Adversarial search of Monte-Carlo simulation for 3D Tic-Tac-Toe

Kohei Kawasaki
kawas009@umn.edu
Carlo Hernandez
herna463@umn.edu

May 11, 2016

1 Introduction

1.1 Brief Description of problem

As the game is getting more complex and deeper strategically, the heuristic function and cost function is getting useless due to absence of absolute evaluation of current state and forward step. That node will be expanded exponentially and it could be said impossible to use $A*$ search or $IDA*$ since both algorithms depends on the validity of evaluation function and strategically complexity causes evaluating a each move to difficult. In alpha-beta pruning, if we put bounds on the possible values of the utility function, the nwe can arrive at bounds for the average without looking at every number. Therefore Monte-Carlo simulation(Tree search) works an alternative evaluation of Alpah-Beta prunning or other search algorithm. From a start position the algorithm play thousands of games against itself, using randomly chosen move and evaluate each note by the statical win percentage. Monte Carlo Tree Search MCTS, does not rely on a positonal evaluation function, however this approach is a general algorithm and can be applied to mamy problems. The most promising result was the game of Go. In this project, I am going to work on 3D tic-tac-toe from the approach of monte carlo tree search. 3D tic-tac-toe is the three dimensional version of commonly played board game. The strategy of game expanded due the expansion of dimension. Assume N as the length of each side, the size of the board game N^3. Thus the maximum tree size is theN^3!. In terms only size of board game, from 5 side length, it is over the game of go. However because of the simplicity of the game rule, its branching factor still is inferior to game of go. For the similarity of game, I implement algorithm, which is commonly used Go, to 3D tic-tac-toe. Here is the categoris of problems: (1) Single player ploblems(called optimization problems), (2) two opponent problems, and (3) problems with multiple opponents. Likewise Go, Chess or Shogi, 3D tic-tac-toe is also non-cooperative two player deterministic game.

1.2 Monte-Carlo Tree Search

Monte Carlo Tree Search is the best first search method that does not require a positional evaluation function based on a stochastic exploration. And piling the result of explorations, the algorithm gradually builds up a game tree in memory, and successively becomes to more accurate estimation of the variable moves. MCTS consists of four main parts. (1) Selection, (2) Expansion, (3) Simulation

Table 1: Different types of problems.

	One player	Two player	Multi Player
Deterministic	TSP, PMP	Go, Chess, Shogi	Chinese Chekers
Stochastic	Sailing Problem	Backgammon	Simplified Catan

and (4) Back propagation. These part are repeated as long as there is time left. The steps are as follows. (1) From the root node, the selection strategy is applied recursively until unexplored node selected, (2) then one unexplored node is added to the tree, (3) one simulated game is played on the node. (4) Finally, the result of this game is back propagated in the tree.

2 Related works

The first Monte-Carlo simulation appeared on early 1990. B. Abramson developed (expected-outcome) model which returns expected value of the games outcome[1]. A standard model of static evaluators is Min-Max tree search. For the difficulty of setting of evaluation, however, the design is viewed as impossible. Moreover, it was too complicated to evaluate the accuracy due to the absence of standard evaluation to grasp a strategy in the entire game. From this perspective, Abramson defined as (Expected-Outcome) values as

$$EO(G) = \sum_{leaf=1}^{k} V_{leaf} P_{leaf}$$

where Node G is given by a players expected payoff over an infinite number of random completions o game, k is the number of leaves in the subtree, P_{leaf} is the probability to be reached in a given random play. At the bottom line, Abramson tired to make a heuristic from the statical evaluation of random game plays. Due to limitation of computer architecture at those days, the test was operated 6x6 Othello. By running a program in a tons of random game, from its value, the player make a decision.

B. Brugmann implemented simulated annealing to the Monte Carlo simulation as their project GOBBLE[3]. At the first glance because of the difference between traveling salesman problem and tree search Game like Go however, the two competing parties playing the game in hostile way and have to optimise two sequences of moves. It seemed simulated annealing cannot be to find local variable due to the discontinuous function of the list of moves. As one of possible solution, through the simulation of random games, algorithm itself is learned. And here is the 2 point of improvement. Evaluation of the moves. at any stage of the game, moves were evaluated by the average score of the game. Selection of the moves. Simulated annealing was used to control the probability to good moves are more likely to be played first.

