"IAgo Vs Othello": An artificial intelligence agent playing Reversi

Jacopo Festa, Stanislao Davino

festajacopo@gmail.com, woods88@hotmail.it

Abstract. This paper presents the application of fundamental Artificial Intelligence algorithms for the realization of programs capable of effectively playing the game of Reversi. This application has been developed as a final project for the course of Artificial Intelligence, held in the academic year 2012/2013 at the Engineering Faculty of the University of Salerno, Italy. Several agents have been developed and tested, either in agent vs. agent and agent vs. human games. The application is freely available for download at the link specified in [8].

Keywords. Reversi, Minimax, Alpha-Beta Pruning, Heuristic Functions, Java, Game

1 Introduction

Our goal is to create an application that uses artificial players for the game of Reversi. This application is named "IAgo Vs Othello" (a wordplay that refers to the famous Shakespeare's tragedy of "Othello"), and is written in Java language.

It allows a human player to challenge any of the implemented artificial intelligences or to spectate a challenge between artificial players, or instead allows two humans to challenge each other. Our work focused on "classic" search algorithms like Minimax and Alpha-Beta Pruning and envisages different strategies in order to compute the next move in a reasonable time for a human opponent.

Different strategies have been implemented, which can be combined with any algorithm. Then we focused on testing and evaluating the performances of our agents, with different algorithms, on varying search depth, the used optimizations, and the strategies.

In Section 2 the theory on which the used algorithms are based upon is introduced, and the heuristic functions used are described too; Section 3 is about the application and the experiments done, and eventually, in Section 4 we will discuss the obtained results and draw some examples of future developments.

2 The Agent

An agent is an entity capable to evaluate the state of the game board and to select a move from the set of the available ones. An agent uses a specific algorithm to explore the available moves and their consequences, and a strategy to evaluate them.

2.1 The algorithm

The algorithms implemented in this work are variations around a basic algorithm, called Minimax. This algorithm comes from Von Neumann's Game Theory [1, chapter 5] [2]; it works mostly like a human would think: starting from his currently available moves, it selects one of them and evaluates its consequences, i.e. how the game state will evolve as a result to that particular move.

In this game state "projection" the algorithm knows that, as the players alternate between each other (unless a special situation occurs), it is the opponent player's turn, so it selects an opponent's move and evaluates its consequences; after this, it is the player's turn again, and a new move is selected.

The algorithm goes on until it reaches a game over state. It keeps track of the moves sequence that brought to that state and explores other possibilities, by going one step backward and selecting other moves. This is done systematically until all the game over states have been evaluated, and the best of these is elected as objective (i.e. a state that means victory for the agent, or if no victory is reachable, a state that means draw).

As the agent knows the sequence that leads to the objective state, it now simply selects the first move that brings the game towards it.

There are two main issues with this approach. Firstly, the chosen sequence is composed by both the player's and the opponent's moves; the agent can make a guess about which move the opponent will choose, but this is only a mere prediction. If the opponent chooses a different move, the game will evolve differently.

This is not a concern at all, because this agent is an optimum player (a player who always tries to maximize its final points) and works with the assumption that the opponent is an optimum player too, struggling to minimize the agent's score, so the agent will select as the opponent's moves those that lead towards a minimization of its final points. If the opponent doesn't choose optimum moves, then the player will get a higher score than expected because, if there was a different set of moves that could further minimize the agent's score, the opponent would have chosen it, so other non-optimum moves can only lead to a smaller score for the opponent, and a greater score for the agent.

The second issue is a more problematic one. Suppose, for the sake of simplicity, that at each step there are 10 available moves, that lead to 10 different game states. For each of these states, there are other 10 moves as well; considering all

the possible 2-moves sequences, we have a total of 100 different states. All the possible 3-moves sequences lead to a total of 1,000 different states, and so on. The number of states to be evaluated grows exponentially the deeper the algorithm goes. For Reversi, with a total of max 60 turns, the number of states to be evaluated has been estimated to be approximately 10^{58} [3].

