Upper Confidence Trees for Game AI

Chahine Koleejan

***Abstract*—Monte-Carlo methods are a branch of algorithms which are starting to bridge the gap between computer programs and top human players in complex games such as Go. This project investigates different parameters in the Upper Confidence Tree algorithm, a popular variant of Monte Carlo Tree Search. Experiments are run in the form of two programs using Monte Carlo Tree Search to select moves facing each other in the games of Othello and Gomoku. The results show that certain parameters clearly affect performance while the impact of others needs to be further investigated.**

I. Introduction

For decades, classic two-player games have been considered an excellent platform for testing AI algorithms[1]. They provide a closed testbed with simple rules refined in order to challenge humans. They also provide clear benchmarks for evaluating an algorithm's performance against other algorithms or against human intelligence.

A recently developed branch of algorithms which has revolutionised game AI[2][3] is that of Monte-Carlo Tree Search(MCTS) algorithms. These build and expand a search tree while evaluating the effectiveness of individual moves by simulating random games. A highly successful variant of MCTS is the Upper Confidence Tree(UCT) algorithm, which applies the Upper Confidence Bound(UCB) to an MCTS algorithm.

This project aims to investigate the effect of changing different parameters in the UCT algorithm. Experiments are run in the form of two computer programs playing against each other, each using a UCT algorithm to select moves.

The games chosen for this project are Othello and Gomoku and the parameters being investigated are the maximum depth of the search tree, the ratio of exploration to exploitation, the number of random simulations and the use of decisive/anti-decisive move selection.

II. Background

A. Monte Carlo Tree Search

Algorithm 1 General MCTS

Input: s_0

Create root node v_0 with state s_0
while Stopping criterion is not met **do**
  v_1 = TreePolicy(v_0)
  $delta$ = DefaultPolicy(s(v_0))
  Backup(v_1,delta)
return move(BestChild(v_0))

The MCTS algorithm involves growing a game tree where each node in the tree corresponds to a single state of the game and the root node corresponds to the state at which the search begins.

MCTS can be described by four phases[4], repeated until a stopping criterion is met, such as a limit on the computation time or on the number of random simulations. These phases are selection, expansion, simulation and backpropagation.

1) Selection: Starting at the root node, the tree is recursively traversed while applying a selection policy until an expandable node is reached. An expandable node is one that is nonterminal and has children which do not belong to the tree.

2) Expansion: A child node is added to the tree according to the available actions. In the basic algorithm a random child is selected.

3) Simulation: A simulation is run from the expanded node according to default policy until an outcome is observed. In the basic algorithm the default policy is simply selecting a random action from the set of legal moves.

4) Backpropagation: The result of the simulation is backpropagated up the selected nodes of the tree in order to update their statistics, typically their number of visits and number of wins.

B. Upper Confidence Trees

Greedy move selection may not be an effective way to construct a search tree as this will typically result in avoiding moves after a few poor outcomes, even if there remains significant uncertainty about the value of these moves. The problem of how to select moves can be modeled as stochastic action selection problems known as multi-armed bandit problems[5].

The UCT algorithm aims to apply the principle of *optimism in the face of uncertainty*[6] by maximising an upper confidence bound on the value of moves. This is done by adding an exploration term to the average reward or the exploration term of the move thus far. This exploration term is highest for action-state pairs which have been the least visited. One commonly used confidence bound is the UCB1[7] bound shown below.

$$UCB1(j) = \bar{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

For a child node j, n_j is the number of times j has been visited and n is the number of times the parent node has been visited. $\bar{X}_j$ is the average reward of j from previous visits and C_p is some constant greater than 0.

The UCB1 bound provides a balance between exploitation of successful moves thus far and the exploration of moves

TABLE I
DEFAULT PARAMETER SETTINGS

Parameter	Value
Number of random simulations	200
C_p	$1/\sqrt{2}$
Maximum depth of search tree	100
Exploration factor	0.5
Decisive/antidecisive moves	false

which have been rarely visited, as each time a node is visited its exploration term decreases.

Provided the number of simulations is large enough, the value of the UCB1 bound converges towards the minimax action values. [8][9]

C. Games

1) Othello: Othello[1] is a two-player board game, played on an 8x8 board. Players take turns placing disks of their assigned color, black or white, on the board. During a play, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are converted to the current player's color.

The goal of the game is to have the majority of disks turned to display your color when the last playable empty square is filled.

2) Gomoku: Gomoku[2] is also a two-player board game where players have colors black and white, typically played on a 19x19 board. Players take turns placing stones of their assigned color on the board. The goal of the game is to have a row of consecutive stones of your color, either horizontally, vertically or diagonally.

