Focused Real-Time Dynamic Programming for MDPs: Squeezing More Out of a Heuristic

Trey Smith and **Reid Simmons**
Robotics Institute, Carnegie Mellon University
{trey,reids}@ri.cmu.edu

Abstract

Real-time dynamic programming (RTDP) is a heuristic search algorithm for solving MDPs. We present a modified algorithm called Focused RTDP with several improvements. While RTDP maintains only an upper bound on the long-term reward function, FRTDP maintains two-sided bounds and bases the output policy on the lower bound. FRTDP guides search with a new rule for outcome selection, focusing on parts of the search graph that contribute most to uncertainty about the values of good policies. FRTDP has modified trial termination criteria that should allow it to solve some problems (within ϵ) that RTDP cannot. Experiments show that for all the problems we studied, FRTDP significantly outperforms RTDP and LRTDP, and converges with up to six times fewer backups than the state-of-the-art HDP algorithm.

Introduction

Markov decision processes (MDPs) are planning problems in which an agent's actions have uncertain outcomes, but the state of the world is fully observable. This paper studies techniques for speeding up MDP planning by leveraging heuristic information about the value function (expected long-term reward available from each state). In many domains, one can quickly calculate upper and lower bounds on the value function. As with the A* algorithm in a deterministic setting, admissible bounds can be used to prune much of the search space while still guaranteeing optimality of the resulting policy.

Real-time dynamic programming (RTDP) is a well-known MDP heuristic search algorithm (Barto, Bradtke, & Singh 1995). Each RTDP trial begins at the initial state of the MDP and explores forward, choosing actions greedily and choosing outcomes stochastically according to their probability.

This paper introduces the Focused RTDP algorithm, which is designed to both converge faster than RTDP and solve a broader class of problems. Whereas RTDP keeps only an upper bound on the long-term reward function, FRTDP keeps two-sided bounds and bases its output policy on the lower bound (Goodwin 1996), significantly improving anytime solution quality and performance guarantees.

FRTDP guides outcome selection by maintaining a *priority value* at each node that estimates the benefit of directing search to that node. Priority-based outcome selection both focuses sampling on the most relevant parts of the search graph and allows FRTDP to avoid nodes that have already converged.

FRTDP has modified trial termination criteria that should allow it to solve some problems that RTDP cannot. RTDP is known to solve a class of non-pathological stochastic shortest path problems (Barto, Bradtke, & Singh 1995). We conjecture that FRTDP additionally solves (within ϵ) a broader class of problems in which the state set may be infinite and the goal may not be reachable from every state.

Relative to existing algorithms, FRTDP is intended to be more robust, converge faster, and have better anytime solution quality before convergence is reached. Experimentally, FRTDP significantly outperformed several other algorithms on all the problems we studied.

Related Work

Action selection based on the lower bound is a frequently occurring idea in decision theoretic search. It was used in (Goodwin 1996) and probably earlier, and applied to MDPs in (McMahan, Likhachev, & Gordon 2005).

LRTDP (Bonet & Geffner 2003b) and HDP (Bonet & Geffner 2003a) are RTDP-derived algorithms that similarly use the idea of avoiding updates to irrelevant states. However, their trials are not restricted to a single path through the search graph, and they do not explicitly select outcomes. Irrelevant states are avoided through a modified trial termination rule.

HSVI (Smith & Simmons 2004) and BRTDP (McMahan, Likhachev, & Gordon 2005) both include the idea of outcome selection, but they prioritize internal nodes by uncertainty rather than the FRTDP concept of priority. We conjecture that FRTDP priorities will lead to better performance than uncertainty values because they better reflect the single-path trial structure of RTDP. Unfortunately, we do not have a performance comparison. HSVI is designed for POMDPs, and we did not compare with BRTDP because we did not become aware of it until just before publication.

