Application of Different Artificial Intelligence Methods on Reversi

Yiwei Han

University of Rochester, Rochester, USA

yhan32@u.rochester.edu

Abstract. Using different algorithms in artificial intelligence to perform adversarial agent games. The most popular adversarial game that could be implemented using artificial intelligence algorithms is chess games. The algorithm this research used includes minimax with improvement and deep reinforcement learning. The goal for this research is to compute the popular game Reversi in different artificial intelligence methods successfully. Moreover, the research seeks for improvements in the heuristic part of minimax algorithm and the combination of deep reinforcement learning with Monte Carlo Tree with Neural Network. This paper uses Reversi as an example to analyze different algorithms, including Minimax, Monte Carlo Tree, and Neural Networks. As a result, both algorithms work successfully. The alpha-beta pruning minimax algorithm with improvement in heuristic function and fixed depth cut-off significantly increase the winning probability and time cost of our artificial intelligence agent. The deep reinforcement learning successfully combined MCTS with neural network to train two agents to complete Reversi with great winning probability.

Keywords: Reinforcement Learning; Minimax; Heuristic; MCTS.

1. Introduction

Adversarial games are most popular type of games among all time. They have been developed as board games long time ago, which two players play together. Nowadays, because of the development of Internet, many adversarial games have been developed as mobile apps. There is no longer the need for two players playing together-artificial intelligence can be the opponent [1]. Since then, many artificial intelligence algorithms have been developed over time to maximize difficulty and winning rate of the artificial intelligent agent. The artificial intelligence algorithms have simple state space search, which requires domain knowledge. Moreover, as time develops, manually extracting domain knowledge is often costly, and human expert data is often expensive, unreliable or even unavailable. Even if reliable expert data are available, the supervised learning system often achieves a performance bottleneck [2-3]. Algorithms like MCTS and deep reinforcement learning are more effective [4]. They only need the basic rules of the game, and learn the game from the beginning, and eventually train the AI to perform effectively and perfectly based on the data given by the training.

This paper will introduce the subsequent sections arranged as follows:

First, this paper will introduce the design of Reversi, including the layout, rules, and the design of the state- space. Second, the paper will introduce basic state space search for games which require the domain knowledge. The first algorithm is Minimax, which is the full state space search of the game. Then, this paper will introduce heuristic Minimax algorithm with alpha-beta pruning and fixed depth cut-off as a modification since the large and unachievable state space because of limited time and space complexity. Thirdly, the paper will introduce deep reinforcement learning methods including Monte Carlo Tree search and Neural Network. The MCTS methods evaluate each step by looking ahead for the moves in the state space while the Neural Network methods evaluates the best next move based on the current situation. The distinction makes them perform differently. Lastly, the paper will evaluate the performance of the deep reinforcement learning with loss function.

2. Design of Algorithms

The design of the algorithm includes 3 parts: modeling of the Reversi game and rule, Minimax algorithm (with its modification), and deep reinforcement learning algorithm. This paper will introduce the design of the algorithms by the above order. The Minimax algorithm and deep reinforcement learning algorithm both are based on the structure of the modeling of Reversi. In the modeling of Reversi, the algorithm includes the rule, winning check, nodes, and state space that better fits the further modeling of the two artificial intelligent algorithms. Both algorithms will use the nodes and state space, as well as condition winning to do searches among states, and rewards of the deep reinforcement learning algorithm.

2.1 Modeling of Reversi

The game is played on a rectangular board divided into squares and the standard board size is 8x8. The pieces are disks with two sides, corresponding to the two players. Traditionally the pieces are dark on one side and light on other [4-5]. When it is their move, a player must place a piece with their color up such that there is a line of one of more pieces of the opponent's color between the new piece and one of the player's own pieces already on the board. That is, each move must "trap" a line of the opponent's pieces between the player's own pieces, including the piece being played. The line of one or more trapped pieces may be horizontal, vertical, or diagonal. After playing their piece, the player flips over the line of trapped pieces so that they all show the player's own color. Note that a player may capture more than one line of their opponent's pieces in a single move. If a player cannot make a legal move, they must "pass". The game ends when neither player can make a legal move. Whoever takes more pieces on the board wins. And there are circumstances to a draw when the number of two-color pieces is the same.

2.2 Minimax

Minimax is a typical adversarial search algorithm. The two agents' goals are in conflict. And its utility is a zero-sum game, which means what good for me is bad for you.

Minimax algorithm is the algorithm that there are two players in the game, one is Max and one is Min. The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree [6]. It expands a node's children before its siblings. The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion. Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit. MAX will select the maximized value and MIN will select the minimized value [7]. The algorithm generates the entire game tree and use the utility function to get to the utility value of the nodes in the terminal state. Then, it uses minimum value for each node in each layer alternate with maximum value for each node in each layer to get the best choice for the state space search.

