Bounded Incremental Real-Time Dynamic Programming*

Changjie Fan Xiaoping Chen
Department of Computer Science
University of Science and Technology of China
cjfan@mail.ustc.edu.cn xpchen@ustc.edu.cn

Abstract

A real-time multi-step planning problem is characterized by alternating decision-making and execution processes, whole online decision-making time divided in slices between each execution, and the pressing need for policy that only relates to current step. We propose a new criterion to judge the optimality of a policy based on the upper and lower bound theory. This criterion guarantees that the agent can act earlier in a real-time decision process while an optimal policy with sufficient precision still remains. We prove that, under certain conditions, one can obtain an optimal policy with arbitrary precision using such an incremental method. We present a Bounded Incremental Real-Time Dynamic Programming algorithm (BIRTDP). In the experiments of two typical real-time simulation systems, BIRTDP outperforms the other state-of-the-art RTDP algorithms tested.

1 Introduction

Markov decision processes (MDPs) have proven to be effective and principled tools for modeling many types of planning and action problems that are subject to uncertainty [1].

The earliest algorithms for solving MDPs were Value Iteration and Policy Iteration. These methods apply Dynamic Programming, and solve the optimal policy of all states in a backward fashion. More recently, algorithms based on forward search have been proposed, for instance, AO* [2], LAO* [3]. Both of them apply heuristic search to solve MDPs; the difference between them is that LAO* can handle the search graph with the existence of loop. In contrast with the Iteration Algorithms, they only compute the optimal policy for the given starting state. In the same vein, new Dynamic Programming algorithms have been designed: Heuristic Search/DP (HDP) [4], Envelope Propagation (EP) [5], and Focused Dynamic Programming (FP) [6].

From the perspective of control, these methods are primarily designed for open-loop systems: the entire policy is computed off-line. However, for many real-world applications, a closed-loop design with feedback is more reasonable. Accordingly, such algorithms as Real-Time Dynamic Programming (RTDP) [7], Labeled RTDP (LRTDP) [8], Bounded RTDP (BRTDP) [9], and Focused RTDP (FRTDP) [10] have been developed.

One technique, such as used in FRTDP, involves recording and constantly updating the upper and lower bounds of the expected reward corresponding to related states. This information guides the branch selection, dramatically improving performance. In addition, these algorithms apply such a criterion that when the difference between the upper and lower bound is small enough, an optimal policy with sufficient precision is obtained. Hence, on the basis of this criterion, the agent focuses its computation on minimizing the difference to compute the whole policy. In a real-time system with uncertainty, when there is a guarantee that an optimal policy with sufficient precision still remains after a certain action performed at current step, then the solving of the remainder policy can be deferred and this action can be performed right away. As a result, the outcome uncertainty of the action can be eliminated by observation and the decision-making is simplified. The problem of determining when the agent should commit to a decision is called the *stopping problem* [11]. In a real-time system with restrictions on decision time for each step, there is an urgent need for the policy to tell which action can be performed at the current step, therefore a more reasonable computational sequence governed by branch selection can improve the performance.

In this paper, we make the following contributions to a real-time Markov decision process: First, we propose a new criterion that deals with the stopping problem, guaranteeing that after the execution at the current step an optimal policy with sufficient precision still remains while avoiding solving the whole remainder policy. Second, we prove that, under some preconditions, such an incremental method can obtain an optimal policy with arbitrary precision over the

*: This work is supported by the NSFC under grant No.60275024 and the 973 Programme under No.2003CB317000.

0-7695-2999-2/07 $25.00 © 2007 IEEE
DOI 10.1109/FBIT.2007.14

whole process, and that the error does not accumulate. Third, we introduce a new branch-selection strategy and design a bounded incremental RTDP algorithm accordingly. Finally, we evaluate our algorithm, BIRTDP, in two typical real-time simulation systems designed on the RaceTrack benchmark problem. The experiments show BIRTDP has a better real-time performance than other state-of-the-art RTDP algorithms that we have tested.

