COMP424 - Artificial Intelligence Omweso Agent Design Project

Leo Wong
260528085

McGill University, Biology and Computer Science
Email: leo.wong2@mail.mcgill.ca

Omweso is a board game part of a family of other variants in the Mancala family, originating from the people of Uganda. The game is played on a board with 4 rows and 8 columns of pits, where seeds are initially placed and sowed throughout the game. Each player begins with 32 seeds which can be placed in any conformation they deem strategical in their corresponding 2×8 grid of pits. Following the preliminary setup with all 32 seeds of each player positioned in the pits of the board, the game begins. The player decides on the move to be played, which consists of picking up all the seeds from one pit containing more than 1 seed on their side of the board, and sowing them in a counter-clockwise manner, until there are no more seeds in that set. At this point, the player's turn could end, relay sowing could follow or a capture could occur. The ultimate goal of the game is to capture the opponent's seeds such that no more moves can be made, meaning that all of the opponent's pits contain 1 seed or less. In order to design a competitive agent, many approaches are explored during this experiment. Greedy algorithms are initially used with the endeavour to snatch possession of the opposing player's seeds. Minimax game algorithm is implemented in order to infer potential moves with the assumption that the opponent plays in an optimal manner. Alpha-Beta Pruning is also employed in order to cut down space and time complexity and increase the depth of the searching algorithm. The Monte Carlo method is considered and its approach discussed. The concepts of randomness and chaos are also included to maximize the winning probability. Results of each implementation are used to assess the efficiency of the AI agent.

Keywords: Omweso; Greedy heuristics; Minimax and Alpha-Beta; Randomness and Chaos

1. INTRODUCTION

Uganda is the country in East African hosting major Omweso tournaments, attracting players from across the globe. Omweso is one of the several variations of board games from the Mancala family, all sharing the same roots. Just like Chess or Checkers, the main objective of this board game is to carry out moves until a goal state is reached, where a winner is determined. Each player begins the match with 32 seeds which they strategically position on their side of the board. Unlike other board games, the initial placement of the seeds is entirely up to the players to decide, meaning that the turn order plays a role in influencing the initial setup. To play the game of Omweso is simple and the rules are straightforward, but to win requires a clever and strategic approach.

1.1. Game Complexity

There is only 1 starting configuration for games like Checkers and Chess, but there are many times more in Omweso. Each player starts the game with 32 seeds which they are free to place in whichever configuration they desire. This makes theoretically 7.5×10^{11} different positions in which the first turn player can possibly place his seeds. This can be calculated from the equation[1] given below:

$$\frac{(p-1+s)!}{(p-1)! \times s} \tag{1}$$

where p is the number of pits and s the number of seeds for each of the players.

Looking at the number of positions the opponent can counter with, which is also 7.5×10^{11}, there are therefore 5.6×10^{23} combinations[2] to start with. As the game progresses and moves are being played, the number of possible configurations of the board state will increase quickly. Taking the immense branching factor into consideration, there is no simple way to decide the outcome of a game without *digging deep* into the tree.

2. THE APPROACH

Given a game with such a great number of states, the best approach would be one which infers well enough the probabilities of subsequent moves along with a high win rate. In order to find the most efficient design, many algorithms are tested using scripts; some are ditched, others are retained and combined to create the most effective agent. The algorithmic approaches and probabilistic concepts considered and implemented are as follows:

- Greedy Algorithm
- Minimax Algorithm
- Alpha-Beta Pruning Algorithm
- Initial Configuration of Seeds
- Randomness and Chaos
- Right Side Capture
- Depth-Based Progressive Annealing

2.1. Greedy Algorithm

The first approach is to design an agent following a greedy heuristic, which looks to return the move with the best short-term outcome. In other words, on a 1-level deep analysis, the best decision is given by the greedy algorithm. Two evaluation functions are considered and compared.

2.1.1. Positions to be Captured

The first evaluation function looks at the number of *capture moves* the opponent could consequently make based on the move of the player. It minimizes the number of ways the opposing agent could snatch the player's seeds, and will choose the best move generating the least of such outcome (0 being the minimum).

2.1.2. Seeds Advantage

The second evaluation function consists of looking at the number of seeds the player and the opponent is in possession of given that the agent makes a particular move. Hence, the design focuses on the difference between the number of seeds each competitor has at the end of the move, and will choose the move which generates a board state benefiting the player the most.

2.2. Minimax Algorithm

Minimax is an appropriate approach for the game of Omweso, perfect for turn-based game play in which the possibility of losing is minimized. Based on this concept, every move would generate a different board state, from which the most advantageous move can be obtained by looking at the most favorable board state for the given player. During the opposing player's turn, the algorithm looks at the worse outcome for the player, in turns the best move is chosen for the opponent. For a LEVEL-1 Minimax, the agent simulates the resulting board states for the set of legal moves and returns the best move. For a LEVEL-2 Minimax, the agent simulates the resulting board states for the set of legal moves as well, but it will then infer the best move for the opposing player for every board states obtained prior. For a given move, the agent not only looks at the outcome of the move, but also takes a step further and assesses the outcome of the opposing player's decision, assuming the best possible one is chosen. Then, the agent chooses the move which is least advantageous for the opponent. Now, as deeper levels of the game-tree are reached, the same process is repeated over and over again, increasing in branching factor at every step.

The evaluation function consisting of counting the number of seeds in a given board state is used to decide which best move to pick. The Max-player maximizes the difference in seeds between itself and the opponent, while the opponent minimizes it. The caveat in this approach is that the assumption of the opposing players making the best move for themselves and the worse for the player is not guaranteed in real game play. The recursive algorithm can be defined as follows:

```
Minimax(boardstate, depth, turnPlayer)
    if(depth is 0 || moveList is empty)
        return difference in number of seeds
    if(turnPlayer true)
        get list of legalMoves
        for move M in legalMoves:
            bs = simulate move M
            score = Minimax(bs, depth-1, false)
        return bestMove
    if(turnPlayer false)
        get list of legalMoves
        for move M in legalMoves:
            bs = simulate move M
            score = Minimax(bs, depth-1, true)
        return bestMove
```

2.3. Alpha-Beta Pruning Algorithm

The Minimax algorithm works well in searching the game-tree and returning winning moves, however, the depth at which it can search is limited. If two agents powered by the same Minimax algorithm but with different search depths battled one another, the one with the more extensive search exploration has a greater probability of winning, since it looks further ahead to predict the best move to make. Hence, implementation of Alpha-Beta pruning helps out with the issue of depth, since it cuts off branches which do not have to be searched. At every search level, there is a cut-off indicating that there is no need to look further in a given branch, since it is guaranteed that nothing beyond would yield a better result. In such way, the exploration of the search-tree is decreased by neglecting unnecessary branches, meaning more computation can be allocated to search deeper into the graph. Retaining the Minimax algorithm utilized before and bringing a few modifications, the algorithm of Alpha-Beta can be derived and defined as follows:

```
AB(boardstate, depth, a, b, turnPlayer)
    if(depth is 0 || moveList is empty)
        return difference in number of seeds
    if(turnPlayer true)
        get list of legalMoves
        for move M in legalMoves:
            bs = simulate move M
            score = AB(bs, depth-1, a, b, false)
            a = Max(a, score)
            if(b <= a)
                break
        return bestMove
    if(turnPlayer false)
        get list of legalMoves
        for move M in legalMoves:
            bs = simulate move M
            score = AB(bs, depth-1, a, b, true)
            b = Max(b, score)
            if(b <= a)
                break
        return bestMove
```

2.4. Initial Configuration of Seeds

For every board state configuration, there is a distinct probability that it leads to a win or a loss. Some positions yield a higher chance of winning, which are considered to be *winning configurations*, while others most probably generate losses. For instance, a state with only one pit containing 2 seeds represents a much higher chance of losing than another state with more seeds. Some configurations are more prone to being inflicted severe captures. In order to find these positions, data of losing configurations are obtained from battling the agent against itself with the goal of finding a repetitive pattern.

2.5. Randomness and Chaos

If two Minimax players are playing against one another, the one reaching greater depths in the search-tree and looking at outcomes of moves the furthest will have the upper hand. Theoretically, two identical agents running on Minimax will have equal chances of winning while going first and second, since they predict one another correctly, where the assumption of the opponent maximizing its moves holds. What if randomness is added, in a controlled manner? Playing a move that is not expected by the opponent could potentially lead their strategy astray. Sporadic occurrences of randomness might progressively tip the balance and lead to a win.

2.6. Right-Side Capture

The rules are such that a capture can only be made if the last seed sowed happens to be in the second row, and that the directly opposing pits on the opponent's board are not empty. Also, the sowing process is counter clock-wise. In this case, attacking and trying to capture the opponents pits on the right-hand side would break their cycle and slow down their sowing[3]. Hence, given that a set of moves returns a board state score of equal value, the moves which lead to captures in the pits $\{0, 1, 2, 13, 14, 15\}$ are prioritized over the other pit positions.

2.7. Depth-Based Progressive Annealing

On many instances, the evaluation function returns an identical value for different moves. Other times, the values could be very close. But if the move does not make the cutoff even if it has a score of 1 point less, the move associated with that state will be discarded. The underlying values of that slightly worse state might in fact lead to an overall better outcome. Since depth is a limiting factor, allowing slightly worse moves to be chosen could possibly lead to better results. For example, setting a cut-off of 5 points would allow any moves generating a worse board state within that score range to be considered and possibly chosen.

3. RESULTS

The initial implementation of the agent based solely on a greedy approach yields results that are not optimal, but expected. Following 10 batches of 100 runs, alternating between starting first and second, the player wins approximately 95% of the time against a random player. This is to be expected, since having a heuristic function still beats a random move most of the time.

Using the Minimax algorithm, the agent wins 100% of the time based on 200 runs tested against the random player. The maximum depth that the agent could reach is 5, without generating timeouts.

Implementation of Alpha-Beta Pruning increased the maximum depth to 8. It wins 100% of the time based on 200 runs tested against the random player. When matched up against Minimax, it wins 67 games out of 100 when it starts first, and 55 when it goes second.

For every loss, the data obtained is used to figure out a pattern of the initial configuration more prone to losing. However, no conclusive observation could be obtained from the dataset.

Randomness and progressive annealing did not produce any significant differences in win rate when played against one another with a sample of 200 runs. It has a slight edge over conservative Minimax players.

4. MOTIVATION FOR THE TECHNICAL APPROACH

Several approaches have been considered and combined together in the process of creating the AI agent. After each implementation of a new algorithm or concept, tests have been run in order to obtain data for efficiency assessment. Depending on the results, the method tends to change accordingly. Hence, there is a motivation behind every step throughout the conception of the agent. Overall, the approach can be defined as a deep-searching engine which implements touches of randomness.

4.1. Greedy, Greedier, Greediest

The first attempt of creating an Omweso agent is to implement the logic based on greedy heuristics. Given that the agent plays a particular move, the algorithm returns the best decision according to greedy evaluation functions. The initial motivation behind this choice of method is because of speed and complexity. Supposing that the greedy algorithm does indeed return an optimal solution, it would be much faster and computationally more efficient than other extensive *heavy-weight* algorithms.

4.2. The Deepest Wins

Discussed in the results section, the greedy approach does not seem to be the best, nowhere near perfect. In the case of Omweso, the greedy algorithms fail to find the globally optimal solution[4], as it only considers the impact of a move one step ahead. To remedy this issue, the concept of searching through the game-tree comes to mind. Minimax is an adequate algorithm to be implemented in 2-PLAYER board games, as it tries to maximize and minimize possible moves from the player and the opponent, respectively. To increase the depth and enhance the time and space complexity, Alpha-Beta Pruning algorithm is used. The motivation is that if the agent can search and look at outcomes further ahead than the opponent, there is a lower risk of choosing a move that leads to an eventual loss.

4.3. Chaotic Behavior, Unexpected!

Having the impression that many classmates are also designing agents based on Minimax algorithms, the battles could be close and the outcome would mainly be defined by the depth of the search engine and the initial placement of the seeds. What if a touch of randomness is included? The reason behind this approach is based on the effect of suprise, when the opposing player does no expect an non-optimal move to be played. Hence, by including some *chaos* in the functionality and logic of the player, it might throw the opponents off. Doing so, slightly worse outcomes at early levels are not completely omitted.

