2012 11th International Conference on Machine Learning and Applications

Using Local Regression in Monte Carlo Tree Search

Arisoa S. Randrianasolo
School of Computing and Informatics
Lipscomb University
Nashville, TN, USA
Email:arisoa.randrianasolo@lipscomb.edu

Larry D. Pyeatt
Department of Mathematics and Computer Science
South Dakota School of Mines and Technology
Rapid City, SD, USA
Email:larry.pyeatt@sdsmt.edu

***Abstract*—In this non-comparative study paper, local regression modeling is added to Monte Carlo Tree Search (MCTS). This local regression modeling is used to replace the random action selection in the simulation part of MCTS. The modified MCTS method and the regular MCTS method were tested against each other on the tic-tac-toe and the connect four games. The results of the experiment indicated that local regression helped the modified MCTS to outperform the regular MCTS on moderately memory demanding games, such as connect four. The performances of the two approaches appear to equal each other on low branching factor and less memory demanding games similar to tic-tac-toe.**

***Keywords*-Monte Carlo Tree Search ; Q-learning; General Game Playing; Local Regression.**

I. Introduction

Creating a real time general game player capable of playing various games without any prior training is a difficult and interesting task. Using reinforcement learning methods appears to be the best approach to create such game players. The games played by the general game players can be of various complexities and can go from a simple tic-tac-toe to a Go game. The thinking or simulation time allowed in each game to be played is very short; a few seconds in general. These facts render it impossible to use straight-forward reinforcement learning methods or tree structures to create a general game player. This situation, however, did not stop many researchers from creating algorithms that are capable of dealing with the curses previously listed. Monte Carlo Tree Search is one of the algorithms commonly used to create a general game player [1] [2]. Each MCTS-based general game player imposes its own modifications to the original and basic MCTS. Similarly, this non-comparative paper provides an explanation for a way in which local regression can be employed to improve the simulation part of the original MCTS algorithm.

II. Monte Carlo Tree Search: MCTS

Monte Carlo Tree Search is a search method closely related to minimax and alpha-beta searches. The learning or the gaming environment is expressed in a tree-like graph. The nodes of the tree represent the states in the environment. Each branch from each node of the tree describes an action that can be taken from the state represented by the node. The tree has starting states and final or goal states. There is a reward, r, associated with taking an action, a, from a state, s. State-action values are created to describe the quality of taking an action, a, while in a state, s. These values are represented by $Q(s,a)$. If the environment is in a state, s, and the learner took an action, a, then the environment transitions into a new state, s', where the leaner can take a new action, a'.

MCTS is composed of four steps[3]. The first step is called *selection*. In this step, the game tree is traversed and an action is selected to move the environment to a new state or a new node. The second step is called *simulation*. In this step, MCTS performs playouts that go from a starting node to an end node. The third step, *back propagation*, consists of forwarding, to each node involved in a playout, the total reward received at the end of a playout. The last step, called *tree expansion*, consists of deciding which node(s) encountered in the simulation is(are) going to be added to the search tree.

MCTS requires a considerable amount of simulation time to be allocated to guarantee that the search converges into an optimal policy. A policy is a mapping from states to actions that will maximize or minimize the long term reward of the learner. The goal in the simulation is to visit all the possible pairs of state-action, in the tree, multiple times[4]. It is not always possible, for real time general game players, to possess an important amount of simulation time. This triggers the numerous attempts by various researchers to modify some or most of the Monte Carlo Tree Search steps.

III. MCTS steps

A. Selection

While traversing the game tree, a greedy selection or an ϵ-greedy selection can be used[4]. The main goal of the selection method is to find a balance between the exploitations of known moves or actions and the the explorations of unseen or unknown moves. The most common used strategy for an action selection while traversing the tree in MCTS is called Upper Confidence bound applied to Tree(UCT)[5]. UCT creates new state-action values using the equation,

$$Q^*_{UCT}(s,a) = Q_{UCT}(s,a) + c * \sqrt{\frac{\lg n(s)}{n(s,a)}}. \quad (1)$$

