Jafar — Intelligent Othello Agent

Joshua Nelson, Tim Cosgrove, Andrew Haigh

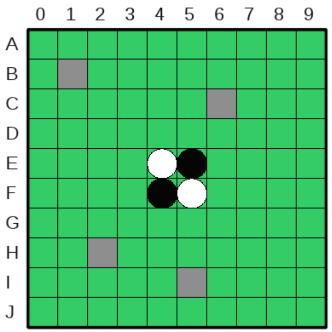
COMP3130 Research Project

May 29, 2012


YEAH, I NAMED THE BIRD IAGO...
LIKE FROM "OTHELLO"...

Problem Overview

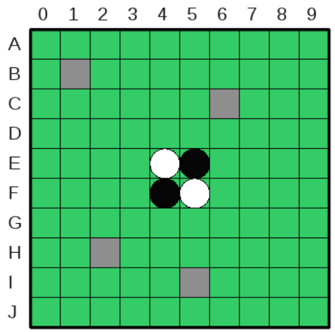
- The Game — 10x10 Modified Othello



(a) Initial Configuration

Problem Overview

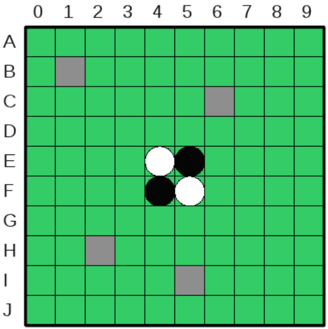
- The Game — 10x10 Modified Othello
- The Problem — Intelligent AI player



(a) Initial Configuration

Problem Overview

- The Game — 10x10 Modified Othello
- The Problem — Intelligent AI player
- Solution basis



(a) Initial Configuration

Solution Structure

- The MetaPlayer class — Utilises knowledge of the game state and creates instances of other players accordingly

Solution Structure

- The MetaPlayer class — Utilises knowledge of the game state and creates instances of other players accordingly
- NegamaxPlayer (varying depth argument)
    - The 'main' AI player; performs a pruned Negamax search (simplification of Minimax)
    - Searches to a depth specified by the MetaPlayer; where it then evaluates the quality of the leaf states

Solution Structure

- The MetaPlayer class — Utilises knowledge of the game state and creates instances of other players accordingly
- NegamaxPlayer (varying depth argument)
  - The 'main' AI player; performs a pruned Negamax search (simplification of Minimax)
  - Searches to a depth specified by the MetaPlayer; where it then evaluates the quality of the leaf states
- Other Player interfaces:
  - OpeningPlayer
  - GreedyPlayer
  - HumanPlayer

Static Evaluation

- The FeatureSet class — Maintains a list of *features*; functions which evaluate a game state based on some criteria of strength

Static Evaluation

- The FeatureSet class — Maintains a list of *features*; functions which evaluate a game state based on some criteria of strength
- LegalMoves Feature
  - Also known as 'mobility'; the number of moves available to the agent
  - Having only a few moves available is very limiting
  - Useful through all stages of the game

More Features

- BlockedAdjacent, CornerPieces, SidePieces

More Features

- BlockedAdjacent, CornerPieces, SidePieces
  - Count of how many pieces we have next to blocked squares, corners and edges
  - Useful as these pieces are harder to capture; easier to 'lock'

More Features

- BlockedAdjacent, CornerPieces, SidePieces
  - Count of how many pieces we have next to blocked squares, corners and edges
  - Useful as these pieces are harder to capture; easier to 'lock'
- Visibility
- StoneCount

TD-λ

- Remaining challenge: decide how important each feature is (the feature 'weights')

TD-λ

- Remaining challenge: decide how important each feature is (the feature 'weights')
- Solution - use the TD λ (Temporal Difference) formula to calculate the new weights
- $w := w + \alpha \sum_{T=1}^{N-1} \tilde{J}(x_t, w) * (\sum_{j=t}^{N-1} \lambda^{j-t} d_t)$

The J Function

$$w := w + \alpha \sum_{T=1}^{N-1} \Delta \tilde{J}(x_t, w) * (\textstyle\sum_{j=t}^{N-1} \lambda^{j-t} d_t)$$

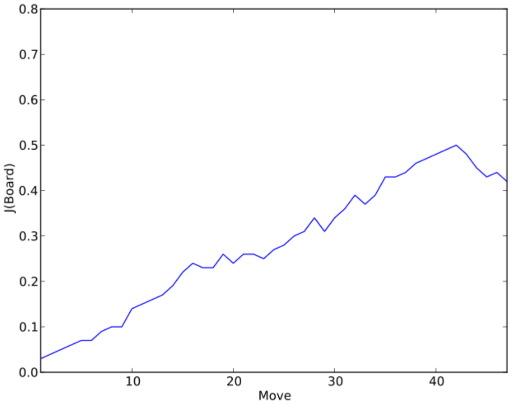
- The J function returns a probability of winning, given a set of weights and a board.
- The perfect J function would always return 0 or 1 precisely.
- We are trying to learn a good approximation to the J function

