Interest Alpha-Beta: A Heuristic Applied to Atari-Go

John Champaign
Software Architecture Group
University of Waterloo
Ontario, Canada, N2L 3G1
jchampaign@uwaterloo.ca

Abstract

Artificial opponents for human games have fascinated researchers and the general public since Baron Wolfgang von Kempelen built a chess-playing machine called "The Turk" for the amusement of the Austrian Queen Maria Theresia in 1769. While this was a hoax, the concept took Europe by storm and has always been an important part of AI research. The goal of this paper is to improve on techniques for playing Atari-Go, a variant of the game Go that has attracted a large amount of research activity. Improvements leading to faster computation time were investigated with the potential of being applied to standard Go game playing.

1 Introduction

1.1 Go Background

Computer Go playing has long been considered one of the hardest problems for game programmers. Second only to chess in terms of research effort and programming time spent, every solution has failed to produce strong opposition for a moderately skilled human player (some estimates place the strongest AI players at the level of a human with about 1 month of experience) [1].

Some researchers and Go enthusiasts are so confident that this is a hard problem that multimillion-dollar pots have been offered to AI programs that can play a strong game. Mr. Ing, one of Taiwan's most successful industrialists and a well-known and influential figure in Go [2], offered 40 Million Taiwanese dollars to any program that could beat a Taiwanese professional player [3]. The Ing Chang-ki Goe Educational Foundation offered 1 Million USD to a program that could beat a selected twelve-year-old player [4] and Professor Elwyn Berlekamp (a venture capitalist and Mathematician affiliated with UC Berkeley) has offered 5 million USD for a program that can beat a 5-dan player [5] (a very strong professional).

Search depth plays a role in the difficulty, as a game board initially has N^2 positions to be played on (with N = 9, 13 & 19 all used, 19x19 being the most popular) where N is a natural number. This could lead to a branching factor of 361 (on the 19x19 board), which is computationally expensive to say the least. The complexity is $O(N^2!)$.

1.2 Atari-Go

A related problem that is easier to solve is that of Atari-Go [6]. This game differs from Go in that rather then capturing territories, the game ends with the first capture (a stone of the opposing colour being surrounded). Work done on Atari-Go is easily applicable to Go.

Atari-Go is, like Go, played on a square board of size N. The board is made up of a grid, with stones being played on the vertices. Obviously on a 1x1 board, the "grid" is a single point, and the first player to move wins. Boards of greater size become increasingly difficult to analyze. Players take turns placing a stone of their colour (black or white), with the black player placing first. Players must play one (and only one) stone each turn. If a group of stones (stones of the same colour, which are connected by lines to other stones) are surrounded (have a stone of the opposite colour at every possible vertex connected to the group by a line) they are removed and the opposing player wins. Free locations connected to a group by a line are known as liberties, and can be used to determine how much danger a group is in. In Figure 1, the white stones have 4 liberties, and the black stones have 5 liberties.

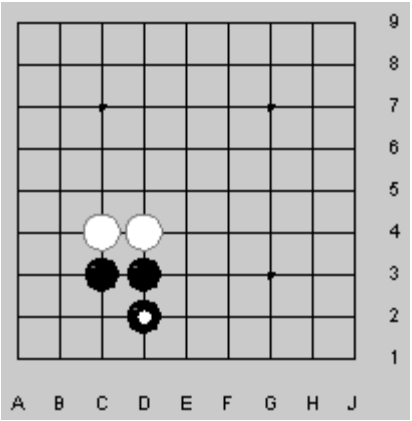

Figure 1. A 9x9 Go board

While the complexity of Atari-Go is the same as Go, the end condition (victory) is more clearly defined and easier to determine.

1.3 Game Play Algorithms

A popular game playing technique for deciding on the optimal solution is the Minimax algorithm. If this algorithm is used and allowed to search to an unlimited depth it will provide perfect play. However "the time cost is totally impractical, but this algorithm serves as the basis for more realistic methods" [7].

Alpha-Beta Pruning is a method of reducing the branching of Minimax, and thus decreasing its computation time. It does this by assuming the opponent is rational, and will not choose a poor move. It then focuses computation on the opponent's best response to each move that can be made. Additionally, if a move is determined to be worse then a previous move, the algorithm aborts searching that branch and moves on to evaluate other moves.