It selects an action by following the calculation,

$$\pi_{UCT}(s) = \arg\max_a Q^*_{UCT}(s,a), \quad (2)$$

where $n(s,a)$ is the number of times that an action, a, has been selected from a state, s, and $n(s)$ is the total number of

978-0-7695-4913-2/12 $26.00 © 2012 IEEE
DOI 10.1109/ICMLA.2012.91

IEEE computer society

visits to a state, s. This means that

$$n(s) = \Sigma_a n(s, a). \tag{3}$$

In the UCT approach, if an action, a, has not been used in a state s, then UCT defaults to selecting, or exploring, this action rather than using the actions that have been used before. UCT is a famous approach in solving the GO game[6]. It has recently become the method of choice in creating a general game player[7]. Previous winners of the IJCAI General Game Playing competitions are believed to be UCT based [1]. More information about the UCT method can be found in a paper by Gelly and Silver [5]. While traversing the game tree, the modified MCTS, explained in this paper, uses UCT as an action selection mechanism.

B. Simulation

The simulation consists of preforming numerous playouts. A recent MCTS-based general game player adopted the following strategy when conducting a playout: If the node or the state encountered in the playout is within the game tree, built in memory, then use UCT as an action selection. In the other case, use a random action selection or some other knowledge based action selection method. In a game or during the simulations, if the resulting state, s', from taking an action, a, is not part of the game tree, then the UCT policy, π_{UCT}, is abandoned and replaced by a random or a non-random policy. This paper is focused on improving the action selection during the game and in the simulation.

C. Back Propagation

Various back propagation methods can be used and there is no single agreed upon method that appears to be the best. MCTS back propagation methods can be similar to those used in Q-learning, Q(λ), TD(0) and TD(λ)[4]. The main idea is to distribute back to the higher nodes in the tree the total reward received at the end of each playout. The back propagation used in this paper is similar to the one used in Watkins'Q(λ)[4].

D. Tree Expansion

To use memory efficiently, only the first state or node that happens to be outside the game tree is added to the tree. Any states encountered, following the first state out of the game tree, will be ignored once the playout reaches an end node and a back propagation is performed.

IV. Local regression

Local regression is a method used to approximate the relationship between a predictor variable, x_i, and a response variable, y_i. This situation is described by a function, f, such that

$$y_i = f(x_i) + \varepsilon_i. \tag{4}$$

f is an unknown function and ε_i is an assumed error value. The goal is to predict the value, $f(x_0)$, for a target variable, x_0. The local regression method uses the predictor variables within the interval

$$[x_0 - span(x_0); x_0 + span(x_0)]$$

when it is computing the value of $f(x_0)$. To calculate $span(x_0)$, let α be a span percentage. If $\alpha = 0.1$, then 10% of the total predictor variables will be used to calculate the value of $f(x_0)$. Let $\Delta_i(x_0) = |x_0 - x_i|$, and let $q = \alpha * N$, where N is the total number of predictor variables, then define

$$span(x_0) = \Delta_q(x_0). \tag{5}$$

For simplicity, assume that $f(x_i)$ is a quadratic function such that,

$$f(x_i) = \beta_0 + \beta_1(x_i - x_0) + \frac{1}{2}\beta_2(x_i - x_0)^2, \tag{6}$$

with x_i in $[x_0 - span(x_0); x_0 + span(x_0)]$. f can be approximated by calculating the parameter $\beta = (\beta_0, \beta_1, \beta_2)'$ as follow,

$$\begin{aligned} \beta^* \quad = \quad & \arg\min_{\beta \epsilon R^3} \textstyle\sum_{i=1}^{n} w_i(x_0)[y_i \\ & -\beta_0 + \beta_1(x_i - x_0) + \tfrac{1}{2}\beta_2(x_i - x_0)^2]^2 \end{aligned} \tag{7}$$

$$w_i(x_0) = W\left(\frac{x_i - x_0}{span(x_0)}\right) \tag{8}$$

$$W(u) = \begin{cases} (1 - |u|^3)^3 & : u <= 1 \\ 0 & : u > 1 \end{cases} \tag{9}$$

x_i can be a single variable or a vector, similar cases can also happen with x_0 and y_i. This section is a simplified introduction to local regression. Any interested readers can acquire more in-depth explanation from the references [8] and [9].