LAO* (Hansen & Zilberstein 2001) is another heuristic search algorithm, but with a control structure unlike RTDP's deep single-path trials. We did compare with LAO* because it was dominated by LRTDP in an earlier study with similar problems (Bonet & Geffner 2003b).

IPS and PPI (McMahan & Gordon 2005) also outperform LRTDP on racetrack problems. We did not compare with these algorithms because they explore backward from a goal state rather than forward from a start state. The distinction is not crucial for the racetrack problem, but forward exploration is required for multiple-reward problems where the set of goal states is ill-defined, and for problems where the set of possible predecessors of a state is not finite (as when POMDPs are formulated as belief-state MDPs).

MDP Model

An MDP models a planning problem in which action outcomes are uncertain but the world state is fully observable. The agent is assumed to know a probability distribution for action outcomes. This paper studies discrete infinite-horizon stationary MDPs, formally described by a set of states $\mathcal{S}$, a finite set of actions $\mathcal{A}$, a successor function C providing the set of states that could result from an action $C^a(s) \subseteq \mathcal{S}$, transition probabilities $T^a(s_i, s_j) = \Pr(s_j | s_i, a)$, a real-valued reward function $R(s, a)$, a discount factor $\gamma \leq 1$, an initial state s_0, and a (possibly empty) set of absorbing goal states $\mathcal{G} \subseteq \mathcal{S}$. Taking any action in a goal state causes a zero-reward self-transition with probability 1.

The object of the planning problem is to generate a stationary policy π that maximizes expected long-term reward:

$$J^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s, \pi\right] \tag{1}$$

Our algorithms generate an approximately optimal policy $\hat{\pi}$ with small regret $J^{\pi^*}(s_0) - J^{\hat{\pi}}(s_0)$ when executed starting from s_0.

Value Iteration and Heuristic Search

An MDP can be solved by approximating its optimal value function $V^* = J^{\pi^*}$. Any value function V induces a greedy policy π_V in which actions are selected via one-step lookahead. The policy induced by V^* is optimal.

Value iteration (VI) is a widely used dynamic programming technique in which one solves for V^* using the fact that it is the unique fixed point of the Bellman update:

$$V(s) \leftarrow \max_a \left[R(s, a) + \gamma \sum_{s' \in C^a(s)} T^a(s, s') V(s')\right]. \tag{2}$$

VI algorithms start with an initial guess for the right-hand side of (2) and repeatedly update so that V gets closer to V^*.

In classical synchronous value iteration (VI), at each step the Bellman update is applied over all states simultaneously. Conversely, asynchronous algorithms update states one at a time. By updating the most relevant states more often and carefully choosing the right update ordering, they often achieve faster convergence.

The heuristic search algorithms we consider are asynchronous VI algorithms that can use additional *a priori* information in the form of admissible bounds h_L and h_U on the optimal value function, satisfying $h_L \leq V^* \leq h_U$. For goal states $s \in \mathcal{G}$, we enforce that $h_L(s) = h_U(s) = 0$. h_L and h_U help the algorithm choose which states to asynchronously update.

Incremental Search Graph Expansion

Every MDP has a corresponding AND/OR search graph. Nodes of the graph are states of the MDP. Each state/action pair of the MDP is represented with a k-connector in the search graph, connecting a state s to its possible successors $C^a(s)$. Individual edges of each k-connector are annotated with transition probabilities.

The *explicit graph* is a data structure containing a subset of the MDP search graph. Heuristic search algorithms can often reach an optimal solution while only examining a tiny fraction of the states in the search graph, in which case generating node and link data structures for the unexamined states would be wasteful. Instead, one usually provides the algorithm with an initial explicit graph containing s_0 and callbacks for extending the explicit graph as needed.

Expanding a node means generating its outgoing k-connectors and successor nodes and adding them to the explicit graph. Explicit graph nodes are either *internal nodes*, which have already been expanded, or *fringe nodes*, which have not. Let $\mathcal{I}$ denote the set of internal nodes and $\mathcal{F}$ the set of fringe nodes.