2 Background

2.1 Notation

A Markov Decision Processes [12] model is a tuple $\langle \mathrm{S,A,T,R} \rangle$. S is a finite set of states, A is a finite set of actions, $\mathrm{T}: \mathrm{S} \times \mathrm{A} \times \mathrm{S} \to [0,1]$ gives the dynamic, we write $T^a(s,s') = P(s' \mid s,a)$ for the probability of reaching state s' when executing a from state s. $\mathrm{R}: \mathrm{S} \times \mathrm{A} \to \mathbb{R}$ is a real-valued reward function. A policy is defined as $\pi : S \to A$, it specifies which action to execute in each state. To evaluate a policy, there is a corresponding value function $V_\pi(s)$, which denotes the expected reward when applying the policy π from the state s. $Q(s,a)$ is defined as an action value function to denote the expected reward when execute action a from state s. By using the Bellman equations, we can update a policy as follows:

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} T^a(s,s') V_\pi(s') \tag{1}$$

$$V_\pi(s) = \max_{a \in A} Q(s,a) \tag{2}$$

$$\pi(s) = \arg\max_{a \in A} Q(s,a) \tag{3}$$

Here γ is a discount factor, $\gamma \le 1$. The optimal policy is denoted as π^*, and the corresponding value function is V^*. ε is the Bellman error, generally, when $V^* - V^\pi \le \varepsilon$, we say the related policy π is ε optimal.

2.2 Related Work

RTDP [7] is the earliest Real-Time Dynamic Programming Algorithm. It is executed over a series of repeated trials. Each trial starts with the initial state and ends with the goal state, then the value function of related states are updated by using Bellman equations. The process is a search in an AND/OR tree. The branch selection consists of action selection and outcome selection in turns. RTDP selects actions by a greedy policy with the heuristic information supplied by the upper bound of the value function, and selects outcomes stochastically according to the possibility distribution of the successor states following the selected action.

Some new search algorithms for MDPs also keep a lower bound of the value function to gain more information to guide the outcome selection. Focused RTDP [10] is such an algorithm. It is very similar to HSVI [13] based on POMDP. The key principle behind them is to decrease the uncertainty of the value function on the initial state s_0, denoted as $\hat{V}(s_0) = V_U(s_0) - V_L(s_0)$. Given the Bellman error ε, when $\hat{V}(s_0) \le \varepsilon$, the resultant policy will be regarded as ε optimal, because the uncertainty of any state is supplied by its successors when update with Bellman equations. In the outcome selection, they prefer the successor state that supplies the most uncertainty. The difference is that FRTDP uses priority to describe the uncertainty and employs an adaptive depth adjust mechanism.

BRTDP also keeps both upper and lower bounds. It introduces a more efficient technique for computing the initial bounds. Furthermore, it applies outcome selection by sampling but also according to uncertainty. This paper focuses on the real-time design, we have taken FRTDP for comparison because it is the latest RTDP algorithm that applies both upper and lower bounds.

Pemberton and Korf [11] developed the first incremental search method for real-time decision making and introduced the idea of incremental decision. However, their method avoided the stopping problem by assuming that there is a strict deadline at each step of decision-making [11]. Another related Algorithm involving incremental decision in real-time process is CLRTA* [14], it does not deal with this problem too. Furthermore, all these methods were designed for deterministic problems, so we will not take them for comparison.

3 Incremental Optimal Real-Time Decision

In this section, we first formulate the convergence criterion employed by FRTDP, then we introduce a new criterion and argue that it is more suitable for real-time decision-making; further, we propose the method of bounded incremental real-time decision-making and prove that the policy obtained is ε optimal.

Upper and lower bounds kept in FRTDP supply an interval estimate of the optimal value function $V^*(s)$. The upper bound $V_U(s)$ is an optimistic estimate; the lower bound $V_L(s)$ is a pessimistic estimate. For all $s \in S$, it holds that $V_L(s) \le V^*(s) \le V_U(s)$. Roughly

speaking, when referring to a lower bound, it will be considered that there exists a policy π_L, that when followed, V_L can be achieved, and we denote it as $V_L = V^{\pi_L}$. Then, we define the upper and lower bound of the action value functions $Q_U^a(s)$, $Q_L^a(s)$ (that are similar to equation (1)) and $a_U(s), a_L(s)$ below:

$$\begin{cases} Q_U^a(s) = R(s,a) + \gamma \sum_{s' \in S} T^a(s,s') V_U(s') \\ Q_L^a(s) = R(s,a) + \gamma \sum_{s' \in S} T^a(s,s') V_L(s') \end{cases} \quad (4)$$

$$\begin{cases} V_U(s) = \max_{a \in A} Q_U^a(s), \ a_U(s) = \arg\max_{a \in A} Q_U^a(s) \\ V_L(s) = \max_{a \in A} Q_L^a(s), \ a_L(s) = \arg\max_{a \in A} Q_L^a(s) \end{cases} \quad (5)$$

For conciseness, when referring to the initial state s_0, we omit the state symbol s_0.

FRTDP uses the convergence criterion:

$$V_U - V_L \le \varepsilon \quad (\text{I})$$

With the definitions above, when this criterion is met, it follows that:

$$V_U \le \varepsilon + V_L \Rightarrow V^* \le V_U \le \varepsilon + V_L = \varepsilon + V^{\pi_L},$$

that is, $V^* - V^{\pi_L} \le \varepsilon$. In other words, if criterion (I) is met, the ε optimal policy has been found.

Notice that if the lower bound of a certain action value function is greater than the upper bound of any other action value functions, then the policy of the corresponding state can be determined without knowing the exact expected reward. Accordingly, we define $V_U^{-a_L}$ as the maximal upper bound action value function except a_L on state s_0 and the corresponding action a_U^{-L}:

$$V_U^{-a_L} = \max_{a \in A, a \ne a_L} Q_U^a, \ a_U^{-L} = \arg\max_{a \in A, a \ne a_L} Q_U^a.$$

Taking into account the error, we present a new criterion below:

Theorem 1 If $V_U^{-a_L} - V_L \le \varepsilon$ (II), there is an ε optimal policy π with $\pi(s_0) = a_L$.

Proof. If $\pi^*(s_0) = a_L$, the proposition holds trivially. Otherwise, $\pi^*(s_0) \ne a_L$, then according to the definition of V^*, $V^* \le V_U^{-a_L}$, it follows that $V^* \le V_L + \varepsilon = V^{\pi_L} + \varepsilon$, that is $V^* - V^{\pi_L} \le \varepsilon$. In other words, if criterion (II) is met, after executing a_L, at least one ε optimal policy still remains. This is the theory basis to obtain the ε optimal policy incrementally.

Theorem 2 If criterion (I) is satisfied, criterion (II) is always met. The converse does not hold. Thus, it implies that criterion (II) is weaker than criterion (I) as a condition.

Proof. $V_U - V_L \le \varepsilon \Rightarrow V_U^{-a_L} - V_L \le \varepsilon$ is trivial. For its converse, figure 1 gives a counterexample.

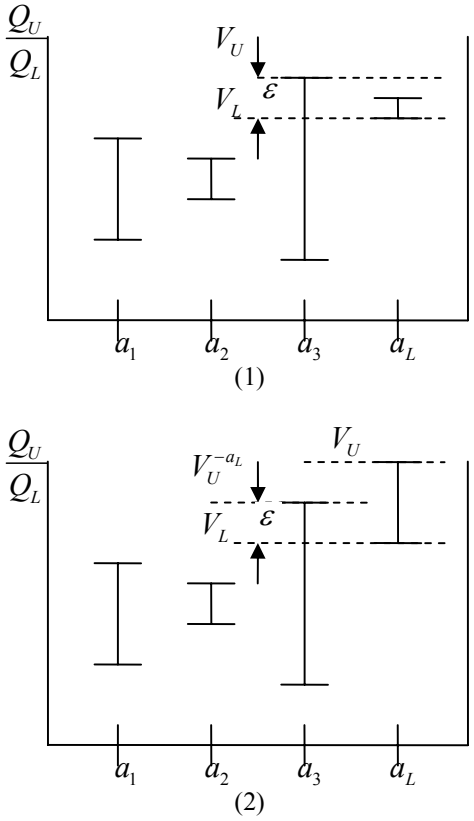


Figure 1. The two figures above show the difference between criterion (I) and criterion (II). Criterion (I) is met in the first figure but not in the second one. Criterion (II) is satisfied in both.

An important question: given the Bellman error ε, and the initial state s_0, if criterion (II) is met at every step of execution, and if we let $V^{a_L}(s_0)$ denote the expected reward of the entire process, then does $V^*(s_0) - V^{a_L}(s_0) \le \varepsilon$ always hold? That is, will the error at each step be accumulated? Let us consider such an example, as shown in figure 2.

Let $\varepsilon = 1$, it is easy to give such initial upper and lower bounds:

$$\begin{cases} V_U(s_0) = 0, V_L(s_0) = -9 \\ V_U(s_1) = -8, V_L(s_1) = -100 \end{cases}$$

By using equation (4), we arrive at:

$$\begin{cases} Q_U^{a_1}(s_0) = -1, Q_L^{a_1}(s_0) = -10 \\ Q_U^{a_2}(s_0) = -9, Q_L^{a_2}(s_0) = -101 \end{cases}$$

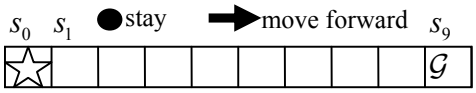


Figure 2. Initial state is at the most left, goal state is at the most right, action a_1 stay, action a_2 move forward one grid, reward of any action is -1.

Therefore, in the initial state s_0 , criterion (II) is met and the first action can be 'stay', and then everything unchanged, and it will repeat forever. Obviously, such a policy is not ε optimal. Essentially speaking, the reason is that from the second step onwards, the action performed is not consistent with the policy corresponding to the lower bound of the previous step. Furthermore, in this case, it is certainly that the initial lower bound does not satisfy the following condition.

Definition 2 If for every $s \in S$:

$$V_L(s) \le \max_a [R(s,a) + \sum_{s' \in S} T^a(s,s') V_L(s')] ,$$

the lower bound is **monotonic** [8].

When this condition is satisfied, if the agent performs the action solved by equations (4) and (5) with the lower bound, then the expected reward will not be below the lower bound in any state. If this condition is not satisfied, there will be no such guarantee. The example above explains such case. Then we have the following theorem that can deal with the stopping problem in a real-time application.

Theorem 3 Given MDPs problems satisfying additional restrictions:

(S1) There exists at least one optimal policy, and by following all the optimal policies the problem can terminate in finite steps.

(S2) The lower bound is monotonic.

If a policy π_d is obtained dynamically in such an incremental way that, at each step, suppose current state is s , when $V_U^{-a_L}(s) - V_L(s) \le \varepsilon$ is satisfied, let $\pi_d(s) = a_L$ and the action a_L is performed immediately, this process continues until the problem terminates. Then π_d is ε optimal.

Proof. Suppose the action performed at each step is consistent with the optimal policy, S1 guarantees the process will terminate in finite steps with the result that $V^* - V^{\pi_d} = 0$. Otherwise, suppose that in a certain step, the action performed deviates from the optimal policy, we know $V^* - V_L \le \varepsilon$ from theorem 1, and S2 guarantee that, the policy π_d which can be obtained will not be worse than π_L , that is, $V^* - V^{\pi_d} \le \varepsilon$.

Notice that there is no restriction of a finite state space. In other words, it can be extended to the POMDP problems with an infinite belief state space.

4 Bounded Incremental RTDP

4.1 Branch-Selection Strategy

In the process of a real-time dynamic programming using both upper and lower bounds, criterion (I) indicates the uncertainty $\widehat{V}$ of the initial state is decided by V_U and V_L . And equations (4) and (5) indicate that each of them correspond to the action value function Q_U^a , Q_L^a which are decided by a group of successor states. Consequently, to decrease $\widehat{V}$, the next branch should be selected among this group of successor states. According to the greedy policy, the most direct way is to select it from the outcome states corresponding to a_U and a_L . For this problem, FRTDP selects a_U for the following reasons: updating a lower bound will only increase it. If the problem has a tree structure, the other actions' lower bounds will never be updated. As a result, the process will select this a_L at the same state forever and be impossible to find better policy any more. Selection of a_U has no such a drawback. Things are different in graph structure, however, we follow the greedy policy in this paper but introduce a new branch-selection strategy based on criterion (II).

With regard to $\widehat{V}$, what is described in criterion (II) can be referred to as action uncertainty, denoted as $\widehat{V^a}$: $\widehat{V^a} = V_U^{-a_L} - V_L$. In the same vein, to decrease $\widehat{V^a}$, the greedy action selected should be the one that has the maximum upper bound of the action value function exclusive of a_L , denoted as a_U^{-L} . Actually, this process is engaged in proving the optimistic expected reward of a_U^{-L} is not ε precision greater than the pessimistic expected reward of a_L . Furthermore, selection of a_U^{-L} results in a decrease of its upper bound as well as an increase of its lower bound, thus, the other actions all have a chance to be selected in the future.

Notice that such a selection policy is only applied to the initial state. In the following successor states, we still select a_U according to criterion(I) for the purpose to eliminate the uncertainty of its parent node, the selection of a_U^{-L} doesn't have such a direct effect.

The other problem involved here is the monotony of lower bound. Theorem 2 requires the lower bound to be monotonic to guarantee ε optimal. In fact, even

without a monotonic lower bound, there exists a solution. We can denote an action $a_?$ corresponding to the lower bound value function V_L, meaning that we have no knowledge of the action a_L. When:

$$V_L(s) > \max_a [R(s,a) + \sum_{s' \in S} T^a(s,s')V_L(s')],$$

actually, the action with maximum lower bound is $a_?$, the lower bound will not be updated. In this case, selection of $a_U^{-?}$ is same as a_U. This agrees with criterion (I). Furthermore, without a decided lower bound action, the process will not give a result action that can be performed which guarantees ε optimality at current step. In any case, when the lower bound value function can be updated, the monotony is satisfied synchronously.

4.2 Real-Time Design

Bounded Incremental RTDP is based on FRTDP but employs a new branch-selection strategy that places more emphasis on real-time behavior.

At each step, when the algorithm obtains a solution meeting the precision requirement or times out, BIRTDP will return an action. After the execution, the outcome will be observed and BIRTDP continues with the next step of decision-making. The pseudocode is presented in algorithm 1. The remainder of this section will focus on explaining the differences to FRTDP. Full details on FRTDP can be found in [10].

Algorithm1 BIRTDP:

```
Function trialRecurse(s, w, depth)
  if isTerminal(s, depth) or TIMEOUT() then
    return end if
  trackUpdateQuality(s ,w, depth)
  a* ← chooseGreedyAction(s, depth)
  s' ← chooseMaxPriSuccessor(s, a*)
  trialRecurse(s', γT^a*(s,s')w ,depth+1)
  update(s)

Function BIRTDP(s_0, ε)
  while V_U^{-a_L} − V_L > ε
    if TIMEOUT() then break end if
    trialRecurse(s_0 ,w=1,depth=0)
    adjustTrailDepth()
  end while
  return a_L
```

```
Function Run(ε)
  (max_depth, s) ← (INIT_DEPTH, INIT_STATE)
  while !isTerminal(s)
    a* ← BIRTDP(s, ε)
    s ← execute(a*)
    max_depth ← max(INIT_DEPTH,
    max_depth − 1)
  end while
```

1) *chooseGreedyAction()* selects a_U^{-L} in the depth corresponding to the current step. When the lower bound is not monotonic or planning with the future steps, it selects a_U.
2) FRTDP employs an adaptive depth adjust mechanism. When the update effect is not obvious in the current *max_depth*, the parameter *max_depth* will be increased. However, after an execution in the real-time environment, it should be adjusted accordingly.
3) In a hard real-time system, time out will also result in the return of *BIRTDP()* and *trialRecurse()*.
4) *execute()* returns the outcome state observed.