V. Idea to improve MCTS

Instead of performing a random action selection when faced with a state that is outside of the game tree, we use local regression to perform a guided action selection. Assume that an out of game tree state, s_i, is encountered. The neighbor states, for the state s_i, can be defined as the states within the interval

$$I_{s_i} = [s_i - span(s_i); s_i + span(s_i)]. \tag{10}$$

$span(s_i)$ is obtained by replacing x_0 by s_i(see previous section). The query or the target variable is the pair, (s_i, a_1). Once a query is obtained, define the new neighbor states as the states in the game tree and in I_{s_i} that have a recorded value for the action a_1. Let a set, S, represents these new neighbor states. Local regression can then be applied to S in order to estimate the value of $f(s_i, a_1)$. In this case, the predictor variables to be used in the local regression consist of all s in S. The response variables consist of all the $Q(s, a_1)$ for all s in S.

It is possible to encounter a situation where the states in the game tree and in I_{s_i} have no recorded values for the action, a_1. In this case, the local regression increases the computation to a three dimensional case. The predictor variables become all pairs, (s, a), for any s in I_{s_i} and in the game tree. The response variables become all the $Q(s, a)$ for all s in I_{s_i} and in the game tree. It can happen that I_{s_i} is empty. In this situation, the default random action selection from the traditional MCTS

method is used.

In this paper, it assumed that f is a quadratic function. By setting $(s_i, a_1) = x_0$, $x_i = (s, a_1)$ for any s in I_{s_i} and in the game tree, $span(x_0) = span(s_i)$ and $span(x_i) = span(s)$ for any s in S and in the game tree, the equations from the previous section can be used without any modification.

The parameter β can be calculated in the following way. For any $s_1, ..., s_n$ in S,

$$Y = \begin{bmatrix} Q(s_1, a_1) \\ . \\ . \\ . \\ Q(s_n, a_1) \end{bmatrix} \tag{11}$$

$$X = \begin{bmatrix} 1 & (s_1, a_1) - (s_i, a_1) & ((s_1, a_1) - (s_i, a_1))^2 \\ . & . & . \\ . & . & . \\ . & . & . \\ 1 & (s_n, a_1) - (s_i, a_1) & ((s_n, a_1) - (s_i, a_1))^2 \end{bmatrix} \tag{12}$$

$$W = \begin{bmatrix} w_1(s_i, a_1) & 0 & 0 & ... & 0 \\ 0 & w_2(s_i, a_1) & 0 & ... & 0 \\ . & 0 & w_3(s_i, a_1) & ... & 0 \\ . & . & . & ... & 0 \\ . & . & . & ... & 0 \\ 0 & 0 & 0 & ... & w_n(s_i, a_1) \end{bmatrix} \tag{13}$$

Matrix multiplication is used to perform the following calculation,

$$Z = X^T * W \tag{14}$$

$$Y = Z * Y \tag{15}$$

$$X = Z * X \tag{16}$$

$$\beta = X^{-1} * Y \tag{17}$$

The value of $Q(s_i, a_1) = \beta(1, 1)$.

VI. Related work

A GO game player, called MoGo [10], was the first game player that employed an UCT action selection and a modified MCTS. MoGo used a Go game specific move selection when the action search happened to go out of the simulation tree. MoGo's strategy consisted of evaluating a move based on a 3x3 Go game sub-board. The evaluation process would then decide if the move was playable or not. The creators of MoGo claimed that this approach was superior to the pure random action selection.