In this framework, one can ask: given the information embodied in a particular explicit graph, what inferences can be drawn about the quality of different policies? And which fringe nodes should be expanded in order to improve quality estimates?

A useful statistic for characterizing a policy π is its *occupancy* $W^\pi(s)$ for each state s. If we consider the distribution of possible execution traces for π and interpret the discount γ in terms of trace termination (i.e., execution terminates at any given time step with probability $1 - \gamma$), then $W^\pi(s)$ is the expected number of time steps per execution that π spends in state s before passing beyond the fringe into the unknown part of the graph.

Formally, occupancy is defined as the solution to the following simultaneous equations ($s' \in \mathcal{I} \cup \mathcal{F}$):

$$W^\pi(s') = W_0(s') + \gamma \sum_{s \in \mathcal{I} - \mathcal{G}} W^\pi(s) T^{\pi(s)}(s, s') \tag{3}$$

where $W_0(s')$ is 1 if $s' = s_0$ and 0 otherwise.[1]

The occupancy at each fringe node indicates its relevance to the policy. In particular, the quality $J^\pi(s_0)$ of a policy satisfies

$$J^\pi(s_0) = J^\pi_{\mathcal{I}} + \sum_{s \in \mathcal{F}} W^\pi(s) J^\pi(s) \tag{4}$$

where $J^\pi_{\mathcal{I}}$ is the expected reward from executing π up to the point where it reaches a fringe node. Given an explicit graph, $J^\pi_{\mathcal{I}}$ can be calculated, but $J^\pi(s)$ for fringe nodes s

[1]If $\gamma = 1$ and π has loops, the occupancy of some states may diverge. However, it converges for the problems and policies that interest us.

cannot, because it depends on information in the unexplored part of the graph.

One way to estimate $V^*(s_0)$ is by keeping bound functions $V^L \leq V^* \leq V^U$ and choosing fringe nodes that help squeeze the bounds interval $|V^U(s_0) - V^L(s_0)|$ down to zero. Eq. 4 can be added and subtracted to get

$$|V^U(s_0) - V^L(s_0)| \leq \sum_{s \in \mathcal{F}} W^{\pi^*}(s)|V^U(s) - V^L(s)| \quad (5)$$

$$\leq \sum_{s \in \mathcal{F}} W^{\pi^*}(s)|h_U(s) - h_L(s)| \quad (6)$$

where the constant $J^\pi_\mathcal{I}$ has dropped out by subtraction, and we have brought in the only available information about the value at fringe nodes, i.e., their heuristic values. This expression makes clear each fringe node's contribution to the uncertainty at s_0. The best possible result of expanding a fringe node s is to decrease its local uncertainty to 0, reducing the uncertainty at s_0 by at most $W^{\pi^*}(s)|h_U(s) - h_L(s)|$, "the occupancy times the uncertainty". Later, we discuss how to approximate this upper bound on uncertainty reduction and use it to guide fringe node expansion.

Real-Time Dynamic Programming

RTDP (alg. 1) is an asynchronous VI algorithm that works by repeated trials; each trial starts at s_0 and explores forward in the search graph. At each forward step, action selection is greedy based on the current value function, and outcome selection is stochastic according to the distribution of possible successor states given the chosen action. When a goal state is reached, RTDP terminates the trial by retracing its steps back to s_0, updating each state along the way. The value function V^U is initialized with a heuristic $h_U \geq V^*$. Like the A* algorithm in the deterministic setting, RTDP often converges without even examining all the states of the problem.

Focused RTDP

Focused RTDP (alg. 2) is derived from RTDP. As in RTDP, FRTDP execution proceeds in trials that begin at s_0 and explore forward through the search graph, selecting actions greedily according to the upper bound, then terminating and performing updates on the way back to s_0. Unlike RTDP, FRTDP maintains a lower bound and uses modified rules for action selection and trial termination.