A key factor to the improvement offered by Alpha-Beta pruning is the ordering of the branches. Ordering these branches perfectly would be hard however. An algorithm capable of doing so could simply be applied directly to the game and Minimax or Alpha-Beta wouldn't be needed. The right heuristic can however get us *close* to the best ordering.

The goal of this project was to deliver a playable Atari-Go AI that can play quickly and perfectly on smaller boards, and provide an enjoyable game for some human players on larger boards. A new technique for ordering the branching in Alpha-Beta Pruning that will decrease the necessary computation was investigated.

2 Related Work

Go has been the subject of many research papers. Although many focus on pattern matching, others focused on optimization techniques for search algorithms and were more applicable to my work.

1) Applicable Go Research:

Muller[8] gives an overview of the current state of research in Go as well as techniques for the representation of knowledge in a Go program. He goes on to discuss the sub problems of the game, and techniques to solve each. His discussion of the "Life and Death" sub problem, determining whether or not a group can be made safe from capture, is applicable to Atari-Go. Additionally, he discusses using a full board minimax search in the endgame when the branching factor has been greatly reduced.

Bouzy and Cazenave [9] describe issues in Go programming such as game complexity; move generation, and tree search. Additionally, they discuss optimization techniques for these areas.

2) Atari Go research:

Cazenave [10] presents an approach to Atari-Go that guarantees victory in a reasonable length of time on boards up to size 6x6. He uses a new AND/OR tree search algorithm based of a theory of combinatorial games.

Cazenave [11] further refines his research in this paper, presenting an algorithm based on threat analysis. It can model existing related algorithms such as Lambda Search and Abstract Proof Search, although it was designed to solve 6x6 AtariGo faster then previous algorithms. Theoretical and experimental comparisons with related algorithms were given.

3) Artificial Go Players:

Enderton [12] details the creation of a Go program called Golem. He focuses on creating simple algorithms based on the fundamental principles of the game, rather then pattern matching.

4) Detailed search algorithm and how it applies to Go:

Thomsen [13] proposes a new method for searching binary game trees that is guaranteed to find the minimax values. It provides large relative reduction in search space over standard alpha-beta while using negligible working memory.

5) Correspondences:

E-mails were exchanged with both Dr. Cazenave and Dr. Müller, recognized experts in Go research, neither of whom was familiar with an approach similar to this. This was seen as some evidence of the originality of this approach.

Compared to these related works, this "interest areas" approach is far simpler. The simplicity of Go suggests that a simple algorithm could be applied to the game in an effective manner. Many researchers working in the area of Go game playing have sought this elusive approach.

3 Approach

The approaches applied to this program involved a simple method for reduction of the required computation and a heuristic used to order the examination of successors during Alpha-Beta pruning.

Board symmetry often allows reduction in computation time. In this approach fact that for the first move only ¼ of the positions on the board need to be considered was exploited. Any other positions can be achieved by rotating the board, and therefore do not need to be considered. This dramatically decreased computation for the initial move (which would always be the most costly if no optimization was done).

An *Interest ordering heuristic* was the heart of our approach. One aspect in regular (non-Atari) Go games that has been observed is that many moves are made close to the last move of the other player, and previous moves in general. Stones are seldom placed along the edges (only when this would push towards a win). When play shifts from one area of the board to another, it is when a player realizes he is at a disadvantage in the area of play (and hopes to improve his position by strengthening himself elsewhere). Obviously in Atari-Go there is no incentive to leave an area of weakness, as the opponent will exploit the opportunity. This leads to greater locality of play, and should improve the results of this approach. Figure 2 shows an example game between 2 humans on a 6x6 board. A "clustering" of stones being placed near previously placed stones can be seen (especially in stones 1-4, 6-12, and 13-18).

