Abstract—Bounded-parameter Markov decision process (BMDP) can be used to model sequential decision problems, where the transitions probabilities are not completely known and are given by intervals. One of the criteria used to solve that kind of problems is the maximin, i.e., the best action on the worst scenario. The algorithms to solve BMDPs that use this approach include interval value iteration and an extension of real time dynamic programming (Robust-LRTDP). In this paper, we introduce a new algorithm, named \( B^2 \)RTDP, also based on real time dynamic programming that makes a different choice of the next state to be visited using upper and lower bounds of the optimal value function. The empirical evaluation of the algorithm shows that it converges faster than the state-of-the-art algorithms that solve BMDPs.

I. INTRODUCTION

Markov Decision Process (MDPs) [1] provides a mathematical framework for modeling the activity of sequential decision making, for example in the areas of planning, operations research and robotics. An MDP models the interaction between an agent and its environment in \( t \) stages-to-go. At each stage \( t \) the robot or software agent makes a choice of an action that have probabilistic effects and decides to perform it producing a future state and a reward. The goal of the agent is to maximize a value function, that can be the expected value of the sum of discounted rewards over a sequence of choices.

Several algorithms, that use dynamic programming, have been proposed to solve MDPs. One classic algorithm is the value iteration [1] that updates the value of all states at each iteration. When the initial state is known, we can use a more efficient solution named Real-Time Dynamic Programming (RTDP) [2] and its extensions Labeled Real-Time Dynamic Programming (LRTDP) [3], Bounded Real-Time Dynamic Programming (BRTDP) [4], Focused Real-Time Dynamic Programming [5] and Bayesian Real-Time Dynamic Programming [6].

However, known extensions of an MDP are more suitable to represent practical problems of great interest for real applications in special an MDP whose probabilities are not completely known and where constraints over probabilities are defined, called Markov Decision Process with Imprecise Probabilities (MDP-IP) that has been proposed in the 1970’s [7]. A particular case of MDP-IP has been proposed in the late 1990’s, called Bounded-parameter Markov Decision Process (BMDP) [8]. A BMDP is an MDP in which the transition probabilities and rewards are defined by intervals. Since the transitions in both of these problems are imprecise, there are infinite models of probabilities to choose.

There are various criteria to evaluate a policy for both MDP-IP and BMDP. One of them is the maximin criterion, which considers the best scenario in the worst case. To solve an MDP-IP, the main computational bottleneck is the need to repeatedly solve optimization problems in order to consider the worst case probabilities [9]–[11]. To solve a BMDP we can explore an additional information: the structure of the intervals. Using that information we can avoid calls to an optimizer by applying a greedy method for choosing the probabilities. The interval value iteration is an algorithm [8] that uses this greedy method to solve BMDPs. There is also a solution for BMDPs that includes an extension of LRTDP, named Robust-LRTDP [12].

In this work, we propose a new algorithm for solving a BMDP based on the BRTDP algorithm [4], named \( B^2 \)RTDP, which converges faster than the Robust-LRTDP by making a better choice of the next state to be visited.

In Section II we introduce the main concepts related to an MDP, we present the definition of a BMDP and existing solutions. In Section III we present a new algorithm for solving BMDPs that uses upper and lower bounds of the value function to do a better choice of the next state to be visited and to verify convergence. In Section IV we evaluate the proposed algorithm in terms of convergence time. Finally, in Section V we show the conclusions.

II. BACKGROUND

A. Markov Decision Process

Formally an MDP is defined by the tuple \( M = (S, A, p, r, \gamma) \), where:

- \( S \) is a finite set of observable states;
- \( A \) is a finite set of actions that can be executed by the agent;
- \( p(s'|s, a) \) is the transition function that represents the probability that the future state be \( s' \) given that the action \( a \) was applied in the state \( s \);
- \( r \) is a function that associates a reward to each state;
\(\gamma\) is the discount factor such that 0 \(\leq \gamma < 1\).

