Wydział Informatyki Elektroniki i Telekomunikacji
Stochastyczne Algorytmy Obliczeniowe

Searching for optimal strategies for Reversi (aka. Othello)

Michał Bizoń
Alina Borysenko
Maciej Imiołek

1. State of the art

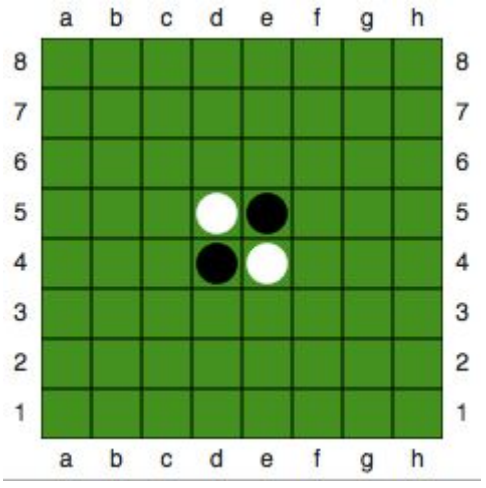
Computer Reversi refers to computer software with capability to play Reversi (Othello). There are many different programs, which are capable of playing the game using different strategies. The most important issue is to find best move, or set of moves which lead to winning the whole game.
So far, programs play the game in one of the following way. First option is „learn" program what moves are better than others. Program plays thousands of games and save courses of the games. With this history of moves, they can then search for what moves cause more wins than others. Second option is to play with some predefined strategy algorithm which can decide which move in is the best at the given time.
Best programs to play Reversi combine this two options. In our project, we focused on providing a platform capable of finding the best algorithm to play Reversi without saving a lot data from previous games. We want to choose best way to play the game, if any exists.

2. Problem definition

2.1. Reversi rules

Reversi is a two-player strategy board game played with pieces (or disks), which are light on one side and dark on the other. Each player has one color assigned to him. The goal of the game is to have the majority of disks turned to display your color when the game ends. Reversi rules are as follows:

- game is played on a 8x8 board
- game starts with two light and two dark pieces on board, positioned in the center, see image below:



Reversi game board

- dark player is the first to move
- during player's turn, the player places a piece on the board, his colour up. A piece must be placed on a square adjacent to opposing player's piece in such a way, that at least one opposing piece is captured
- capture occurs when there is a straight line (horizontal, vertical or diagonal) between a piece that is being placed on the board and any other piece owned by the acting player, with at least one opposing player's piece in between. The pieces in between are turned over
- if a player to act has no available moves to make he passes
- game ends when both players have no available moves to make, whoever has the majority of pieces in his colour wins

2.2. Problem

The aim of this project is to automate the process of finding the optimal strategies for Reversi game. A tournament platform for Reversi strategies will be created. Strategies will be matched against each other in round-robin tournaments and winners will then create successor strategies that will advance to next iterations.

2.3. Strategies

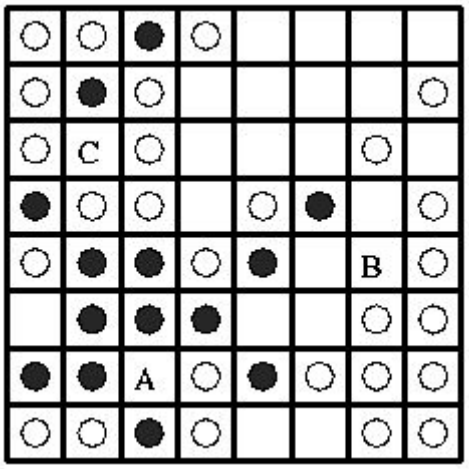
2.3.1. Non-parametrized

- **Random moves** - self-explanatory, not considered in experiments
- **Greedy algorithm** - makes moves that yield the most points at a given time
- **Corners** - aims to capture closest board corners

2.3.2. Parametrized

1. **Stable discs** - aims to capture squares that can't be recaptured
   Parameters:
   - *stable-disk-weight* - capture stable disk or capture something that yields more points