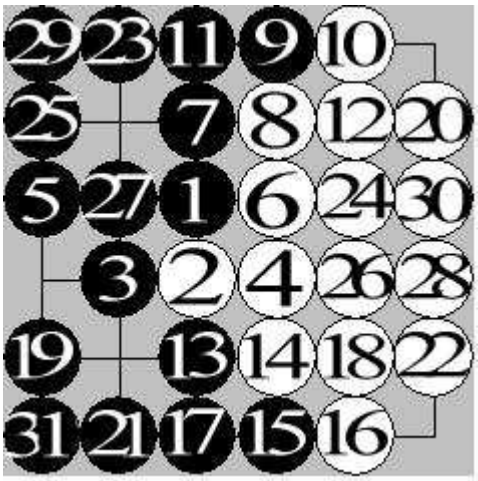

Figure 2. Interest Table

1	1	1	1	1	1
1	4	2*2	2	4	1
1	2*2	0	2*3	2	1
1	2*2	0	2*3	2	1
1	4	2*2	2	4	1
1	1	1	1	1	1

Figure 3. Atari-Go game

This interest heuristic gives different portions of the board greater weight during searches, and considers these positions first when pruning. "Interest areas" are updated as the game progresses. This update consists of setting the interest value of the location where the stone was played to 0, and doubling the value of any adjacent stones.

An example "interest areas" array of this game after the second move is given in Figure 3. Setting the outer vertices to 1, the innermost vertices to 3 and the vertices in between to 2 generated this initial map. The corner positions of the second outermost square are called *handicap* locations and are often viewed as strong opening moves, thus these were given a weight of 4 in the initial "interest map".

```
function interest-update
      (interestMap, position)
returns the updated interestMap

inputs: interestMap, an array
representing the Go board and
the current interest value of
each position on it
        Position, one vertice on
the play board

interestMap[Position] ← 0
for each s in adjacent(position)
do
  interestMap[Position] ←
  interestMap[Position] * 2
end

return InterestMap
```

Figure 4. The interest heuristic

Using this map, the search could be focused for the next move using the 4 positions first, then the 3's, then the 2's and finally 1's. The expectation was that this would allow enough focus of computational power to provide a reasonable artificial opponent who can choose moves in a more useful length of time.

Alpha-Beta searches have the attractive quality that they make the same assertion as Minimax searches do; that it guarantees the best possible result if not depth limited. Additionally, it satisfies this far more quickly in most situations. This approach extends Alpha-Beta in a similar fashion, providing the same result more quickly.

4 Other Approaches

Many researchers take a different approach to Go.

Brute force searching was attempted using Minimax, Alpha-Beta Pruning, and Greedy searches. This was applied in a selective manner: it was depth limited to allow a "maximum time" for the program to consider moves. As a heuristic, the number of liberties of the computer player's groups (defensive play), and the number of liberties of the opponent's groups (offensive play) were used. This provided a basis for evaluation of our new approach rather then providing a solution. This area has been thoroughly explored in past research.

Pattern matching is the direction most serious approaches to Go playing have taken. Progress was achieved without introducing expert knowledge into the project, so this approach was rejected. The elegance and simplicity of Go makes these cumbersome pattern-matching systems unwieldy by comparison. It is my belief that there exists an approach to playing Go that is a simple as the game itself, yet provides good results.

5 Hypothesis

A heuristic based on an initial table similar to Figure 3, and updated according to the algorithm given in Figure 4 can be applied to the ordering of successors in an Alpha-Beta search of Atari-Go moves that will provide equivalent results to a non-ordered Alpha-Beta search but with less computation time.

6 Experimental Design

Time was spent trying to find an existing Go program, which could be modified to use this interest heuristic. One Java applet that was found on the Internet [14] plays Atari-Go using Greedy, Minimax and Alpha-Beta algorithms against one another as well as playing

against humans. This applet was modified to introduce the interest heuristic and tested against the three algorithms available [15].

The main question of this experiment is "How much faster can Alpha-Beta solve a problem with the interest heuristic then without?" This was answered by running games with algorithms playing against each other, then reversing algorithms, playing again and comparing the times. This was performed for different board sizes and for different depths of the algorithms to try and decide how universal the results are.

Of secondary interest is evaluating our confidence that this heuristic maintains the results provided by Minimax. Therefore all games had their results compared to ensure that in the same situation, Minimax, Alpha-Beta and modified Alpha-Beta all win or lose. If one of these algorithms wins or loses in a given situation so should the other two. Also of interest is the speed improvement of the modified Alpha-Beta heuristic over traditional Minimax. This is not of paramount interest however, because part of this increase in speed comes from Alpha-Beta pruning.

