Point-based Incremental Pruning for Monte-Carlo Tree Search

Bo Wu, Yanpeng Feng
Education Technology and Information Center
Shenzhen Polytechnic
Shenzhen, China
wubo@szpt.edu.cn

***Abstract*—Monte-Carlo tree search (MCTS) combines the generality of stochastic simulation and the accuracy of tree search, which has attracted the great attention of scholars. However, the MCTS search requires a sufficient number of iterations to converge to a good solution, which is more difficult to optimize. In order to solve this problem, this paper presents a point-based incremental pruning (PIP) for Monte-Carlo tree search. Instead of reasoning about the whole policy trees space when pruning the cross-sums of the value functions during policy construction, our algorithm uses boundary belief points to perform exact pruning, and exploits intermediate points to perform approximate pruning by generating policy trees, then uses real-time belief states to get the optimal policy in policy execution. The theoretical analysis and empirical results indicate that PIP can speed up the tree search efficiently.**

***Keywords*-Partially observable Markov decision processes (POMPDs); Monte-Carlo tree search (MCTS); incremental pruning**

I. INTRODUCTION

Monte-Carlo Tree Search (MCTS) is a method for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results. It has already had a profound impact on Artificial Intelligence (AI) approaches for domains that can be represented as trees of sequential decisions, particularly games and planning problems [1]. The basic algorithm involves iteratively building a search tree has four steps applied per search iteration [2]. 1) Selection: Starting at the root node, a child selection policy is recursively applied to descend through the tree until the most urgent expandable node is reached. 2) Expansion: One (or more) child nodes are added to expand the tree, according to the available actions. 3) Simulation: A simulation is run from the new node(s) according to the default policy to produce an outcome. 4) Backpropagation: The simulation result is backed up (i.e. backpropagated) through the selected nodes to update their statistics.

The key idea is to evaluate each state in a search tree by the average outcome of simulations from that state. MCTS provides several major advantages over traditional search methods [3]. It has outperformed previous planning approaches in challenging games such as Go [4]. The fastest offline Partially observable Markov decision processes (POMDPs) solvers today are based on point-based approach [5]. This approach reduces the complexity of planning in the belief space B by representing B as a set of sampled beliefs and planning with respect to this set only. Moreover, Many algorithms have focused on getting exact solution by improving the efficiency of updating and pruning vectors or policy trees [6]. One of the fastest general online POMDP solvers today is Partially Observable Monte-Carlo Planning (POMCP) [3]. Starting from the current belief b, POMCP performs best first search in the belief space [7].

However, computing the optimal policy for large-scale MCTS is known to be intractable because of the exponential complexity. In order to solve this problem, we present a point-based incremental pruning (PIP) algorithm for MCTS.

The remainder of the paper is structured as follows: in Section II we review the POMDPs framework. In Section III we review the MCTS model. In Section IV we describe the PIP algorithm. Experimental results from RockSample problem are showed in Section V. Finally, we wrap up with some conclusions in Section VI.

II. POMDPS

Formally, a POMDP model is presented as a tuple (S, A, Z, T, O, R),its schematic diagram of sequential decision-making is described in Fig.1 as below [8].

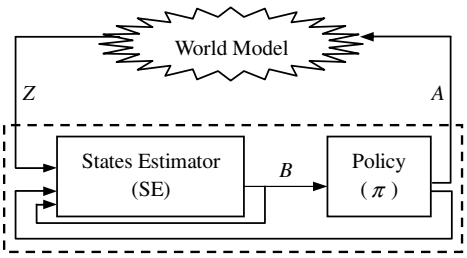

Figure 1. The model of POMDP

- $S = \{s_1, s_2, s_3, ..., s_n\}$ is the set of all the environment states, which are not directly observable. Agent can only compute a belief over the state space S.
- $A = \{a_1, a_2, a_3, ..., a_m\}$ is the set of all possible actions. Actions stochastically affect the states of the world.

