2010 International Conference on Technologies and Applications of Artificial Intelligence

Pruning in UCT Algorithm

Jing Huang, Zhiqing Liu,Benjie Lu,Feng Xiao
BUPT-JD Institute of Computer GO
Beijing University of Posts and Telecommunications
Beijing, China 100876
flyanjj@gmail.com, zhiqing.liu@gmail.com, xfeng1986@gmail.com

***Abstract*—UCT is a Monte-Carlo planning algorithm that, with in a given amount of time,computes near-optimal solutions for Markovian decision processes of large state spaces.It has gained much attention from there search community and been used in many applications since its publication in 2006, because of its significant improvement of the effectiveness of Monte-Carlo planning computation. This paper proposes a modification of the UCT algorithm, which can prune certain Markovian decision process actions and their associated states during the Monte-Carlo planning computation. The pruning of actions and states is performed based on properties of underlying UCB algorithms of UCT. This paper proves that it is highly unlikely for the pruned actions and states to be in the solution path returned by the UCT algorithm, making the pruning modification almost just as good as the original algorithm. Additionally, the pruning modification may reduce the size of the Markovian decision process state space, and thus improves the effectiveness of the original algorithm. Experimental results in computer GO demonstrate the effectiveness of pruning in the UCT algorithm.**

***Index Terms*—UCT algorithm; Monte-Carlo planning; pruning condition; territorial information**

I. Introduction

For large state-space Markovian Decision Problems[?], Monte-Carlo planning is one of a few viable approaches to find near-optimal solutions. But original Monte-Carlo method[?] does not use any guidance including online knowledge or offline knowledge, its effectiveness in finding near-optimal solutions is very limited. With the publication of the UCT algorithm in 2006[?], near-optimal solutions to Markovian Decision Problems can be computed more effectively. The UCT algorithm is an application of UCB algorithms[?] to tree search, in which each tree node is treated as an independent multi-arm bandit, its child-nodes are independent arms of the bandit, and child-nodes are visited based on the UCB algorithms. Because of the UCT algorithm, we can achieve reward maximization by balancing using any knowledge that already acquired and attempting new actions to further increase knowledge. This is known as the exploitation-exploration dilemma in reinforcement learning[?]. At present, the UCT algorithm has been successfully used in solving many decision problems, including computer games, such as the game of GO[?], Backgammon[?], Chinese Checkers[?], Spades[?], Hearts-Shooting the Moon[?] and so on. At the same time, it can be used to choose fast implementations for the fast Fourier transforms[?] as well.

Although, the UCT algorithm is much more effective than the plain vanilla form of Monte-Carlo planning, it does not address the state-space problem of Markovian Decision Problems, as such its effectiveness can be severely limited when the state space of Markovian Decision Problems is large. In order to address this limitation, this article proposes a modification of the UCT algorithm that integrates state-space pruning in Monte-Carlo planning. More specifically, this paper presents three pruning conditions under which actions of state pruning may occur. The first two pruning conditions are based upon properties of underlying UCB algorithms in UCT, and are thus domain-independent. We prove in this paper that it is highly unlikely for the pruned actions and states to be in the solution path returned by the UCT algorithm, making the pruning modification almost just as good as the original algorithm. The last pruning condition is domain-dependent, based on territorial information in the game of GO, because computer GO is our primary application domain. Domain-independent and domain-dependent pruning conditions are complimentary, and can be used in UCT pruning individually or jointly. Our experimental results in computer GO demonstrate the effectiveness of pruning in the UCT algorithm.

The rest of paper is organized as follows: Section II discusses absolute pruning condition, our first domain-independent pruning condition based on properties of the underlying UCB algorithms, and proves that the UCT algorithm with pruning under the absolute pruning condition is equivalent to the original UCT algorithm. Section III discusses relative pruning condition, our second domain-independent pruning condition based on properties of the underlying UCB algorithms, presents algorithms to compute upper bound of visits to be used in the relative pruning condition, and proves that it is with a high probability that the UCT algorithm with pruning under the relative pruning condition is equivalent to the original UCT algorithm. SectionIVdiscusses the domain-dependent pruning condition based on territory information of computer GO. SectionV presents an enhanced UCT algorithm, based on these pruning conditions discussed above. SectionVI presents experimental results of the enhanced UCT algorithm in computer GO. This paper is complete with discussions of conclusion remarks and future works in Section VII.

