Raymond Liu

*Épreuve Synthèse* abstract

420-LCU Computer Programming

Robert Vincent

May 13, 2022

**Analysis of Computer Player Algorithms in Reversi**

Raymond Liu, Marianopolis College, Montreal, Quebec, Canada

In the third assignment of this course, students were instructed to create an algorithm to choose moves for the board game Reversi. In practice, Reversi computer players are frequently based on the paradigm of minimax. Firstly, a brief introduction to minimax is provided. Then its technical details are further explored, notably implementation and issues in the context of efficiency. Secondly, Alpha – Beta pruning, a variation of minimax that seeks to address this deficiency will be explored in how its modified implementation fixes minimax’s inefficiency. Lastly, the effectiveness and efficiency of these algorithms and a combined algorithm are quantified and analyzed from data gathered from 100 simulations of Reversi for each.

Keywords: Reversi, Computer Player, Minimax, Alpha-Beta pruning, Efficiency, Computer Algorithms

**Analysis of Computer Player Algorithms in Reversi**

Raymond Liu

Marianopolis College

420-LCU Computer Programming

Robert Vincent

May 13, 2022

INTRODUCTION

For the past 40 years AI techniques have been applied to board games such as chess, checkers or Reversi, also known as Othello. These are examples of two player, *zero sum,* and *perfect information* games. *Zero sum* means that the gain of one player is the loss of its opponent, while *perfect information* means that all relevant information is open to both players (Lim, 8-9). Computer players are particularly effective in these kinds of games as they can leverage their superior computational abilities to accurately measure the desirability of each move using zero-sum and perfect information.

Reversi is played on an 8x8 board where a player must place a piece such that a straight line is formed between it and one of their pieces with atleast one opponent piece between them. All opponent pieces on that line are then flipped. The player with the most pieces at the end of the game wins. In 1997, the world Reversi champion Tekashi Murakami was defeated by Logistello, a computer player based on minimax (Buro,2). Modern computer players now consistently beat their human counterparts, however still frequently incorporate some variation of minimax. The purpose of this paper is to provide a brief overview of minimax, its common optimization alpha – beta pruning and to provide an analysis of their effectiveness in Reversi.

MINIMAX

Introduced in 1928 by John Van Neumann, minimax is the most well-known decision rule for two-player, zero sum games (Roberts). Minimax’s principal idea is that the computer, or the maximizer, will seek to maximize its score, while its opponent, the minimizer will try to minimize the maximizer’s score. Assuming its opponent plays perfectly (chooses the best move at every possible instance), the computer will look ahead into future states of the game and choose the optimal move. The way the computer does this is through a tree. Starting from a *root node* representing the current game state, branches with node are created representing a new game state after a move is played. Each additional level of nodes is said to increase the depth of the search by one. The nodes at the end of the branches are called *terminal nodes* and are what ultimately affect which path is chosen. A tree allows the computer to envision all possibilities up to the terminal nodes at a predefined depth or until no more legal moves can be played. In combination with other techniques such as weight tables, computer players can “solve” games, meaning it is impossible for them to lose. Simpler games such as tic tac toe or versions of Reversi with a smaller board (4x4 and 6x6) are considered to be solved.

An important metric when quantifying the efficiency of an algorithm is its time complexity, which is the amount of time taken by an algorithm to run, as a function of different parameters of the input. Time complexity is frequently represented by “big O” notation, which is the mathematical notation that describes the limiting behavior as it approaches its asymptotic value (usually infinity). For traditional minimax, this time complexity is , where b is the number of branches per node or the branching factor, and d is the depth of the tree (Megalooikonomou). Minimax’s time complexity is constant due to the nature of its depth first search. The algorithm will traverse as far down a branch as it can, then backtracks until it finds an untraversed branch and repeats the process. This process is continued until all branches and nodes have been searched, resulting in a left to right search of the terminal nodes.

While doing a search that includes all nodes is not an issue for smaller games such as tic-tac-toe where the amount of game states is low, it is in more complex games. This is especially true in games with a high branching factor. Reversi on average has a branching factor of 10 (Norvig, 646) and 60 moves played in a normal game, resulting in a total of possible moves. This number is relatively small in comparison to other games such as chess where the branching factor is 35 and the average number of moves per game is 80, resulting in possible moves to consider (Koch). As it may be evident, the enormous number of nodes that need to be traversed by the standard minimax algorithm may begin to pose performance issues in systems with less computational power.