III. EXPERIMENTAL DESIGN

A. Default parameter settings

The default parameter settings are shown in Table I. These are used unless the parameter in question is being investigated.

B. Parameters being investigated

1) Maximum search tree depth: This is the maximum allowed depth of a node in the search tree. The depth of a node is the minimum number of of edges it takes to get to it from the root node. The values tested range from 1 to 7.

2) Exploration factor: This is a measure of how much emphasis is put on exploitation of rarely visited moves compared to the exploration of previous successful moves. For an exploration factor of x, the exploitation factor is (1-x). These values are multiplied by their respective terms inthe UCB1 bound at the time of node selection. The values tested range from 0, corresponding to pure exploitation to 1, corresponding to pure exploration.

3) Number of rollouts: The number of random games the algorithm simulates every turn before selecting a move. The values tested range from 1 to 1000.

[1]https://en.wikipedia.org/wiki/Reversi

[2]https://en.wikipedia.org/wiki/Gomoku

4) Decisive and antidecisive moves: A decisive move is defined as one that leads immediately to a win, and an antidecisive move is one that prevents the opponent from making a decisive move on their next turn. Having this boolean parameter turned on means that prior to using the standard policy, the algorithm checks whether either player has a decisive move and if so, plays it.

C. Performance Measures

1) Win percentage: The most important performance measure is how often the program achieves its goal of winning the game. This is measured against a constant opponent when varying a parameter.

2) Winning margin: Applicable to games which use a point system to determine a winner, in this project Othello. This is the difference between the winning player's final score and the losing player's final score. This is only considered in cases where the program being tested wins the game. The higher the winning margin, the better the performance.

3) Number of moves: Applicable to games with a winning condition, in this case Gomoku, this is simply the number of moves it takes for the winning player to win. The lower the number of moves taken, the better the performance.

IV. RESULTS AND DISCUSSION

The results are presented in tables II to VIII as well as in Figures 1 to 4. All results come from experiments run on Othello, other than table VIII which refers to Gomoku. Other experiments run on Gomoku follow a similar trend as those presented in this report.

The values in the tables are from the point of view of the program with parameter in the first column of the table. The values presented are averages of 1000 runs in most cases. In certain cases, such as the number of rollouts being high, the values are averages of 200 runs.

From Figure 1, it is clear that as the number of random simulations run increases, so does the strength of the program. Against all opponents, increasing the number of simulations improved the win percentage. This is what one would expect as more simulations being run means the algorithm gets to observe more outcomes for the possible moves and gets a better estimate of their true value, eventually leading to better moves being selected in general.

By the same reasoning the winning margin also improves as the number of simulations increases. Figure 2 presents the score difference instead of the margin so as to demonstrate that even if the program loses, the amount it loses by decreases as the number of simulations is increased.

The experimental results fail to show a clear trend when it comes to the effect of the maximum depth of the search tree, presented in Figure 3.

The effect of the exploration factor is shown in Figure 4. Against all opponents, the trend is rougly in the shape of a bell curve, in most cases peaking at the halfway mark. This shows both exploration and exploitation are essential for good performance, and that an approximately even balance of the

TABLE II
WIN PERCENTAGES FOR VARYING NUMBER OF ROLLOUTS

	10	50	100	200	500	750
50	86.9	-	-	-	-	-
100	93.4	66.8	-	-	-	-
200	97.3	82.9	66.8	-	-	-
500	98.9	94.1	78.0	70.0	-	-
750	99.0	97.0	91.0	84.0	63.0	-
1000	99.0	96.0	95.0	91.0	65.0	66.0

TABLE III
WIN MARGINS FOR VARYING NUMBER OF ROLLOUTS

	10	50	100	200	500	750
50	20.2	-	-	-	-	-
100	21.9	17.7	-	-	-	-
200	24.8	21.4	19.2	-	-	-
500	27.4	24.0	24.3	22.9	-	-
750	30.4	25.0	23.4	22.6	17.8	-
1000	28.8	28.0	24.9	23.9	19.0	15.9

TABLE IV
WIN PERCENTAGES FOR VARYING MAXIMUM DEPTHS OF THE SEARCH TREE

	1	2	3	4	5	6
2	52	-	-	-	-	-
3	62	52	-	-	-	-
4	52	62	52	-	-	-
5	58	59	63	49	-	-
6	63	54	55	47	46	-
7	63	54	46	48	59	52

TABLE V
WIN MARGINS FOR VARYING MAXIMUM DEPTHS OF THE SEARCH TREE

-	1 2	2	3	4	5	6
2	14.8	-	-	-	-	-
3	14.6	16.5	-	-	-	-
4	16.5	16.1	17.5	-	-	-
5	15.0	15.3	16.9	17.8	-	-
6	17.9	15.2	14.4	14.9	14.3	-
7	17.0	14.9	14.8	17.6	16.9	16.7

two is optimal. Conversely the figure also shows that focusing on only one of the two aspects results in a very poor win percentage.