II. Absolute Pruning Condition

Domain-independent pruning conditions are derived solely based on properties of the UCB algorithms of multi-armed

978-0-7695-4253-9/10 $26.00 © 2010 IEEE
DOI 10.1109/TAAI.2010.38

bandit problem. The UCB algorithms are used in the UCT algorithm to balance exploration and exploitation in action selection for each Markovian decision process state encountered in Monte-Carlo planning. In a UCT-based Monte-Carlo planning process with a given starting state $\mathcal{S}$, a number of Monte-Carlo simulations will be conducted from $\mathcal{S}$ within a given amount of time, a near-optimal solution returned as the result of the UCT process consists of a path of states starting from $\mathcal{S}$, in which each next state is the most visited state from its previous state.

Consider the k-armed bandit problem. Let a_i denote the arms, v_i the current number of visits on a_i, w_i the current number of wins on a_i, $v = \sum_i v_i$ the current total number of visits on all arms, and $w = \sum_i w_i$ the current total number of wins on all arms, where $i \in \{1, \cdots, k\}$. We clearly have $v_i \geq w_i$ and $v \geq w$ because the number of wins is always bounded by the number of visits. Let $\bar{v}_i$ denote the number of visits to be played on a_i, of which $\bar{w}_i$ denote the number of wins. Let $\bar{v} = \sum_i \bar{v}_i$ the total number of visits to be played on all arms, of which $\bar{w} = \sum_i \bar{w}_i$ denote the number of wins. Similarly, we also have $\bar{v}_i \geq \bar{w}_i$ and $\bar{v} \geq \bar{w}$. Let $V = v + \bar{v}$ denote the total number of visits on all arms. We now can state the first pruning condition as follows:

Condition 1. Absolute Pruning Condition: *a_i can be pruned if $\exists j$ such that $v_j > V/2$, where $j \in \{1, \cdots, k\} \cap i \neq j$.*

It is obvious that, If

$$v_j > V/2, where j \in \{1, \cdots, k\} \cap i \neq j \tag{1}$$

then

$$\sum_{i \neq j} v_i = V - v_j < V/2 \tag{2}$$

hence

$$v_i < V/2 < v_j (i \neq j) \tag{3}$$

Because the UCT algorithm always selects the most visited node in each step of path selection in return, pruned nodes are impossible to be the most visited ones due to the absolute pruning condition, and as such cannot appear in the solution path returned by UCT. Therefore, the path result returned from the UCT algorithm with absolute pruning condition is thus consistent with the result returned from the original UCT algorithm, making our pruning modification equivalent to the original algorithm.

III. Relative Pruning Condition

The absolute pruning condition can be relaxed with an introduction of a new concept called "upper bound number of visits" for a given arm at a given moment in the UCT process, or just "upper bound" in short. Upper bound of an arm specifies the largest number of visits that are possibly to be conducted on the arm. Due to the fact of the UCB algorithms that an arm with a higher winning rate will accumulate more visits, upper bound of an arm a_i at a given moment in the UCT process can only be achieved when all subsequent simulation results on a_i are always "win", and when all subsequent simulation results on the other arms are always "lose". Let u_i denote the upper-bound number of visits on a_i to be visited at this situation. For sure, if the total number of visits on a_i is still less than the current number of visits on a_j, a_i can be safely pruned. Here the total number of visits on an arm can be computed as the sum of its current number of visits and its upper bound number of visits. Therefore we can state the second pruning condition as follows:

Condition 2. Relative Pruning Condition: *a_i can be pruned if $\exists j$ such that $v_j > v_i + u_i$, where $j \in \{1, \cdots, k\} \cap i \neq j$.*

Obviously, situations in which the upper bound of an arm is indeed reached are rare. This is because the visited number of the arm a_i, whose average reward is not very high during the simulation so far, is unlikely to become perfect in subsequent simulation. Generally speaking, if the current winning rate of a_i is

$$r_i = w_i/v_i \tag{4}$$

it will be replaced by

$$r_i = 1 - \alpha(1 - r_i), \alpha \in [0, 1] \tag{5}$$

if

$$f(r_i) = 1 - \alpha(1 - r_i) - r_i = (1 - r_i)(1 - \alpha) \geq 0 \tag{6}$$

then

$$f'(r_i) = \alpha - 1 \leq 0 \tag{7}$$

from equations (6) and (7), we know that arms with low winning rates are will change more acutely. In order to ensure the arms are pruned safely, the relative pruning condition assumes better winning rates for arms with lower winning rates currently.