ALPHA-BETA

Introduced in the early 1960’s by Alexander Brudno (Marsland, 6), Alpha – Beta pruning, also known as α-ẞ is an optimization of minimax that seeks to improve its efficiency by decreasing the number of nodes that need to be evaluated. By removing branches that do not influence the final result, the tree can be “pruned”, producing the same final result as traditional minimax (Russel and Norvig, 167) but with sometimes significant performance increases. In fact, a chess computer player that searches to a depth of 4 with 36 branches per node will need to evaluate over a million terminal nodes in a traditional minimax. By applying an alpha – beta prune, this number reduces to 2000, a reduction of 99.8% (Levy,102)

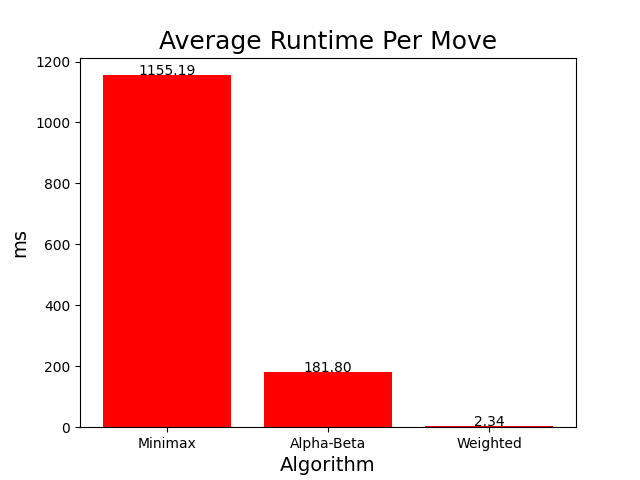
α-ẞ prunes branches by tracking two values. Alpha, the best maximizing score so far found, and Beta, the best minimizing score so far found. Depending on if it is the maximizing or minimizing player’s turn, the algorithm will compare the values of its branches to the current alpha or beta. For example, on the minimizing player’s turn, it will traverse down its branches and begin evaluating. As soon as a branch returns a value greater than Alpha, it can be pruned as the minimizing player would not play that branch. Upon completion of the search, Beta is updated to be the best minimizing value found or remains the same if one is not found. A similar process is done for the maximizing player, except each branch is checked to Beta and the value updated is Alpha. (Lim, 19)

While an α-ẞ generally presents an improvement in runtime over traditional minimax, it is not consistent. This is due to the depth first nature of the search, similar to minimax. If the most optimal result is in the far right of the tree, α-ẞ will nonetheless have to search the entire tree like minimax. However, if the most optimal result is in the far left of the tree, α-ẞ will prune all branches to the right of each node, resulting in the most performance gains. α-ẞ’s runtime in the best case is , while its worst case is , (Norvig, 616) averaging at around . Effectively this means that α-ẞ pruning on average can search double the depth of its search in the same amount of time compared to minimax.

MINIMAX & α-ẞ IN PRACTICE

In order to observe the differences between minimax and α-ẞ in practice, 100 games of Reversi were simulated against a randomized move selector for each of the following algorithms: a traditional minimax algorithm with a search depth of 3, a α-ẞ pruned minimax algorithm with a search depth of 3 and a weighted table combined with a minimax algorithm with a search depth of 1. The inclusion of the simple algorithm combining minimax and a desirability table for each position of the board originally created for Assignment 3 is meant to simulate real world algorithms that take a multi pronged approach to their gameplay strategy.

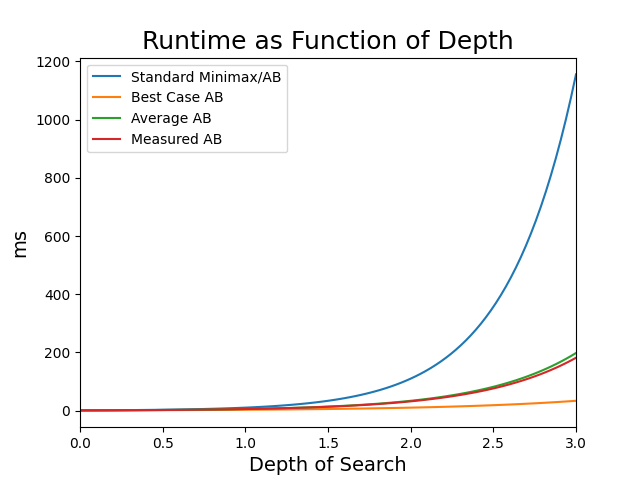
Unsurprisingly, as indicated in Figure 1 below, traditional minimax was the worst performing with an average runtime of 1155.19 milliseconds per move. α-ẞ presented an improvement of over 6 times at 181.80 milliseconds per move while the combined algorithm took a mere 2.34 milliseconds per move.