Boozy and Helmstetter developed two Go programs based on same basic idea[2]. That derives from their two hypothesis. First if it is possible to terminate search process of game, the process provides the best move and this process does not necessarily need domain dependent knowledge, however, its cost is exponential by the search depth. Second, knowledge-based go programs seemed impossible to improve. Thus, their paper explored a hybrid solution of these two idea, which is little knowledge based. As an domain-dependent knowledge part, they defined of an eye, and in the eye random programs not to play. Next thing is evaluation of terminal position for a random

game. There are two two ways to evaluate it,for the first move of the game only, or for all moves played in the as if they were the first to be played. In their project latter one is implemented. The Advantage is the one random game helps evaluate almost all possible moves at the root, yet as a drawback, move order is ignored which is crucial to this game.

3 The Monte Carlo Tree Search Idea

The basis of Monte Carlo Simulation came out of the limitations of other Artificial Intelligence. Classic approaches from the past relied on on a game dependent heuristic to give an AI an estimate of the current game state. However, making making mid state evaluators that reliably evaluated heuristic values are too complex and depend too much on the nature of the game. [6] Monte Carlo Tree Search does not have this problem because it does not rely on heuristics to get a guess of the game state. Rather, it builds simulations of the game and then builds statistics to choose the next action with the highest win rate of the simulations. This is done using four stages: Selection finds a state in the tree and decides based on the statistics aggregated which node to explore. There is a balance that needs to be achieved such that it does not keep picking a local optimum to allow the exploration of a possible better game state. Expansion is when the game reaches a node that is currently not in the search tree, and a new node is added. From there, Simulation takes over, making random moves for both the parties. Backpropagation happens when the simulation reaches a terminal node, and the result of that simulation is sent upwards throughout the path that the selection phase took, updating the win rates of those nodes.

4 Model Based Reinforcement Learning for Playing Atari Games

One of the challenges of using the Monte Carlo Search Idea is that the Selection phase can become computationally expensive if it keeps selecting nodes that lead to regions of the search tree that doesn't look promising. Another way of balancing exploration and exploitation is to use weighted Monte Carlo Tree Search. This AI gives each action node from the current state a probability of being picked in selection based on its current win rate and then a node is randomly picked. In contrast to uniform random selection or select highest win rate node, this allows for the possibility of picking a low win node and possibly discovering a better action while still favoring nodes with better win rates, making the algorithm as a whole more reactive. In Model Based Reinforcement Learning for Playing Atari Games, the game of pong was used to evaluate the effectiveness of normal Monte Carlo Tree Search and weighted Monte Carlo, the weighted version was found to react faster to the ball than uniform random Monte Carlo. [9] At long distances, the paddle wont even move to the general vicinity of the ball because the goal state is too deep in the search tree to be found. In weighted, this is mitigated by giving a majority of simulation time to more promising actions.

5 Parallel Monte Carlo Tree Search

With the advent of new hardware paradigms, implementations of Monte Carlo can take advantage of these paradigms. One paradigm that Monte Carlo is set to take advantage of is parallelism in hardware. From this, four implementations Monte Carlo Search have arisen. Of the four, the

two that give insight are Leaf and Root Parallelization.[7] Leaf is one way to apply parallelism to the simulation phase of MCTS. That is, multiple threads are assigned the task of carrying out simulations. This allows for multiple samples on one selection phase, giving a more accurate distribution of win rates across the search nodes. Root on the other hand seeks to parallelize from the root up. This means that full phase of MCTS are parallelized and the grand total of each search nodes are totaled up and the best action is selected based on that.

6 UCT

Upper Confidence bounds to applied to Trees strategy(UCT) was advocated and developed by Kocsis and Szepesvari(2006) in order to find a near optimal action in large state-space Markovian Decision Problems(MDPs)[10]. The previous approach to this problem was And the strategy is used widely nowadays in selection of each node to be expanded from its easiness to implement. From the two primary aspects of Monte Carlo planning algorithms, which is (1) small error of probability even if the algorithms is halted prematurely, and the returning the best action if enough time is given. The UCT Algorithm outline is here, In a bandit problem, K actions are defined in accordance with the set of random payoffs $X_{it}, i = 1, ...K, t \geq 1$, where each i is the index of a game. And this algorithm, keeps track of the average rewards $X_{i,Ti(t-1)}$ for all the arms and the distribution function is bounded by the upper confidence.

$$I_t = argmax_{i \in i,..K} X_{i,Ti(t-1)} + c_{t-1,Ti(t-1)}$$

where $C_{t,s}$ is a bias sequence.

$$C_{t,s} = \sqrt{\frac{2ln(t)}{s}}$$

Even though, their theoretical analysis is consistent in the optimal action as the sample number grows to infinity, in the experiment of sailing domain they found out the significant error level.