Now, we present a method to calculate upper-bound number of visits u_i for arm a_i. From the paper of Auer et. al.[4], we know that,

$$E[T_i(n)] \leq \frac{8 * \ln n}{\triangle_i^2} + 1 + \frac{\pi^2}{3} \tag{8}$$

where $\triangle_i = \mu^* - \mu_i$, μ_i denotes the winning rate of a_i, and μ^* denote the winning rate of the best arm a_{max}. (I.e., the visit number on a_{max} is the highest.) Then, let $E[T_i(n)]$ denote the number of visits on a_i during n simulations, clearly the expression in the right hand side of the inequality of (8) is the value of u_i.

If

$$g(r_i) = 1 - \alpha(1 - r_i) \tag{9}$$

Then

$$g'(r_i) = \alpha > 0 \tag{10}$$

Namely, if the winning rate on one arm is better than the winning rate on another arm, then after the transformation mentioned above, the relationship between the two arm will remain unchanged. Because of this property, the final result will keep same with the original UCB algorithm. The a_{max} whose win rate is the best will become the arm that we want to find out.

IV. Pruning Condition based on GO Territory

The above two pruning conditions are both domain-independent. The section describes a domain-dependent pruning condition for computer GO, it uses territorial information in GO to prune certain states in Monte-Carlo planning. GO, to certain extents, is a territorial game between two players, who alternatively place stones of their own color on to the GO board, occupying some territory. The player who occupies more territory at the end is the winner of the game.

Certain facts of territory information in GO exist based on GO knowledge. Generally speaking, the territory information of both GO players shall not change dramatically during one round of play (i.e., two plies). In other words, when both players play competitively in GO, one player shall not allow dramatic territory change in one round of play if he plays the second; and he shall not expect to cause dramatic territory change in one round of play if he plays the first. We shall use the term "hot point" to refer to the responding playing point in one round of play which ensures that no dramatic territory change occurs.In figure 1, the points which are marked cross are "hot points".

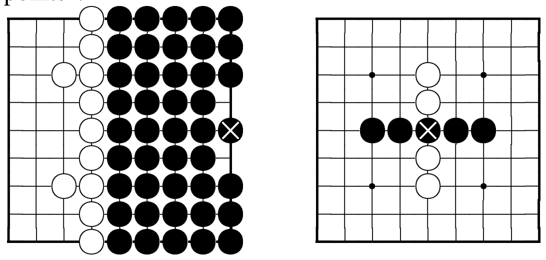


Figure 1. Hot Point

Given this informal definition of hot points, they can be computed as follows: For a given board B, let $P(B)$ denote its parent board, i.e., the last board position leading toward this board, $S(B)$ denote the set of its sibling boards, i.e., the set of all children boards of $P(B)$. Let A denote the number of points on the GO board, which is typically 81 or 361. For a given board B, and a given point p on B, let T_p^B denote the territory value of p. Let T^B denote the territory value of B, and we naturally have

$$T^B = \sum_{p \in B} T_p^B \tag{11}$$

Let Δ_B denote the change of territorial value of board B with respect to its parent $P(B)$, and we have

$$\Delta_B = T^{P(B)} - T^B \tag{12}$$

Let $E(\Delta_B)$ denote the expected change of territorial value of board B at each point, and we have

$$E(\Delta_B) = \frac{\Delta_B}{A} \tag{13}$$

We can compute the variance of the change of territorial value of board B in terms of its expectation as follows

$$D(\Delta_B) = E(\Delta_B^2) - E(\Delta_B)^2 \tag{14}$$

For a given board B, a measurement of its territorial change with respect to its parent $P(B)$, denoted as $\Phi(B)$, can be computed as follows

$$\Phi(B) = \frac{D(\Delta_B)}{\sum_{C \in S(B)} D(\Delta_C)} \tag{15}$$

$\Phi(B)$, in the range of $[0, 1]$, measures how stable a board is with respect to territory.