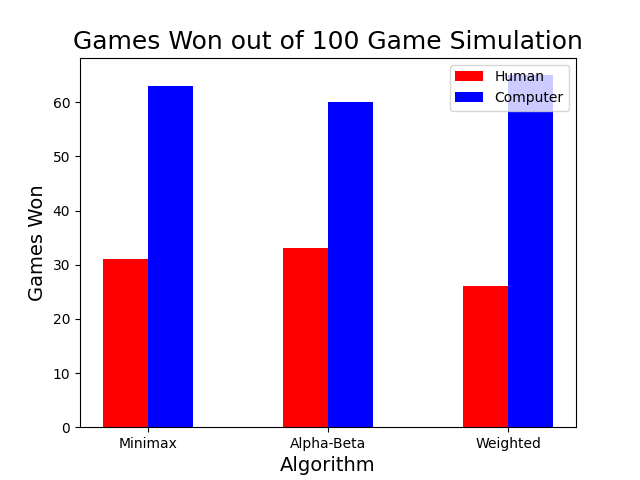
Some basic arithmetic with the results gathered from the minimax runtime also confirms Reversi’s theoretical branching factor of 10, which was measured as 10.49 during simulations.

Assuming that α-ẞ had the same branching factor during its simulations, its overall time complexity throughout the simulation can be calculated to

This time complexity is roughly in line with the expected average , although α-ẞ performed slightly better than expected. These results have been graphed below in Figure 2.



What is interesting to note is that not only did the combined algorithm prove to be by far the most effective in terms of runtime, but it also performed the best in terms of games won as indicated in Figure 3 below, winning 65 games compared to the 63 and 60 of minimax and α-ẞ respectively. This emulates real – world computer players, which almost always combine some variation of the minimax algorithm with other relevant algorithms for the game. Logistello for example uses Probcut, a further improved variation of minimax in conjunction with game stage dependent tables and an opening book (Buro, 3)



CONCLUSION

As highlighted by the results of the simulations, while minimax and α-ẞ are effective tools in designing computer players, using a combined approach is not only much more efficient but also produces better results. This is illustrated by the superior performance of the combined algorithm as well as the design choices of real-world algorithms such as Logistello. This observation can be extended beyond just the design of computer players. As computers tackle ever-increasingly difficult problems, the optimal solution will often involve approaching the problem from multiple angles.

**References**

Buro, Michael. “LOGISTELLO - A Strong Learning Othello Program.” *NEC Research Institute* , 1997, https://skatgame.net/mburo/ps/log-overview.pdf

Chen, Janet, et al. “Strategies of Play.” *CS Stanford* , Stanford University, 1998, https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/game-theory/Minimax.html#:~:text=The%20Minimax%20algorithm%20is%20the,choice%20that%20a%20player%20makes.

“Computer Othello.” *Wikipedia*, Wikimedia Foundation, 13 May 2022, https://en.wikipedia.org/wiki/Computer\_Othello.

“Depth-First Search (DFS).”*Brilliant Org,* Brilliant.org, May 13, 2022, https://brilliant.org/wiki/depth-first-search-dfs/

Great Learning Team. “Why Is Time Complexity Essential and What Is Time Complexity?” *GreatLearning Blog: Free Resources What Matters to Shape Your Career!*, My Great Learning, 1 Apr. 2022, <https://www.mygreatlearning.com/blog/why-is-time-complexity-essential/#:~:text=Time%20complexity%20is%20the%20amount,you%20understand%20time%20complexity%20clearly>.

Koch, Christof. “How the Computer Beat The Go Master.” *Scientific American*, Scientific American, 19 Mar. 2016, https://www.scientificamerican.com/article/how-the-computer-beat-the-go-master/#:~:text=Given%20that%20a%20typical%20chess,on%20how%20to%20program%20a.

