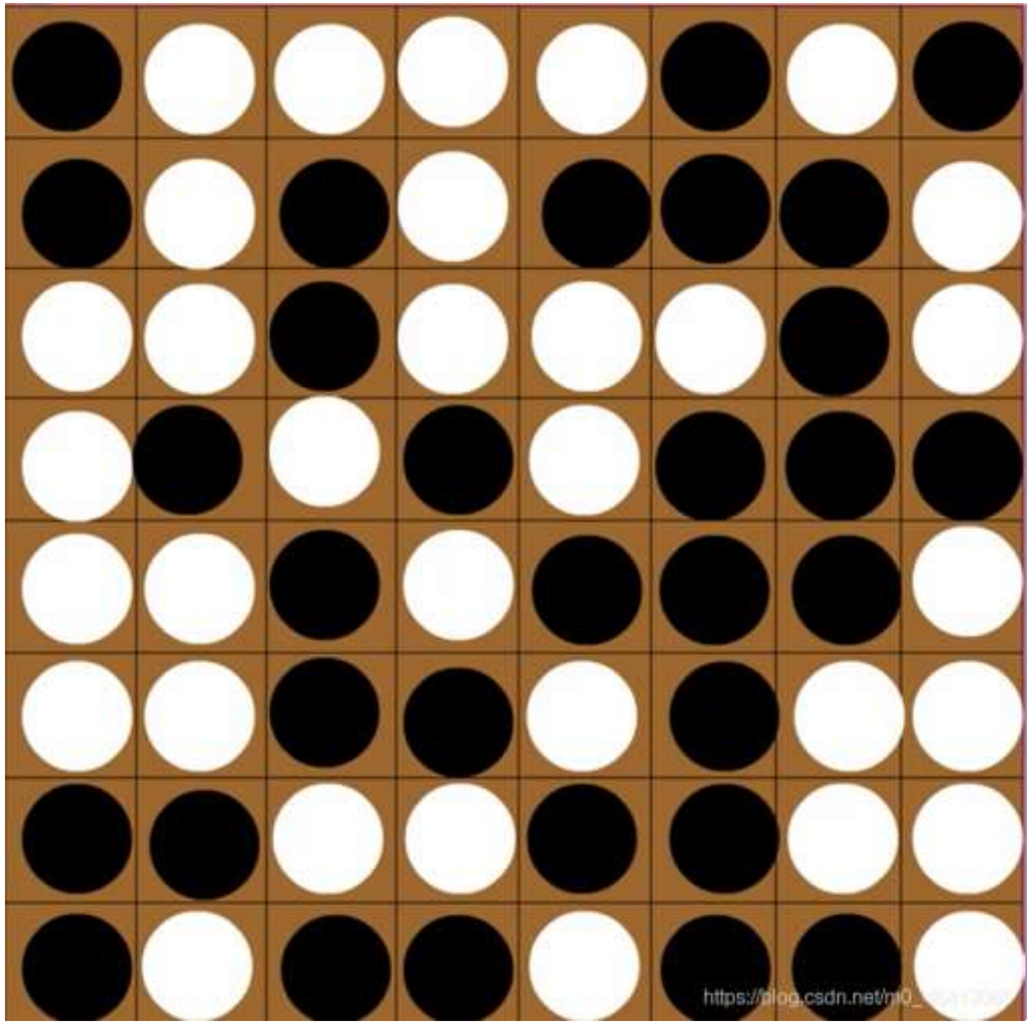
Artificial Intelligence Project

鞠屹昂 杨在洲 李盛忻 黄冠杰 刁斫

Topic: Reversi(黑白棋) Agent

Rules:

1. Start from four pieces
2. when placing a new piece, opponent's pieces in the middle will be reversed
3. piece must be placed at locations that can reverse opponent's pieces.
4. Winner is the player with most pieces