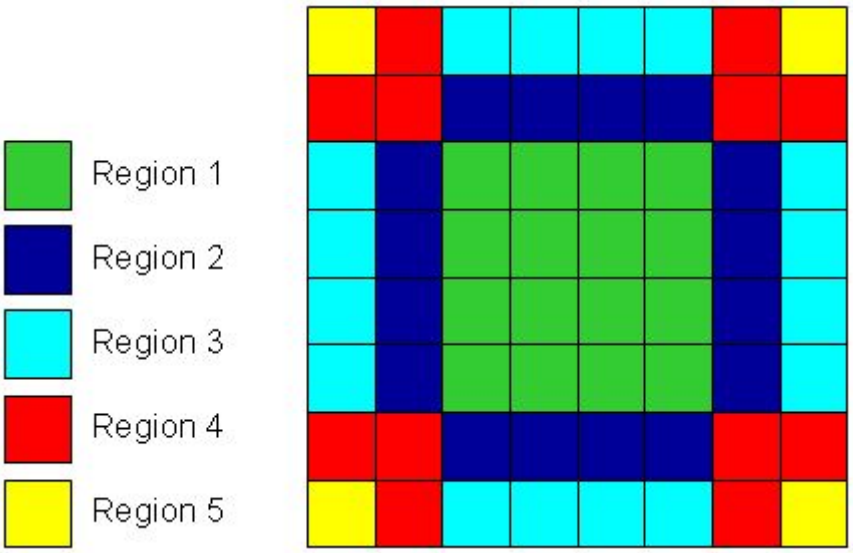
   Value range - [0, 1] - 1 always capture stable, 0 always capture for points
   Value step - 0.1



A, B, C - are stable fields for white disc.

2. **Positional** - based on risk map, aims to capture low-risk squares and avoid high-risk squares
   Parameters:
   - *risk-map* - array of 5 integers - risk value for each region

   Value set for each region - {0, 1, .. , 99 }



Risk map for positional strategy

3. **Mobility** - aims to make such moves that maximize number of your moves and minimize number of opponent's moves
   Parameters:
   - *maximize* - decide whether to maximize self moves or minimize opponent's moves

   Value range - [0, 1] - 1 always maximize self moves, 0 always minimize opponent's moves

   Value step - 0.1

4. **Mixed strategy** - used one strategy at first, changes to another strategy when condition is met
   Parameters:
   - *strategy1* - strategy to use at the beginning of the game
   - *changeCondition* - number of moves to make before changing strategy

   Value set - { 0, 1, 2, .. , 19 } - 0 indicates that only strategy 1 should be used throughout the game
   - *strategy2* - strategy to use after changeCondition is met

2.4 Proposed solution

2.4.2 Model

Each strategy will be represented as an specimen containing the following properties:

- *gamesPlayed* [int] - number of games played by the stratgy specimen
- *gamesWon* [int] - numbers of games won
- *earlyGameStrategy* [strategy] - strategy that should be used from the start of the game
- *strategySwitchPoint* [int] - number indicating after how many moves should the specimen change strategy to lateGameStrategy - if the value is 0, only earlyGameStrategy should be used throughout the game
- *lateGameStrategy* [strategy] - strategy that should be used when {strategySwitchPoints} moves are made

2.4.3 Initial population

- 5 strategies - 2 non-parametrized, 3 parametrized
- each parametrized - 8 different parameter sets
- 26 singular strategies + 34 randomly mixed multiple strategies
- 60 strategies in total

2.4.4 Mutation

If at any point mutation of specimen's strategies is to be performed, a decision tree determines the details of the mutation. The decision tree requires the following parameters:

- *disturbEarlyGameStrategyChance*
- *replaceEarlyGameStrategyChance*
- *disturbStrategySwitchPointChance*
- *useOnlyEarlyGameStrategyChance*
- *disturbLateGameStrategyChance*
- *replaceLateGameStrategyChance*

The decision tree has the following structure:

```
DisturbSpecimensStrategies(disturbEarlyGameStrategyChance,
        replaceEarlyGameStrategyChance, disturbStrategySwitchPointChance,
        useOnlyEarlyGameStrategyChance, disturbLateGameStrategyChance,
        replaceLateGameStrategyChance)
{
   # check to see if distirbance should be performed
   if random(0,1) > win%
      # check to see whether to disturb earlyGameStrategy
      if random(0,1) < disturbEarlyGameStrategyChance
         # check to see whether to disturb parameters or change strategy
         if random(0.1) < replaceEarlyGameStrategyChance
            ChangeStrategy(earlyGameStrategy)
         else
            DisturbParameters(earlyGameStrategy)

      # check to see whether to disturb strategySwitchPoint
      if random(0,1) < disturbStrategySwitchPointChance
         if random(0,1) < useOnlyEarlyGameStrategyChance
            strategySwitchPoint = 0
         else
            strategySwitchPoint = random(1,20)
      # check to see whether to disturb lateGameStrategy
      if random(0,1) < disturbLateGameStrategyChance
         # check to see whether to disturb parameters or change strategy
         if random(0.1) < replaceLateGameStrategyChance
            ChangeStrategy(lateGameStrategy)
         else
            DisturbParameters(lateGameStrategy)
}
```