For the actual trials, 48 games were played. Eight of these were on a board of size two which any of them should have been able to easily analyze. 12 of these were on a board of size 3 and used a search depth of 6, which allowed every algorithm to play each other. The hope was that this would show the similarity in performance of the non-Greedy algorithms and demonstrate the difference in computation time. On a board size of 4 the interest Alpha-Beta was played as black and white against the traditional Alpha-Beta (with a search depth of 6) and the Minimax (at a search depth of 4). The computation required for Minimax at larger board sizes and search depths became prohibitive after this point. 4 games were played with the traditional Alpha-Beta and the interest Alpha-Beta at board sizes 5 and 6 with search depth of 4.

A small sample of humans were asked to play against the algorithm and asked their impressions.

7 Results

The raw data is attached as Appendix A. Readers are encouraged to use the modified java applet created [16] to replicate any of these trials.

An artificial Atari-Go player was produced that can guarantee a win [17] on smaller NxN boards (provided it chooses who moves first), and provides a challenge to beginner players at larger sizes. It plays with the same strength (see Figure 6) as a similar depth search using Minimax (or Minimax with Alpha-Beta pruning) but with faster computation in many situations (see Figure 5).

It was found that the modified Alpha-Beta pruning technique, using the heuristic presented in this report gave an average speed improvement of 11.95% over a standard Alpha-Beta pruning. The interest Alpha-Beta search obtained the same results as Minimax and Alpha-Beta, but in a shorter time. One game (Greedy vs. Minimax, size 3) can be seen in Figure 6 that does not support this, and will be discussed in the next section.

Interest Alpha-Beta was determined to be, on average, 10.3 times faster that Minimax. Minimax became so slow on larger boards that running trials became prohibitively expensive.

While the Greedy algorithm was computed quickly to the point that it does not appear on the time scale presented in Figure 6, its performance was poor and chaotic, even at small board sizes, and thus it wasn't given serious consideration.

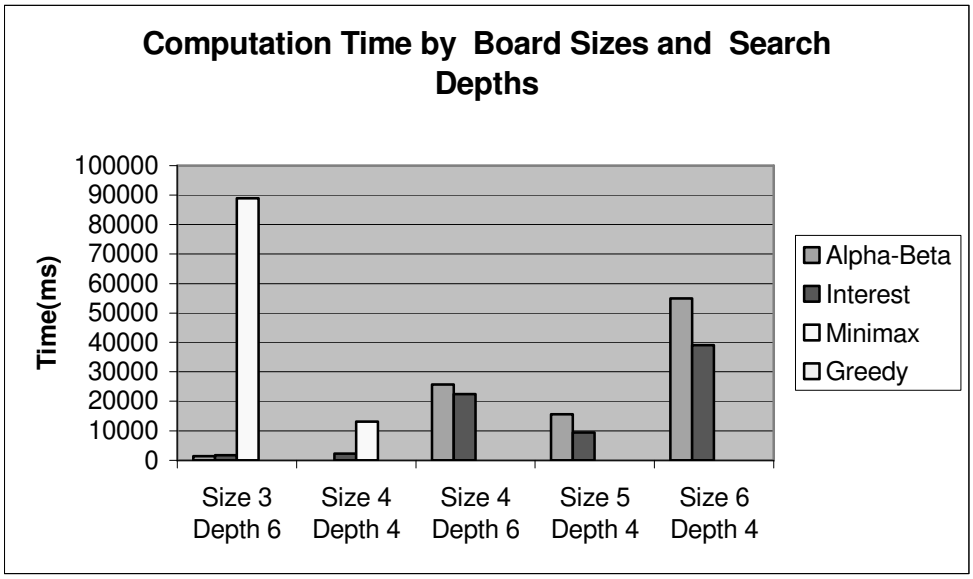

Figure 5. Computation Time

Size	algorithms	winner
3	Interest vs. Alpha-Beta	black
	Alpha-Beta vs. Interest	black
	Interest vs. Minimax	black
	Minimax vs. Interest	black
	Alpha-Beta vs. Minimax	black
	Minimax vs. Alpha-Beta	black
	Interest vs. Greedy	black
	Greedy vs. Interest	black
	Minimax vs. Greedy	black
	Greedy vs. Minimax	white
4	Interest vs. Alpha-Beta	white
	Alpha-Beta vs. Interest	white
	Interest vs. Minimax	white
	Minimax vs. Interest	white
5	Interest vs. Alpha-Beta	white
	Alpha-Beta vs. Interest	white
6	Interest vs. Alpha-Beta	white
	Alpha-Beta vs. Interest	white