978-1-5386-3013-6/17 $31.00 © 2017 IEEE
DOI 10.1109/ICISCE.2017.119

- $Z = \{z_1, z_2, z_3, ..., z_l\}$ is the set of all possible observations. Observation, contaminated by sensor noise, is usually an incomplete projection of the world state.
- T is the state transition probability distribution, $T(s_i, a, s_j) = P(s_j|s_i, a)$, represents the probability of ending in state s_j if the agent performs action a in state s_i. Since T is a conditional probability distribution, so $\sum_{s \in S} T(s_i,a,s_j) = 1, \forall (s, a)$.
- O is the observation probability distribution, $O(s_i, a_j, z_k) = P(z_k|s_i, a_j)$, is the probability that the agent will perceive observation z_k upon executing action a_j in state s_i. Since O is a conditional probability distribution, so $\sum_{z \in Z} O(s_i,a,z_k) = 1, \forall (s, a)$.
- R is the reward function, $R(s, a) : S \times A \times Z \mapsto$, is the reward obtained by executing action a in state s. The objective of POMDP planning is to optimize action selection. The goal of agent is to collect as much reward as possible over time.

We assume in this paper that R is bounded, S, A, Z are finite. In this paper, unless otherwise specified, superscript represents time (t) and subscript stands for the specific variables. Such as, s_i represents the ith state in the set of S, s^t represents the state at time t, $P(s^t = s_i)$ represents the probability when state is s_i at time t. Sometimes, if necessary, the current state is denoted s and the next state is denoted s'. For instance, s represents the *current* state, and s' represents the *next* state.

III. MONTE-CARLO TREE SEARCH

Monte-Carlo tree search (MCTS) uses Monte-Carlo simulation to evaluate the nodes of a search tree [9]. The search tree T contains one node, $n(s)$, corresponding to each state s that has been seen during simulations. Each node contains a total count for the state, $N(s)$, and an action value $Q(s, a)$ and count $N(s, a)$ for each action $a \in A$. Simulations start from the root state s^0, and are divided into two stages. When state s^t is represented in the search tree, $s^t \in T$, a tree policy is used to select actions. After each simulation s^0, a^0, s^1, a^1, ... , s^T with outcome z, each node in the search tree, $\{n(s^t) \ | s^t \in T \}$, updates its count, and updates its action value $Q(s^t, a^t)$ to the new MC value. This update can also be implemented incrementally, without reconsidering previous simulations, by incrementing the count and updating the value towards the outcome z [9]. The MC value can be computed as follows.

$$Q(s, a) = \frac{\sum_{i=1}^{N(s)} I^i(s, a) z^i}{N(s, a)} \tag{1}$$

where z^i is the outcome of the ith simulation; $I^i(s, a)$ is an indicator function returning 1 if action a was selected in state s during the ith simulation, and 0 otherwise. The updating processes can be computed as follows.

$$N(s^t) \leftarrow N(s^t) + 1 \tag{2}$$

$$N(s^t, a^t) \leftarrow N(s^t, a^t) + 1 \tag{3}$$

$$Q(s^t, a^t) \leftarrow Q(s^t, a^t) + \frac{z - Q(s^t, a^t)}{N(s^t, a^t)} \tag{4}$$

The partially observable Monte-Carlo planning (POMCP) algorithm [3] is an online planning method that extends the Monte-Carlo tree search (MTCS) method to partially observable Markov decision processes (POMDPs). Each simulation starts from a state sampled from the belief, chooses actions based on the multi-armed bandit algorithm UCB1. If child nodes for all actions of the current node are added into the tree, actions are selected to maximize an upper confidence bound on the action value.

$$Q^{\oplus}(s, a) = Q(s, a) + c\sqrt{\frac{\log N(s)}{N(s, a)}} \tag{5}$$

$$a^* = \arg\max_a Q^{\oplus}(s, a) \tag{6}$$

where c is a bias parameter which defines the proportion of exploitation and exploration. If $c = 0$, the UCB1 policy becomes a greedy policy.

