COMP3130 Othello Project

Team Reverwesome

Josh Godsiff Jarrah Bloomfield

May 29th, 2012

Problem Outline

- Modified Othello Game
- 4 randomly removed squares
- 10x10 board rather than the traditional 8x8
- Network communication with server
- 150 seconds total time per player
- Design an agent to play intelligently

Solution Outline

- Static evaluation function based on feature weights
- Temporal Difference Learning
- Negamax search with alpha beta pruning
- Concurrent searching
- Time management
- C++ and Ada interfacing

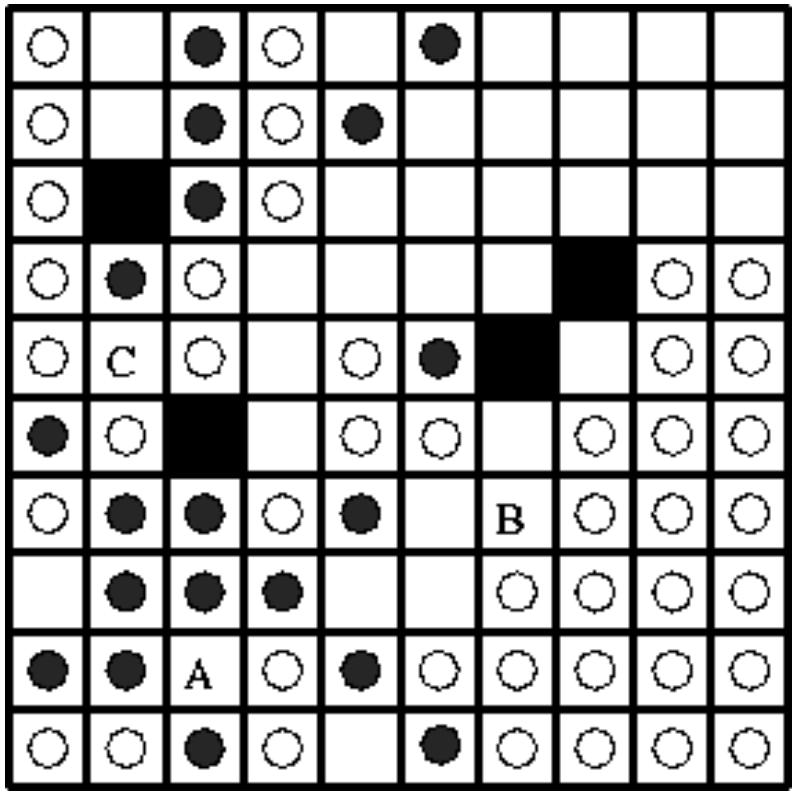
Static Evaluation

- Use features weights to evaluate value of board states
- 15 weights for each piece position
  - based on (naive) 8-way symmetry
  - in actuality, only a 4-way symmetry exists. However, in practice this doesn't make a lot of difference.
- Weight for mobility - number of moves if it was our turn
- Weight for stability - number of stable pieces we own
- Weight for internals - number of internal pieces we own

Piece Stability

- Stability is the inability for a piece to ever be flipped. For example, corners are always stable.
- All 4 lines must be stable in one direction
- Line stability from the whole line being 'full'
- Line stability from the whole direction being stable and full in your favour
- Because of the removed squares, piece stability became especially important ahead

Piece Stability



Stable positions for white

Mobility

- Mobility is a measure of the number of moves a player has in a given board state.
- High mobility tends to be much more important in the early to mid game than the number of pieces a player has, since Reversi boards can change very quickly.
- Much less useful in the later game, where we want to focus on pieces and stability.
- Has the unfortunate side-effect of making the branching factor quite large, meaning we cannot search to great depth.