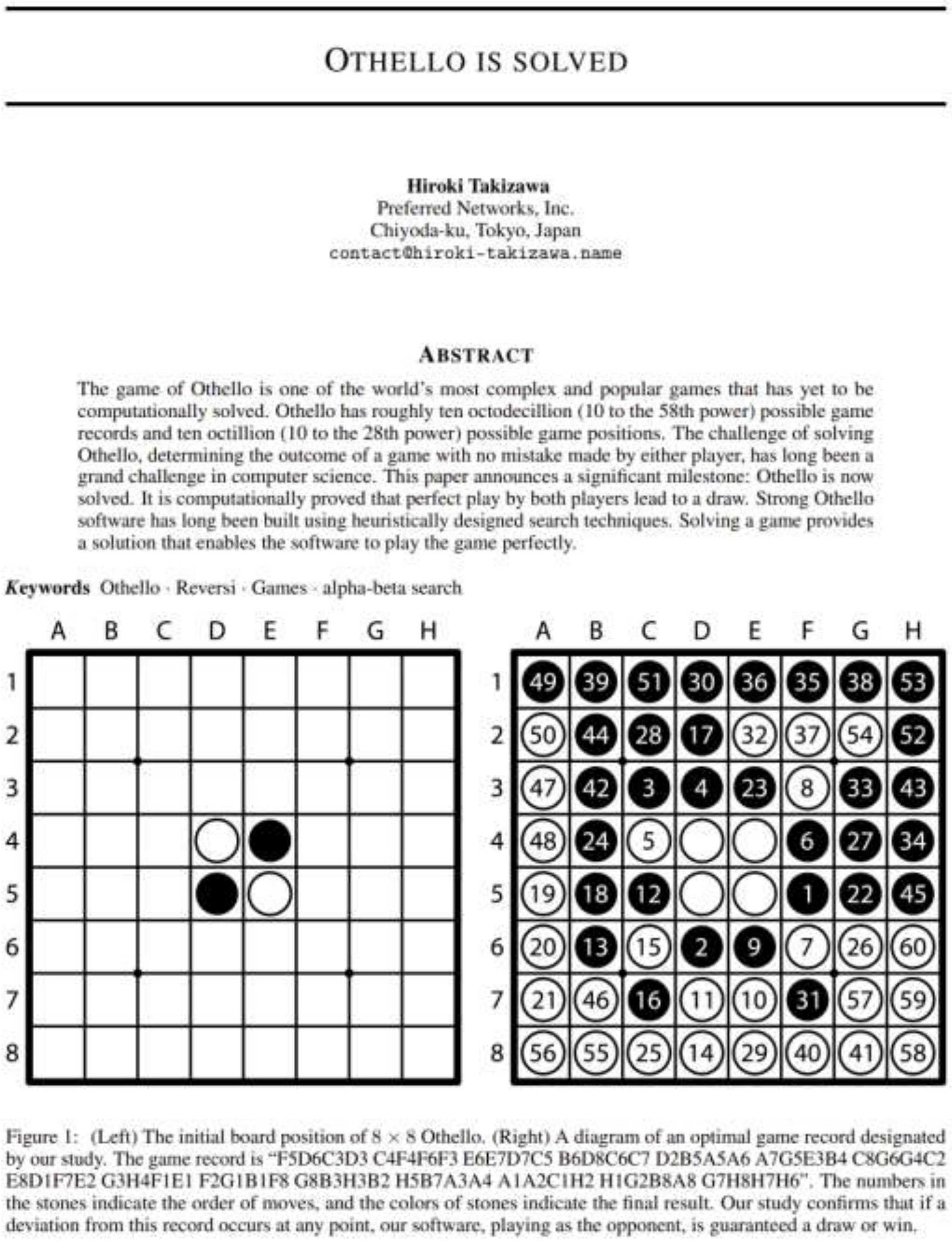
Motivation

Reason why we choose this topic:

Latest news:

Reversi(or Othello) was finally solved by Hiroki Takizawa at 2023.10 using the latest computer cluster and improved algorithm.

The rules are explicit and the state space is easy to represent



OTHELLO IS SOLVED

Hiroki Takizawa
Preferred Networks, Inc.
Chiyoda-ku, Tokyo, Japan
contact@hiroki-takizawa.name

ABSTRACT

The game of Othello is one of the world's most complex and popular games that has yet to be computationally solved. Othello has roughly ten octodecillion (10 to the 58th power) possible game records and ten octillion (10 to the 28th power) possible game positions. The challenge of solving Othello, determining the outcome of a game with no mistake made by either player, has long been a grand challenge in computer science. This paper announces a significant milestone: Othello is now solved. It is computationally proved that perfect play by both players lead to a draw. Strong Othello software has long been built using heuristically designed search techniques. Solving a game provides a solution that enables the software to play the game perfectly.

Keywords Othello · Reversi · Games · alpha-beta search

Figure 1: (Left) The initial board position of 8 × 8 Othello. (Right) A diagram of an optimal game record designated by our study. The game record is "F5D6C3D3 C4F4F6F3 E6E7D7C5 B6D8C6C7 D2B5A5A6 A7G5E3B4 C8G6G4C2 E8D1F7E2 G3H4F1E1 F2G1B1F8 G8B3H3B2 H5B7A3A4 A1A2C1H2 H1G2B8A8 G7H8H7H6". The numbers in the stones indicate the order of moves, and the colors of stones indicate the final result. Our study confirms that if a deviation from this record occurs at any point, our software, playing as the opponent, is guaranteed a draw or win.

Possible Methods

We plan to implement these methods:

1. Greedy: Simply pursue most reversed pieces at each step
2. Monte Carlo Tree Search
3. Reinforcement Learning based on Q-values

Monte Carlo Tree Search

Learn from some existing frameworks of Reversi, figure out how to implement MCTS tree search, design an interface for testing.

Roxanne Policy: Combination of some useful policies, could be helpful in guiding our implementation.

Reinforcement Learning based on Q-values

Considerations:

2^{28} possible states, we cannot store them all, so we fit the Q-value function with a network instead.