Exploring so many states cannot be done at all with currently available computers: computing time required to evaluate a single move would be more than the age of the universe itself! Our objective is to have agents capable to compute a move in a reasonable time for a human opponent, i.e. a few seconds.

In order to do this, beside Minimax, we have implemented another well known algorithm, called Alpha-Beta Pruning [1, chapter 5]; it works like Minimax, except for the fact that it stops the evaluation of a possible move when it knows that this move is always worse than another one already evaluated. Alpha-Beta Pruning is a good optimization of Minimax because achieves the same results using less time and memory, as less moves and less states are evaluated.

Even using Alpha-Beta Pruning, though, computing time is still too much heavy. So, the search is cut when a certain maximum depth is reached. With this variation, *terminal states* (i.e. states evaluated at the max depth) generally aren't game over states, and the agent does not know if they are good or bad ones. To evaluate these states we must give the agents some knowledge that can be used to determine what is the right thing to do in a particular game situation. Different knowledge (called "*heuristics*") lead to different strategies and different playing styles.

2.2 Heuristics

2.2.1 Heuristic on Score (HS)

The simplest heuristic function for the game of Reversi calculates the score of the player (i.e. the number of disks of his color currently on the game's table). Intuitively, may seem like a good strategy, because in the end, victory goes to the player with the highest number of disks of his color. Actually, the score during the game may greatly vary from one turn to the other in such a way that having more disks in a certain game state does not necessarily represents a real advantage. This heuristic does not have good performance, but is useful to compare the behavior of other and more complex heuristic relative to a "beginner" player behavior.

2.2.2 Heuristic on Mobility (HM)

With this heuristic, we do not consider the actual score of the player, instead we calculate some parameters:

- *Mobility of the player:* the number of available moves;
- *Potential mobility of the player:* the number of empty slots next to at least one disk belonging to the opponent;
- *Potential mobility of the opponent:* the number of empty slots next to at least one player's disk;

It's important to note that if a player cannot play legal moves (in other words his mobility is equal to zero), he is obliged to pass the turn, that is a serious disadvantage for him and a big advantage for the opponent, who probably will win the game. This way to estimate the utility is much better than HS, but is anyway far from the optimum because, in late game, mobility tends to decrease for both the players.

2.2.3 Heuristic on Mobility and Corners (HMC)

One fundamental strategy in Reversi is to focus on capturing the corners. These represent very important positions. A disk is *stable* if it can't be turned to the opponent's color; any disk placed in a corner is, by definition, stable and makes stable every adjacent disk along the edges. For this reason, in addition to mobility, a good heuristic function has to consider the corners and the positions near them. Taking the corners is more important than keeping an high mobility, and so we appoint an heavy weight to the corners in the evaluation. We also assign a negative score to those positions that give to the opponent access to the corners as well, because those are undesirable positions.

2.2.4 Heuristic on Mobility, Corners and Edges (HMCE)

Corner positions are fundamental, but positions along the edges are important too; this heuristic function tries to take into account this fact by considering the number of disks the player has on each edge, and subtracting the number of opponent's disks to this value. With this heuristic, the agent tries to play more on the edges (in addition to the corners) and tries to stop the opponent from taking them.

2.2.5 Heuristic on Mobility, Corners, Edges and Stability (HMCES)

HMCE confers a positive score to a state of the game in which the player has more disks along the edges than the opponent (including corners), and a negative score otherwise. However, the strategic meaning of the disks along the edges is more important if those disks are also stable, i.e. they can't be captured by the opponent. An edge disk is stable when:

- It is placed in one of the four corners;
- It is placed in a row or in a column of disks of the same color that ends at least in one of the four corners.

We chose to evaluate only the stability at the edges positions, as these positions are easier to compute; central disks' stability, after all, depends from the stability at the edges (due to the recursive nature of the stability definition), and it is less important.