Piece Internality

- A piece is internal if it has no empty neighbours
- Harder to dislodge, can result in higher mobility

Temporal Difference Learning

- Used a fairly standard TD(λ) policy.
- Play whole game, recieve reward $r \in \{1, 0, -1\}$ for win/draw/loss.
- For each board b_k position in the game, find the direction to adjust the weight in, based on making a small change and seeing if it improves/worsens the result.
- Then run through all of it is successors, and calculate the weighted difference between it's value and the successor's.
- $\sum_{t=k}^{T} \lambda^t \left(V(b_t) - V(b_k) + \mathit{feedback} \right)$

Learning Strategies

- $\alpha = 0.0001$
  Learning weight. Tend to get divergence if we set it any higher.
- $\epsilon = 0.15$
  Used for ϵ-greedy policy. The chance we'll pick a random move instead of a good one.
- $\gamma = 0.9$
  Discount factor on how much we weight boards far into the 'future' when training.
  Scales as γ^t, where t is the number of boards in the futre we're looking.

Negamax with alpha beta pruning

- Adversarial search algorithm
- Minmax algorithm modified to negate the value at each step
- Attempts to maximise reward against adversary who is actively minimising reward
- α - β pruning cuts sections of the game tree that an opponent would never choose

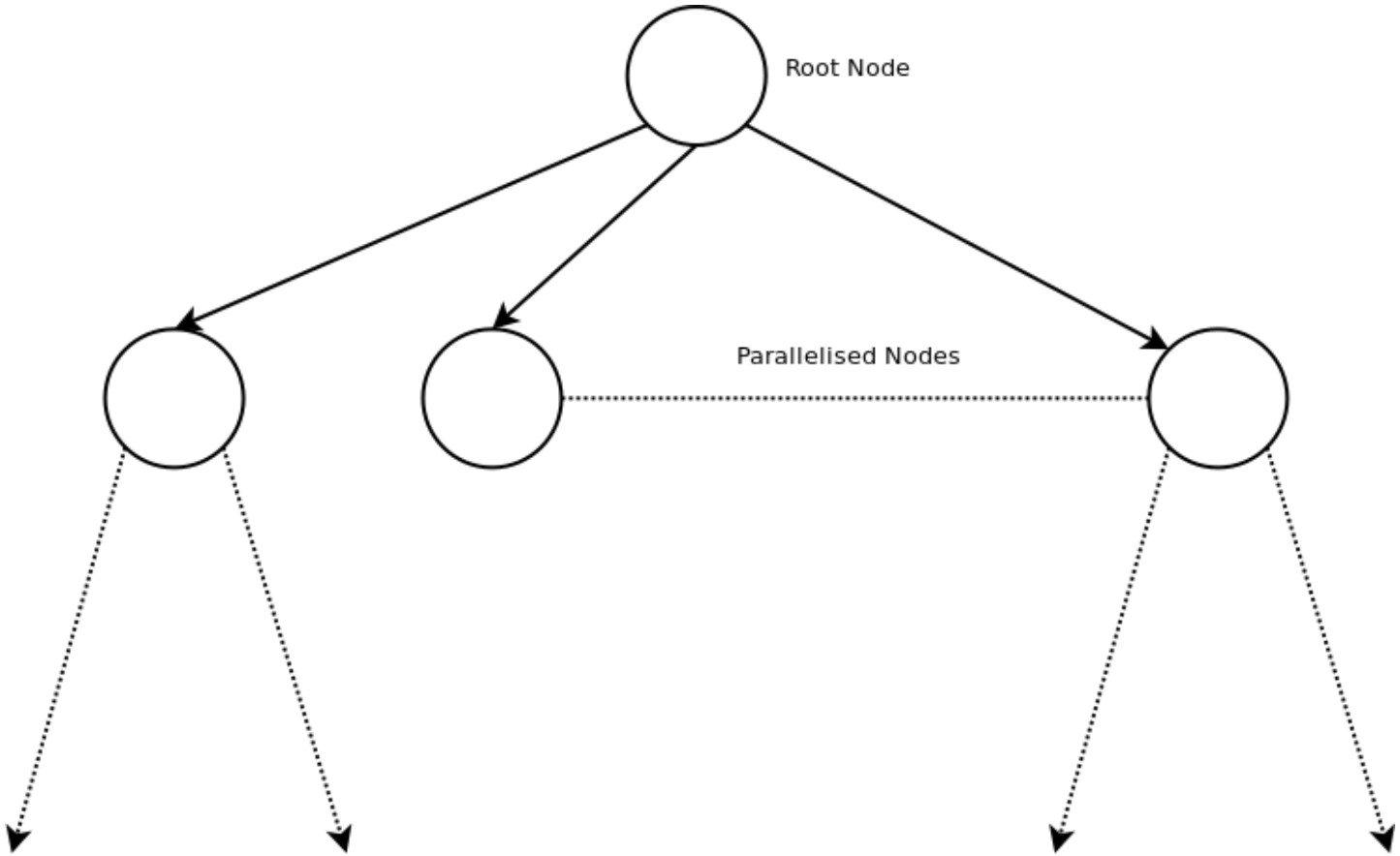
Negamax features

- Search to a moderate depth if close to the start of the game
- Search to a deep depth if close to the end of the game
- Add depth if exploring along a forced move
- Use static evaluation feature weights to evaluate how far in our favour the board value is
- Terminals in our favour worth ∞, in opponent's favour worth $-\infty$.
- This means we avoid a loss at all costs, and take a win at all costs (for example wiping out a player early)

Concurrency

- Use concurrency to distribute MinMax search across multiple CPUs, thereby (hopefully) reducing the time taken for a search.
- Have to find a model of parallelism which balances putting processing time where it's needed vs overhead in communication between threads.
- We settled for a simplistic model - give each thread one of the 'top-level' MinMax nodes to search through. When it's done, give it another one.
- Has the downside that $\alpha\beta$ pruning data does not get updated until a thread finishes its node.