Figure 6. Algorithm Victors

3 humans played against this system at size 5 with depth 4 and found it to be a satisfying game. Comments included that it "was hard", "played slowly" (although interestingly often the human players took longer to decide on moves than the computer did), "it's smarter than me" and "it's great!"

An interesting, and unexpected, result of this work was that it clearly demonstrated which player has the advantage in Atari-Go on boards of size 3-6. The black player should win on a board of size 3, while the white player should win on the others.

This experiment answered the main question, however more extensive trials could further explore this area.

It was noticed that the improvement in speed became more pronounced at large board sizes (interest Alpha-Beta was 30.6% faster in a size 6, depth 4 games, while it was 45.6% slower in a size 3, depth 6 games). It is believed that there is an overhead associated with maintaining an interest map and spending time deciding on the ordering of the Alpha-Beta branching. This overhead pays off on larger boards, but can sometimes be a hindrance on smaller boards. Since 45.6% slower means about one additional second per move at board size 3, while 30.6% faster can represent up to 20 seconds less per move at board size 6 this overhead doesn't seem unreasonable.

Figure 5 clearly shows this increasing performance difference, as the board size increases.

Given a system with time constraints, Interest Alpha-Beta's reduced computation time would allow additional calculations, a deeper search and would thus result in a stronger player. For example, if Minimax could perform a depth 5 search in a given time, and the modified Alpha-Beta was able to perform a depth 8 search, the Alpha-Beta player would have an advantage.

The one odd result was in the size 3 game of Greedy vs. Minimax where Minimax won (and should have lost). This was due to the implementation of the Greedy algorithm playing somewhat randomly, and in this game it made a poor choice of opening moves.

8 Lessons Learned

This project demonstrated the difficulty of Go game playing. This technique produced modest improvements in the performance of a simplified version of the game. Much work remains to be done before a serious Go artificial opponent can be produced.

The need remains to further improve the computation speed for analyzing Atari-Go positions. While these refinements certainly exist, they are without a doubt more complicated then this approach.

While modifying the java applet, it was discovered that it did not use pure Alpha-Beta pruning, but actually performed a rough filter on the successor positions before applying them (it only considered moves that were adjacent to stones already on the board). It was modified it so that it was a standard Alpha-Beta pruning. This algorithm performed better then this rough filter, but more extensive trials comparing the two would have merit.

The results of this project are not exhaustive. The trial results, while suggesting a certain level of performance, are not the same as a proper comparison of the algorithms. A more comprehensive proof, in addition to trial timings [18], that this heuristic guarantees the same result as minimax would be valuable. Additionally, a statistical analysis of why this ordering results in the performance gains it does would be necessary to establish the legitimacy of this technique.

Unusual board situations, and "Life and Death" problems [19] should be studied in addition to practical games (from first

move to last). Humans would add a greater range of conditions as well. This would test the robustness of these algorithms further.

Human participants to use the program extensively and provide detailed feedback would be valuable. Testers were selected randomly for participation, but serious Go players could have provided insight. Another line of research would be a systematic investigation on the human impact of the game.

A suggestion from one reviewer of this paper was to extend the algorithm to infer information about its opponent. This approach presented assumes that the opponent is an expert player, while a more aggressive play style would be suited for opponents who were clearly not experts. Clearly distinct from the work done, this idea has merit.

9 Conclusion

A heuristic was produced that, when applied to the ordering of successors in an Alpha-Beta search of Atari-Go, provided moves that produced equivalent results to a non-ordered Alpha-Beta search but required less computation time.

