Tree Search vs. Monte-Carlo in Go for Small Boards

Team 1

Introduction

The game of Go is a conundrum in the field of artificial intelligence. The number of rules and types of pieces are smaller than those of chess or even checkers, games which have both seen the successful application of various AI tree search, pruning and heuristic techniques. However, computer agents for Go are currently unable to compete with strong amateur players on the full-sized board or regularly defeat professional players on even the smallest board. This is because several aspects of Go make traditional tree search techniques impracticable for Go. These elements include the huge branching factor of the board, the percentage of potential moves which are legal and the non-monotonic count of pieces on the board.

We chose to focus on this problem as it has remained unsolved since the first computer Go agent was devised by Albert Zobrist in 1970 [1]. In particular, we focus on solving games on small sized boards, where the branching factor is more manageable. In this paper, we develop a tree search algorithm to play go, which we test against the state-of-the-art Monte-Carlo algorithms.

Related Work

Despite the challenges faced by classical approaches, the field of computer Go has seen dramatic advances since the use of Monte-Carlo techniques to evaluate the best next move was first proposed in 2006 [2].

This approach for computer Go has several weaknesses, however. For example, it is always possible in a Monte-Carlo simulation, which relies on random sampling, that a clear best move is simply not contemplated. It follows from this that these agents can fail to pick the best sequence of play especially when the order of moves is important, which is quite often the case in Go. Furthermore, each position is represented individually, and Monte-Carlo agents make no attempt to represent relations between related positions. This means that these approaches are entirely depended on a random sampling to explore the gamespace.

Some researchers have partially mitigated these weaknesses with hybrid algorithms, such as UCT-RAVE, which "forms an online generalization between related positions, using the average outcome of each move…combining this rapid but biased estimate of a move's value with the slower but unbiased Monte-Carlo estimate" [3]. However, even UCT-RAVE, like the basic Monte-Carlo method it is based on, is not exhaustive and can be shown to overlook an optimal play if the random sampling is poor.

Approach

We use mature algorithms to develop four autonomous Go playing agents and pit them against each other using the online match making service "KGS Go." This service allows people from around the world to play Go against one another over the internet. Our players log onto the Go server with a screen name just like a person would, enter a game, and start playing. (As an aside, this could be interesting to set up as a Turing test, since the server provides a human opponent no indication that he or she is playing against a computer.) We test the tree-search algorithm's effectiveness versus Monte-Carlo by having them play a series of 10 games against one another on 7 by 7 board.

The four players are outlined below:

1) Random: This player makes the first legal move it can find at random. It was developed as an incremental step towards a more sophisticated player, but proved to be an interesting sparing-partner simply because it is non-deterministic and never plays the same

game twice. Two heuristic tree-search algorithms playing against each other will make identical moves every game.

2) Minimax: This player uses simple adversarial planning as described in the textbook [4]. Every turn, the player builds an exhaustive tree of potential moves and the opponent's potential replies, up to a certain preset "look-ahead depth," and then selects the move leading to the maximum-value leaf node. This player relies heavily on an evaluation function to determine the relative value of different plays. We chose to evaluate the board as the difference in scores each player would earn if the game ended immediately using Chinese scoring, as shown below.

$$Value = 378 + BlackScore - WhiteScore - 2*komi \quad \text{[Eq. 1]}$$

BlackScore and WhiteScore are the total number of stones played, plus the territory controlled by the players. This player is able to play intelligently, but is quite slow since the tree's branching factor is absurdly large (up to $boardSize^2$). We were not able to search deeper than three-moves into the future, primarily due to speed constraints.

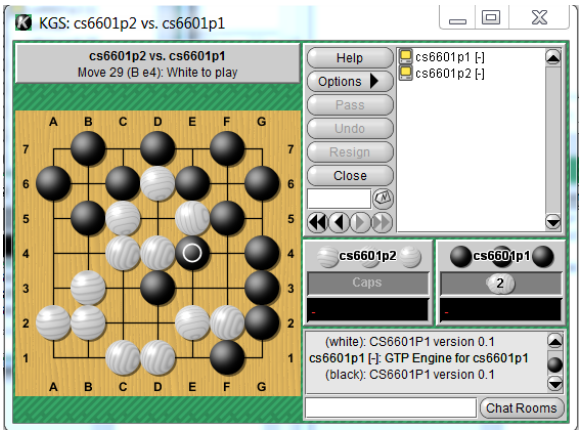
3) Alpha-Beta: This player is identical to the minimax player in terms of overall strategy, but attempts to search deeper by pruning off branches that are guaranteed not to lead to a high-value leaf node. This is done by keeping track of the current minimum and maximum values encountered during the search so far and not searching any sub-trees which are dominated by already searched space. This allows the algorithm to search deeper than simple minimax, and search faster. We set ours to 6 plies.