In an MDP with infinite horizon, the agent performs actions in an infinite number of stages. Thus, the solution for an MDP is a policy, a function that maps states into actions \((\pi : S \rightarrow A)\). The goal is to find an optimal policy \((\pi^*)\) that maximizes the value function over states defined as the sum of the expected discounted rewards obtained from the initial state \(s_0 = s\), in an infinite horizon following actions \(\pi(s)\):

\[
v^\pi(s) = E_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t \cdot r_t | s_0 = s \right]. \tag{1}\]

To find \(\pi^*(s)\), we use the optimal value function over states given by the Bellman Equation \([1]\), i.e.:

\[
v^*(s) = \max_{a \in A} \{ r(s) + \gamma \sum_{s' \in S} p(s'|s,a)v^*(s') \}. \tag{2}\]

Thus, an optimal policy \(\pi^*(s)\) w.r.t. \(v^*(s)\) can be found by the equation:

\[
\pi^*(s) = \arg\max_{a \in A} \{ r(s) + \gamma \sum_{s' \in S} p(s'|s,a)v^*(s') \}. \tag{3}\]

The classic algorithm for solving MDPs is the value iteration algorithm \([1]\), which updates the value function for all states \(s \in S\) at each iteration \(t\) (i.e., a synchronous algorithm) by the following equations:

\[
Q^{t+1}(s,a) = \{ r(s) + \gamma \sum_{s' \in S} p(s'|s,a)v^t(s') \}, \tag{4}\]

\[
v^{t+1}(s) = \max_{a \in A} \{ Q^{t+1}(s,a) \}, \tag{5}\]

where \(Q^{t+1}(s,a)\) is the function that associates a real value for the quality of using action \(a\) in state \(s\). It was proved that \(v^{t+1}(s)\) converges to the optimal value function \(v^*(s)\) when are performed infinite updates. The greedy policy with respect to \(v^{t+1}\) can be found by:

\[
\pi(s) = \arg\max_{a \in A} \{ Q^{t+1}(s,a) \}. \tag{6}\]

Note that if the set of states is relatively large then the execution time of this algorithm is impractical \([2]\).

In an special case of MDP, named Stochastic Shortest Path (SSP) it is also given an initial state \(s_0 \in S\) and a possibly empty set of goal states \(G \subset S\), where every action applied to \(s \in G\) leads to the same state \(s\) with probability 1.

RTDP \([2]\) is a synchronous algorithm for solving MDPs that differently from the value iteration algorithm does not perform the computation of the value function for all states in each iteration, but only for the states reachable from the initial state by the greedy policy. It updates the states encountered during simulations based on sampling, called trials. RTDP chooses the next states to be visited by sampling them based on the transition probability, i.e.:

\[
s' \sim p(\cdot|s,a). \tag{7}\]

This algorithm has two advantages over the value iteration algorithm: (i) it returns a good policy at any time; and (ii) it returns an optimal result without exhaustive exploration. However, one of its shortcomings is that the unlikely states are ignored and therefore the convergence to the optimal value becomes slow \([3]\). The disadvantage of RTDP comes from the use of the greedy exploration method that selects the states with higher probability more frequently. Thus, this approach makes the most impactful updates; but in the search for the optimal solution, the path with lower probability also needs to be updated \([3]\).

Due to the convergence problem presented by the RTDP algorithm, some extensions were proposed, including: Labeled Real-Time Dynamic Programming (LRTDP) \([3]\) and Bounded Real-Time Dynamic Programming (BRTDP) \([4]\).

LRTDP uses a methodology for labeling states that have already converged to avoid revisiting them. An state \(s\) is labeled as solved (converged) if all reachable states from \(s\) in a depth-first search, using the best actions, have a residual error smaller than a defined tolerance, \(\epsilon\). The states that have converged are no longer updated and then the states that have not yet converged are prioritized \([3]\).

With the same motivation, i.e., to decrease the convergence time to the optimal solution, BRTDP algorithm have two value functions, \(v_L\) (\(v\) lower) and \(v_U\) (\(v\) upper), initialized with lower and upper bounds for the value function, respectively. The two value functions are updated for each state visited during a trial and the convergence of a state is measured by the difference between \(v_L\) and \(v_U\). These two bounds must converge to the same value in infinite steps \([4]\). Using this information BRTDP focuses the search on areas where there is greater uncertainty in the state values adjusting the probability to prioritize them and thus it converges faster \([4]\).