IV. INCREMENTAL PRUNING FOR MONTE-CARLO TREE SEARCH

A. Policy Tree

Policy tree [10] is constituted by actions and observations, a t-step policy tree can be constructed by the (t-1)-step policy tree. Each node of the policy tree represents the action that should be taken if it is reached. Each branch represents the observation obtained after taken the action. If P is a t-step policy tree, then the value of executing in state s is:

$$V_{p^t}(s) = R(s, a_i) + \gamma \sum_{s' \in S} T(s, a, s') \cdot \sum_{z_i \in Z} O(s', a, z_i) V_{p_{a_i, z_i}^{t-1}}(s') \tag{7}$$

The value of executing a policy tree from some belief states can be represented by:

$$V_p(b) = \sum_{s \in S} b(s) V_p(s). \tag{8}$$

Let $\alpha_p = \langle V_p(s_1), V_p(s_2), \cdots V_p(s_n) \rangle$, then $V_p(b) = b \cdot \alpha_p$. According to the definition of policy trees, we could construct several different t-step policy trees by changing the combination of a and z. Different t-step policy trees have same level and structure, but different root nodes and branch nodes. Let P be the set of all t-step policy trees, then,

$$V_t(b) = \max_{p \in P} b \cdot \alpha_p. \tag{9}$$

It can be seen that $V_t(b)$ is piecewise-linear and convex. A t-step policy tree is a set of policy trees with different

combination of actions. We represented the set of the t-step policy tree by $P_t = \bigcup_{a \in A} p_t^a$, and the value set can be represented by $\bigcup_{p_i \in P} V_{p_i}(b)$. For all belief states b in B, we calculate $V_{p_i}(b)$ by (7) and (8), and get the optimal policy tree by $V_t(b) = \max_{p_i \in P} V_{p_i}(b)$.

As we know the policy tree can be generated by iteration, the root has $|A|$ choices, $|A|$ is the total number of the elements in action set. So there are $|A|$ policy trees in layer 0, $|A||A|^{|Z|}$ policy trees in layer 1, and $|A|^{1+|Z|+|Z|^2+\cdots+|Z|^t}$ policy trees in layer t. The number of t-step policy trees grows exponentially with $|Z|$. Thus, the policy trees must be pruned to solve large-scale POMDPs.

As mentioned above, the number of the t-step policy trees grow exponentially with $|Z|$. The t-step policy tree is composed of a root a, Z branches, and a $(t-1)$-step policy tree. Suppose the number of $(t-1)$-step policy trees is $|P_{t-1}|$, then the number of the t-step policy trees is $|A| \cdot |P_{t-1}|^{|Z|}$. As the scales of $|A|$ and $|Z|$ are certain in a given POMDPs model, we prune the policy trees in the process of policy trees generation, then get the optimal policy at some certain b.

To reduce the size of the t-step policy trees, we apply the point-based theory into the policy trees pruning, and propose the Point-based Incremental Pruning (PIP) algorithm. According to the different selections of the belief points.

B. point-based incremental pruning

When the scale of observation is large, we use the fuzzy distance-based belief selection method to generate belief points set, then use these belief points to prune the policy trees. Suppose the t-step policy trees set is P_t, each member is $p_{t,i}$, the initial belief point set is B. Each belief points in B construct N policy trees with root $\{a_1, a_2, ..., a_n\}$, as shown in Fig. 3.