Now we can state our third pruning condition as follows

Condition 3. Territorial Pruning Condition: *Board B_i can be pruned if $\exists$ board $B_j \in S(B), B_i \neq B_j$ such that $\Phi(B_j) \notin [M, N]$ and $\Phi(B_i) \in [M, N]$, where M and N are two predetermined constants.*

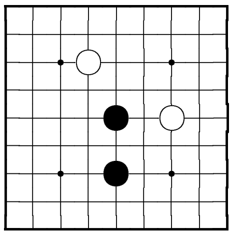
If $\Phi(B_j) > N$, it means board B_j is beneficial to player who is at a disadvantage. If $\Phi(B_j) < M$, it suggests B causes relatively small territory swing and board B_j is beneficial to player who is at a advantage. From different standpoint, the player will choose some action to defend or attack. So, this hot points are all important. N and M mentioned in the paper are based on the experiment and experience.

Obviously, the above discussions assume that territorial value can be measured for each point on the GO board. While it is difficult to measure this value statically, it can be computed as an expectation in Monte-Carlo planning. In other words, we can enhance the original Monte-Carlo simulation to return territorial information. Expected values from a large amount of simulations will measured territory accurately.

More specifically, we modify the Monte-Carlo simulation used as the evaluation method in the UCT algorithm as follows: Each Monte-Carlo simulation will obtain the full territory information of a complete game. After a number of simulations we can compute the average occupancy of every point. In fact, let n denote the total number of simulations conducted from the current board. For a given point p on the board, let $b_p(n)$ denote the number of times in which the point is occupied by Black, $w_p(n)$ denote the number of time in which the point is occupied by White, and $e_p(n)$ denote the number of time in which the point is empty. Obviously, we have $n = b_p(n) + w_p(n) + e_p(n)$. As such, the territorial value of a point p in board B can be computed as follows

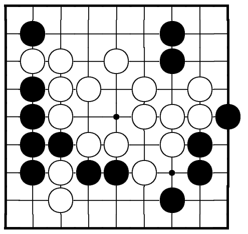
$$T_p^B = \frac{b_p(n)}{n} - \frac{w_p(n)}{n} \tag{16}$$

Figure 2 shows territorial values of each point of in an early game, while Figure 3 shows territorial values of each point of in an ending game. For clarity, the territorial values are into the range of [0, 10]. We also know that the closer to 10 the value is, the higher probability the point belongs to black. On the hand, the closer to 0 the value is, the higher probability the point belongs to white. We can get every point's territorial information value in this way.



```
3·3·3·3·3·3·3·3·3
3·3·3·3·3·3·3·3·3
4·4·4·3·4·3·3·3·3
4·4·4·4·5·4·3·2·2
4·4·4·5·7·4·1·2·2
4·4·5·5·6·4·3·3·3
4·4·5·5·7·5·4·3·3
4·5·5·5·5·5·4·4·3
5·5·5·5·5·5·4·4·4
```

Figure 2. Stable State



```
3·3·3·2·2·3·4·4·4
3·3·2·1·2·3·4·4·3
2·0·0·0·0·2·4·3·2
4·5·0·0·0·0·1·0·3
5·5·0·0·0·0·0·0·7
5·5·5·0·0·0·0·7·7
5·5·2·3·3·0·3·7·7
5·5·2·2·3·5·7·7·7
4·3·2·3·3·5·7·7·7
```

Figure 3. not Stable State

V. Pruning in Monte-Carlo Planning

A. Pruning with relative pruning condition

It is not suitable to use UCB algorithm with relative pruning condition in every simulation. There are 2 primary reasons. On the one hand, as the node added, the cost of evaluation process with pruning condition will cause the number of simulation time decreasing. That may become the reason of result in a deviation. On the other hand, if the win rate doesn't keep stable, it is possible to prune some relatively good node. Thus, we often use this pruning method after 1 second simulation. Algorithm 1 describes this method:

Algorithm 1 Pseudocode of UCT with Pruning Condition

```
1: function UCTwithPruningCondition(rootNode)
2: node[0]:=rootNode; i=0;
3: while node[i] is not leaf do
4:     if one second passed then
5:     $pruning\ the\ nodes\ which\ meet\ the\ pruning\ conditions$
6:     end if
7:     node[i+1]:=descendByUCB1(node[i]);
8:     i:=i+1;
9: end while
10: updateValue(node,-node[i].value);
```

To be specific, if we use the UCT algorithm with absolute pruning condition and relative pruning condition at the same time, actually we use the relative pruning condition only. Because any arm's upper-bound number of visits must be less than V/2. In addition, we can use the first two pruning condition with the territorial pruning condition simultaneously.