### B. Bounded-Parameter Markov Decision Process

A BMDP is an extension of an MDP, where the transition probability between states, rather than being defined by a real number, is defined by a range of real numbers. This imprecision in probabilities can be used to: (i) represent the lack of information of the proposed model; or (ii) reduce the number of states of an MDP that has a large number of states \([8]\).

In the example of the transition diagram given in Figure 1, arcs are labeled by actions and transition probabilities; nodes are labeled by states and rewards. In this diagram we have states \(s_0, s_1\) and \(s_2\) with their rewards; and actions \(a_0\) (dashed line), \(a_1\) (solid line) and \(a_2\) (dotted line) with their transition probabilities. For example, given that the agent is in state \(s_0\), the probability of going to state \(s_1\) by applying an action \(a_1\) is defined by the interval \([0.4 \ 0.6]\).

Formally a BMDP is defined by the tuple \(M = (S, A, P, r, \gamma)\), where \([8]\):

- \(S\) is a finite set of observable states;
- \(A\) is a finite set of actions that can be executed by the agent;
- \(P(s'|s,a)\) is the transition probability function between states given by intervals
Fig. 1. Example of a transition diagram for a BMDP.

\[ [p_{\text{min}}(s'|s,a), p_{\text{max}}(s'|s,a)] \]. These intervals are closed and \( 0 \leq p_{\text{min}}(s'|s,a) \leq p_{\text{max}}(s'|s,a) \leq 1 \). To ensure that the intervals of transition probabilities have well formed values it is necessary that (i) the sum of \( p_{\text{min}}(s'|s,a) \) for all successor states for a pair (state,action), be less than or equal to 1; and (ii) the sum of \( p_{\text{max}}(s'|s,a) \) for all successor states for a pair (state,action) be greater or equal to 1.

- \( r \) is a function that associates a reward (real value) to each state;
- \( \gamma \) is the discount factor such that \( 0 \leq \gamma < 1 \), when solving a stochastic shortest path problem, we adopt \( \gamma = 1 \).

Since the transition probability between states is defined by an interval, there are infinite models, i.e., we can choose innumerable precise probability values within the intervals of probability such that the sum of the transition probabilities for all successor states for a pair (state,action) be equal to 1.

It is possible to define various criteria to evaluate a policy. In this work, we are interested to find the best action in the worst case scenario (maximin criterion or also known as robust solution).

For BMDPs that adopt this criterion it has been proved that there is a deterministic stationary policy whose value function is the solution of the Bellman equation [8], given by:

\[
v^*(s) = \max_{a \in A} \min_{p \in P} \{ r(s) + \gamma \sum_{s' \in S} p(s'|a,s)v^*(s') \},
\]

where \( p \) is a transition probability measure that is belong to the probability interval \( P \). In this equation, the agent chooses the action that maximizes the value \( v(s) \) in the worst case scenario (i.e., when the probability that minimizes \( v(s) \) was chosen).

a) Robust value iteration.: In the robust value iteration algorithm [8], the value function is updated for all states in each iteration:

\[
v^{t+1}(s) = \max_{a \in A} \{ Q^{t+1}(s,a) \},
\]

where \( Q^{t+1} \) is given by:

\[
Q^{t+1}(s,a) = \min_{p \in P} \{ r(s) + \gamma \sum_{s' \in S} p(s'|a,s)v^t(s') \}.
\]

To find the worst model (\( \min_{p \in P} \)), i.e., to find the probabilities that make the value for the pair (state,action) be the smallest possible, the method \text{WORSTMODEL} (Algorithm 1) is used. For this, the algorithm sorts the reachable states \( S' \subset S \) by the value function in ascending order and tries to assign valid higher probabilities for states with lower value function [8]. In this algorithm \( p_{i}^{\text{max}} \) means \( p_{\text{max}}(s_i'|s,a) \) and \( p_{i}^{\text{min}} \) means \( p_{\text{min}}(s_i'|s,a) \).