4) Monte-Carlo: This is the algorithm currently used by state-of-the-art computer Go players [2], despite being deceptively simplistic. It makes X random moves at Y levels of recursion and then selects the "best" one according to the value function above [Equation 1]. We tried many values for the X and Y parameters before settling on 3 recursive plays and 10 plays per level. This algorithm is very fast, making it appealing for commercial application.

Evaluation

We focused on the matches between the Alpha-Beta player and the Monte-Carlo player. This is because minimax is strictly worse than Alpha-Beta since it is slower, and the random opponent did not win any games, unsurprisingly.

Qualitatively, we found that Alpha-Beta will make tactically solid defensive formations, visible as the diamond shapes in the upper portion of this screenshot. Monte-Carlo tends to explore more of the board and is quicker to play the center, but fails to develop solid defensive structures.



Quantitatively, we show the results of our ten game series below.

Game Number	Monte Carlo Score	Alpha Beta Score	Starting Player
1	20	29	AB
2	5	37	MC
3	15	34	MC
4	18	29	AB
5	14	35	AB
6	17	30	MC
7	0	49	AB
8	10	38	MC
9	14	35	MC
10	13	36	AB

Table 1: Monte-Carlo vs Alpha-Beta

Note that Alpha-Beta won every single game, regardless of playing first or second. This supports our hypothesis that tree-search strategies will outperform the state-of-the-art Monte-Carlo on small boards.

There is another interesting metric that isn't contained in the table – the average amount of time it takes each algorithm to make a move. This was impossible to quantify given our experimental setup, since we rely on an internet based client-server program to host the game and provide rules enforcement, and therefore our speed is affected by a number of factors including internet latency and the amount of other people using the server at the same time. However, it is clear that Alpha-Beta is much faster than basic minimax, and Monte-Carlo is faster than both of them. We speculate that Monte-Carlo's speed is the reason it is so popular in commercial applications.

As a point of interest, we also perform favorably against amateur human opponents, but we omit the games played against internet strangers in this table, since we cannot be sure which policy they are using.

Discussion

The experimental evidence we collected supports our hypothesis that tree-search methods would beat Monte-Carlo in a game of Go played on a small (7x7) board. While we didn't develop any particular novel algorithms for this project, we were able to implement several different strategies and verify their relative merits. For example, Alpha-Beta's 6 ply look ahead gives it an enormous advantage in the late game where a single move can change the game dramatically. It is also more likely to make plays that will lead to the capture of an opponent's stone, and more defensive in protecting its own stones. Monte-Carlo, on the other hand, is more likely to play the center of the board, and makes its moves more quickly.

We also learned the importance of a good heuristic function for pruning the tree-search. A good heuristic can save a lot of time, especially in the beginning of the game when most moves are roughly equivalent score-wise. The heuristic also heavily impacts the branching factor, as an Alpha-Beta tree with an optimal heuristic has a branching factor of sqrt(boardSize) and similar tree with a very poor heuristic has a branching factor of boardSize. The average case for a random heuristic is ¾ boardSize. The heuristic used as the value function affects the number of states explored dramatically.

For this reason, developing a "better" value function would be an excellent topic for future work. This may allow the autonomous agents to play on larger boards as well. At the present, the largest solved Go board we were able to verify in the literature is only 5x5, which has over 400 billion legal states. In this respect, the fact that our agent was able to play reliably well on a 7x7 board is already a significant contribution. (All autonomous Go playing programs on larger boards are forced to use Monte-Carlo.) However, we believe that tree-search could be possible even on these large boards with the right heuristic.

Unfortunately, we underestimated the amount of scaffolding necessary to connect one agent to another online and we ran out of time before we could pursue this direction. Even so, preliminary results suggest that tree-search is more than capable for competing on small boards. In retrospect, we would have leaned more on premade tools so that we can get straight to the interesting stuff.

References

[1] Zobrist, A. *A New Hashing Method with Application for Game Playing*. Technical Report #88, Computer Science Department, The University of Wisconsin, Madison: 1970.

[2] L. Kocsis and C. Szepesvari. *Bandit based Monte-Carlo Planning.* Computer and Automation Research Institute of the Hungarian Academy of Sciences, Budapest, Hungary: 2006.

[3] Gelly, S., and Silver, D. *Combining online and offline learning in UCT.* In 17th International Conference on Machine Learning, 273–280. 2007.

[4] Russel, S. and Norvig, P. *Artificial Intelligence: A Modern Approach, Third ed*. New York: Prentice Hall, 2010.