Table VIII presents the results of running a program which played decisive moves whenever available against one which didn't consider them. The results are overwhelmingly in favor of using such moves, with a perfect win record when doing so. The average number of moves taken to win a game is also significantly reduced when using decisive moves.

V. CONCLUSION

The investigation showed that as one would expect, increasing the number of random simulations run by the MCTS algorithm increases the strength of a program. It also increases the winning margin in games which a score-based system.

The investigation also showed that a balance between exploitation and exploration is required for good performance.

The use of decisive and antidecisive moves was shown to drastically increase the strength of a program in games with a winning condition.

TABLE VI
WIN PERCENTAGES FOR VARYING EXPLOITATION TO EXPLORATION RATIOS

	0	0.2	0.4	0.5	0.6	0.8
0.2	72	-	-	-	-	-
0.4	69	64	-	-	-	-
0.5	76	61	51	-	-	-
0.6	75	46	48	46	-	-
0.8	72	43	37	34	41	-
1	11	11	1	2.5	0	2

TABLE VII
WIN MARGINS FOR VARYING EXPLOITATION TO EXPLORATION RATIOS

	0	0.2	0.4	0.6	0.8
0.2	17	-	-	-	-
0.4	20.9	16.2	-	-	-
0.6	25.72	17.3	17.8	-	-
0.8	19	17.5	19.5	15.8	-
1	26.5	31.1	16.0	/	41

TABLE VIII
EFFECT OF USING DECISIVE AND ANTIDECISIVE MOVES

	On	Off
Win%	100	50
Number of moves taken to win	44	220

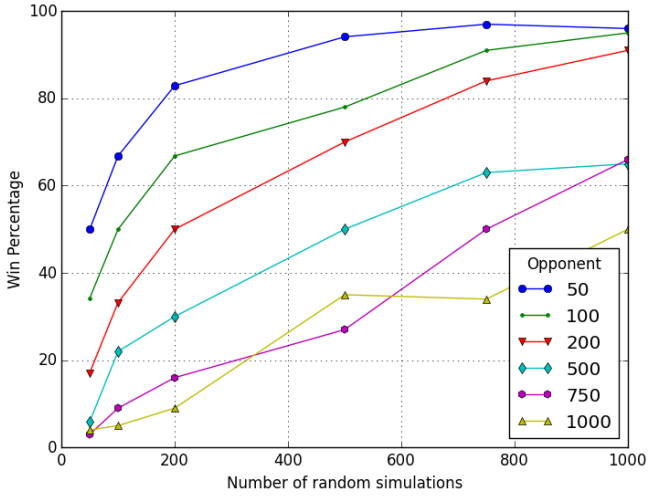

Fig. 1. Number of random simulations vs Win Percentage against an opponent with a constant number of random simulations

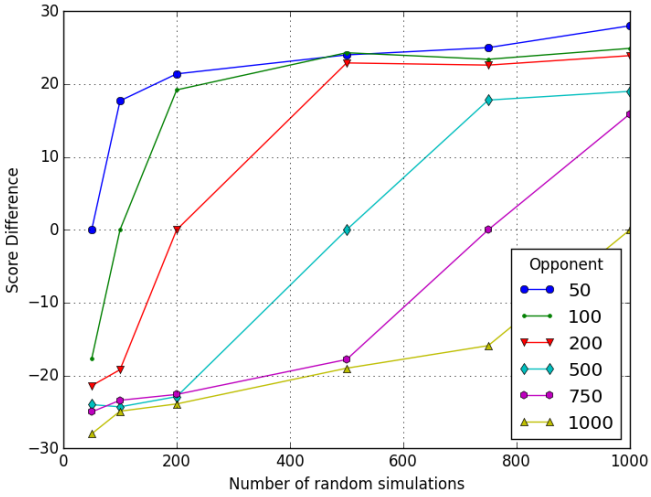

Fig. 2. Number of random simulations vs Score Difference against an opponent with a constant number of random simulation