7 PBBM

As one of the other selection part strategy of Monte Carlo, in 2006 Coulom developed PBBM(Probability to be Better than Best Move)[8]. In their paper their attempt of a new framework is combining Min-max tree search algorithm into it by not backing up the min-max value close to the root, the average value at each depth like a Min-Max, but by general backup operator, which is defined below. Most of the monte carlo algorithm rely on the central limit theorem that is approached a normal distribution with mean μ and variance. In their progressive purning, the standard deviation of the best move is taken into account. However, it is very insecure in tree search, because the payback of random simulations are not identically distributed as the search tree expanded, move probabilities are changed. For the sake of dodging the dangers of completely pruning a move, it must be considered to allocate the reduced probability of exploring a bad move, instead of cutting off the move which is supposed to bad move at this moment.

$$u_i = exp(-2.4 \frac{\mu_0 - \mu_i}{\sqrt{2(\sigma_0^2 - \sigma_i^2)}}$$

where μ_0 and σ_0 are the value and standard deviation of the best move, respectively. Similarly, μ_i and σ_i are the value and the standard deviation of the move under consideration.

8 Objective Monte Carlo

Objective Monte Carlo developed in 2006 by Chaslot et al[5]. Objective Monte Carlo consists of two part, the first one is a move selection strategy and the next one is a new back propagation strategy.
In the first part, suppose Vm is the current evaluation of the move m, and σ is the standard deviation of Vm. They defined the probability of the move m to be superior to O_{bj} as

$$U_m(O_{bj}) = \frac{\sum_{i=O_{bj}}^{\infty} D_m(x)}{\sum_{i=-\infty}^{\infty} D_m(x)}$$

where,

$$D_m(S) = N(V_m, \frac{\sigma}{\sqrt{n}})$$

In the second back propagation part, their strategy returns the value relying of standard deviation of the move m to the value of move m. the Min-Max values measured by all child node in a max of random number, so it is overestimated in a sence. To avoid this error, the value returned by the $U_m(O_{bj})$ represent the urgency of the a move, the value estimated by the back propagation strategy should be close to the average value of the child node in the beginning of the experiments. As a consequence it does not require any evaluation function in the usage of this selection strategy and the backpropagation strategy. Therefore, this is applicable to any game where it is difficult or impossible to create an evaluation function without parameter tuning.

9 Parallel Algorithm

A parallel Monte Carlo Tree Search Algorithm is provided Tristan and Nicolas to UCT algorithm[4]. They provide Mater Slave algorithm fro MCTS and With a single slave and five seconds per move our algorithm scores 40.5% against GNU Go, with sixteen slaves and five seconds per move it scores 70.5%. The master process is responsible for descending and updating the UCT tree. The slaves do the playouts that start with a sequence of moves sent by the master. The master starts sending the position to each slave. Then it develops the UCT tree once for each slave and sends them an initial sequence of moves.The slave process loops until the master stops it with an END GAME message, otherwise it receives the board, the color to play

10 Conclusion

As the field of AI increases, the prevalence of Monte Carlo implementations are increasing.. All versions, while having their unique qualities that improve its performance, still are for the most part are the same in their philosophy. Such a powerful idea that it can be implemented in most situations where other AIs would fail show that Monte Carlo method is the preferred method for games that are complex and dynamic.

References

[1] Bruce Abramson. Expected-outcome: A general model of static evaluation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(2):182–193, 1990.

[2] Bruno Bouzy and Bernard Helmstetter. Monte-carlo go developments. In *Advances in computer games*, pages 159–174. Springer, 2004.

[3] Bernd Brügmann. Monte carlo go. Technical report, Citeseer, 1993.

[4] Tristan Cazenave and Nicolas Jouandeau. A parallel monte-carlo tree search algorithm. In *Computers and Games*, pages 72–80. Springer, 2008.

[5] GMJB Chaslot, Jahn-Takeshi Saito, Bruno Bouzy, JWHM Uiterwijk, and H Jaap Van Den Herik. Monte-carlo strategies for computer go. In *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91, 2006.

[6] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008.

[7] Guillaume MJ-B Chaslot, Mark HM Winands, and H Jaap van Den Herik. Parallel monte-carlo tree search. In *Computers and Games*, pages 60–71. Springer, 2008.

[8] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and games*, pages 72–83. Springer, 2006.

[9] Justin Fu and Irving Hsu. Model-based reinforcement learning for playing atari games.

[10] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.