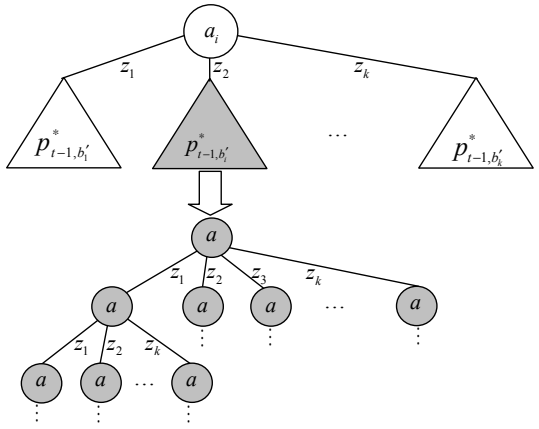

Figure 2. Policy Tree with root a_i is constructed by k subtrees

For each b_i, a_i, z_i, b'_i is calculated by belief state updating Equation, and is estimated by fuzzy distance-based belief selection method in EXPAND (B_0, b'). After getting B_{new}, the new policy tree will be constituted. If $b'_i \in B_{new}$, it is simple that take the optimal policy p^*_{t-1,b'_i} as the z_i branch. Otherwise, if $b'_i \notin B$, we calculate the distance $D(b', B_{new})$ between b'_i and B_{new}, and get the closest neighbor $\tilde{b}_i$ of b'_i in B_{new}, then take the optimal policy $p^*_{t-1,\tilde{b}_i}$ as the z_i branch. After all policy trees are construed by t-step iterations, purging must be executed to reduce the number of policy trees. In PIP, we only keep the optimal policy tree with max value of each belief point in B_{new}, and all other policy trees will be purged. The algorithm for PIP is showed below (Algorithm 1).

Algorithm 1 Point-based Incremental Pruning

```
Initialize Vp_{t,i}(a_i, b_i) = 0, P_t = ∅;
for each b_i ∈ B do
  for each a_i ∈ A do
    for each z_i ∈ Z do
      Subtree.Root = a_i;
      calculate b' by (7);
      B_new = EXPAND(B_0, b');
      if (b' ∈ B_new) then
        Subtree.Branch(z_i) = p^{a_i}_{t,b_i};
      else
        b̃ = min_{b∈B} ||b − b'||_1;
        Subtree.Branch(z_i) = p^{a_i}_{t,b̃_i};
      end if
    end for
  end for
end for
calculate V_{p^{a_i}_{t,b_i}}(b_i) using by (7) and (8);
p*_{t,b_i} = arg max_{p^{a_i}_{t,b_i}} V_{p^{a_i}_{t,b_i}}(b_i);
P_t = purge(⋃_{b_i ∈ B_new} p*_{t,b_i});
```

The complexity of traditional policy trees algorithms is $|S|^2|Z||P_t|$. Without pruning, the number of t-step policy trees is $|A|^{1+|Z|+|Z|^2+\cdots+|Z|^{t-1}}$. Where, $1 + |Z| + |Z|^2 + ... + |Z|^{t-1} = \sum_{i=0}^{t-1} |Z|^i = \frac{|Z|^t - 1}{|Z| - 1}$. So the complexity of the algorithm is $O(|S|^2|Z||A|^{\frac{|Z|^t-1}{|Z|-1}})$. In PIP algorithm, the number of t-step policy trees is $|B|$ at most after pruning. The complexity of PIP algorithm is $|S|^2|Z||B|$. $|B|$ is the key to reduce complexity.

V. EXPERIMENTAL RESULTS

We applied PIP to the benchmark RockSample problem. An instance of RockSample with map size $n \times n$ and k rocks is described as RockSample[n, k]. The POMDP model of RockSample[n, k] is as follows.

- The state set S. The state space is the cross product of $k+1$ features. The position of rover $X_P = \{(1,1),(1,2),,(n,n)\}$, k binary features $X_R^i = \{Good, Bad\}$ that indicate which one of the rocks are

good. The terminal state is EXIT. So, the number of states in RockSample[n, k] is $n^2 \times 2^k + 1$.

- The action set A. The rover can select from $k + 5$ actions: $\{North, South, East, West, Sample, Check_1, ..., Check_k\}$. The first four are deterministic single-step motion actions. The *Sample* action samples the rock at the rover's current location. If $X_R^c = Good$, then the reward $R = 10$, and $X_R^c = Bad$; else if $X_R^c = Bad$, then the reward $R = -10$. When the terminal state is EXIT, $R = 10$. All other moves have no cost or reward.
- The observation set Z. Each $Check_i$ action applies the rover's long-range sensor to rock X_R^i, returning a noisy observation from $\{Good, Bad\}$. The noise in the long-range sensor reading is determined by the efficiency factor $\eta(X_P, i)$.