TD: Temporal difference

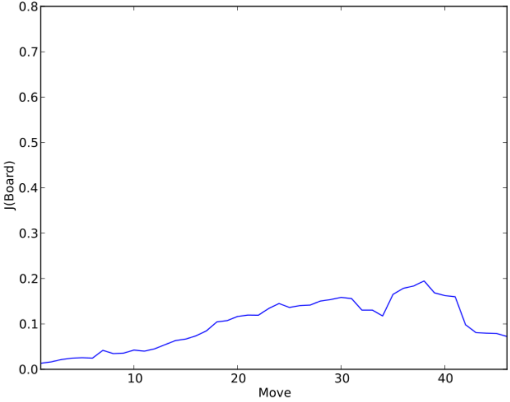
$$w := w + \alpha \sum_{T=1}^{N-1} \Delta \tilde{J}(x_t, w) * \left(\sum_{j=t}^{N-1} \lambda^{j-t} d_t\right)$$

- d_t is the *Temporal Difference* between successive game states.
- The key observation is that for an ideal J function this would always be zero.
- $\Delta \tilde{J}(x_t, w)$ corrects each weight according to whether it was pointing us in the right direction.

Initial J function
Lost game

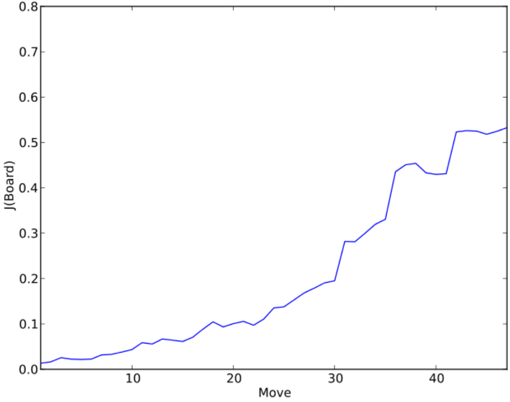


J function after 6000 iterations
Lost game

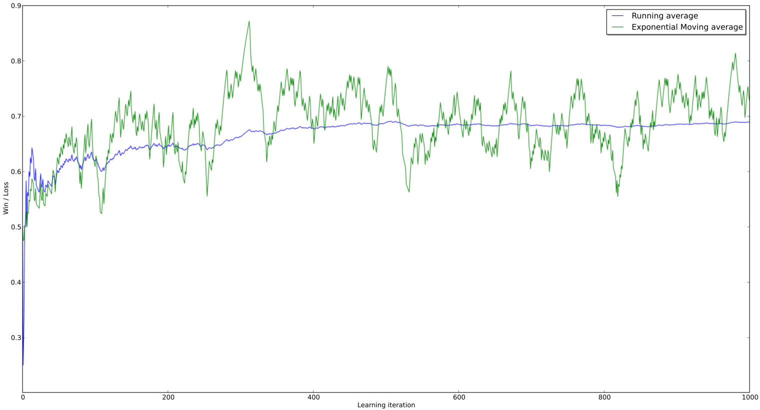


J function after 6000 iterations

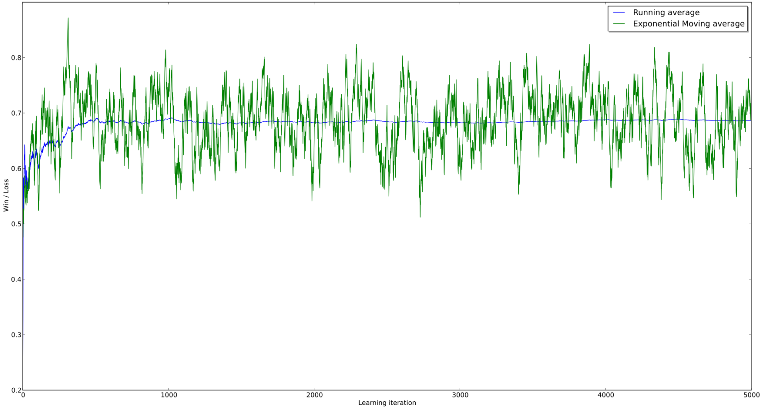
Won game



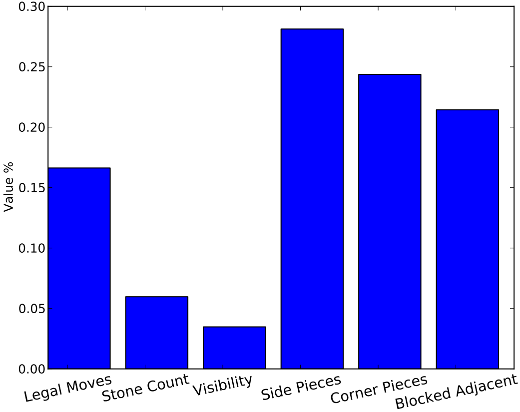
The first 1000 learning iterations



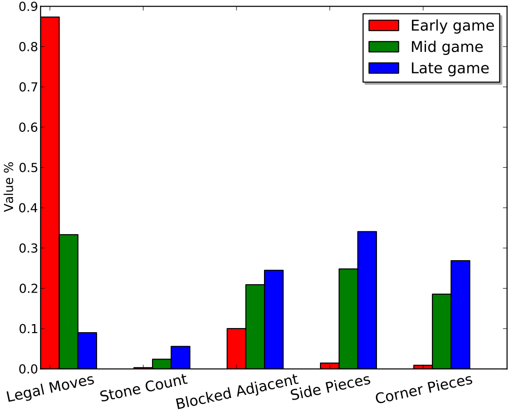
The first 5000 learning iterations



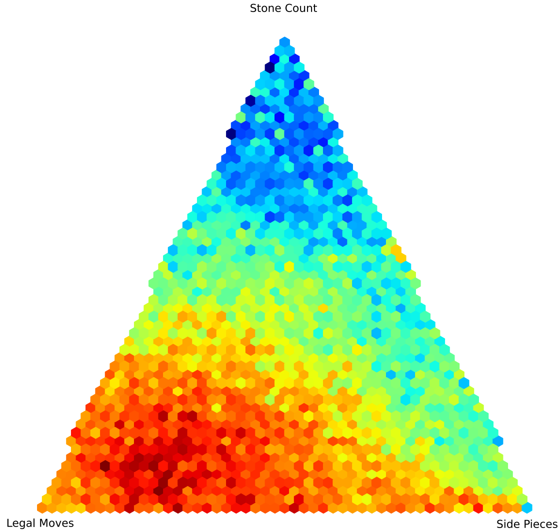
Self play - learnt weights



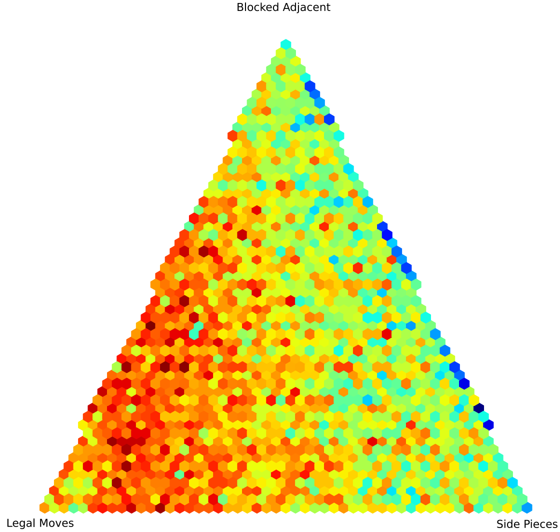
Self play, stage learning weights
4 ply for 1300 games



Feature weight space visualisation



Feature weight space visualisation



The Elo Rankings Arena

- Custom made genetic algorithm
- Pits a group of randomly generated agents against each other for Elo style ranking points
- Creates new agents from those who perform best

The Elo Rankings Arena

- Custom made genetic algorithm
- Pits a group of randomly generated agents against each other for Elo style ranking points
- Creates new agents from those who perform best
- Used more as a demonstration of the learning process
- Interesting to see it come to the same conclusions as TD-λ

Future Improvements

- More attention to blocked squares