Using a Lower Bound

RTDP keeps an upper bound V^U that is initialized with an admissible heuristic $h_U \geq V^*$, and its output policy is the greedy policy induced by V^U. In contrast, FRTDP keeps two-sided bounds $V^L \leq V^* \leq V^U$ and outputs the greedy policy induced by V^L.

There are two main benefits to keeping a lower bound. First, if h_L is uniformly improvable[2], the greedy policy induced by V_L has value at least as good as $V_L(s_0)$—in other

[2]Applying the Bellman update to a uniformly improvable function brings the function everywhere closer to V^* (Zhang & Zhang 2001).

Algorithm 1 RTDP

```
function initNode($s$):
  {implicitly called the first time each state $s$ is touched}
  $s$.U $\leftarrow$ $h_U(s)$
function QU($s, a$):
  return $R(s,a) + \gamma \sum_{s' \in C^a(s)} T^a(s,s')\ s'$.U
function backup($s, a$):
  $s$.U $\leftarrow$ $\max_a$ QU($s, a$)
function trialRecurse($s$):
  if $s \in \mathcal{G}$ then return end if
  $a^* \leftarrow \text{argmax}_a$QU($s, a$)
  trialRecurse(chooseSuccessorStochastically($s, a^*$))
  backup($s$)
function RTDP($s_0$):
  loop trialRecurse($s_0$)
```

Algorithm 2 Focused RTDP

```
function initNode($s$):
  {implicitly called the first time each state $s$ is touched}
  ($s$.L, $s$.U) $\leftarrow$ ($h_L(s), h_U(s)$); $s$.prio $\leftarrow$ $\Delta(s)$
function $\Delta(s)$:
  return $|s$.U - $s$.L$| - \epsilon/2$
function QL($s, a$):
  return $R(s,a) + \gamma \sum_{s' \in C^a(s)} T^a(s,s')\ s'$.L
function QU($s, a$):
  return $R(s,a) + \gamma \sum_{s' \in C^a(s)} T^a(s,s')\ s'$.U
function backup($s, a$):
  $s$.L $\leftarrow$ $\max_a$ QL($s, a$)
  $(u, a^*) \leftarrow$ max, argmax$_a$QU($s, a$)
  $\delta \leftarrow |s$.U $- u|$; $s$.U $\leftarrow u$
  $(p, s^*) \leftarrow$ max, argmax$_{s' \in C^{a^*}(s)} \gamma T^{a^*}(s,s')\ s'$.prio
  $s$.prio $\leftarrow$ $\min(\Delta(s), p)$
  return $(a^*, s^*, \delta)$
function trackUpdateQuality($q, d$):
  if $d > D/k_D$ then
    $(q_{\text{curr}}, n_{\text{curr}}) \leftarrow (q_{\text{curr}} + q, n_{\text{curr}} + 1)$
  else
    $(q_{\text{prev}}, n_{\text{prev}}) \leftarrow (q_{\text{prev}} + q, n_{\text{prev}} + 1)$
  end if
function trialRecurse($s, W, d$):
  $(a^*, s^*, \delta) \leftarrow$ backup($s$)
  trackUpdateQuality($\delta W, d$)
  if $\Delta(s) \leq 0$ or $d \geq D$ then return
  trialRecurse($s^*, \gamma T^{a^*}(s, s^*)W, d + 1$)
  backup($s$)
function FRTDP($s_0$):
  $D \leftarrow D_0$
  while $s_0$.U - $s_0$.L $> \epsilon$ do
    $(q_{\text{prev}}, n_{\text{prev}}, q_{\text{curr}}, n_{\text{curr}}) \leftarrow (0, 0, 0, 0)$
    trialRecurse($s_0, W = 1, d = 0$)
    if $(q_{\text{curr}}/n_{\text{curr}}) \geq (q_{\text{prev}}/n_{\text{prev}})$ then $D \leftarrow k_D D$
  end while
```

words, one can interrupt the algorithm at any time and get a policy with a performance guarantee. The second benefit is that empirically, up to the point where V^L and V^U converge, policies derived from V^L tend to perform better. Policies derived from the upper bound are often "get rich quick" schemes that seem good only because they have not been thoroughly evaluated. The obvious drawback to keeping a lower bound is that updating it increases the cost of each backup. In practice, we observe that adding lower bound calculation to the HDP algorithm increases wallclock time to convergence by about 10%, but with substantial benefits to anytime performance.

