Efficiency and Performance of Search Algorithms on Real Applications

Uchenna Akujuobi
Uchenna.akujuobi@kaust.edu.sa
King Abdullah University of Science and Technology

ABSTRACT
Okwe is a popular game played by the Igbo people of south-eastern Nigeria. This game is a member of the big Mancala family game. This paper presents an artificial intelligence (AI) designed to play Okwe efficiently. This paper also tries to show the difference in efficiency and performance in using different search algorithms to design the AI. The search algorithms used in this paper are Random/Brute force, heuristic, minimax and alpha-beta algorithms. The result of this work is a program capable of winning human opponents.

Keywords
Game theory, minimax, alpha-beta, heuristic, brute force, mancala, okwe, mancala game

INTRODUCTION

Okwe is a variant of the mancala game a two-player board game played over the world. More than 800 names of traditional mancala games are known, and almost 200 invented games have been described [1]. Okwe is played on a board with 12 holes, 6 each of which belong to a player and the other 6 belonging to the opponent. Each player has 1 bigger hole at each end where captured seeds are placed. The game is also played on the ground also where the holes are carved into the ground.

At the beginning of the game, each of the 12 player holes contains 4 seeds each. These seeds might be stones, marble, palm kernel, but usually, round pebbles are used. Okwe is a two player alternating game, the goal being to capture more seeds than the opponent. A turn consists of picking the seeds in any hole on the player's side and placing each seed into each hole in an anti-clockwise direction. The game starts by scatting the start configuration of 4 seeds in each hole (cell). In this paper, I will use cell to represent holes. To capture the seeds in a cell, the total number of seeds in that cell must be completed to 4 (usually after the first round of play). Okwe has 5 rules:

- A player must begin his move from his own area
- When a player makes a move, he must take all the seeds from the selected cell
- He cannot capture seeds from his opponent's cells except if the seed completing the cell is the last seed he has.
- A player should only stop playing only when the last seed is placed in an empty cell.
- When there are only 8 seeds on the board, the last player to capture seeds would take all the 8 seeds.

Figure 1 shows the start state of the board at the beginning of the game.

The search space of the game is not so huge but it has some rather interesting characteristics when it comes

to developing an artificial intelligence for it. To the best of my knowledge, there is no publication in any literature that deals with building an AI for Okwe up to now.

Even though the Search space is not huge, different search algorithms can either be ineffective with high performance or be effective or have low performance. In order to design an efficient game, an efficient algorithm with good or close to optimum performance must be used.

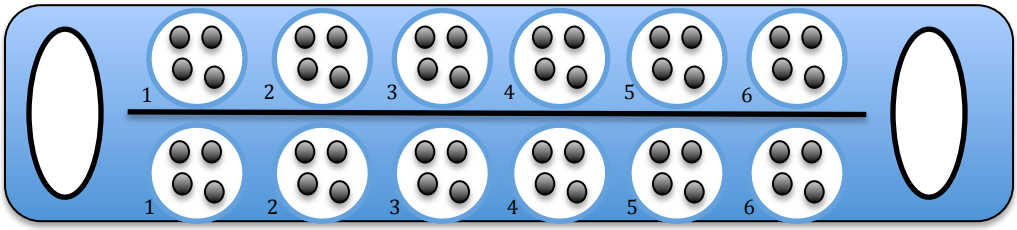

Figure 1. The initial start state of the game

One open problem in the study of Mancala family of games is as follows: What heuristics can be used for playing Mancala games by a computer [2].

In this paper, I propose heuristics for the game and implement the game with some algorithms and run experiments and compare results to determine the best of them all. This program was implemented both in C++ (visual studio) and C# (unity) but the experiments are done mostly with the C++ console implementation to avoid delays in graphical rendering and animations but real life experiments for testing the efficiency would be done on the C# implementation since it doesn't consider time taken but the number of wins, lose or draws.

The rest of the paper is organized as follows: section 2 talks about the game classification, section 3 talks about the implementation, section 4 talks about the experiments and results gotten, section 5 talks about the conclusion and section 6 lists some of the future works.

2. CLASSIFICATION

Okwe can be classified as a two-player zero-sum game. The simplest mathematical description of a game is the strategic form [3]. In strategic form, a two player zero-sum game can be described by a triplet (X,Y,A), where

X is a nonempty set, the set of strategies of player I

Y is a nonempty set, the set of strategies of player II

A is a real-valued function defined on $X \times Y$.(Thus, A(x, y)is a real number for every $x \in X$ and every $y \in Y$.)

The interpretation is as follows. Simultaneously, player I chooses $x \in X$ and Player II chooses $y \in Y$, each unaware of the choice of the other. Then their choices are made known and I wins the amount A(x, y) from II [3]. This means a win from Player I is an automatic loss for Player II. Thus, a win in Okwe is represented by +1, a loss by -1, and a draw by 0. The game task environment can be specified using PEAS (Performance measure, Environment, Actuators, sensors).

Okwe uses a heuristic value to evaluate the next possible action to take

up to a given depth. The out come of the game is used as the performance measure (i.e win, lose or draw). It is fully observable, static, deterministic, sequential and discrete and the environment can be interpreted as a competitive multi-agent. The agents in Okwe are rational.

3. IMPLEMENTATION

In this section, I will present the different implementation methods and algorithms used in this paper. First, I will present the heuristic function used for the search tree. Then, second, I will present the brute force search algorithm, heuristic based search, minimax algorithm and the alpha-beta (pruning) search algorithms. The board is represented by 2 arrays. One is a 2x6 array, which represents the holes for the game play, and the other array is a 2x1 array, which represents the number of seeds captured by each player.

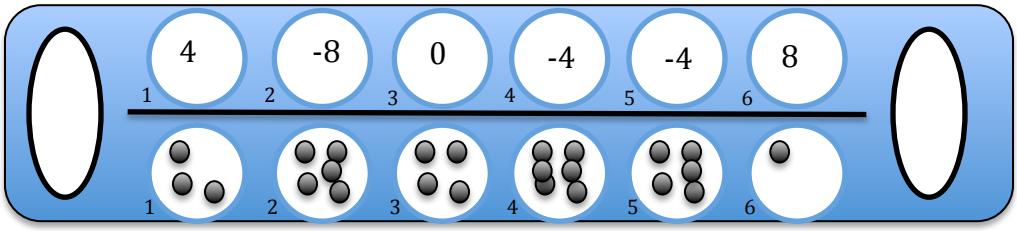

Figure 2. An Example of the board heuristic evaluation for player I

3.1 HEURISTIC FUNCTION

The heuristic function is a way to inform the search about the direction to a goal. It provides an informed way to guess which neighbor of a node will lead to a goal [4]. In Okwe, the higher the seeds captured, the better the move. But also, we should also not forget the number of seeds captured by the opponent after a given action. For instance, if it is player I's turn, and there are only two possible actions A_1 $\{X_{mc1}, X_{oc1}\}$ and A_2 $\{X_{mc2}, X_{oc2}\}$. Where X_{mc} is the total seed that would be captured by Player I and X_{oc} the total seeds to be captured by Player II. One can't say for sure that action A_1 is better than A_2 just because $X_{mc1} > X_{mc2}$ since it is possible that $X_{oc1} > X_{oc2}$. Thus, the total seeds captured by the opponent should be taken into consideration in the heuristic function.

The heuristic function is given by the total number of seeds capture by a player subtracted from the total number of seeds captured by the opponent. Thus, the higher the value, the better chance the move has to be selected. Figure 2 shows an example of heuristic evaluation during the game for player I using the heuristic function. In the example, the best action for player I is to select the seeds to play from the 6^{th} hole.

3.2 RANDOM/BRUTE FORCE SEARCH

This search algorithm does need to use the heuristic function and doesn't try to evaluate the outcome of any action. This search algorithm randomly selects an action and checks if it is a possible action or not. If the test fails, it chooses a different action and repeats the same test. This it will do till it finds a possible action and then makes that move. In the field of algorithms, it is sometimes helpful to make decisions randomly

instead of spending lots of time on deciding which alternative is the best choice, especially when the time required to obtain the optimal choice is prohibitively high [5]. Obviously, this is not an efficient algorithm but I used it to evaluate the efficiency of the other algorithms and the result gotten was interesting.

In order for it not to go into an infinite check-select-check execution since it possible that all six actions might not be valid, I added a stop after each 10 random selections without making an actual move (couldn't select a possible action) it gives up its turn. The advantage of brute force search algorithm is that it would require less time to make a move if it actually finds a possible action and also doesn't require any extra space.

3.3 HEAURISTIC BASED SEARCH

This search takes each node and returns the heuristic value using the heuristic function. The action to be taken is determined by evaluating the heuristic values of each node and then the node with the highest heuristic value is selected. Typically a trade-off exists between the amount of work it takes to derive a heuristic value for a node and how accurately the heuristic value of a node measures the actual path cost from the node to a goal [4]. In this implementation, I went for the later.