5 Experiments

Our experiments are based on the RaceTrack benchmark problem [7]. In this problem, a car is driven from the initial state to the goal state in a grid map with obstacles. A state is composed of a 2-dimensional position and a 2-dimensional speed in a vector of the form $(x, y, \dot{x}, \dot{y})$. The possible actions accelerate the car and are described by a vector of the form $(\ddot{x}, \ddot{y})$, where $\ddot{x}$ and $\ddot{y}$ are integer valued in the set {-1,0,1}. When the car accelerates, there is a probability of slipping resulting in an acceleration vector (0,0). If the car encounters an obstacle, it will be reset to the initial state with a zero speed. Any action in any state will result in an immediate reward of -1. When the car arrives at one of the goal states, the process will terminate.

Our experiments use the standard maps small-b and large-b. For convenience of the real-time test, we select a fixed initial state. Suppose the bottom-left corner of the map is denoted as (0,0), small-b uses an initial state at (1,5), and large-b uses (1,1). The goal states are left unchanged. In addition, we use two other maps named small-b-m with an added goal state at (7,15) in small-b and to large-b-m add several goal states from (33,11) to (33,17) by a side of the middle

raceway in large-b. The probability of slip is uniformly set to 0.1 as the problem default. The computational environment was a Windows XP, AMD64 3200+ CPU, 1G RAM.

5.1 Experimental Results

There are three parts to our experiments. The first one is off-line test in which BIRTDP is compared with LRTDP, HDP, and FRTDP. The other experiments are integrated real-time tests with two typical real-time simulation environments. The emulator simulates the execution, returns the stochastic outcome to the algorithm, and judges whether the process terminated. As a consequence of the need to do some slight modification of the algorithms to be tested in the simulation environments, we only compared BIRTDP with FRTDP, which was the state-of-the-arts RTDP algorithm that applies both upper and lower bounds. All these experiments use a uniform initial upper bound set to 0, and a lower bound set to -1000. In real-time tests, before the process terminated, the computational results of the upper and lower bounds are retained for reuse. When arriving at the goal state, they are cleaned.

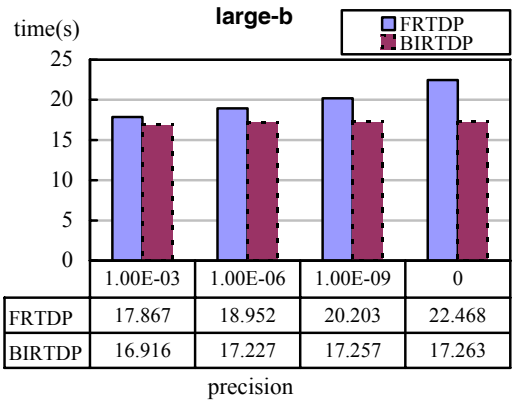
Table 1. Off-Line Test

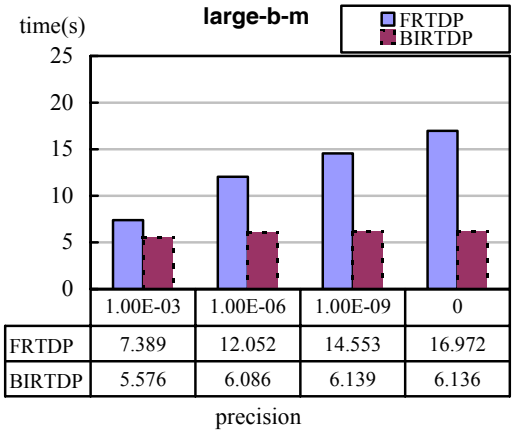
Algorithm	Time(s)	Reward	Nodes
Small-b(9312s,9a)			
HDP	1.9415	-13.2643	8690
LRTDP	1.6287	-13.2645	9110
FRTDP	1.3917	[-13.2647,-13.2637]	9130
BIRTDP	**1.36925**	**[-13.2793,-13.1941]**	**9122**
Small-b-m(9312s,9a)			
LRTDP	1.72150	-5.4374	7735
HDP	1.43900	-5.4374	7770
FRTDP	0.97700	[-5.4377,-5.4367]	8036
BIRTDP	**0.34050**	**[-5.4602,-5.4140]**	**4376**
Large-b(23880s,9a)			
HDP	27.45700	-24.4460	36075
LRTDP	24.67650	-24.4295	35886
FRTDP	17.18850	[-24.4478,-24.4468]	35251
BIRTDP	**16.18300**	**[-24.4752,-24.4434]**	**34821**
Large-b-m(23880s,9a)			
LRTDP	24.84600	-8.5254	34911
HDP	17.32250	-8.5249	34722
FRTDP	7.15100	[-8.5262,-8.5253]	25363
BIRTDP	**5.30550**	**[-8.5500,-8.5225]**	**20250**