Algorithm 1 Worst Model for pair (state,action) [8]

1. procedure \text{WORSTMODEL}(s: state, a: action, v: value function)
2. \( S' = (s'_1, \ldots, s'_i) = s \).SortNextStates(a)
3. \( i = 1 \)
4. \( \text{limit} = \sum_{s' \in S'} p_{i}^{\text{min}}(s'|s,a) \)
5. while \( \text{limit} - p_{i}^{\text{min}} + p_{i}^{\text{max}} < 1 \) do
6. \( \text{limit} = \text{limit} - p_{i}^{\text{min}} + p_{i}^{\text{max}} \)
7. \( p(s'_i|s,a) = p_{i}^{\text{max}} \)
8. \( i = i + 1 \)
9. end while
10. \( j = i \)
11. \( p(s'_j|s,a) = 1 - (\text{limit} - p_{i}^{\text{min}}) \)
12. for all \( m \in \{j + 1, \ldots, k\} \) do
13. \( p(s'_m|s,a) = p_{m}^{\text{min}} \)
14. end for
15. end procedure

In the example of Figure 2, we show: the reachable states \( S' = \{s_1, s_2, s_3, s_4, s_5\} \) (from state \( s_0 \) applying action \( a_0 \)); the value function for each state; and the transition probability intervals. The \text{WORSTMODEL} method starts with the set of reachable states \( S' \) already ordered by the value function. The algorithm finds the state \( s_j \), such that for all states \( s_i \) with \( i < j \), \( p(s'_i|s,a) = p_{i}^{\text{max}} \) (line 7) and from \( m > j \), \( p(s'_m|s,a) = p_{m}^{\text{min}} \) (line 13). Note that for \( s_j \), it is not possible to assign \( p_{i}^{\text{max}} \) even though for the next states be assigned \( p_{\text{min}}^{m} \) (since this would result in a sum of probabilities greater than 1). In the example of Figure 2, since \( j \) is equal to 4, it is assigned \( p_{4}^{\text{max}} \) to \( s_1, s_2 \) and \( s_3 \) and for \( s_5 \) is assigned \( p_{5}^{\text{min}} \). For the state with index \( j \) it is assigned \( 1 - (\text{limit} - p_{i}^{\text{min}}) \) (line 11), in this example, we assign \( 0.18 \) to \( s_j = s_4 \). Note that \( \text{limit} \) is initialized with \( \sum_{s' \in S'} p_{i}^{\text{min}}(s'|s,a) \) (line 4).

b) Robust-LRTDP: The Robust-LRTDP algorithm [12] finds a robust optimal solution for BMDPs when \( s_0 \) and \( G \) are given. Robust-LRTDP calls the method \text{WORSTMODEL}.
(Algorithm 1) to find the worst possible model for each pair (state, action) considering a value function $v_u$ ($v$ upper). Likewise the LRTDP [3] algorithm, Robust-LRTDP increases the speed of convergence to the optimal solution using the technique of labeling converged states.

III. THE ALGORITHM $B^2$RTDP

Among the disadvantages of Robust-LRTDP algorithm are: (i) the state is only considered converged if all reachable states by the greedy policy also have converged; and (ii) all states not converged, have the same priority (proportional to their transition probability) to be visited during the trials.

To cope with the limitations of Robust-LRTDP algorithm, in this work we propose to use the ideas of BRTDP [4] to solve BMDPs, i.e., to find the best action in the worst case scenario. We call this solution $B^2$RTDP (Algorithm 2).

The $B^2$RTDP algorithm maintains upper and lower bounds of the optimal value function $v_u$ and $v_l$, defined as $v_u(s) \geq v^*(s), \forall s \in S$ and $v_l \leq v^*(s), \forall s \in S$, respectively. By computing the difference between $v_u$ and $v_l$ it is possible to prioritize the visit of states with a large amount of uncertainty in their values. The values $v_u$ and $v_l$ of states visited during a trial are updated using Equations 9 and 10. Like for the BRTDP [4] we inherit the property that successive updates of the upper and lower bounds preserve the admissibility and converge monotonically to $v^*(s)$, i.e.,

$$\lim_{t \to \infty} v^u_t(s) = v^*(s) = \lim_{t \to \infty} v^l_t(s). \quad (11)$$

In Figure 3 we show how $v_u(s)$ and $v_l(s)$ are supposed to converge monotonically to $v^*(s)$. Note that the difference between the upper and lower bounds can be used as a measure of uncertainty in the state value.