2.2.6 Heuristic on Mobility, Corners, Edges and Stability, Time-variant (HMCEST)

During a Reversi match, several strategies can be used depending on which phase the game is in. A Reversi match requires 60 turns (unless someone succeeds to win earlier). For this reason we decide to evaluate the scores on mobility and stability depending on the current game turn, assigning more or less importance to both strategies depending on the current phase the game is in. This heuristic works by computing separately the mobility-related score and the stability-related one; then the final score is given by multiplying these scores by some weights and sum them with the corner score (which is not weighted). Let w_m and w_s be the mobility and stability weights, respectively, and T the current turn number; weights are determined as follows:

- $w_m = 1.5$ and $w_s = 0.5$ if $T \leq 20$;
- $w_m = 1.5$ and $w_s = 0.5$ if $20 \leq T \leq 40$;
- $w_m = 1.5$ and $w_s = 0.5$ if $T \geq 40$.

3 Playing with "IAgo Vs Othello"

The application has been written in Java 1.6; we chose this programming language in order to have a cross-platform, self-contained application that can be instantly downloaded and launched, and because of the libraries that come with Java, which allow to make easily complex applications and good user interfaces.

"IAgo Vs Othello" is available for free download at [8], and it is released under a GPL3 license.

All the agents have been written from scratch, based on pseudocode available in [1, chapter 5]; there are, however, some variations with respect to the original version, because the algorithm has to manage the situation in which there isn't any available move; according to Reversi rules the player has to pass the turn, so the node currently evaluated must create a new child node without any variations on the game state, where now is the opponent's turn. If also the opponent has no available moves, then this child node represents a game over state.

The implemented algorithms are:

- Minimax;
- Alpha-Beta Pruning;
- H-Minimax (Minimax with support for heuristic functions);
- H-Alpha-Beta Pruning;
- Randomizer (simply selects the move randomly);
- Greedy (selects the move that flips the maximum number of opponent's disk).

Minimax and Alpha-Beta Pruning, without heuristics, cannot be used on a 8x8 table, so they haven't been tested in our experiments; they can choose a move in a reasonable time on a 4x4 table though. Minimax and Alpha-Beta-Pruning (and their heuristic versions too) have a time complexity that goes like $O(b^d)$ and $O(b^{d/2})$, respectively, according to the theoretical ones, where *b* is the *branching factor* (mean number of moves available in any game state), and *d* is the maximum depth. Computing times have been also experimentally verified.

The application allows a user to play against an artificial agent, or against another human; it also allows to spectate a match between two artificial agents and see the outcome of it.

There are two different interfaces available to start a new game; the complex one allows to generate one or two personalized agents by selecting an algorithm and some options, like the heuristic function and the max depth; the basic one presents the user with five different predefined agents, which correspond to different levels of difficulty.

To evaluate the agent performance, we let the agents we have developed play against each other and collect information from the matches, like the number of victories, defeats, draws, mean root branching factor for each turn, mean time required to compute the next move for each turn, etc., for each agent. We use also agents that use the same strategy but they evaluate the moves at different depths: these maximum depths are: 1 (only the currently available moves are evaluated), 3, 5 and 7. All the agents in the shown experiment use the same algorithm (H-Alpha-Beta Pruning) in order to keep computing time in a reasonable limit. The goal of this experiment was to measure how depth affects both computing time and game performance, and to evaluate the quality of the tested strategies. Results are shown in Table 3.1.

This experiment has shown that:

- As max depth increases, we can see a better playing ability, which is demonstrated by the raw victories/defeats ratio;
- As max depth increases, we can see mean computing time for each move strongly increases, in according to their exponential time complexity;
- At the maximum tested depth (= 7), we can see that HMC is the heuristic that behaves better. This can be explained by the fact that, as depth increase, a more complex heuristic may cause an agent to be distracted by more, simultaneous, and less important objectives, while a simpler one can focus towards more important ones; this may suggest that, as an objective, stability is somewhat less

important than mobility and the edges capturing; maybe an even better agent can be created by properly weighting these scores, like we tried to do with the HMCEST heuristic.