1) Off-Line Test: The objective is to compare performance in such a way that all algorithms return the policy for the first step with quality guarantee. The new algorithm only solved the action for the first step with the guarantee that an ε optimal policy still remained and the other algorithms solved the whole policy in their original ways. With ε set to 0.001, the result is shown in table 1. The last column is nodes expended by the algorithms.

2) Soft Real-Time Test [15]: When the process began, the agent could execute action in anytime, and after the execution it could continue the decision-making. The only requirement was to achieve the goal as soon as possible. Each of the tests were performed on large-b and large-b-m 500 times automatically and averaged. We recorded only the total decision time used and excluded the execution time which was measured in simulation steps, because the execution time were nearly the same under an ε smaller than 0.001. The result is shown in table 2.

Table 2. Soft Real-Time Test

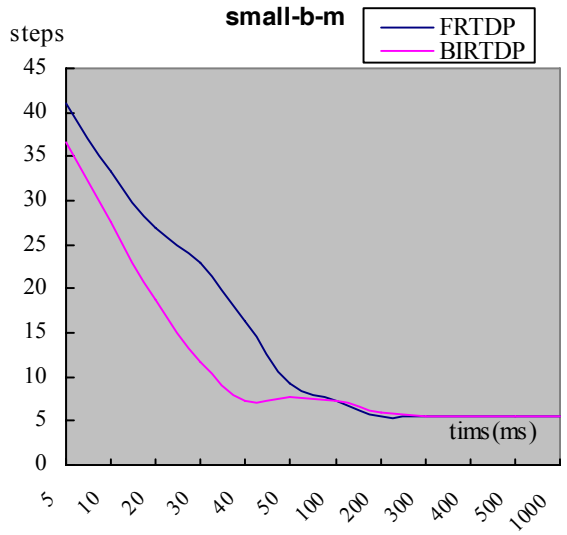




3) Hard Real-Time Test [15]: There is time restriction for each decision step. The agent must make a decision and perform an action in each time slice. The hard real-time test is performed on small-b-m. Each test was repeated 1500 times. Because all action rewards are -1, we record the average simulation steps

used to achieve the goal instead. The result is shown in table 3.

Table 3. Hard Real-Time Test



5.2 Discussions

In the off-line test, BIRTDP did outperform the other algorithms, but not achieved our expected effect. An analysis reveals that this is principally caused by the graph structure and the existence of a loop. For instance, suppose an action translates the current state into itself with probability 1 and receives a negative immediate reward $-R$, until the upper and lower bounds of the current state satisfy $V_U - V_L \le \varepsilon + R$, criterion (II) will not be met. In such case, whichever criterion is used, it is necessary to solve almost all of the problem.

In the real-time tests, our results demonstrate that BIRTDP has better real-time characteristics. The results of soft real-time test show that BIRTDP outperforms FRTDP most dramatically in the following two circumstances: (1) Environments that have a much larger state space compared to the region from the initial state to the goal state. (2) Solutions that require much higher quality. In fact, these tests demonstrate that BIRTDP finds the optimal policy without the exact value function for each step. The results of hard real-time test show BIRTDP has a better transient response characteristic. For instance, when the decision time for each step was restricted to 30ms or 40ms, the quality of solution nearly doubled, and BIRTDP required significantly less decision time given for each step to obtain an approximate optimal solution. Such results are due to the new branch-selection strategy that computes a better action at the current step with limited time. A more intuitive explanation is that, compared with the other branch-selection strategies, it does not directly care about how good the policy is, but rather, which action performed at current step will not eliminate the chance of getting a satisfying result.