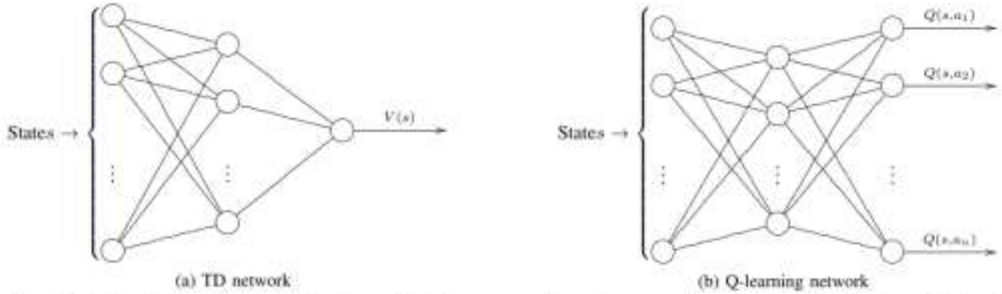


Figure 2. Topologies of function approximators. A TD-network (a) tries to approximate the value of the state presented at the input. A Q-learning network (b) tries to approximate the values of all the possible actions in the state presented at the input.

Reinforcement Learning in the Game of Othello: Learning Against a Fixed Opponent and Learning from Self-Play

Michiel van der Ree and Marco Wiering (*IEEE Member*)
Institute of Artificial Intelligence and Cognitive Engineering
Faculty of Mathematics and Natural Sciences
University of Groningen, The Netherlands

***Abstract*—This paper compares three strategies in using reinforcement learning algorithms to let an artificial agent learn to play the game of Othello. The three strategies that are compared are: Learning by self-play, learning from playing against a fixed opponent, and learning from playing against a fixed opponent while learning from the opponent's moves as well. These issues are considered for the algorithms Q-learning, Sarsa and TD-learning. These three reinforcement learning algorithms are combined with multi-layer perceptrons and trained and tested against three fixed opponents. It is found that the best strategy of learning differs per algorithm. Q-learning and Sarsa perform best when trained against the fixed opponent they are also tested against, whereas TD-learning performs best when trained through self-play. Surprisingly, Q-learning and Sarsa outperform TD-learning against the stronger fixed opponents, when all methods use their best strategy. Learning from the opponent's moves as well leads to worse results compared to learning only from the learning agent's own moves.**

I. Introduction

Many real-life decision problems are sequential in nature. People are often required to sacrifice an immediate pay-off for the benefit of a greater reward later on. Reinforcement learning (RL) is the field of research which concerns itself with enabling artificial agents to learn to make sequential decisions that maximize the overall reward [1], [2]. Because of their sequential nature, games are a popular application of reinforcement learning algorithms. The backgammon learning program TD-Gammon [3] showed the potential of reinforcement learning algorithms by achieving an expert level of play by learning from training games generated by self-play. Other RL applications to games include chess [4], checkers [5] and Go [6]. The game of Othello has also proven to be a useful testbed to examine the dynamics of machine learning methods such as evolutionary neural networks [7], n-tuple systems [8], and structured neural networks [9].

When using reinforcement learning to learn to play a game, an agent plays a large number of training games. In this research we compare different ways of learning from training games. Additionally, we look at how the level of play of the training opponent affects the final performance. These issues are investigated for three canonical reinforcement learning algorithms. TD-learning [10] and Q-learning [11] have both been applied to Othello before [9], [12]. Additionally, we compare the on-policy variant of Q-learning, Sarsa [13].

In using reinforcement learning to play Othello, we can use at least three different strategies: First, we can have a learning agent train against itself. Its evaluation function will become more and more accurate during training, and there will never be a large difference in level of play between the training agent and its opponent. A second strategy would be to train while playing against a player which is fixed, in the sense that its playing style does not change during training. The agent would learn from both its own moves and the moves its opponent makes. The skill levels of the non-learning players can vary. A third strategy consists of letting an agent train against a fixed opponent, but only have it learn from its own moves. This paper examines the differences between these three strategies. It attempts to answer the following research questions:

- How does the performance of each algorithm after learning through self-play compare to the performance after playing against a fixed opponent, whether paying attention to its opponent's moves or just its own?
- When each reinforcement learning algorithm is trained using its best strategy, which algorithm will perform best?
- How does the skill level of the fixed training opponent affect the final performance when the learning agent is tested against another opponent?

Earlier research considered similar issues for backgammon [14]. There, it was shown that learning from playing against an expert is the best strategy. However, in that paper only TD-learning and one strong fixed opponent were used. When learning from a fixed opponent's moves as well, an agent doubles the amount of training data it receives. However, it tries to learn a policy while half of the input it perceives was obtained by following a different policy. The problem may be that the learning agent cannot try out its own preferred moves to learn from, when the fixed opponent selects them. This research will show whether this doubling of training data is able to compensate for the inconsistency of policies. It is not our goal to develop the best Othello playing computer program, but we are interested in these research questions that also occur in other applications of RL.

In our experimental setup, three benchmark players will be used in both the train runs and the test runs. The results will therefore also show possible differences between the effect this

Evaluation

After implementing these methods,

1. We take the implemented methods to fight Game AI and calculate their win rate
2. We let the three agents fight with each other in order to find an optimal one