- Initial nodes can take more time than they otherwise might have.
- But still end up with an $O(n)$ speed up, where n is the number of cores.

Concurrency



Concurrency model

Time management

- Vary the search depth depending on time left and how long the previous move took
- Below 5 seconds, drop to depth 2
- Below 15 seconds, drop to depth 4
- If we have extra time, increase to 8
- If we have less time, decrease to 5
- Default to 7

C++ and Ada

- We took the C++ sample client code and used it to call procedures in Ada
- C++ client listened for new messages and saved them
- C++ called Ada procedures
- Procedures called entries on Ada's Main Task
- Messages and information passed using direct edits to shared memory

C++ and Ada Experiences

- Make the communication as simple as possible
- Don't call entries or complex data structures
- C++ decapitalises everything
- Avoid C++ and Ada running computation concurrently near shared data
- C++ and Ada store arrays in different fashions

Other ideas (Monte Carlo)

- Use monte carlo prediction as a substitute for reward to help with learning of mid-game states
- Tends to make the algorithm learn more quickly, and on-the-fly
- Can also be used as a predictor of how good a board state is.

Other ideas (MinMax)

- Optimised search ordering - expand nodes by their value in our piece feature weights
- Store searches between moves. Could have saved a huge amount of computation time.
- Vary depth of search based on branching factor and volatility of the state.

Other ideas (Evolutionary Algorithm)

- Use an evolutionary algorithm strategy to play agents off against each other.
- Use feature values as 'genes'.
- 'Breeding' via exchanging features. Don't need a heuristic for which to exchange, as features tend to be relatively independent.
- Mutation via small random change to a feature.
  e.g. from a uniform distribution around (-1,1), or
  Normal distribution with mean 0.
- Essentially still doing gradient decent. Can be used as a substitute for an ϵ-greedy policy.

Other ideas (Utilising opponent time)

- Compute further down the game tree while the opponent is thinking
- Could allow significantly greater search depths to be reached
- Required memory management and large data structures
- Ada would be very well positioned to perform this task

Winning Speech

- Friends, family, wives, pets, neighbours etc.
- