Monte-Carlo Approximation of Temperature

Tristan Cazenave

LAMSADE
Université Paris-Dauphine
Paris France
cazenave@lamsade.dauphine.fr

Abstract. Monte-Carlo tree search is a powerful paradigm for the game of Go. We propose to use Monte-Carlo tree search to approximate the temperature of a game, using the mean result of the playouts. Experimental results on the sum of five 7x7 Go games show that it improves much on a global search algorithm.

1 Introduction

Monte-Carlo Go has recently improved to compete with the best Go programs [6–8]. We are interested in the use of Monte-Carlo methods when there are independent games. In such cases it might be interesting to analyze the games independently instead of considering them as a unified game.

Section 2 describes related works. Section 3 presents the Monte-Carlo algorithms we have tested. Section 4 details experimental results. Section 5 concludes.

2 Related Works

In this section we expose related works on Monte-Carlo Go. We first explain basic Monte-Carlo Go as implemented in GOBBLE in 1993. Then we address the combination of search and Monte-Carlo Go, followed by the UCT algorithm, and previous works on the approximation of temperature.

2.1 Monte-Carlo Go

The first Monte-Carlo Go program is GOBBLE [3]. It uses simulated annealing on a list of moves. The list is sorted by the mean score of the games where the move has been played. Moves in the list are switched with their neighbor with a probability dependent on the temperature. The moves are tried in the games in the order of the list. At the end, the temperature is set to zero for a small number of games. After all games have been played, the value of a move is the average score of the games it has been played in first. GOBBLE-like programs have a good global sense but lack of tactical knowledge. For example, they often play useless Ataris, or try to save captured strings.

2.2 Search and Monte-Carlo Go

A very effective way to combine search with Monte-Carlo Go has been found by Rémi Coulom with his program CRAZY STONE [6]. It consists in adding a leaf to the tree for each simulation. The choice of the move to develop in the tree depends on the comparison of the results of the previous simulations that went through this node, and of the results of the simulations that went through its sibling nodes.

2.3 UCT

The UCT algorithm has been devised recently [9], and it has been applied with success to Monte-Carlo Go in the program MOGO [7, 8] among others.

When choosing a move to explore, there is a balance between exploitation (exploring the best move so far), and exploration (exploring other moves to see if they can prove better). The UCT algorithm addresses the exploration/exploitation problem. UCT consists in exploring the move that maximizes $\mu_i + C \times \sqrt{log(t)/s}$. The mean result of the games that start with the c_i move is μ_i, the number of games played in the current node is t, and the number of games that start with move c_i is s.

The C constant can be used to adjust the level of exploration of the algorithm. High values favor exploration and low values favor exploitation.

2.4 Thermography

Thermography [2] can be used to play in a sum of combinatorial games. In Go endgames, it has already been used to find better than professional play [12], relying on a computer assisted human analysis. A simple and efficient strategy based on thermography is Hotstrat, it consists in playing in the hottest game. Hotstrat competes well with other strategies on random games [4], but it can be improved taking into account the subgame type [1]. Another approach used to play in a sum of hot games is to use locally informed global search [11, 10]. Our previous work on evaluating the temperature evaluated goals temperature on a single board using a Monte-Carlo method [5]. In this paper, we either use Hotstrat or Minimax to evaluate the subgames built for each separate board.

3 Search Algorithms

In this section, we present the different algorithms we have tested. They all use the score of a playout for UCT instead of the usual probability of winning because the difference in points is meaningful to approximate the temperature.

3.1 Global Search

The direct application of UCT to playing on several boards is to do a global search. A move can be made on any board, the normal UCT tree is developed, and the playout are played separately on each separate board. In our implementation of global search, the color to play after the UCT tree descent starts the playout on the first board, then when the game on the first board is over, the other player starts the playout of the second board, and so on up to the completion of all the playouts on all boards.

3.2 Dual Search

Dual search consists in performing two UCT searches on each separate board. The first one always starts with Black, and the second one always starts with White. In each search, after the descent of the UCT tree, a normal playout is played on the separate board. Each search is allocated the same number of playouts. At the end of the searches, the program knows the mean value of the playouts starting with a Black move (μ_{Black}), and the mean value of the playouts starting with a White move (μ_{White}). The temperature of the board is approximated with $(\mu_{Black} + \mu_{White} - size \times size)/2$.

The player to move chooses to play the best UCT move of the board with the greatest approximated temperature.

3.3 Threat Search

Threat search consists in performing four UCT searches on each separate board. The first one always starts with Black, and the second one always starts with White. After the first search is completed, Black knows the best UCT move. The third search always starts with the best Black move and it is folowed by the descent of an UCT tree that also starts with a Black move (so all the playouts starts with two Black moves). The fourth search is the equivalent for White of the third search. Each search is allocated the same number of playouts. At the end of the searches, the program knows the mean value of the playouts starting with a Black move, the mean value of the playouts starting with a White move, the mean value of the playouts starting with two Black moves, and the mean value of the playouts starting with two White moves. It either use these values to compute the temperature with HotStrat, or to perform a Minimax search on all the values of all the boards.

The player to move chooses to play the best UCT move of the board with the greatest approximated temperature, or the best UCT move returned by Minimax.

4 Experimental Results

The random games are played using the same policy as in MOGO [7]. We tested the algorithm on a game composed of five 7x7 boards. The komi is set to 7.5 points. Therefore the maximum number of points is 252.5 for White, and 245.0 for Black.

In table 1 the results of games between the global search algorithm and the dual search algorithm are given. The algorithms use the 0.3 UCT constant. For each algorithm, the table gives the mean number of points against the approximation algorithm.

Table 1. Results of the global search program against the dual search program

Size	Playouts	Black	mean number of Black points	White	mean number of White points
$5 \times$ 7x7	1,000	Dual	199.21	Global	53.28
$5 \times$ 7x7	1,000	Global	47.13	Dual	205.37

In table 2 the results of games between the threat search algorithm and the dual search algorithm are given.

Table 2. Results of the threat search program against the dual search program

Size	Playouts	Black	mean number of Black points	White	mean number of White points
$5 \times 7x7$	1,000	Threat	127.03	Dual	125.45
$5 \times 7x7$	1,000	Dual	120.50	Threat	131.96

The threat search algorithm is twice slower as the dual search algorithm. We played the dual search algorithm against another dual search algorithm that is twice slower in order to compare with the threat search algorithm. Table 3 gives the results of games between the two dual search algorithms. Results of the dual threat algorithm with 2,000 playouts are similar to the results of the threat search algorithm with 1,000 playouts and they take the same time.We also give in table 3 the result of the dual algorithm with 2,000 playouts against the threat algorithm with 1,000 playouts. The dual algorithm has a clear win. Using the threat search algorithm is more complicated and gives worse results than the more simple dual search algorithm.

Table 3. Results with different numbers of playouts

Size	Black	mean number of Black points	White	mean number of White points
$5 \times 7x7$	Dual(1,000)	118.39	Dual(2,000)	134.10
$5 \times 7x7$	Dual(2,000)	128.13	Dual(1,000)	124.34
$5 \times 7x7$	Threat(1,000)	118.58	Dual(2,000)	133.86
$5 \times 7x7$	Dual(2,000)	131.79	Threat(1,000)	120.68

In the previous experiment, the threat search algorithm uses HotStrat to choose the board to play, in order to test if HotStrat was a potential problem, we replaced it with a Minimax on the tree composed of the first two moves for Black and for White for all the boards. Minimax gave results very similar to HotStrat. The behavior of the threat search algorithm is not due to HotStrat.

5 Conclusion

When a game is composed of independent games, it is better to approximate the temperature using seperate Monte-Carlo tree searches on each game than using a global Monte-Carlo search. When we tested the algorithms that evaluate threats, we obtained results comparable to the more simple dual search algorithm.

References

1. Cherif R. S. Andraos, Manal M. Zaky, and Salma A. Ghoneim. Comparative study of approximate strategies for playing sum games based on subgame types. In *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 212–219. Springer, 2006.
2. E. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways*. Academic Press, 1982.
3. B. Bruegmann. Monte-Carlo Go. Technical report, 1993.
4. T. Cazenave. Comparative evaluation of strategies based on the value of direct threats. In *Board Games in Academia V*, Barcelona, Spain, 2002.
5. T. Cazenave. Goal threats, temperature and monte-carlo go. In *Games of no chance 3*, Banff, Canada, 2005.
6. R. Coulom. Efficient selectivity and back-up operators in monte-carlo tree search. In *Computers and Games 2006*, Volume 4630 of LNCS, pages 72–83, Torino, Italy, 2006. Springer.
7. S. Gelly, Y. Wang, R. Munos, and O. Teytaud. Modification of UCT with patterns in monte-carlo go. Technical Report 6062, INRIA, 2006.
8. Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *ICML*, pages 273–280, 2007.
9. L. Kocsis and C. Szepesvàri. Bandit based monte-carlo planning. In *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2006.
10. M. Müller, M. Enzenberger, and J. Schaeffer. Temperature discovery search. In *AAAI 2004*, pages 658–663, San Jose, CA, 2004.
11. M. Müller and Z. Li. Locally informed global search for sums of combinatorial games. In *Computers and Games 2004*, LNCS, pages 273–284, Ramat-Gan, Israel, 2004. Springer.
12. W. Spight. Go thermography - the 4/21/98 jiang-rui endgame. In R. Nowakowski, editor, *More Games of No Chance*, pages 89–105. Cambridge University Press, 2002.