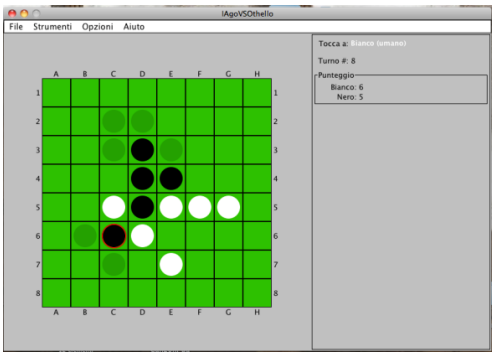

Fig. 3.1. Game situation in a Reversi game. Legal moves for the current player (the white one in this example) are highlighted with dark green disks.

Agent	Victories	Defeats	Mean computing time (microseconds)
H-ABPruning-D1-HMC	44	295	116
H-ABPruning-D3-HMC	115	223	3498
H-ABPruning-D5-HMC	169	171	61762
H-ABPruning-D7-HMC	329	11	1079018
H-ABPruning-D1-HMCE	99	240	124
H-ABPruning-D3-HMCE	176	163	3614
H-ABPruning-D5-HMCE	250	90	66650
H-ABPruning-D7-HMCE	300	40	1152387
H-ABPruning-D1-HMCES	99	241	134
H-ABPruning-D3-HMCES	190	150	3837
H-ABPruning-D5-HMCES	260	80	69279
H-ABPruning-D7-HMCES	300	40	1223536
H-ABPruning-D1-HMCEST	65	264	127
H-ABPruning-D3-HMCEST	139	191	3774
H-ABPruning-D5-HMCEST	210	130	72598
H-ABPruning-D7-HMCEST	260	80	1089245

Table 3.1. Victories, defeats, and mean computing time for each tested agent.

A series of games against human players have been done too, using agents corresponding to five different difficulty levels; so far, human players have performed well against *Beginner*, *Easy* and *Medium*. Some have been able to defeat *Hard*, but no one could defeat *Very Hard*. We plan to arrange a Reversi contest with more human players to play against the artificial agents, in a strictly con-

trolled environment, in which we can collect data such as victories/defeats ratios, computing time, etc. for both human players and the artificial agents. As there are many available implementations of agents playing Reversi, like Logistello [6] and Cassio [7], we also plan to try our agents against those programs.

4 Conclusions and Future Development

Despite the relative simplicity of the implemented algorithms, these reveal some kind of intelligent behavior; some behaviors we seen with HMC and more advanced heuristic include:

- The agent forces the opponent to select bad moves, from which the agent itself can benefit;
- The agent tries to drive the game flow towards situations in which it can take over a corner; but sometimes it just doesn't take control of the corner immediately, but, knowing that the agent can take it in a second time because the position isn't threated by the opponent, the agent prefers to focus on taking other strategic positions and then takes the corner later;
- Some moves seem to be stupid at first sight (like moving in the center of the table instead of the edge), but, a few turns later, they reveal to be part of a complex scheme that the agent choose for its strategy.

The algorithms implemented and presented in this paper are, however, "basic" ones; our application could be extended by implementing and testing new algorithms based on search trees exploration (*Beam Search*, for example), or to approach the problem differently by implementing an agent capable of learning to play through accumulated game experience.

5 References

[1] Stuart *Russell*, Peter *Norvig, Artificial Intelligence: A Modern Approach (3rd edition), Prentice Hall, ISBN-13: 978-0136042594;*
[2] John von Neumann, *On the Theory of Games of Strategy*, "Contributions to the Theory of Games", n. IV, 13-42, 1959;
[3] Victor Allis, *Searching for Solutions in Games and Artificial Intelligence*, Ph.D. Thesis, University of Limburg, Maastricht, 1994;
[4] Brian Rose, *Othello: A Minute to Learn... A Lifetime to Master*, 2005: http://othellogateway.com/rose/book.pdf;
[5] Reversi Wikipedia page: http://en.wikipedia.org/wiki/Reversi;
[6] Logistello Homepage: https://skatgame.net/mburo/log.html;
[7] Cassio Homepage: http://cassio.free.fr/;
[8] IAgo VS Othello download link: http://nclab.diiie.unisa.it/iago_vs_othello/IAGO_VS_OTHELLO.jar