3. Software

3.1 Structure

Software code is entirely written in Java and is divided into 3 packages - engine, algorithm, and tournament. Engine package contains tools for playing Reversi games between two players. Abstraction for player algorithms is also in engine package. Algorithm package contains concrete implementations of player algorithms.

Tournament package contains platform for performing tournaments to determine the best of the strategies implemented in algorithm package.

3.2 Game engine

The game engine can use various algorithms to play. The game state is computed using board evaluation.

3.3 Algorithms

Algorithms package holds strategies, which decide how to play (choos moves to play). Also there is one algorithm which can play two different strategies. First one up to some predetermined number of moves, and later the second one.
Algorithms also can mutate some strategies, change their parameters or even combine two different ones.

3.4 Tournament engine

The tournament engine package holds implementation of evaluation of the best Reversi strategy. It uses round-robin tournaments that make strategies compete against each other and select the best players of tournament.

4. Testing plan

4.1 Tournament selection

4.1.1 Child and parents succession

In each iteration, do 20 times:

- randomly chose two 10-strategies sets
- two round-robin rounds in each set
- best-of-two match between each strategy - starting strategies alternate
- two winning strategies populate and together with their child advance to next iteration

4.1.2 Only child succession

In each iteration, do 60 times:

- randomly chose two 10-strategies sets
- two round-robin rounds in each set
- best-of-two match between each strategy - starting strategies alternate
- two winning strategies populate, but only their child advances to next iteration
-

4.2 Mutation

Before winning strategies populate:

- decide whether to mutate (disturb) specimens based on its winning percentage
- if mutation is to occur, specimens' stats (games played, games won) are rest

4.3 Determining winners

After a set number of iterations a list of all surviving specimen is sorted by win percentage in a descending manner. Should there be any specimen within (subject to change) 5% win rate margin, a second tournament in performed.

5. Results

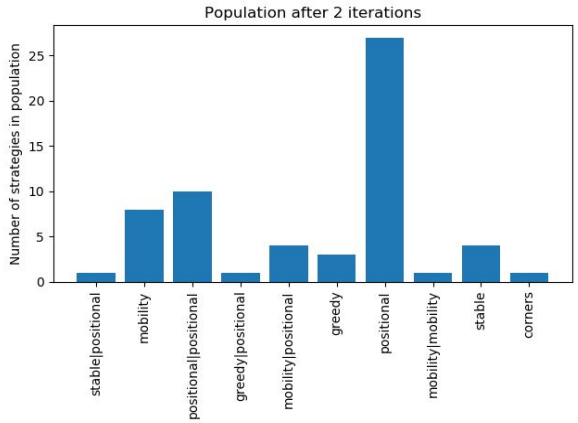
5.1 Experiment 1

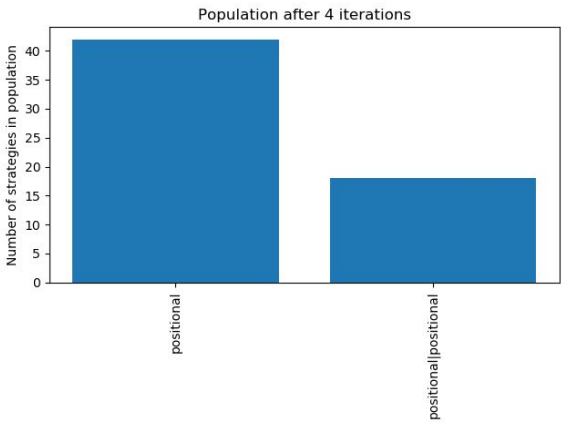
Experiment details:

- population size: 60
- iterations: 15
- tournament size: 8
- extra: mutation disabled