VI. Experimental Results

The experimental data come from 9×9games between the original version of LinGo or LinGo with pruning condition and GnuGo3.8. To be specific, we played 500 games for each CASE and the simulation time used to get one step is 10 seconds. Because UCT algorithm promotes the Go program's intelligence, we use these pruning condition in LinGo directly to prove the effect of UCB algorithm with all kinds of pruning condition. CASE1 to CASE4 describe the result when we used UCT algorithm with relative pruning condition, CASE5 is the result when we use the UCT algorithm with territorial pruning condition and CASE6 describe the result when we use the second condition and the third condition. The following is the result,the first column describes the LinGo's version, the second column records the win rate of the games between different version of LinGo and GnoGo3.8, and the third column records how much time will LinGo spend on in one game:

$version$	win_rate	$time(s)$
$Original\ version$	87 ± 3.4	126.2 ± 2.7
$CASE1(\alpha = 1.0)$	84.0 ± 3.7	119.4 ± 3.0
$CASE2(\alpha = 0.9)$	87.9 ± 3.3	118.5 ± 2.7
$CASE3(\alpha = 0.8)$	89.5 ± 3.1	118.9 ± 2.8
$CASE4(\alpha = 0.7)$	88.1 ± 3.2	119.9 ± 2.9
$CASE5$	92.0 ± 3.2	120.9 ± 2.6
$CASE6(\alpha = 0.8)$	92.4 ± 2.2	120.1 ± 2.3

From the result of experiments we can find that the performance of LinGo using UCB algorithm with pruning condition is better than the original LinGo's. First of all, the win rate of programs which use the UCB algorithm with pruning condition increase to 90%. Secondly, we will spend less time to play a game of Go.

VII. Conclusions

Although the experiment is effective, there is some limitations. When we use the UCB algorithm with absolute pruning condition or relative pruning condition, just a few nodes will be pruned. Because the pruning condition is very strict, we have to find some loose pruning condition to prune more nodes in simulation. To be specific, we will try to find other method to get upper bound of visits when we use the UCB algorithm with relative pruning condition. Meanwhile, when we use the territorial information pruning condition, we can give a more accurate interval to make that less nodes which is not important about game's result will be staying.

References

[1] Michael L. Littman, Thomas L. Leslie Pack Kaelbling. On the Complexity of Solving Markov Decision Problems. Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence(UAIC95):394-402,1995.

[2] F. James. Monte Carlo theory and practice. Reports on Progress in Physics, Vol.43, NO.9:1145-1189,1980.

[3] L. Kocsis and C. Szepesvari. Bandit based monte-carlo planning. The 15Th European Conference on Machine Learning(ECML), pages 282-293,2006.

[4] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. Machine learning, 47(2/3): 235-256, 2002.

[5] Rémi Munos, Olivier Teytaud. Modification of UCT with Patterns in Monte-Carlo Go. Technical Report 60-62, INRIA, France, November 2006.

[6] Sylvain Gelly, YizaoWang. Exploration exploitation in Go: UCT for Monte-Carlo Go. Twentieth Annual Conference on Neural Information Processing Systems(NIPS),2006.

[7] F. Van Lishout, G. Chaslot, and Jos W.H.M. Uiterwijkm. Monte-Carlo Tree Search in Backgammon. Computer Games Workshop :175-184, 2007.

[8] Mark H. M. winands, Yngvi Björnsson and Jahn-Takeshi Saito. Monte-Carlo Tree Search Solver. Computer and Games: 25-36, September 2008.

[9] F. de Mesmay, Arpad Rimmel, Yergen Voronenko, Markus Püschel. Bandit-based Optimization on Graphs with Application to Library Performance Turning. Proceedings of the 26th International Conference On Machine Learning:729-736, 2009.