6 Conclusion

In this paper, we have proposed a new criterion based on the upper and lower bound to deal with the stopping problem in the real-time Markov decision process. With this criterion satisfied, there is a quality guarantee to stop decision-making and execute the policy at current step without solving the whole remainder policy. As a result, in the soft real-time system that does not appoint a deadline for decision-making, this criterion usually helps the agent confirm an action to be performed earlier at each step. The direct benefit is that the subsequent observation will eliminate the outcome uncertainty and some of the related computation can be avoided. However, we point out that the lower bound must be monotonic to guarantee that the error will not accumulate when performing the action corresponding to the lower bound at each step. In such an incremental manner, one can obtain an optimal policy with arbitrary precision over the whole process. In hard real-time systems with a restriction of decision-making time for each step, if the same branch-selection strategy is applied but neither of the two criteria mentioned in this paper can be met before the deadline at each step, then irrespective of the criterion used, the solution quality will be the same. We have introduced a new branch-selection strategy that focuses the computation on solving the partial policy related to current step. This branch-selection strategy is more efficient both in soft and hard real-time systems. The principles proposed in this paper are simple and can be readily integrated into RTDP methods. We present the BIRTDP algorithm accordingly.

BIRTDP outperforms the other tested algorithms in two typical real-time simulation systems. The results also show that, with quality guarantees, BIRTDP is affected less by irrelevant states, and is less sensitive to the high precision requirement. Furthermore, in the hard real-time test with a strict restriction on decision time, BIRTDP also shows a better transient response characteristic.

Acknowledgements

Thanks to the group members Zhiwei Song, Jianmin Ji, Feng Wu, Wensong Wei, and Yuanmi Chen for discussions. Many thanks to Benjamin Johnston and the anonymous reviewers for their constructive comments on this paper. The authors are also grateful to Trey Smith for the publication of the ZMDP software package.

References

[1] Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. Journal of Artificial Intelligence Research, 11: 1-94, 1999.

[2] Nilsson, N. J. (1980) Principles of Artificial Intelligence, Tioga Publishing, Palo Alto, CA, 1980.

[3] Hansen, E. A., and Zilberstein, S. (2001). LAO*: a heuristic search algorithm that finds solutions with loops. Artif. Intell., 129, 35-62.

[4] Bonet, B. and Geffner, H. (2003). 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.

[5] Dean, T., Kaelbling, L. P., Kirman, J., and Nicholson, A. (1995). Planning under time constraints in stochastic domains. Artif. Intell., 76, 35-74.

[6] Ferguson, D., and Stentz, A. T. (2004). Focused dynamic programming: Extensive comparative results (Technical Report CMU-RI-TR-04-13). Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[7] Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. Artif. Intell., 72, 81-138.

[8] Bonet, B., and Geffner, H. (2003). Labeled RTDP: Improving the convergence of real-time dynamic programming. In Proc. of ICAPS-03 (pp. 12-21).

[9] 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.

[10] Trey Smith and Reid Simmons. (2006). Focused Real-Time Dynamic Programming for MDPs: Squeezing More Out of a Heuristic. In Proc. of AAAI, 2006.

[11] Pemberton, J. C., and Korf, R. E. (1994). Incremental Search Algorithms for Real-Time Decision Making. In Proc. of AIPS-94 (pp. 140-145).

[12] Puterman, M. (1994). Markov decision processes. John Wiley and Sons, New York, 1994.

[13] Trey Smith and Reid Simmons. (2004). Heuristic Search Value Iteration for POMDPs. In Proc. of UAI-04.

[14] Chernov, A., and Schmidhuber, J. (2006). Asymmetric Planning for Incremental Real-Time Heuristic Search. Technical Report No. IDSIA-15-06.

[15] Liu, C., and Layland, J. W. (1973). Scheduling Algorithms for Multiprogramming in A Hard-Real-Time Environment. In Journal of the ACM 20(1): 46-61, 1973.