$$\eta(X_P, i) = 2^{-d(X_P, i)/d_0} \tag{10}$$

Where, $d(X_P, i)$ is the Euclidean distance between position X_P and the position of rock X_R^i, which decreases exponentially as a function of Euclidean distance from the target. d_0 is a constant specifying the *half efficiency distance*, if $d = d_0$, then η = 1/2. At η = 1, the sensor always returns the correct value. At η = 0, it has a 50/50 chance of returning *Good* or *Bad*. At intermediate values, these behaviors are combined linearly. The initial belief is that every rock has equal probability of being *Good* or *Bad*. The discount factor $\gamma = 0.95$.

Each algorithm was tested on 10 different initial conditions (which rocks were higher valued and which were lower valued) and each scenario was tested 30 times. We evaluated the performance of PIP by the average total discounted reward and standard deviation. The initial belief state is the same for all runs and the discount factor $\gamma = 0.85$. Each algorithm was stopped after a maximum of 1000 steps or the maximum time of 36000s. Compare the reward and time affect the performance of Rollout, POMCP, and PIP algorithms, the results are showed in Table I.

Table I
THE REWARD OF ALGORITHMS

Algorithm	RockSample[11,11] $\|S\|$=247808	RockSample[15,15] $\|S\|$=7372800
Rollout	8.70 ± 0.29	7.56 ± 0.25
POMCP	20.01 ± 0.23	15.32 ± 0.28
PIP	20.97 ± 0.24	18.34 ± 0.21

From the experimental results, we can conclude that PIP algorithm gets more rewards compared with Rollout and POMCP algorithms.

VI. CONCLUSION

The paper proposes a Point-based Incremental Pruning (PIP) algorithm for solving the complexity of MCTS. PIP is used to avoid the exponential growth of the size of policy trees by selecting the optimal policy trees on specific belief points. The algorithm analysis and the experimental results indicate that PIP can get more rewards to get the optimal policy, and it is efficient for solving large-scale POMDPs problems.

ACKNOWLEDGMENT

This work was supported by the Guangdong Natural Science Foundation of China under Grant 2016A030310026.

REFERENCES

[1] C. B. Browne, E. Powley, D. Whitehouse, and S. M. Lucas, "A survey of monte carlo tree search methods," IEEE T COMP INTEL AI, vol. 4, pp. 1-43, 2012, doi: 10.1109/TCIAIG.2012.2186810 .

[2] G. M. J.B. Chaslot, S. Bakkes, I. Szita, and P. Spronck,"Monte-Carlo Tree Search: A New Framework for Game AI," Proc. Artif. Intell. Interact. Digital Entert. Conf., Stanford Univ., California, 2008, pp. 216-217.

[3] D. Silver, and J Veness, "Monte-Carlo planning in large POMDPs," Advances in neural information processing (NIPS), vol. 23, pp. 2164-2172, 2010.

[4] D. Silver, A. Huang, C. J. Maddison, et al, "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, pp. 484-9, 2016, doi: 10.1038/nature16961.

[5] J. Pineau, G. Gordon, and S. Thrun, "Anytime point-based approximations for large POMDPs," J Artif Intell Res, vol. 27, pp. 335-380, 2006.

[6] N. L. Zhang, and W. Zhang, "Speeding up the convergence of value iteration in partially observable Markov decision processes," J Artif Intell Res, vol.14, pp. 29-51, 2001.

[7] H. Kurniawati, and V. Yadav, "An Online POMDP Solver for Uncertainty Planning in Dynamic Environment," Robotics Research, Springer International Publishing, 2016.

[8] B. Wu, H. Y. Zheng, Y. P. Feng, "Point-based online value iteration algorithm in large POMDP," Appl Intell, vol. 40, pp. 546-555, 2014.

[9] S Gelly, and D Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go," Artif Intell, vol. 175, pp. 1856-1875.

[10] B. Wu, H. Y. Zheng, Y. P. Feng, "Point-based incremental pruning for partially observable Markov decision processes," J Comput Inform Sys, vol. 10, pp. 5103-5112, 2014.