5.1.1 Only child succession

- total number of of matches: 50400

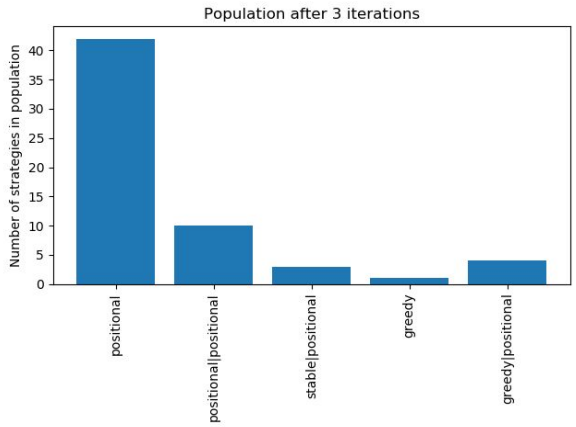


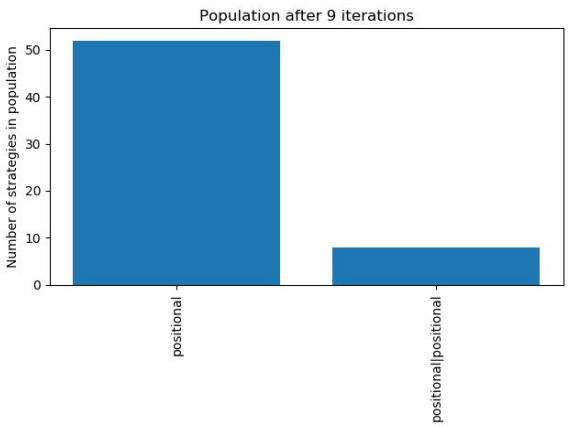


As early as in 4th iteration the whole population consists of only positional strategy players.

5.1.2 Parents and child succession

- total number of of matches: 16800





Positional strategy players still dominate. This time the population consists of 100% positional strategy players as late as in 9th iteration.

5.1.3 Experiment 1 conclusion

Positional strategy players dominate tournaments. Due to lack of mutation all winning positional strategy players advance to next iterations unchanged, which results in population containing only positional strategy players quickly. The top positional strategy players have 65-70% win rate.

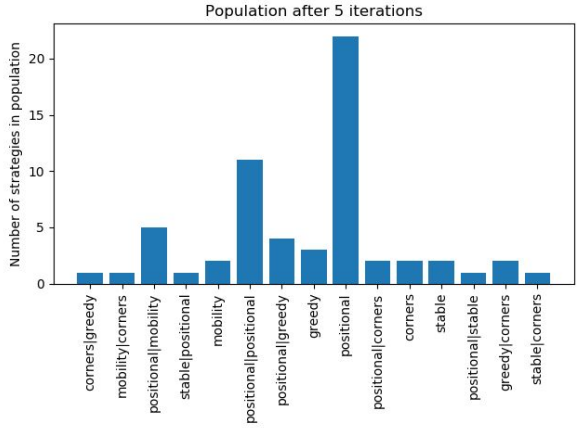
5.2 Experiment 2

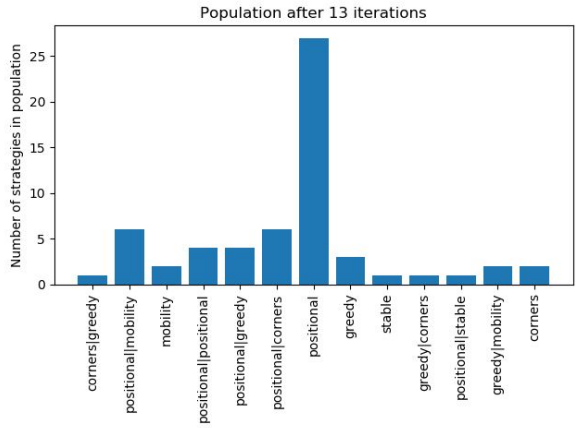
Experiment details:

- population size: 60
- iterations: 50
- tournament size: 10

5.2.1 Only child succession

- total number of of matches: 270000

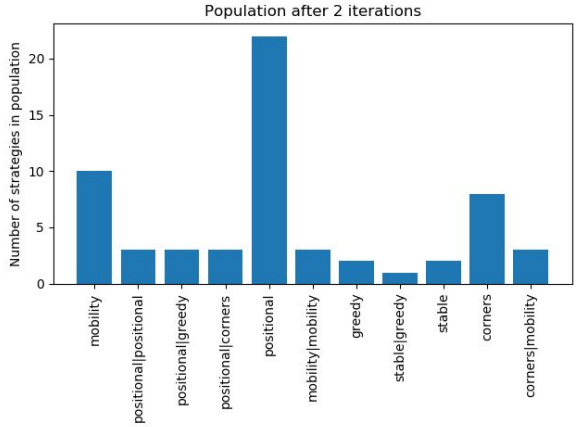


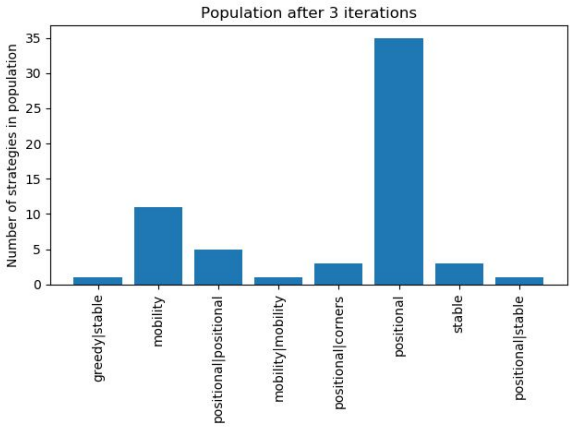


Positional strategy players still dominate all match-ups. Positional strategy players occupy the top of the leaderboard as early as fifth iteration, but don't take up over 50% of the population until 12th iteration.

5.2.2 Parents and child succession

- total number of of matches: 90000





Positional strategy players are at the top of the population already after 2nd iteration. This time, because of the fact that parents as well as a child advance to next iteration, positional strategy players make for over 60% of the population after 3rd iteration.

5.2.3 Experiment 2 conclusion

Introduction of mutation allowed for population to be more diversified during most of the experiment in case of only child succession. Child and parents succession introduced a kind of elitism, where most of the players advancing to next iterations are positional strategy players. Mutation ensures they never make for 100% of the population. The top positional strategy players still have 65-70% win rate.

5.3 Experiment 3

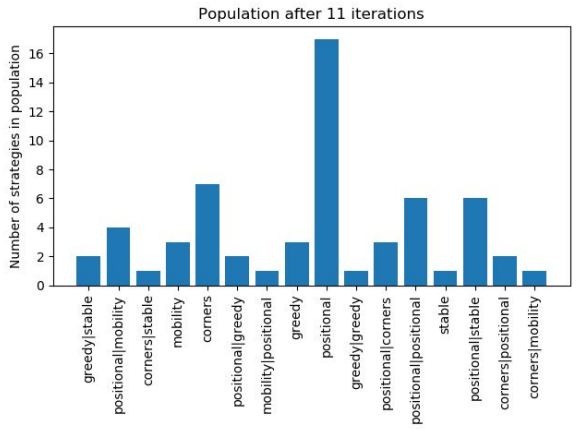
Experiment details:

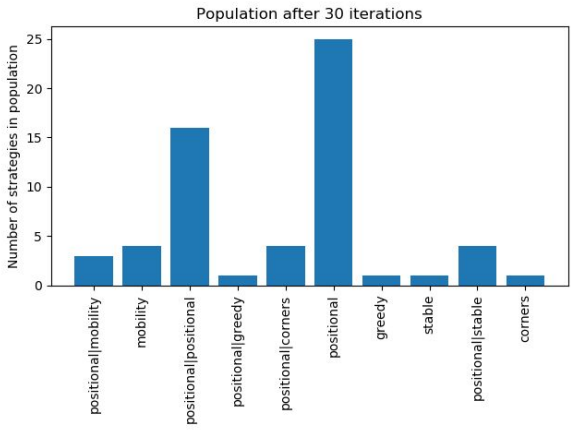
- population size: 60
- iterations: 30
- tournament size: 5

We observed that positional strategy players tend to dominate all other strategies and quickly make for most of the population. We tried to tackle those problems by reducing the tournament size. Performing two 10-player tournaments at once means that 33% of all population (20 out of 60) take part in those tournaments, therefore the chance of there being multiple positional strategy players chosen is high.