Levy, David. “Alpha-Beta Soup.*”* *MacUser,* Jan. 1986, https://archive.org/details/MacUser8601January1986/page/n107/mode/2up

Lim, Yew Jin. “On Forward Pruning in Game-Tree Search.” *National University of Singapore*, 2007, <https://web.archive.org/web/20110718141259/http://www.yewjin.com/storage/papers/PhDThesisLimYewJin.pdf>.

Marsland, T.Anthony. “Computer Chess Methods”. *University of Alberta*, 15 Dec. 1990, https://web.archive.org/web/20220422005343/https://webdocs.cs.ualberta.ca/~tony/OldPapers/encyc.mac.pdf

Megalooikonomou, Vasileios. “Lecture 7.” *CIS Temple Edu,* Temple University, 2003, https://cis.temple.edu/~vasilis/Courses/CIS603/Lectures/l7.html#

Norvig, Peter. “18 Search and the Game of Othello.” *Paradigms of Artificial Intelligence Programming*, Morgan Kaufmann Publishers, San Francisco, CA, 1992, pp. 596–655.

Russell, Stuart, and Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd ed.)*. Pearson Education, 2010.

MARIANOPOLIS COLLEGE

SCIENCE PROGRAM ÉPREUVE SYNTHÈSE

**SELF-EVALUATION FORM ON PROGRAM GOALS**

NAME: Raymond Liu ID: 2031256

ASSESMENT ACTIVITY: Épreuve Synthèse

SUPERVISING TEACHER: Robert Vincent COURSE: 420-LCU

DATE: May 13, 2022

|  |  |  |  |
| --- | --- | --- | --- |
| **PROGRAM GOALS** | | **BRIEF EXPLANATION OF HOW YOUR ACTIVITY CONTRIBUTED TO**  **MEETING THE PROGRAM GOALS** | |
| **1**  Learning and use of scientific vocabulary | | Used and explained relevant vocabulary to topic of research paper. Examples include abstract concept like a tree in computer science, or time complexity and big O notation. | |
| **3**  Use of graphical representation | | Used graphs generated through MatPlotLib to visualize and aid analysis of data gathered through simulations. | |
| **4**  Ability to observe and analyze | | Analyzed data gathered through simulations and compared them to theoretical values/results. | |
| **6**  Use of mathematical tools | | Use of basic algebra to analyze and draw meaningful conclusions from data gathered from simulations. | |
| **8**  Data processing technology | | Used Python to both generate, and analyze data gathered for report. Used computer to aid in research, type research paper and to run simulations. Used graphing tools (Desmos) to double check validity of MatPlotLib generated graphs | |
| **11**  Written communication | | Communicated information and findings through written paper. Conducted extensive reading of other academic papers during research. | |
| **12**  Autonomous work | | Independently researched paper, conducting simulations of algorithms, gathered, and analyzed data. | |
| **17**  **Stimulation of intellectual curiosity** | | Further explored the concept of algorithms seen in class, conducted further research into the concept and applied it in practice in simulations. | |
| **23**  **Application of knowledge to new situations** | | Used newly acquired knowledge of algorithms, matplotlib and Python during research paper. Further developed knowledge of algorithms during research and matplotlib during data graphing. Refined Python skills while implementing algorithms | |
| **24**  **Integrative activities** | | Integrated writing, math and programming skills while writing paper and conducting research. | |

Comments:

The Épreuve Synthèse proved to be much more challenging than I initially expected as I lacked a lot of the knowledge behind the algorithms I had originally planned to write about and had to learn many concepts of computer science while writing. This resulted in having to reduce the scope of the paper multiple times to ensure I was well versed in what I was writing about. Finding published papers also proved to be difficult as most originated from 1960-1990 and used advanced concepts in computer science that I had difficulty understanding. However, online resources proved to be helpful in aiding me understand the parts relevant to the paper. In addition, many of the articles cited by Wikipedia were either inaccessible, or sketchy at best, forcing me to make the less-than-optimal choice of using Wikipedia as references. The simulation portion of the paper also proved to be challenging but enjoyable as I had to learn how adapt the 3rd assignment to implement both minimax and alpha-beta while also finding a way to track the metrics I wanted to. MatPlotLib was frustrating to work with and I would have much preferred to simply export my data to excel, but I nonetheless appreciated the exposure to the library. I was also surprised by the effectiveness of the original algorithm I had created for Assignment 3. Overall, I found the Épreuve Synthèse to be a challenging but fulfilling experience. I garnered a greater appreciation for the intricacies of computer science and this experience solidified my desire to pursue my university studies in this field.