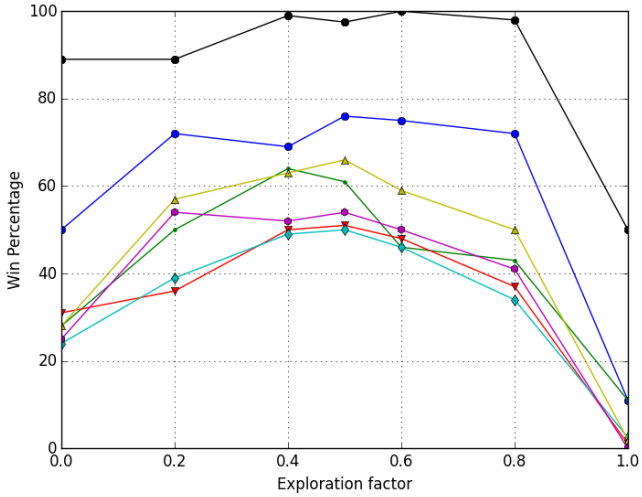

Fig. 4. Exploration Factor vs Win Percentage against an opponent with a constant exploration factor

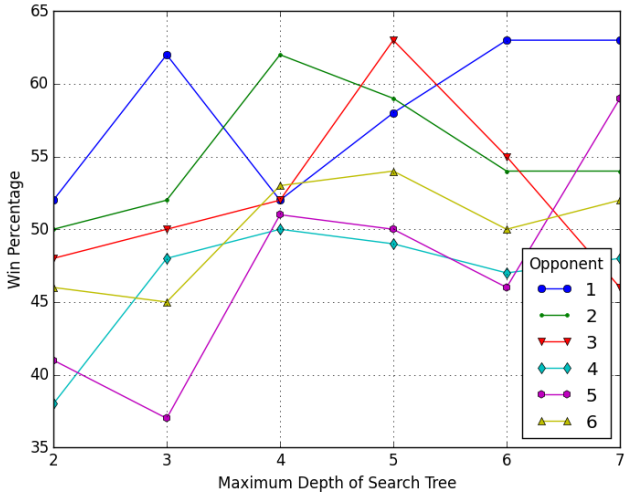

Fig. 3. Maximum Depth of Search Tree vs Win Percentage against an opponent with a constant maximum depth

The effect of the depth of the search tree is currently unclear and remains to be further investigated.

A. Future work

This investigation used the games of Othello and Gomoku as testbeds for the experiments. These are relatively simple games and the strength of top computer programs far exceeds that of top humans. In the future a more complicated game such as Go, where simply increasing the number of simulations or the size of search tree is not a reasonable option due to the size of the search space being too large, will be used.

More parameters, including more complicated ones, than those in this project could be investigated. Examples of these are different tree policies than the UCB1 bound, selection enhancements such as heuristics, different search-tree representations such as those considering move transpositions, move pruning and Rapid Action-Value Estimation, a recent and very promising extension to MCTS.

To get a better idea of the performance of the algorithms, further benchmarks could be considered. This could include performance against human players or other programs whose strength can be quantified, for example using a rank, commonly used in games such as Chess and Go.

REFERENCES

[1] Sylvain Gelly, David Silver, Csaba Szepesvari. The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions, 2008

[2] Cameron B. Browne and co. A survey of Monte Carlo Tree Search Methodsl. In *IEEE Transactions on Computational Intelligence and AI in Games*, 2012, Vol.4, No.1

[3] Sylvain Gelly, David Silver. Monte-Carlo tree search and rapid action value estimation in computer Go In *Artificial Intelligence 175*,2011

[4] G. M. J. -B. Chaslot, S.Bakkes, I.Szita, and P. Spronck. Monte-Carlo tree search: a new framework for Game AI In *Proc. Artif. Intell. Interact. Digit. Entertain. Conf.*,2008, pp. 216-217

[5] H.Robbins. Some aspects of the sequential design of experiments. In *Bulletin of the American Mathematics Society*,1952, 58:527-535

[6] T. L. Lai and H.Robbins. Asymptotically efficient adaptive allocation rules In *Advances in Applied Mathematics*,1985, 6:4-22

[7] P.Auer, N. Cesa-Bianchi, P.Fischer. Finite-time analysis of the multi-armed bandit problem In *Mach. Learn*,2002, Vol. 47, no.2

[8] L.Kocsis and C. Szepesvari. Bandit based Monte-Carlo planning In *Proc. Eur. Conf. Mach. Learn.*,2006, pp. 282-293

[9] L.Kocsis, C. Szepesvari and J. Willmenson. Improved Monte-Carlo Search, 2006