Hilmar and Yngvi [1] listed a few of the action selection methods used in the simulation step of an MCTS. The first approach, called Move-Average Sampling Technique(MAST), stored quality values, $Q_h(a)$, for each action, a, seen in the simulation. $Q_h(a)$ was calculated from the back propagation values received by each node in the playout. Instead of choosing a random action during the simulation, MAST used a Gibbs distribution to assign a selection probability, $P(a)$, to each action. The second simulation action selection method was called Tree-Only MAST. This differed from MAST by only using the back propagation values of the nodes in the tree and the first node outside the tree to calculate $Q_h(a)$. The third action selection method was called Predicate-Average Sampling Technique(PAST). PAST calculated a predicate-action value, $Q_p(p, a)$, instead of the action value, $Q_h(a)$. The predicate p could be any predicate associated with a state, s. The fourth action selection method, called Features-to-Action Sampling Technique(FAST), used feature detection and TD(λ)[4] to create the action value, $Q(a)$, which was used to replace the $Q_h(a)$ in MAST.

Using the back propagation values as a way to assign value to the possible actions that can be taken in a state is a common idea to improve MCTS. A method called Rapid Action Value Estimation(RAVE)[5], for example, used the back propagation value to not only create a state-action value, $Q(s, a)$, but also an approximate state-action value, $Q_RAVE(s, a')$. In $Q_RAVE(s, a')$, the action, a', was used in one or more nodes, under s, during a playout. $Q_RAVE(s, a')$ gives an estimate of the quality of choosing the action, a', while in a state, s. It helps MCTS to make an informed action selection during the simulation instead of a random one. Other Approaches, like the one used by Mhat and Cazenave [2], used the results of a certain number of parallel simulations to create a combined evaluation for each move.

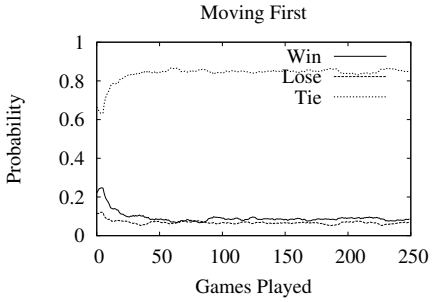
VII. Experiment

An experiment was conducted to measure the performance of MCTS equipped with a local regression against the original MCTS. From this section of the paper to the end, the MCTS implementation that uses a local regression will be referred to as MCTS-reg. The original MCTS and MCTS-reg used UCT as an action selection method for nodes in the game tree. They both used Watkins' Q(λ) back propagation method for the state-action values. They both added one node at a time to the tree after each playout. The difference was located in the action selection for nodes outside the game tree.

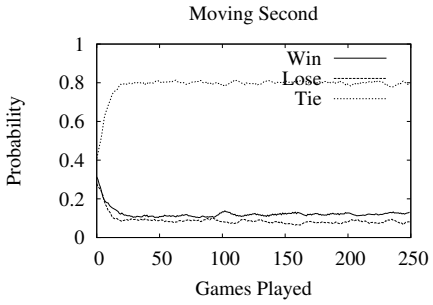
Tic-tac-toe and connect four were used as the test beds for the experiment. A total of 1000 runs were performed for each test bed. Each run contained 500 games of connect four or 500 games of tic-tac-toe. Both players, MCTS and MCTS-reg, were given 10 seconds of simulation time before responding with an action. The players alternated taking the first move in each game. At the end of each run, the state-action values learned from all of the 500 games were cleared and were changed to random values. There is a considerable advantage for the player that moves first in each game. This is the reason why the results of this experiment are expressed by two graphs. The first graph, labeled (a), expresses the cases where MCTS-reg moves first. The second graph, labeled (b), expresses the cases where MCTS-reg moves second.

VIII. Conclusions and Future Works

The performance of the original MCTS and MCTS-reg appear to equal each other in the tic-tac-toe game. In Figure 1, the probability for a tied game is high and reaches 80%. This situation happens because of the low branching factor associated with tic-tac-toe. The game tree associated with this game can be easily stored in the memory of a modern computer. This enables the original MCTS approach to be able to visit most, if not all, of the possible combinations of state-action pairs. The local regression computation in MCTS-reg limited the number of simulations performed to be inferior

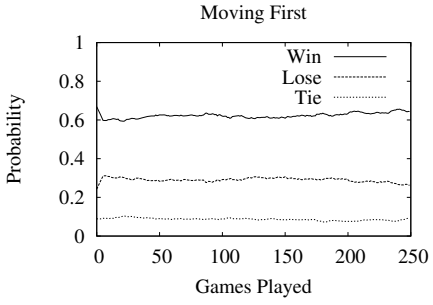


(a) MCTS-reg has first move.

