

The variation and progress of Monte carlo

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

April 27, 2016

1 Introduction

In this article, we explore the various implementations of Monte Carlo Simulations and their impact within the environment they are used. Some of these implementations only change at a structure level, while others change at a high level to fit the needs of a specific problem. These implementations include Model Based Monte Carlo Search for Playing Atari Games, Parallelized Monte Carlo Tree Search, Monte Carlo Planning for Real Time Strategy Games, [insert kohei topics]

2 Expected-outcome

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 difficult 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.

3 Monte Carlo evaluation by average score

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.

4 Partially knowledge based system

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.

5 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.

6 Simulation Based Approach to General Game Playing

Because of nature of Monte Carlo Simulations not being reliant on game dependent mid-state evaluators to obtain heuristics, this approach can be implemented in various games. A modification to Monte Carlo Search Trees simulation phase, Upper Confidence-bounds applied to Trees or UTC

for short, is introduced in a General Game Playing Competition and won in 2007. [10] This approach modifies the Selection aspect such that it gives a good balance between exploration and exploitation of nodes. To do this, not only is the win rate of each possible action at the current game state kept track, but also the amount of times that action is visited. This allows the AI to prioritize its simulations elsewhere if the action has already been simulated multiple times, allowing for the discovery of better choices in a global sense rather than just focusing on the local best choice, and balances it with the limited time and memory that the AI has to simulate and retrieve statistics for each possible move. GGP consists of a variety of games, each with their own level of computational difficulty, and thus it is important that the algorithm be able to prioritize its selection of action nodes properly such that it won't need new hardware or more time.

7 Monte Carlo Planning in RTS Games

While the assortment of games may present Monte Carlo a good approach for game AIs it still only covered games that are for the most part, turn based. In Real Time Strategy, there is no turn based action, and all actions of players can be done regardless of each other. Many of the supposed AIs for RTSs are mainly scripts, resulting in players wanting to play against other human players since the AIs are considered beginner level. An implementation of Monte Carlo idea, Monte Carlo Planning can handle RTS. It follows the same approach as Monte Carlo Tree search, but instead of searching nodes, it searches plans that are randomly generated and then evaluated via repeated simulation and backtracking. Each plan is one of three types of plans: unit control, combat planning, and strategic planning. [8] These plans are randomly generated and simulated forwards to see which one gives the best win rate. The benefit of using this is that it does not have to rely on perfect information. Most AI need to rely on either the heuristic to guess the optimal solution, or have perfect information of the game state to evaluate, which is extremely hard in multiplayer RTS games, sometimes even impossible. Planning via Monte Carlo does not have to rely on that because it relies on statistics and randomness to sample the game state for the best plans.

8 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. [11] At long distances, the paddle won't 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.

9 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.

10 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)[12]. 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 $X_{it}, i = 1, \dots, K, t \geq 1$, 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.

$$I_t = \operatorname{argmax}_{i \in 1, \dots, K} X_{i,Ti(t-1)} + c_{t-1,Ti(t-1)}$$

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

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

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.

11 PBBM

As one of the other selection part strategy of Monte Carlo, in 2006 Coulom developed PBBM(Probability to be Better than Best Move)[9]. 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 be a bad move at this moment.

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

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.

12 Objective Monte Carlo

Objective Monte Carlo developed in 2006 by Chaslot et al[5]. Objective Monte Carlo consists of two parts, the first one is a move selection strategy and the next one is a new back propagation strategy.

In the first part, suppose V_m is the current evaluation of the move m , and σ is the standard deviation of V_m . 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 on standard deviation of the move m to the value of move m . The Min-Max values measured by all child nodes in a max of random number, so it is overestimated in a sense. To avoid this error, the value returned by the $U_m(O_{bj})$ represents the urgency of the move, the value estimated by the back propagation strategy should be close to the average value of the child nodes 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.

13 Parallel Algorithm

A parallel Monte Carlo Tree Search Algorithm is provided by Tristan and Nicolas to UCT algorithm[4]. They provide Master Slave algorithm from 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

14 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] Michael Chung, Michael Buro, and Jonathan Schaeffer. Monte carlo planning in rts games. In *CIG*, 2005.
- [9] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and games*, pages 72–83. Springer, 2006.
- [10] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *AAAI*, volume 8, pages 259–264, 2008.
- [11] Justin Fu and Irving Hsu. Model-based reinforcement learning for playing atari games.
- [12] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.