3.4 MINIMAX SEARCH

A competitive 2-player game such as Okwe can be represented using a tree. The players' move can be modeled using a structure known as adversarial game tree [5]. In minimax, the current player is denoted as MAX whose goal is to select the best move available and the opponent is denoted as MIN whose goal is to reduce the best possible outcome of any MAX move. For an efficient AI, MAX must choose a strategy that will lead to a winning terminal state no matter what MIN does, and the strategy includes the best move for MAX for each possible move by MIN. Minimax assumes it is playing against an infallible opponent and, therefore, must determine the optimal strategy for MAX, and thus have to decide what the best first move is.

The root of the search tree is the start state of the game board and the children nodes are the possible moves for the current player with children nodes, which are the moves available for the opponent, and so fourth. Figure 3 shows an example of the game tree for Okwe.

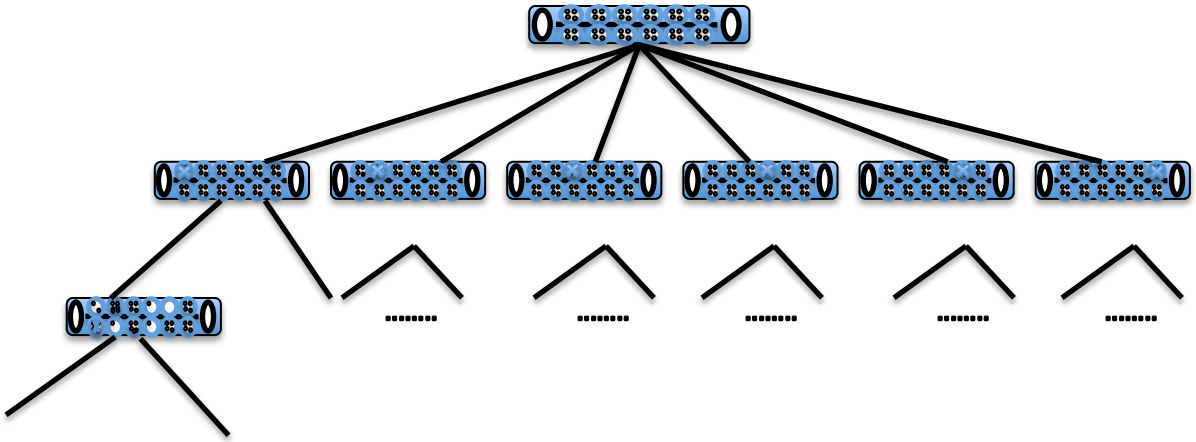

Figure 3. The game tree of Okwe showing the possible max moves and one possible min move at the first level

MINIMAX-VALUE (n)=

$$\begin{cases} UTILITY\ (n) & if\ n\ is\ a\ terminal\ state \\ \max\left(s \in Successors(n)\right) & MINIMAX-VALUE(s)\ if\ n\ is\ a\ MAX\ node \\ \min\left(s \in Successors(n)\right) & MINIMAX-VALUE(s)\ if\ n\ is\ a\ MIN\ node \end{cases}$$

Formula 1. Calculating a minimax function

A minimax search is a recursive algorithm for finding the next move for any given player. All the possible continuations to the desired level are considered, evaluated and assigned a value using the heuristic function as it goes. The minimax value is normally determined using formula 1 [6]:

The leaves of the game tree are the final game states where the outcome of the game is obvious. In order to determine the best move, the whole tree is searched. The amount of work a minimax search generates increases exponentially as a move is examined to a greater depth [6]. In minimax, if the maximum depth of the tree is *m*, and there are b legal moves at each point, then the time complexity is $O(b^m)$, and the memory complexity is *O(bm)*. In Okwe, the branching factor b is 6 thus, the time complexity is $O(6^m)$ which is infeasible. I reduced the memory complexity by generating successors one at a time which reduced the complexity to *O(m)* which is better.

3.5 Alpha-beta Pruning

As seen from section 3.4, the minimax search algorithm is efficient but not very good in performance especially when the search space is huge. The time complexity has to be reduced in other to get a good performance. Alpha-beta pruning does this. Alpha-beta is a tree-search procedure that is faster than minimax but still equivalent in the sense that both procedures will always choose the same depth-1 successor at best, and will assign the same value to it [9].

Alpha-beta is faster because it skips checking the branches which values doesn't affect the outcome taken, thereby pruning the branches. On getting to these branches, alpha-beta algorithm jumps over them to the next available branch. This action of jumping over these branches is called Alpha or beta cut off. Alpha, which is the maximum value of possible actions found at any choice point along the path for MAX, can be associated with MAX. Beta with is the opposite of Alpha is the minimum value of possible actions found at any choice point along the path for MIN, and can be associated with MIN.