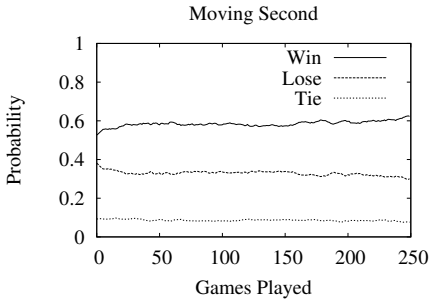


(b) MCTS-reg has second move.

Fig. 1: Comparison of MCTS-reg and MCTS using the tic-tac-toe game. In (a), MCTS-reg makes the first move. In (b) MCTS-reg makes the second move.



(a) MCTS-reg has first move.



(b) MCTS-reg has second move.

Fig. 2: Comparison of MCTS-reg and MCTS using the connect four game. In (a), MCTS-reg makes the first move. In (b) MCTS-reg makes the second move.

to the number of simulations in the regular MCTS. Despite this situation, MCTS-reg still managed to keep its winning probability slightly above its losing probability.

MCTS-reg clearly outperformed the original MCTS in the connect four game. This game has a moderately high branching factor. It is also more memory demanding. Due to the limit on the simulation time and the testing server's memory, it was impossible to visit all pairs of state-action involved with this game during the experiment. While the original MCTS played randomly when it encountered a new state that had not seen before, MCTS-reg used local regression to make an informed selection by analysing previously seen states. The resulting performance is expressed by Figure 2. It can be confirmed that MCTS-reg was far superior to MCTS in both cases of the experiment on connect four.

Figure 1 and Figure 2 show that local regression has the potential to provide Monte Carlo Tree Search a high quality simulation and an excellent out of tree action selection. Using a local regression method appears to give a higher policy to MCTS when facing a memory demanding game. The performance of MCTS equipped with local regression appears to equal the original MCTS in low branching factor task. As it is confirmed by Figure 2, the informed action selection approach performs better than the random action selection.

More testing in other test domains are planned to be carried out in future work. The performance of MCTS-reg needs to be measured on more complex and more memory demanding games so that better analysis can be accomplished. Resources limited us to not be able to execute such testing at the moment. Comparing the strength of the local regression action selection to other non-random action selections is also planned in our future work.

REFERENCES

[1] F. Hilmar and B. Yngvi, "Simulation-based approach to general game playing," in *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*, Chicago, Illinois, 2008, pp. 259–264.

[2] J. Mhat and T. Cazenave, "A parallel general game player," *KI journal*, vol. 25, no. 1, pp. 43–47, 2011.

[3] M. H. M. Winands, Y. Bjrnsson, and J.-T. Saito, "Monte-carlo tree search solver," in *Computers and Games*, 2008, pp. 25–36.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge,Massachusetts: The MIT Press, 1998.

[5] S. Gelly and D. Silver, "Combining online and offline knowledge in uct," in *ICML '07: Proceedings of the 24th international conference on Machine learning*, 2007, pp. 273–280.

[6] M. Enzenberger and M. Mueller, "an open-source framework for board games and go engine based on monte-carlo tree search," *Technical Report TR 09-08, Dept. of Computing Science. University of Alberta, Edmonton, Alberta, Canada*, Dec 2009.

[7] M. Genesereth and N. Love, "General game playing: Overview of the aaai competition," *AI Magazine*, vol. 26(2), pp. 62–72, 2005.

[8] W. S. Cleveland and S. J. Devlin, "Locally weighted regression: An approach to regression analysis by local fitting," *Journal of the American Statistical Association*, vol. 83, no. 403, pp. 596–610, 1988.

[9] R. B. Cleveland, W. S. Cleveland, J. E. Mcrae, and I. Terpenning, "Stl: A seasonal-trend decomposition procedure based on loess," *Journal of Official Statistics*, vol. 6, no. 1, pp. 3–73, 1990.

[10] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of uct with patterns in monte-carlo go," INRIA, France, Tech. Rep., November 2006.