5.3.1 Only child succession

- total number of of matches: 36000

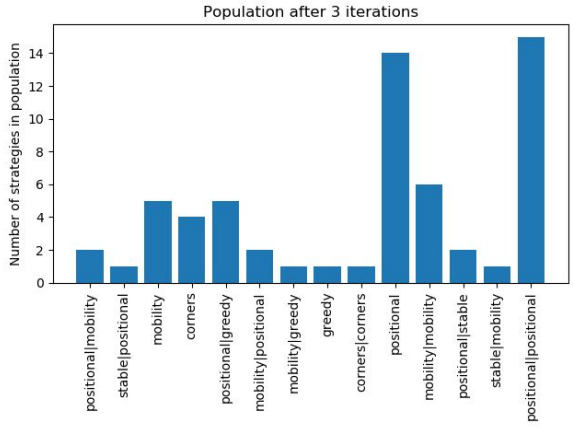


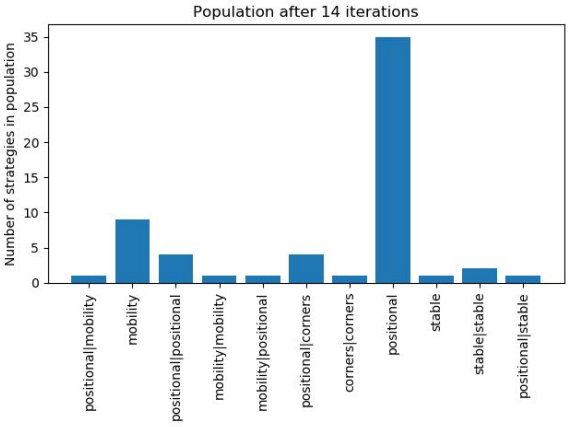


The rate at which positional strategy players populate the strategy set was reduced. It now takes 11 iterations for them to make for 25% of the population and 30 iterations to make for 60% of the population.

5.3.2 Parents and child succession

- total number of of matches: 12000





Still the same elitism problem with parents and child succession exist. Rate at which positional strategy players populate most of the strategy set is a lot higher than when only child specimen advantec to next iteration.

5.3.3 Experiment 3 conclusion

Tuning down the size of tournaments helped strategies other than positional become more competitive in terms of how many of them exist in the population. In terms of win rate there was a slight increase in positional strategies win rate - the top positional strategy players have won 70-75% of their matches, and all other strategies don’t stand a chance.

5.4 Experiment 4

Experiment details:

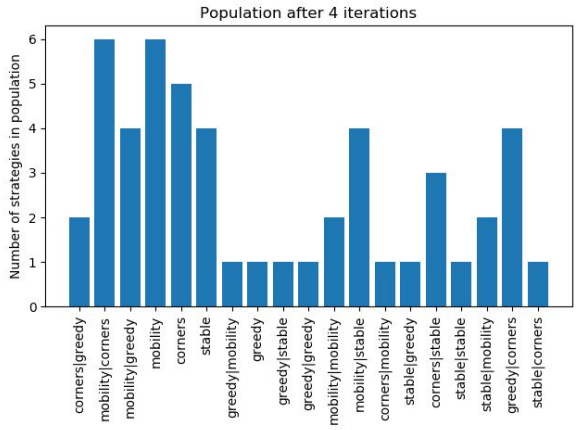
- population size: 50

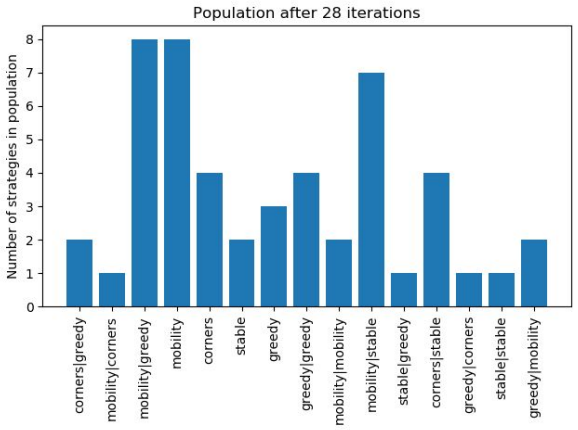
- iterations: 30
- tournament size: 5
- extra: positional strategy excluded

Positional strategy players dominated all previous experiments. This experiment was performed to find out how other strategies cope against each other.