![Fig. 3. Successive updates of the upper and lower bounds preserve the admissibility of the estimation $v_u$ and $v_l$, and converge monotonically to $v^*(s)$.](image)

In the initialization step of the algorithm, the value function of the states, $v_u$ and $v_l$, are initialized with admissible upper and lower bounds, respectively. For example, for a BMDP with infinite horizon, we can use the highest and lowest possible value for the value function, computed as follows: $v_u = r_{\text{max}}/(1-\gamma)$ and $v_l = r_{\text{min}}/(1-\gamma)$, with $r_{\text{max}}$ and $r_{\text{min}}$ being the greatest and smallest reward for all states, respectively. Thus, at the beginning the difference $v_u - v_l$ is the greatest possible value for all states. After the initialization, several trials are executed (Algorithm 2, line 3). Each trial starts with the initial state, then $v_u$ and $v_l$ are updated for the states encountered during the simulations (Algorithm 3, line 6 and 8) and a greedy action w.r.t. $v_u$ is chosen (Algorithm 3, line 7). $B^2$RTDP prioritizes the choice of the next state (Algorithm 3, line 9) according to the transition probability weighted by the value $v_u - v_l$ using the SAMPLENEXTSTATEB procedure (Algorithm 4, line 15), where it is used $b(s')$, given by:

$$b(s') = p(s'|s,a)(s'.v_u - s'.v_l). \quad (12)$$

Note that to compute $b(s')$, we need a probability measure $p(s'|s,a)$ for each $s' \in S$, e.g. the one based on the worst model w.r.t. $v_u$. The next state is drawn using $b(s')/B$ (Algorithm 4, line 23), where $B$ is a normalization variable given by $B = \sum_{s' \in S} b(s')$ (Algorithm 4, line 20). At the end of the trial, the values $v_u$ and $v_l$ are updated in reverse order for all states visited during the trial (Algorithm 3, lines 11-15).

In the $B^2$RTDP, the end of a trial happens when we find a goal state or when all the values of the reachable states, using the greedy policy, have converged, i.e., the difference between $v_u$ and $v_l$ for all reachable states, using the greedy policy, is smaller than $\epsilon$ (Figure 3). For this, the normalization variable is compared to the difference between $v_u$ and $v_l$ of the initial state divided by a constant called adaptation criterion $r > 1$ (Algorithm 4, line 21).

It is important to note that the WorstModel method is called by the Update method (Algorithm 4, line 12) whenever Q is computed (Algorithm 4, line 7). Thus, $B^2$RTDP algorithm calls the WorstModel (Algorithm 1) when is going forward and backward in the trial, and for each of the value functions, $v_u$ and $v_l$. Since the computation of the worst model is costly, we also propose a variation of the $B^2$RTDP algorithm, called $B^2$RTDP-I, where only one call is made to the WorstModel method w.r.t. $v_u$ and use the same probabilities to compute $v_l$.

In the next section, we empirically verify that this algorithm still converges.

Algorithm 2

```python
1: procedure $B^2$RTDP($s$ : state , $\epsilon$ : float)
2: while $s.v_u - s.v_l > \epsilon$ do
3:    $B^2$RTDPTRIAL($s$)
4: end while
5: end procedure
```

Algorithm 3

```python
1: procedure $B^2$RTDPTRIAL($s$ : state)
2: $x = s$
3: traj = EMPTYSTACK
4: while $x = \text{null}$ or $s \in G$ do
5:    traj.PUSH($x$)
6:    $x,v_u =$x.UPDATE($v_u$)
7:    $a = x$.BESTACTION($v_u$)
8:    $x,v_l =$x.UPDATE($v_l$)
9:    $x = x$.SAMPLENEXTSTATEB($a$, $s$)
10: end while
11: while $\text{traj}.\text{EMPT}Y()$ do
12:    $x = \text{traj}.\text{POP}()$
13:    $x$.UPDATE($v_u$)
14:    $x$.UPDATE($v_l$)
15: end while
16: end procedure
17:
```
starting from the initial state, must choose the path to follow until reaching one of the terminal states marked with (+1) and (-1). In each state the possible actions to be taken are go-up, go-down, go-left and go-right. The probability that the action has the desired effect is 0.8, otherwise the action leads the agent to other direction (Figure 4 (b)). Moreover, when the agent chooses an action that leads to a wall, it remains in the same location. For example, from the state (2, 3), there is a probability of 0.8 that action go-right moves the agent to (3, 3) and a probability of 0.2 to remain in the same state (2, 3). Every time the agent reaches a state, it receives a reward of -0.04, except for the terminal states that have rewards 1 and -1.

To generate BMDPs problems, we have included probabilities defined by intervals, which models the problem more realistically. We have generated 10 problems with different grid sizes.

For the three algorithms, we use the acceptable error \( \epsilon = 1 \times 10^{-3} \). The value of the adaptation criterion used for the experiments was \( \tau = 10 \) for \( B^2RTDP \) and \( B^2RTDP-I \) algorithms.

In Figure 5, we compare the convergence time of the algorithms given in seconds for the 10 problems. We note that until the problem 40x40, \( B^2RTDP \) has a slightly worse performance than \( B^2RTDP-I \) and Robust-LRTDP. Note that while \( B^2RTDP \) makes four calls to the \( WORSTMODEL \) method, Robust-LRTDP only makes two, one for each state visited during a trial (forward) and one in reverse order (backward). We know that \( B^2RTDP \) focuses the search in areas where there is greater uncertainty in the value; however the time spent on calls to the \( WORSTMODEL \) method makes it inefficient. This was one of the reasons to propose a new version of the algorithm, \( B^2RTDP-I \), which reuses the probabilities computed during the update of \( v_u \) in the update of \( v_l \). Thus, we maintain the property of the algorithm that searches in areas of greatest uncertainty by performing only two calls to the \( WORSTMODEL \) method for each state visited during a trial. This is shown in Figure 5 where the performance of the \( B^2RTDP-I \) algorithm was almost equal to Robust-LRTDP until the problem 40x40 and an order of magnitude faster than the Robust-LRTDP for larger instances (50x50 to 80x80).

In Figure 6 we show how the value of the initial state \( (v_u(s_0)) \) is updated over the time until achieving convergence, for the 60x60 problem, using the Robust-LRTDP and \( B^2RTDP-I \) algorithms. Both algorithms start with the heuristic value 100. We note that both algorithms have similar behavior except when they are close to convergence. While Robust-LRTDP takes approximately 1000 seconds to converge, \( B^2RTDP-I \) converges in less than 200 seconds. Thus, we verified that \( B^2RTDP-I \), which updates the states with the greatest uncertainty in their values, converges 5 times faster than the Robust-LRTDP algorithm for the Navigation 60x60 problem.

In Figure 7 we show the number of updates performed by the Robust-LRTDP and \( B^2RTDP-I \) algorithms along the time for the Navigation 60x60 problem. The \( B^2RTDP-I \) algorithm performs more updates than Robust-LRTDP in the same amount of time. This confirms our hypothesis that the verification of convergence of Robust-LRTDP, that uses the labeling
procedure of converged states, can be computationally more expensive than the verification of convergence of B^2RTDP-I, which simply uses the difference between the values v_u and v_l.

V. CONCLUSION

The main contribution of this paper is to propose a new algorithm for solving Bounded-parameter Markov Decision Process, that maintains upper and lower bounds of the optimal value function in order to prioritize the search in areas where there are greater uncertainty on the state value function.

The best version of the proposed algorithm, B^2RTDP-I showed up to one order of magnitude speedup in comparison to the state-of-the-art algorithm, Robust-LRTDP, for the largest instances of the Navigation domain (50x50 to 80x80). Showing a trend to increase this factor for ever larger instances (Figure 5).

REFERENCES


