieve this value. However, we show that in this case $v_u^t(x) = v_u^0(x)$, and so we can switch to π^0 to achieve the value. Suppose v_u^{t-1} was the value function just before BRTDP backed up x most recently. Then, BRTDP assigned $v_u^t(x) \leftarrow \min_{a \in A \cup \phi} Q_{v_u^{t-1}}(x, a)$. Since v_u^0 is monotone (for $\mathcal{M}'$, on which BRTDP is running), $Q_{v_u^{t-1}}(x, a) \geq Q_{v_u^t}(x, a)$, and so the only way we could have $v_u^t(x) < \min_{a \in A} Q_{v_u^t}(x, a)$ is if the auxiliary action ϕ achieved the minimum, implying $v_u^t(x) = v_u^0(x)$.

Thus, we conclude that via this transformation it is reasonable to assume BRTDP is initialized with monotone upper bound, implying that at any point in time BRTDP can return a stationary policy with provable performance guarantees. This policy will be greedy in $\mathcal{M}'$, but may be non-stationary on $\mathcal{M}$ as it may fall back on π^0. This potential non-stationary behavior is critical to providing a robust suboptimal policy.

6. Experimental Results

We test BRTDP on two discrete domains. The first is the 4-dimensional racetrack domain described in (Barto et al., 1995; Bonet & Geffner, 2003b; Bonet & Geffner, 2003a; Hansen & Zilberstein, 2001). Problems (A) and (B) are from this domain, and use the `large-b` racetrack map (Bonet & Geffner, 2003a). Problem (A) fixes a 0.2 probability of getting the zero acceleration rather than the chosen control, similar to test problems from the above references. Problem (B) uses the same map, but uses a dense noise model where with a 0.01 probability a random acceleration occurs. Problems (C) and (D) are from a 2D gridworld domain, where actions correspond to selecting target cells within a Euclidean distance of two (giving 13 actions). Both instances use the same map. In (C), the agent accidentally targets a state up to a distance 2 from the desired target state, with probability 0.2. In (D), however, a random target state (within distance 2 of the current state) is selected with probability 0.01. Note that problems (A) and (C) have fairly sparse noise, while (B) and (D) have dense noise.

	S	$v_{\pi_d}(\mathsf{s})$	$v_u(\mathsf{s})$	$v_{\pi'}(\mathsf{s})$	$v^\star(\mathsf{s})$	$v_d(\mathsf{s})$
A	21559	29	63	32	23	19
B	21559	1.3e10	26.9	20.1	19.9	19.0
C	6834	15283	1642	485	176	52
D	6834	7662	182.1	117.1	116.7	63.0

Figure 2. Test problem sizes and start-state values.

Figure 2 summarizes the sizes of the test problems, as well as the quality of various initializations. The $v_{\pi_d}(\mathsf{s})$ column gives the value of the start state under policy π_d, the greedy policy for the value function v_d of the problem's deterministic relaxation (see Section 2). The $v_u(\mathsf{s})$ column gives the value computed via DS-MPI. We also give the value of s under $\pi' = \text{greedy}(v_u)$ and under the optimal policy, as well as $v_d(\mathsf{s})$. Figure 2 shows that DS-MPI can produce high-quality upper bounds that have high-quality greedy policies, despite running in only $\mathcal{O}(|S| \log |S|)$ time rather than the $\mathcal{O}(|S|^3)$ needed to compute v_{π_d}.

6.1. Anytime Performance

We compare the anytime performance of BRTDP to RTDP on the test domains listed in Figure 2, considering both *informed* initialization and *uninformed* initialization for both algorithms. Informed initialization means RTDP has its value function initially set to v_d, and BRTDP has v_ℓ set to v_d and v_u set by running DS-MPI. For uninformed initialization, RTDP has its value function set uniformly to zero, and BRTDP has v_ℓ set to zero and v_u set to 10^6.

Figure 3 gives anytime performance curves for the algorithms on each of the test problems. We interrupt each algorithm at fixed intervals to consider the quality of the policy available at that time. Rather than simply evaluating the current greedy policy, we assume the executive agent has some limited computational power and can itself run RTDP on a given initial value function received from the planner. (This assumption results in a fairer comparison for RTDP, since that algorithm's greedy policy may be improper.) To evaluate a value function v, we place an agent at the start state, initialize its value function to v, run RTDP until we reach the goal, and record the cost of the resulting trajectory. The curves in Figure 3 are the result of 100 separate runs of each algorithm, with each value function evaluated using 200 repetitions of the above testing procedure.