FRTDP maintains a lower bound and outputs the greedy policy induced by the lower bound. It also uses the lower bound during its priority calculation for outcome selection, as described below.

Outcome Selection

Whereas RTDP chooses an action outcome stochastically, FRTDP outcome selection attempts to maximize the improvement in the quality estimate of the greedy policy π_U by expanding the fringe node s with the largest contribution to the uncertainty of $J^{\pi_U}(s_0)$.

FRTDP allows the user to specify a regret bound ϵ. If there is an envelope of nodes s that all satisfy $|V^U(s) - V^L(s)| \leq \epsilon$, then FRTDP algorithm termination can be achieved without expanding any more fringe nodes. And it is perhaps easier to achieve a condition where $|V^U(s) - V^L(s)| \leq \epsilon/2$ for a majority of nodes in the envelope, with uncertainties not too large at the rest. Thus, FRTDP can safely terminate a trial when it reaches a state whose uncertainty is very small. We define the *excess uncertainty* of a state $\Delta(s) = |V^U(s) - V^L(s)| - \epsilon/2$ and terminate any trial that reaches a state with $\Delta(s) \leq 0$. (This is one of two trial termination criteria; see the next section.)

FRTDP outcome selection is designed to choose the fringe node s that maximizes $W^{\pi_U}(s)\Delta(s)$ (occupancy times excess uncertainty). But due to computational constraints, it prioritizes nodes via an approximation scheme that only guarantees the best fringe node in certain special cases. FRTDP recursively calculates a priority $p(s)$ for each node s, such that choosing the successor state with the highest priority at each step causes the trial to arrive at the maximizing fringe node. The recursive update formula is

$$p(s) \;=\; \Delta(s) \;\; (\text{fringe})$$
$$p(s) \;=\; \min(\Delta(s), \max_{s' \in C^{a^*}(s)} \gamma T^{a^*}(s, s') p(s')) \;\; (\text{internal})$$

where the action a^* is chosen greedily according to the upper bound. $p(s)$ is recalculated along with $V^U(s)$ and $V^L(s)$ at each update of a node.

The priority update rule is guaranteed to lead FRTDP to the best fringe node only in the case that the search graph is a tree. In a general graph, there are two confounding factors that violate the guarantee. First, $W^\pi(s)$ is the expected amount of time π spends in s, *adding up all possible paths* from s_0 to s. Maximizing $p(s)$ at each step effectively prioritizes fringe nodes according to their maximum occupancy along the *single most likely path* (in a tree there is only one path). Second, since after trial termination FRTDP performs updates back to s_0 along only one path instead of along all paths, priorities at internal nodes can be inconsistent with respect to their successors.

In practice we find that, despite multi-path violations of the assumptions on which the priority is based, choosing outcomes by priority is better than choosing them stochastically. There may also be more accurate priority update schemes that mitigate multi-path error—the current scheme was chosen to keep overhead small and retain the trial-based framework of RTDP.

Adaptive Maxumum Depth Termination

With the excess uncertainty trial termination alone, FRTDP is a usable search algorithm. However, as with RTDP, poor outcome selection early in a trial could lead into a quagmire of irrelevant states that takes a long time to escape.

FRTDP's *adaptive maximum depth* (AMD) trial termination criterion mitigates this problem by cutting off long trials. FRTDP maintains a *current maximum depth* D. A trial is terminated if it reaches depth $d \geq D$. FRTDP initializes D to a small value D_0, and increases it for subsequent trials. The idea is to avoid over-committing to long trials early on, but retain the ability to go deeper in later trials, in case there are relevant states deeper in the search graph.