There are many ways to implement the alpha-beta algorithm. At first, I tried using stack implemented using singly linked list, but noticed some overhead in creating a new node, pushing and popping the values whenever we want to access the value. Thus, I decided to just use a single variable on each branch that either shows the MIN value or the MAX value, which increased performance.

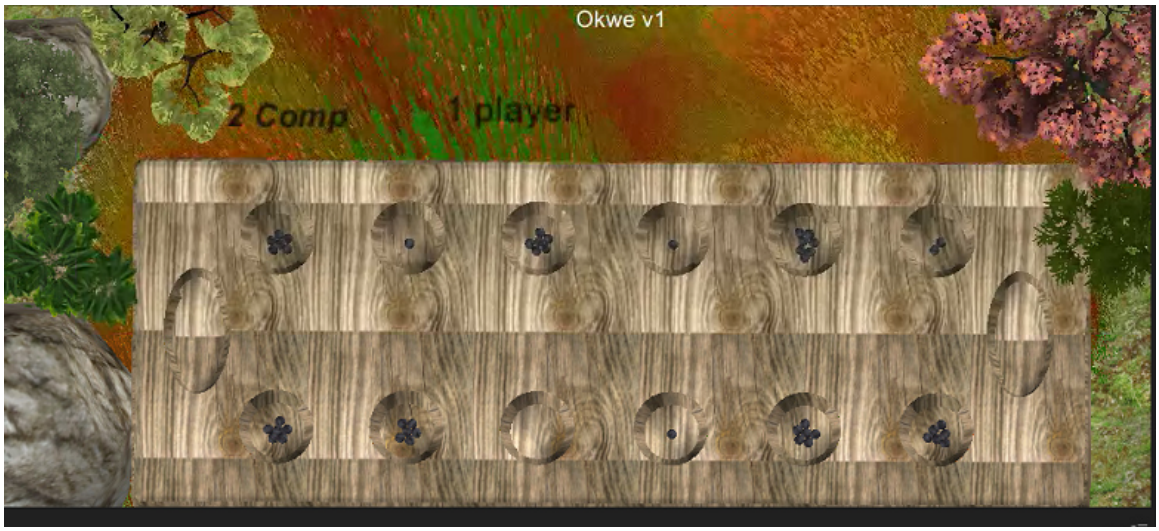

Figure 4. Screenshot of the game during game play

4 EXPERIMENTS

To show the efficiency and performance of the algorithms in playing the game, I conducted some experiments. In this section, I will discuss the results gotten from the experiments. Figure 4 shows a screen-shot of the developed game screen.

Performance

To show the performance of the minimax and Alpha-beta search, I measured the time taken for both the minimax and Alpha-beta algorithms to make a play. This includes the time of tree transversal and the time of playing the game. This measurement was made on the first gameplay. Figure 5 shows the time taken for both algorithms. From the graph, we can see that the time for minimax grows exponentially and it is obviously clear that alpha-beta is the best when it comes to performance. This is because of the large amount of state visited by the minimax search.

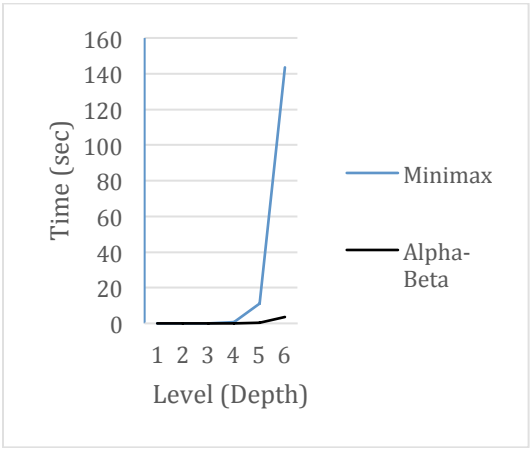

Figure 5. Time taken for completing first game move for alpha-beta and minimax.

Minimax visits the whole states, and since in real world, the search space can be very huge, minimax is not a feasible option. Figure 6 shows the total number of nodes visited by each algorithm. This measurement also was made on the first gameplay.

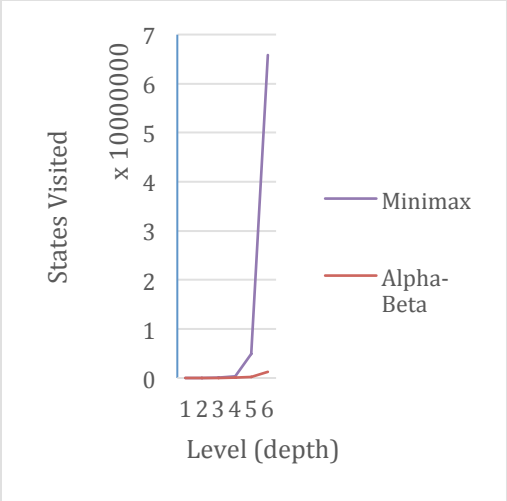

Figure 6. Number of states visited by both minimax and alpha-beta algorithms.