BRTDP performs 4 backups for each state on the trajectories it simulates: one each on v_u and v_ℓ on the way down, and one each on the way back. RTDP performs only one backup per sampled state. Because of locality, BRTDP has better cache performance and lower

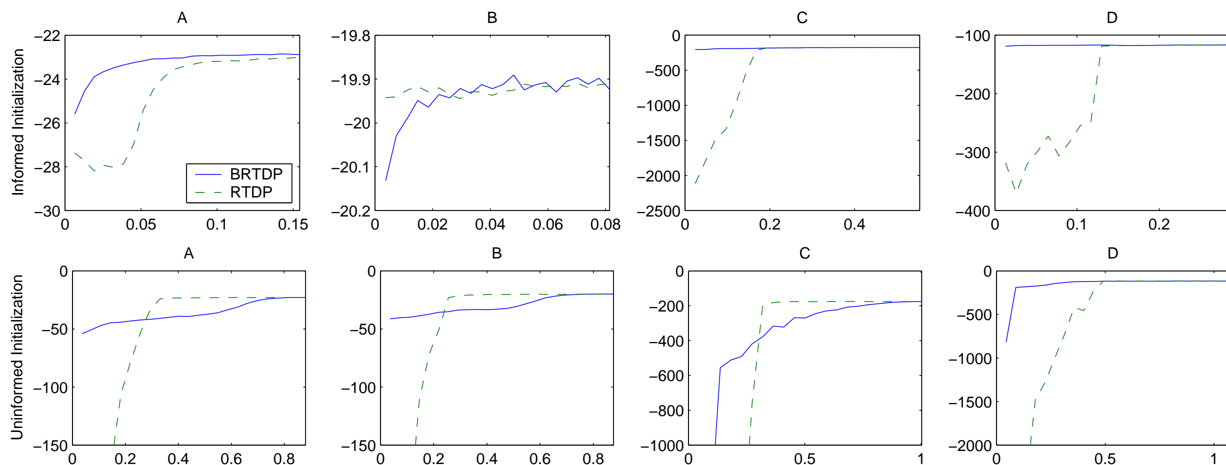

Figure 3. Anytime performance of informed and uninformed RTDP and BRTDP: the first row is for the informed initialization, and the second for uninformed. The X-axis gives number of backups ($\times 10^5$), and the Y axis indicates the current value of the policy; Y-axis labels are (negative) rewards, so higher numbers are better. Note the differences in scales.

overhead per backup, and on the test problems we observed 1.5 to 3 times more backups per unit of runtime than RTDP. Thus, if Figure 3 were re-plotted with time as the X-axis, the performance of BRTDP would appear even stronger. So, our evaluation has handicapped BRTDP in two ways: we compare it to RTDP in terms of number of updates rather than CPU time, and we evaluate RTDP-trajectories rather than stationary policies, even though stationary policies taken from BRTDP have provable guarantees.

Several conclusions can be drawn from the results in Figure 3. First, appropriate initialization provides significant help to both RTDP and BRTDP. Second, under both types of initialization, BRTDP often provides much higher-reward policies than RTDP for a given number of backups (especially with a small number of backups or with informative initialization), and we never observed its policies to be much worse than RTDP. In particular, on problems (C) and (D) BRTDP is nearly optimal from the very beginning. This, combined with the fact that BRTDP provides performance bounds even for stationary policies, make BRTDP an attractive option for anytime applications.

6.2. Off-line Convergence

We compare off-line convergence times for BRTDP to those of LRTDP and HDP.[5] Again, we consider both informed and uninformed initialization. Informed LRTDP and HDP have their value functions initialized to v_d, while uninformed initialization sets them to zero. Time spent computing informed initialization values is not charged to the algorithms. This time will be somewhat longer for BRTDP, as it also uses an upper bound heuristic; however, in our experiments this time is dominated by the algorithm runtime.

We evaluate the algorithms by measuring the time it takes to find an α-optimal partial policy. For BRTDP, since we maintain upper and lower bounds, we can simply terminate when $(v_u(\mathsf{s}) - v_\ell(\mathsf{s})) \leq \alpha$; we used $\alpha = 0.1$ in our experiments. As discussed in Section 3 we initialized v_u to a monotone upper bound, so the greedy policy with respect to the final v_u will be within α of optimal. The other tested algorithms measure convergence by stopping when the max-norm Bellman error drops below some tolerance ϵ. Without further information there is no way to translate ϵ into a bound on policy quality: we can incur an extra cost of ϵ at each step of our trajectory, but since our trajectory could have arbitrarily many steps we could be arbitrarily suboptimal by the end. To provide an approximately equivalent stopping criterion, we used the following heuristic: pick an optimal policy π^* and let $\ell^\star(x)$ be the expected number of steps to reach g from x by following π^*. Then take $\epsilon = \alpha/\ell^\star(\mathsf{s})$. This heuristic yielded $\epsilon = 0.004$, 0.005, 0.001, and 0.002 for problems (A) through (D).