FRTDP performance for any particular problem depends on how D is adjusted, so it is important that whatever technique is used be relatively robust across problems without manual parameter tuning. We chose to adjust D adaptively, using trial statistics as feedback. After each trial, FRTDP chooses whether to keep the current value of D or update $D \leftarrow k_D D$.

The feedback mechanism is fairly ad hoc. Each update in a trial is given an *update quality score* $q = \delta W$ that is intended to reflect how useful the update was. δ measures how much the update changed the upper bound value $V^U(s)$. W is a single-path estimate of the occupancy of the state being updated under the current greedy policy. After each trial, D is increased if the average update quality near the end of the trial ($d > D/k_D$) is at least as good as the average update quality in the earlier part of the trial. Refer to the pseudocode of alg. 2 for details.

The racetrack problems used in our experiments were designed to be solved by RTDP, so it is no surprise that they are particularly benign and suitable for deep trials. In the racetrack domain, AMD termination hurts performance slightly overall, as early trials are less efficient before D grows large. However, AMD improves performance on some more challenging problems (not reported here). For all the listed results, we used AMD with $D_0 = 10$ and $k_D = 1.1$.

Model Assumptions and Convergence

Stochastic shortest path problems (SSPs) are MDPs that satisfy additional restrictions (Bertsekas & Tsitsiklis 1996):

S1. All rewards are strictly negative.

S2. $\mathcal{S}$ is finite.

S3. There exists at least one *proper policy*, that is, a policy that reaches $\mathcal{G}$ from any state with probability 1.

RTDP requires another condition:

R1. All policies that are improper must incur infinite cost for at least one state.

Under conditions S1-S3 and R1, RTDP's V^U value function is guaranteed (with probability 1) to converge to V^* over the set of *relevant states*, i.e., states that can be reached by at least one optimal policy (Barto, Bradtke, & Singh 1995). Unsurprisingly, there is a corresponding result for FRTDP.

Theorem 1. *Under conditions S1-S3 and R1 and setting* $\epsilon = 0$*, FRTDP's* V^L *and* V^U *bounds are guaranteed to converge to* V^* *over the set of relevant states.*

We conjecture that FRTDP can also approximately solve a broader class of SSP-like problems that satisfy:

F1. There are global reward bounds R_L and R_U such that for every (s, a) pair, $R_L \leq R(s, a) \leq R_U < 0$.

F2. There exists at least one policy that (a) reaches $\mathcal{G}$ *starting from* s_0 with probability 1, and (b) has positive occupancy for only a finite number of states.

F3. h_L and h_U are uniformly improvable.

Conjecture 2. *Under conditions F1-F3, FRTDP is guaranteed to terminate with an output policy whose regret is bounded within* ϵ*.*

FRTDP should terminate under these weaker conditions because (unlike RTDP) each trial is capped at a finite maxumum depth D; thus poor decisions early in a trial can always be reconsidered in the next trial. The mechanism for adjusting D should not affect the termination guarantee, as long as the sequence of cap values increases without bound.

The weaker conditions allow FRTDP to solve problems that RTDP cannot. For example, POMDPs formulated as belief-space MDPs naturally have an infinite state set. FRTDP can still solve them if they satisfy F1-F3. FRTDP can also solve problems with "one-way doors", in which poor early action choices lead to states from which the goal is unreachable, as long as there is a policy guaranteed to reach the goal starting from s_0.