5.4.1 Only child succession

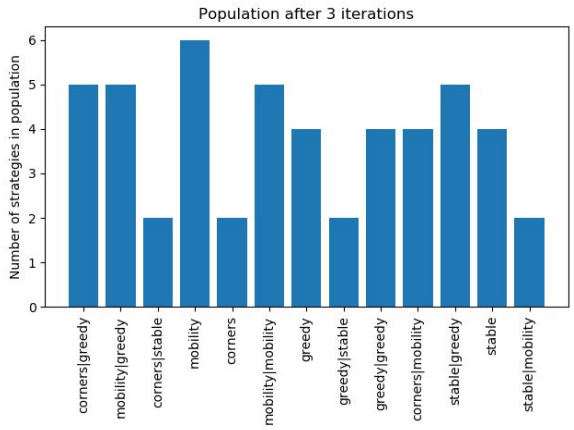
- total number of of matches: 30000

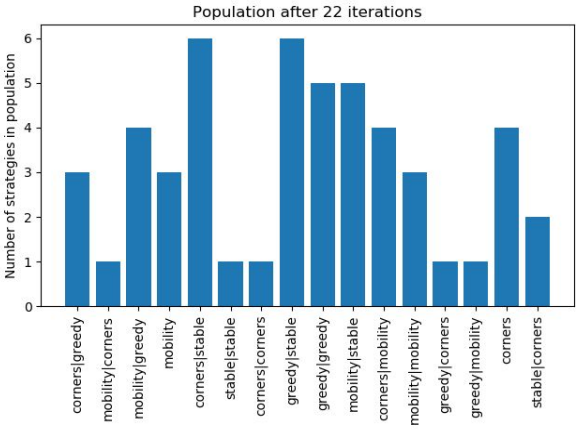




5.4.2 Parents and child succession

- total number of of matches: 10000





5.4.3 Experiment 4 conclusion

Strategies are much more evenly represented throughout the population. It is only at the end of the experiment run where players using mobility strategy are being more commonly represented in the population. While the population isn't dominated by one particular strategy in regard to rarity, the top 3 strategies in terms of win rate were always either solo or combinations of mobility strategy players win low parameter value. This means that mobility strategy performs better when minimizing the amount of moves the opponent can make.

6. Conclusion

This project shows that strategies to play Reversi are not all equal. Out of the set of strategies we implemented, there was one that stood out and dominated all others in terms of win rates, and that was the positional strategy. The top positional strategies averaged at about 65-70% win rate which is not a lot, but we must take into account that positional strategy player usually made for over 50% of the population, so a lot of matches were between players using positional strategy.
The first two experiments have shown us that the algorithm we implemented for finding the best strategy can be quite elitist, resulting in population quickly becoming full of positional strategy players. We tackled that in the third experiment by adjusting tournament size which resulted in strategies being more evenly represented in the population for the most of the experiment and only becoming prevalent of positional strategy players just a few iterations before the end.

7. Sources/Bibliography

- Application to play Reversi. Genetic algorithms to moves. Each player makes random moves. Genetic algorithm to search best moves in own move's history. Different depth search. (Reversi on 10x10 board)
  https://www.lri.fr/~hansen/proceedings/2011/GECCO/companion/p739.pdf
- Article about teaching computer Othello strategies without expert knowledge. Using two ways: neural network and min-max search algorithm. Individual neural network discovered various game-playing strategies, e.g. positional, and later mobility (after 1000 generations).
  http://www.xiaotu.com/pub/ChonS05a.pdf
- Playing Reversi also with Genetic Algorithms. Main purpose is to show that player learns better from large "strategies" set is better than from small one.
  http://gamelearninglab.nctu.edu.tw/ctsun/GA%20learning%20in%20game%20playing.pdf
- Develop new strategies to play Othello based on artificial evolution of neural networks. New strategies play against random-moves and αβ-search. Neural network quickly learn positional and mobility strategies.
  http://nn.cs.utexas.edu/downloads/papers/moriarty.discovering.pdf
- Note briefly describing LOGISTELLO, one of today's strongest Othello programs.
  http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.114.1746&rep=rep1&type=pdf

- This article has shown how Othello programs evolved from classic hand-tuned to sophisticated learning systems which have surpassed human playing strength.
  https://skatgame.net/mburo/ps/compoth.pdf