The winning probability for the Minimax algorithm is 100%. It is optimal [8]. It searches through all the state space and makes the optimal move. However, the time complexity limits how far minimax algorithm can go.

2.2.1 Heuristic Minimax with Alpha-Beta Pruning with Fixed Depth

The Minimax algorithm's time complexity is extremely large. It is O(b^m), where b is the max number of children a node can value, and m is the length of the longest path [2]. It has exponential search space complexity, which is hard to achieve if the searching space is too large. In this case, the time complexity of 4x4 Reversi is achievable for the algorithm, however, the time complexity for larger Reversi board is too big, which makes it impossible for use to use the basic Minimax to perform the search. As a solution, we used Heuristic Minimax with Alpha-Beta Pruning with fixed depth, which is a modification for the original Minimax algorithm.

Minimax with alpha beta pruning is a way to reduce the state space search for the algorithm. In the algorithm, two parameters are needed: an alpha value which holds the best MAX value found for MAX node; and a beta value which holds the best MIN value found for MIN node, the remaining

children can be aborted if alpha ≥ beta [7]. It can significantly reduce the number of nodes searched. As a result, both time and space complexity will be improved than the original Minimax Algorithm.

2.2.2 Heuristic Function

Heuristics are a way to guide a search in the right direction toward a target. It offers a methodical approach for determining which node's neighbor will lead to a particular objective [9]. For the heuristic evaluation function for Othello, we used a method that calculates the best possible score based on the rule. The Heuristic Function's score is based on checking number of corner occupied and number of next to corner occupied then give each score a weight. Taking the corner is a good thing, so we weight this as positive. Next to corner is not a good thing, so we weight this as negative. We give the weight for taking the corner for 30 multiply the different number of AI and human, while give the weight for taking the tile next to the corner for -15 multiply the different number of AI and human. Then, the algorithm applied a final score that is the sum of 800 times the corner score and the 350 times the next score.

2.2.3 Depth Choose

In our experiment, we did several fixed depth checks. For the 8x8 standard Reversi, when the depth set to 13, the time consuming for the artificial intelligence for one state will be more than a minute, however, if we limit the depth to 11 or less, the time consumed is pretty short. Moreover, even with small number for depth like the depth of 6, our accuracy is pretty well, and the time consumed limits to less than 15 seconds. The time elapsed for the artificial intelligence to find an optimal state is quick at first, then get relatively slow when it comes to later states, then turns quicker towards the end of the game [10].

2.3 Reinforcement Learning

2.3.1 Sturcture of Reinforcement Learning in Reversi

Fig 1. The general structure of deep reinforcement learning

```
[[ 1.  1. -1. -1. -1.  1. -1.]
 [-1. -1. -1.  1. -1. -1. -1.]
 [ 1.  1.  1. -1. -1. -1.  1.]
 [ 1. -1.  1.  1.  1. -1.  1.]
 [-1. -1. -1.  1.  1.  1. -1.]
 [ 1.  1. -1.  1. -1. -1.  1.]
 [-1.  1.  1. -1.  1.  1.  1.]]
```

Fig 2. Training example by deep reinforcement learning algorithm between two AI agents playing together

Reinforcement learning is a machine learning method that intelligent agents take actions in an environment in order to maximize cumulative reward [3]. The basic structure of reinforcement learning is shown in the figure 1. The agent observes changes in the environment and makes actions based on its current state. Every time an action is made, the environment will change, the agent will get a new state, and then choose a new action to continue to execute. The basis for selecting an action in the current state is the Policy, which assigns the probability of selection to each action. When each action is completed, there is a reward to that action, and the next action will be based on the reward policy of the model [6]. Moreover, in our reinforcement learning method, two artificial intelligent

agents can be played together. That is, it will play with itself switching from black to white repeatedly during games, in our case, switching between -1 and 1 as black and white (Figure 2).

2.3.2 Basic Components

(1) Agent

The agent for this problem is the Reversi for each side, which is trained for this project and written in this paper.

(2) Environment

The environment of this problem is the general rule and the winning method of Reversi. The winning rule for the Reversi defines the reward feedback of our model. In the process of our reinforcement learning process, the algorithm needs to do a sequence of actions to maximize the probability of the winning of the artificial intelligence, which maximize the cumulative rewards.

(3) State and action space

The state space is the positions on the board which are empty, and able to construct a move. It is the same as the observation state. The action space is inside the state space that the artificial intelligence is able to move under the rule on the board.

(4) Reward

The winning rule of the Reversi determines the reward section of the algorithm. For example, taking the corner will increase the winning probability of one player. In the training method, after each training for the game, the algorithm repeatedly receives feedbacks based on the reward, and finally finishing train the artificial intelligent which can be played almost perfect Reversi.

2.3.3 Design of the Reinforcement Learning

The design of the process includes three parts, the training data producing process which two agents plays together, using neural network to activate and optimize the model and the evaluation of the model. This part combines the Monte Carlo Tree search as a simulation to the Q-learning function and neural network to a whole reinforcement learning process. We trained the model by using the self-play method to get millions of data based on the outcomes of the Reversi of each game. Those data will be transferred to the neural network and then being processed. As part of Common layers, the value head and policy head output the initial probability of the current chess evaluation value and the initial probability of actions for the next step. MCTS then executes the simulation process based on the two output values. To be more specific, a simulation represents a policy improvement operation in the Policy Iteration reinforcement learning algorithm. Several simulations result in an improved search probability. We will input the sampled action into the environment in accordance with the search probability and samples action. An example of how Policy Iteration reinforces learning is represented by this step. For training and optimization, the sample will be fed into the neural network. The position, return value and sear probability will become a set of samples to be further performed.

3. Evaluation

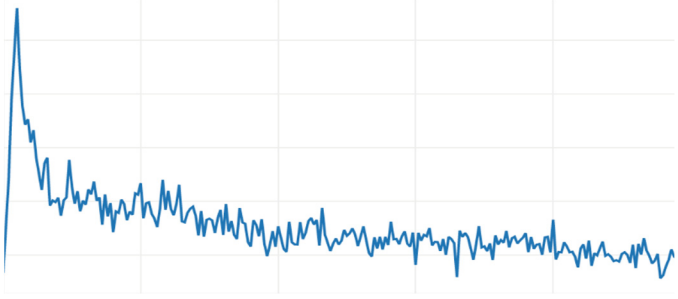

Fig 3. Loss curve with more epoch runs

The figure 3 shows the loss process in our reinforcement training process. The blue line represents the total loss, including policy loss and value loss. It follows the general loss curve of the training

process. With the number of epoch process numbered, the loss decreases in an exponential curve, then becomes relatively steady approaching the end [5]. Although it follows the general loss curve, the loss does not approach 0 in the end. However, with more epochs being ran and iterations, the actual accuracy and rate of winning will be increased since it searched more thoroughly through the state space, though it does not show on the graph.

4. Conclusion

Both the Minimax and reinforcement learning algorithm works well for the Reversi problem. The winning rate for the computer (artificial intelligence) using Minimax should be 100%, since the algorithm searches through all the state space for the next step given a state. The Heuristic Minimax with Alpha-Beta Pruning with fixed depth works well and quick for the 8x8 standard board too. When the cut off depth is around 10 layers, generally, human player could not beat the artificial intelligence. There are rare circumstances which human players can beat the artificial intelligence, but it is only due to the depth that the algorithm search is limited. For the reinforcement learning algorithm.

In general, this paper concluded several methods in which the artificial intelligent algorithms can use in game models. Both Minimax and reinforcement learning method performs well and not time consuming. The accuracy is high enough to beat general human player, thus both methods and be developed as mobile games. Moreover, with the application of the fixed depth cut-off of the minimax algorithm method, mobile app could be developed as several difficulty levels, which fits the public demands better.

For future work, the reinforcement learning needs data to be trained millions of times, which requires a huge amount of time. Proper code modification could speed up the training time. Furthermore, enable GPU utilization could also be increased. By using multiple cores as well as parallel computing, the time consuming will be significantly decreased. Available computing power for researchers has been increasing exponentially over the computing power provided by multiple computing units.

References

[1] Xin, C. Artificial intelligence application in mobile phone serious game. In 2009 First International Workshop on Education Technology and Computer Science 2009, 2:1093-1095.

[2] CIS603 S03, Lecture 7, Temple University https://cis.temple.edu/~vasilis/Courses/CIS603/Lectures /l7. Html.

[3] Yuxi Li, Deep Reinforcement Learning: An Overview, 2018 2 (6). https://arxiv.org/abs/1701.07274.

[4] Wu, Q., Spiryagin, M., Cole, C., & McSweeney, T. Parallel computing in railway research. International journal of rail transportation, 2020,8(2), 111-134.

[5] Spiring, F. A. The reflected normal loss function. Canadian journal of statistics, 1993, 21(3), 321-330.

[6] Icarte, R. T., Klassen, T. Q., Valenzano, R., & McIlraith, S. A. Reward machines: Exploiting reward function structure in reinforcement learning. Journal of Artificial Intelligence Research, 2022, 73, 173-208.

[7] Knuth, D. E., & Moore, R. W. An analysis of alpha-beta pruning. Artificial intelligence, 1975, 6(4), 293-326.

[8] Stuart R & Peter N (3rd Edition). Artificial Intelligence A Modern Approach, 2012,683-695.

[9] David P. & Alan M. Artificial Intelligence: foundations of computational agents, Cambridge University Press, 2017.

[10] Plaat, A., Schaeffer, J., Pijls, W., & de Bruin, A. Best-first and depth-first minimax search in practice. 2015, arXiv preprint arXiv:1505.01603.