4.4. One Step Back, Two Steps Forward

Progressive annealing is a concept that would allow worse moves to be chosen. Suppose at a certain level of depth the agent faces the decision between two outcomes separated by a marginal difference, it would choose the better one. However, the other move could potentially lead to a better outcome on the long run. As the algorithm goes deeper and deeper, the chances of allowing a worse move to be picked decreases.

4.5. Burn the Bridge

If the communication is severed between the front row and the back row on the opponent's side, it is much more difficult for them to carry out a capture, since relay sowing might be less probable to occur. Prioritizing captures on the right side of the opposing player's board removes the pits connecting their rows.

5. DISCUSSION

In this section, the pros and cons from the implementation of the agent is discussed, as well as the potential elements which could improve the current model.

5.1. Advantages and Disadvantages

The method used in the creation of the agent is based mainly on the Minimax algorithm, which explores the entirety of the game-tree up to a certain level. The advantage of employing such approach is that the moves are deterministic such that every possible board state can be simulated, up to a certain depth. The addition of Alpha-Beta Pruning helps decrease the computation cost, which leads to deeper exploration. Another advantage in the approach is that randomness is included in the process, which could give better results that would have been omitted otherwise. Random moves in a controlled regulation system could potentially be a good counter-attack to traditional deterministic agents. The implementation also attempts to disrupt the flow between the seeds on the opponent's board in order to defend against captures.

If being able to explore every single board state to a given level is a strength, then the level to which it can explore is a weakness. Not being able to search deep enough could cause a handicap against an opponent simulating further moves ahead. Random configurations and actions, albeit scarce and sporadic, could technically be risky and lead to losses. Not having a guaranteed starting *winning* configuration represents a flaw that could be improved.

5.2. Potential Improvements

The current implementation is not the best it could be nor is it perfect. There are different algorithms or concepts that could be applied in order to optimize and improve the agent.

5.2.1. Monte Carlo Algorithm

The depth at which the search tree can be explored is limited by time and space complexity. Instead, a depth limit could be set and when that level is reached, another algorithm takes action. The Monte Carlo method relies on random sampling to provide the best result[5]. The search in the branches for Minimax are deterministic, but at a certain depth, it becomes too computational exhaustive. This is the reason why repeated sampling could help infer deeper without being a liability complexity-wise. And again, the concept of randomness comes into play and could eventually lead to better outcomes.

5.2.2. Seeds Count and Pit Positioning

Instead of counting the difference in the number of seeds in each player's possession, a better evaluation could also take into consideration the number of pits that contain 1 seed or less. For instance, given that the board state of the opponent contains a total of 15 seeds, it would be useful to know the positioning of those seeds. All 15 seeds spread across the board with 1 seed per pit is a losing state, but a state with 15 seeds distributed into 4 pits still has a chance of fighting.

5.2.3. Initial Winning Positions

Although the tests did not return any significant pattern or configuration concerning the starting positions, there definitely are starting combinations of seeds placement that yield a higher percentage of winning compared to others. Given that a *winning* starting configuration could eventually be determined, the chances of winning are increased.

6. CONCLUSION

In order to design a competitive agent capable of winning at Omweso, several different algorithmic approaches are explored and some of them combined together with the endeavour to increase the probability of winning games. Omweso requires strategic thinking and state inference to make the best move. It would be interesting to see how professional Omweso player in the world plan their strategies, and how the agent would fare against the best of them.

7. REFERENCES

1 http://www.geocities.ws/omweso/board_games_in academia_v_omweso.pdf

2 Mayega, J. V. Omweso: A Mathematical Investigation of an African Board Game. Department of Mathematics, Makerere University, Kampala (Uganda) 1974.

3 https://groups.yahoo.com/neo/groups/mancala games/conversations/messages/74

4 http://people.csail.mit.edu/hpirsiav/papers/tracking cvpr11.pdf

5 http://en.wikipedia.org/wiki/Monte_Carlo_algorithm