Acknowledgements

The author thanks Dr. T. Cazenave, Dr. M. Müller, Sorin Gherman, Dr. V. Keselj, Dr. R. Mann, Robert Warren, Davor Svetinovic, Lijie Zou and Xinyi Dong for their contributions to this research.

References

1. M. Müller, Computer Go. Artificial Intelligence 134 (2002) 145-152.
2. http://www.usgo.org/ingfoundation/ingbio.html
3. http://www.reiss.demon.co.uk/webgo/compgo.htm
4. http://www.ishipress.com/chess&go.htm
5. http://www.ishipress.com/chess&go.htm
6. Also called “First Capture Go”, “The Capture Game”, “The Capturing Game”, “Capture Go”, and “Capture One”
7. Russle, S. and Norvig, P. (1995). Artificial Intelligence A Modern Approach. Prentice-Hall, New Jersey.
8. M. Müller, Computer Go. Artificial Intelligence 134 (2002) 145-152.
9. B. Bouzy and T. Cazenave, Computer Go: An AI oriented survey. Artificial Intelligence 132 (2001) 39-103.
10. T. Cazenave, La Recherce Abstraite Graduelle de Preuve. RFIA-02, Angers 2002.
11. T. Cazenave, A Generalized Threats Search Algorithm. CG 2002, Edmonton.
12. H. D. Enderton, Technical report CMU-CS-92-101, Carnegie Mellon University, 1991. Report available by ftp from bsdserver.ucsf.edu.
13. T. Thomsen, Lambda-search in game trees – with application to Go. ICGA J. (4) (2000) 203-217
14. http://javaboutique.internet.com/Go/
15. This was a valuable exercise as the area of research for my Master’s topic is re-engineering and my advisor and I hope to get a paper out of the process I went through adapting this applet.

16. Available at http://www.math.uwaterloo.ca/~jchampai/goapplet.html. Requires Java 1.4 plugin.
17. I define "guarantee a win" as an algorithm that, when playing first or second (depending on the board size), can always win despite any moves the opponent makes. On a 1x1 board, the first player always wins, so every algorithm can "guarantee a win" at this size. On a 2x2 board, the second player always wins by playing diagonally opposite the first stone played, thus it is easy to design an algorithm that "guarantees a win" on a 2x2 board.
18. Don Knuth is known to have once written "Beware of bugs in the above code; I have only proved it correct, not tried it"
19. "Life and Death" problems are a well known Go training technique where a board position is shown, and the player is asked to demonstrate how the group can be guaranteed to be kept alive, or taken.

Appendix A: Raw Data

size	depth	algorithm	time(total ms)	time(avg ms)	win?
2	2	c V h	397	183	n
		h v c	42	21	y
		a v h	136	68	n
		h v a	92	46	y
		g v h	32	16	n
		h v g	4	4	n
		m v h	36	18	n
		h v m	66	33	y
3	6	c	6703	1675	y
		v a	1988	662	n
3	6	c	5643	2821	y
		v m	42842	42842	n
3	6	c	6662	1665	y
		v g	36	12	n
3	6	a	6167	2055	y
		v c	2435	1217	n
3	6	m	418919	139639	y
		v c	2869	1434	n
3	6	g	75	25	y
		v c	2604	1302	n
3	6	a	6241	2080	y
		v m	37508	18754	n
3	6	m	411445	102861	y
		v a	1994	664	n
3	6	a	6227	1556	y
		v g	34	11	n
3	6	g	41	10	y
		v a	2101	700	n
3	6	g	26	8	n
		v m	51426	17142	y
3	6	m	846966	211741	y
		v g	45	15	n
4	6	c	168120	24017	n
		V a	156485	22355	y
		a	202977	28996	n
		V c	146328	20904	y
	4	c	14593	2279	n
		V m	80850	11550	y
	4	m	102141	14591	n
		V c	14999	2142	y
5	4	c	104641	9512	y
		V a	155821	15582	n
5	4	a	163232	13602	y
		V c	103422	9402	n
6	4	c	626505	39156	y
		V a	882167	58811	n
6	4	a	818313	51144	y
		V c	583046	38869	n

h = human player, c = interest alpha-beta search, a = alpha-beta search, m = minimax search, g = greedy search, y/n in the win column stands for win/lose with y being the winner.