Experimental Results

We evaluated the performance of FRTDP on problems in the popular racetrack benchmark domain from (Barto, Bradtke, & Singh 1995). States of racetrack are integer vectors in the form $(x, y, \dot{x}, \dot{y})$ that represent the discrete position and speed of the car in a 2D grid. The actions available to the car are integer accelerations $(\ddot{x}, \ddot{y})$ where both $\ddot{x}$ and $\ddot{y}$ are drawn from $\{-1, 0, 1\}$. The car starts in one of a set of possible start states. The goal is to maneuver the car into one of a set of goal states. Some cells in the grid are marked as obstacles; if the car's path intersects one of these cells, it is reset back to one of the start states with zero velocity. Uncertainty in this problem comes from "skidding". Each time the agent takes an acceleration action, with probability p the car skids: the commanded action is replaced with $(\ddot{x}, \ddot{y}) = (0, 0)$.

Because FRTDP focuses on outcome selection, we also wanted to study increasing the amount of uncertainty in the problem. We did this in two ways. First, we examined performance with $p = 0.1$ (the standard case) and $p = 0.3$ (increased chance of skidding, marked by adding a suffix of `-3` to the problem name). Second, we tried increasing the number of possible outcomes from an error. We call this the "wind" variant (marked by adding a suffix of `-w`). In the wind variant, with probability $p = 0.1$ an additional acceleration is added to the commanded acceleration. The additional acceleration is drawn from a uniform distribution over 8 possible values: $\{(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)\}$. The idea is that instead of skidding, a "gust of wind" provides additional acceleration in a random direction.

We selected two racetrack problems whose maps have been published: `large-b` from (Barto, Bradtke, & Singh 1995) and `large-ring` from (Bonet & Geffner 2003b). With three versions for each problem, our results cover a total of six problems.

We selected three heuristic search asynchronous VI algorithms to compare with FRTDP: RTDP, LRTDP, and HDP. In addition, we implemented a modified version of HDP that maintains a lower bound and uses that as the basis for its output policy. We call this algorithm HDP+L.

Following (Bonet & Geffner 2003b), all algorithms were provided with the same admissible upper bound heuristic h_U, calculated by a domain-independent relaxation in which the best possible outcome of any action is assumed to always occur. Formally, the Bellman update is replaced by

$$V(s) \leftarrow \max_a \left[R(s, a) + \gamma \max_{s' \in C^a(s)} V(s') \right] \tag{7}$$

The time required to calculate h_U is not included in the reported running times for the algorithms.

There is no trivial way to calculate an informative admissible lower bound for a racetrack problem. A formally correct way to handle this, with minor algorithm modifications, is to set $h_L(s) = -\infty$ for all non-goal states s. However, dealing with infinite values would have required some extra bookkeeping, so for convenience we supplied $h_L(s) = -1000$, which is a gross underestimate of the actual $V^*(s)$ values. In principle, the finite lower bound could allow FRTDP to prune some additional low-probability outcomes, but this did not happen in practice. See (McMahan, Likhachev, & Gordon 2005) for discussion of how to efficiently calculate a more informative lower bound.

Fig. 1 reports time to convergence within $\epsilon = 10^{-3}$ for each (problem, algorithm) pair, measured both as number of backups and wallclock time. Our experiments were run on a 3.2 GHz Pentium-4 processor with 1 GB of RAM. We implemented all the algorithms in C++; they were not thoroughly optimized, so the number of backups required for convergence was measured more reliably than the wallclock time (which could probably be substantially reduced across the board). Included in wallclock time measurements is the time required to check racetrack paths for collisions; collision checking was performed the first time a state was expanded, and the results were cached for subsequent updates.