Efficiency

To show the efficiency of each algorithm, I set each algorithm to play against each other. Since alpha-beta is a faster version of minimax, I used alpha-beta as the AI algorithm in the mini tournament. Table 1 shows the number of seeds won by each search algorithm and the number of steps taken in the game. A step represents one complete move by both players. Alpha-beta, which is the most intelligent, played better than others did. It however, drew with another alpha-beta with a lower depth. This is can be explained. Since the game involves removing seeds from the board, after a few steps (depending on the players), the search space reduces and using a lower search limit would be as efficient as using a higher search limit.

To show the efficiency of the search algorithms in real game play, I explained the game to people, made them play the game once to get used to the interface and rules, and after that, they played against alpha-beta of limit 5 lookahead and the random brute force search. Table 2 shows the result gotten from this experiment. Only one person could win the AI using the alpha-beta search while the brute force suffered some loss, some draws, and a win. This clearly shows that the efficiency of the alpha-beta search is high.

	Seeds Won by player I	Seeds Won by player II	Steps
Alpha-Beta level 6 vs Random(brute force)	40	8	13
Alpha-Beta level 6 vs Heauristics	28	20	15
Alpha-Beta level 6 vs Alpha-Beta level 3	24	24	7
Heauristic vs Random(brute force)	40	8	29

Table 1. Winning statistics of the Okwe AI

	Person 1	Person 2	Person 3	Person 4	Person 5	Person 6
Alpha-Beta (5)	Win	Win	Win	Win	Win	Draw
Random(Brute)	Lost	Draw	Draw	Lose	Win	Lose

Table 2. Win/lose results of AI against humans

5 CONCLUSION

A move using minimax and alpha-beta are more efficient the other search algorithms in this paper but they are much slower than others because of the time taken to "think" of a move to make. The others don't need to do these classifications and are thereby, faster to make a possible move. This is a typical efficiency and performance tradeoff.

Minimax visit all nodes. Which might lead to not only to performance problems but also to efficiency problems. For instance, using the game rules, it gets to a state where the move goes into an infinite sequence loop and since minimax visits all states, it gets to this loop state and goes to an infinite loop. I treated this as a special case in minimax but alpha-beta solves this problem but cutting out the branch containing the "non-useful" nodes. The probability of the alpha-beta getting into such problems is low.

The number of states alpha-beta visits is much lesser than in minimax, but it is also huge. In applications with a huge number of states, this might not be the best search to use. Thus, other faster algorithms have to be considered.

6 FUTURE WORK

The search algorithms implemented in this paper are not the only ones available. I would like to extend this efficiency and performance test to other algorithms to show how good or how bad they scale for real applications like Okwe. There are also some graphical and performance changes I need to make to the game for it to perform well. Some of the changes include: Implementing the game using vectors to make it faster and providing a better UI for user experience. Finally, I would like to release the game to the public both on mobile phones and on computers.

REFRENCES

[1] "Mancala." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. 22 July 2004. Web. 21 October 2014

[2] Donkers, H.H.L.M., Uiterwijk, J.W.H.M. and Voogt, A.J.D.V., Mancala Games- Topics in Artificial Intelligence and Mathematics. Step by Step Proceedings of the 4th Colloquium Board Games in Academia, 2002

[3] Thomas S. Ferguson, Game Theory, Second Edition, 2014. Mathematics Department, UCLA,pg 4,7

[4] David Poole and Alan Mackworth, Artificial Intelligence: Foundations of Computational Agents, Cambridge University Press, 2010

[5] Hong, Tzung-Pei, Huang, Ke-Yuan and Lin, Wen-Yang. "Adversarial Search by Evolutionary Computation.." Evolutionary Computation 9 , no. 3 (2001): 371-385.

[6] Russel, S. and Norvig, P.. Artificial Intelligence: A Modern Approach. : Pearson Education Inc., 2003.

[7] Borovska, Plamenka and Lazarova, Milena. "Efficiency of parallel minimax algorithm for game tree search.." Paper presented at the meeting of the CompSysTech, 2007.

[8] Slagle, James R. and Dixon, John K.. "Experiments With Some Programs That Search Game Trees.." J. ACM 16 , no. 2 (1969): 189-207.