As expected, on (B) and (D), the problems with dense noise, BRTDP significantly outperformed the other algorithms. On (D), uninformed BRTDP is 3.2 times faster than uninformed HDP, and informed BRTDP is 6.4 times faster than informed HDP. Un-

[5]Improved LAO* is very similar to HDP without labeling solved states, and (Bonet & Geffner, 2003a) shows HDP has generally better performance, so LAO* was not considered in our experiments.

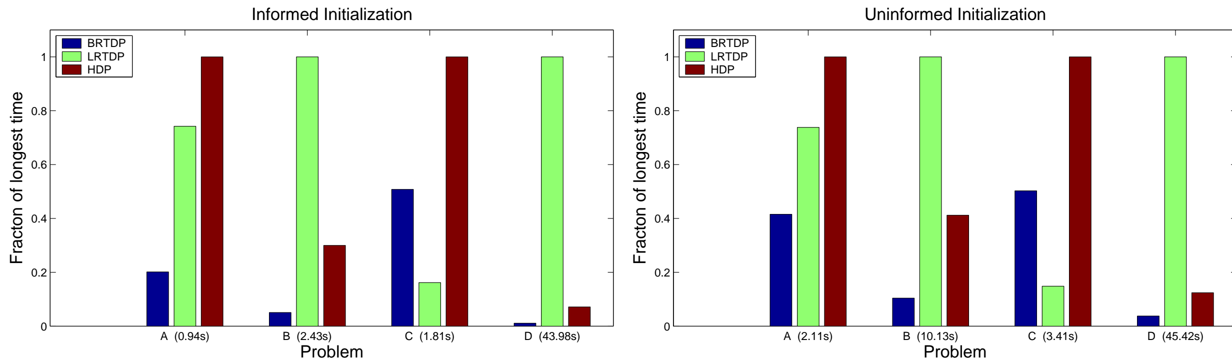

Figure 4. CPU time required for convergence with *informed* (left) and *uninformed* (right) initialization of the algorithms.

informed BRTDP outperforms informed HDP on (D) by a factor of 1.8. More importantly, on (B) and (D) HDP and LRTDP visit all of S before convergence, while BRTDP does not: for example, on (B), informed BRTDP visits 28% of S and only brings $|v_u(x) - v_\ell(x)| \leq \alpha$ for 10% of S. We could make the performance gap arbitrarily large by adding additional states to the MDPs.

7. Conclusions

We have shown BRTDP paired with DS-MPI is a powerful combination for both offline and anytime applications. BRTDP can converge quickly when other algorithms cannot, and it can return policies with strong performance guarantees at any time. In future work we hope to generalize DS-MPI and apply it to other algorithms, as well as continue to develop BRTDP.

Acknowledgments This work was supported in part by NSF grant EF-0331657.

References

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artif. Intell.*, *72*, 81–138.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Bonet, B., & Geffner, H. (2003a). Faster heuristic search algorithms for planning with uncertainty and full feedback. *Proc. 18th International Joint Conf. on Artificial Intelligence* (pp. 1233–1238). Acapulco, Mexico: Morgan Kaufmann.

Bonet, B., & Geffner, H. (2003b). Labeled RTDP: Improving the convergence of real-time dynamic programming. *Proc. of ICAPS-03* (pp. 12–21).

Dean, T., Kaelbling, L. P., Kirman, J., & Nicholson, A. (1995). Planning under time constraints in stochastic domains. *Artif. Intell.*, *76*, 35–74.

Ferguson, D., & Stentz, A. T. (2004). *Focussed dynamic programming: Extensive comparative results* (Technical Report CMU-RI-TR-04-13). Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

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

McMahan, H. B., & Gordon, G. J. (2005). Fast exact planning in markov decision processes. *To appear in ICAPS*.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., & Liang, E. (2004). Inverted autonomous helicopter flight via reinforcement learning. *ISER*. Springer.

Roy, N., Gordon, G., & Thrun, S. (2004). Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research*. To appear.

Smith, T., & Simmons, R. (2004). Heuristic search value iteration for pomdps. *Proc. of UAI 2004*. Banff, Alberta.