The observed speedup of FRTDP convergence compared to HDP, measured in terms of number of backups, ranges from 2.9 up to 6.4. Our initial expectation was that FRTDP

Algorithm	large-b	large-b-3	large-b-w	large-ring	large-ring-3	large-ring-w
RTDP	5.30 (5.19)	10.27 (9.12)	149.07 (190.55)	3.39 (4.81)	8.05 (8.56)	16.44 (91.67)
LRTDP	1.21 (3.52)	1.63 (4.08)	1.96 (14.38)	1.74 (5.19)	2.14 (5.71)	3.13 (22.15)
HDP	1.29 (3.43)	1.86 (4.12)	2.87 (15.99)	1.27 (4.35)	2.74 (6.41)	2.92 (20.14)
HDP+L	1.29 (3.75)	1.86 (4.55)	2.87 (16.88)	1.27 (4.70)	2.74 (7.02)	2.92 (21.12)
FRTDP	**0.29 (2.10)**	**0.49 (2.38)**	**0.84 (10.71)**	**0.22 (2.60)**	**0.43 (3.04)**	**0.99 (14.73)**

Figure 1: Millions of backups before convergence with $\epsilon = 10^{-3}$. Each entry gives the number of millions of backups, with the corresponding wallclock time (seconds) in parentheses. The fastest time for each problem is shown in bold.

would show more speedup on the `-3` and `-w` problem variants with more uncertainty; in fact its speedup was about the same on `-3` problems and smaller on `-w` problems. We do not yet understand why this is the case. By construction, HDP and HDP+L have identical convergence properties in terms of the number of backups required. As measured in wallclock time, lower bound updating for HDP+L introduces an additional cost overhead of about 10%.

Fig. 2 reports anytime performance of three of the algorithms (HDP, HDP+L, and FRTDP) on the two problems where FRTDP showed the least convergence time speedup (`large-ring-w`) and the most speedup (`large-ring-3`) relative to HDP. The quality (expected reward) of an algorithm's output policy was measured at each epoch by simulating the policy 1000 times, with each execution terminated after 250 time steps if the goal was not reached. Error bars are 2σ confidence intervals. The two algorithms that output policies based on a lower bound (HDP+L and FRTDP) are seen to have significantly better anytime performance. In fact, for `large-ring-w`, FRTDP reaches a solution quality of -40 with about 40 times fewer backups than HDP. In each plot, the solid line indicating FRTDP solution quality ends at the point where FRTDP achieved convergence.

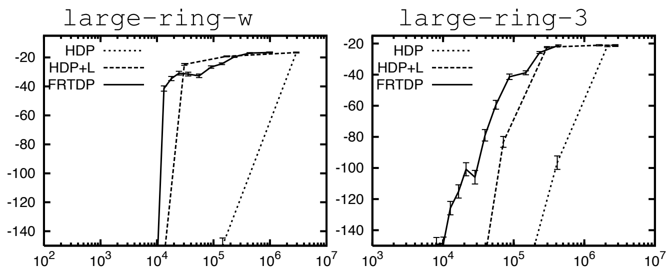

Figure 2: Anytime performance comparison: solution quality vs. number of backups.

Conclusions

FRTDP improves RTDP by keeping a lower bound and modifying outcome selection and trial termination rules. These modifications allow FRTDP to solve a broader class of problems, and in performance experiments FRTDP provided significant speedup across all problems, requiring up to six times fewer backups than HDP to reach convergence.

We also examined the separate performance impact of using a lower bound by implementing both HDP and HDP+L. This technique can be usefully applied on its own to any RTDP-like algorithm.

References

Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1-2):81–138.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

Bonet, B., and Geffner, H. 2003a. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proc. of IJCAI*.

Bonet, B., and Geffner, H. 2003b. Labeled RTDP: Improving the convergence of real time dynamic programming. In *Proc. of ICAPS*.

Goodwin, R. 1996. *Meta-Level Control for Decision Theoretic Planners*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon Univ., CMU-CS-96-186.

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

McMahan, H. B., and Gordon, G. J. 2005. Fast exact planning in Markov decision processes. In *Proc. of ICAPS*.

McMahan, H. B.; Likhachev, M.; and Gordon, G. J. 2005. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proc. of ICML*.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for POMDPs. In *Proc. of UAI*.

Zhang, N. L., and Zhang, W. 2001. Speeding up the convergence of value iteration in partially observable Markov decision processes. *Journal of AI Research* 14:29–51.