Future Improvements

- More attention to blocked squares
- Negamax optimisations (better transposition tables, prob-cuts)

Future Improvements

- More attention to blocked squares
- Negamax optimisations (better transposition tables, prob-cuts)
- Better use of pre-computed data (such as the Opening Book)

Future Improvements

- More attention to blocked squares
- Negamax optimisations (better transposition tables, prob-cuts)
- Better use of pre-computed data (such as the Opening Book)
- More board features (locked squares, open squares, etc.)

Future Improvements

- More attention to blocked squares
- Negamax optimisations (better transposition tables, prob-cuts)
- Better use of pre-computed data (such as the Opening Book)
- More board features (locked squares, open squares, etc.)
- Generic features (features learnt by the agent)

Future Improvements

- More attention to blocked squares
- Negamax optimisations (better transposition tables, prob-cuts)
- Better use of pre-computed data (such as the Opening Book)
- More board features (locked squares, open squares, etc.)
- Generic features (features learnt by the agent)
- More exploration in learning - randomise the initial board and play from that to explore more options

Design Methodology

- C with a python interface
  - Used this for tic-tac-toe warm up problem, decided against due to development overhead

Design Methodology

- C with a python interface
  - Used this for tic-tac-toe warm up problem, decided against due to development overhead
- Java
  - Sacrifice low level speed improvements for high level language features and built in data structures.

Design Methodology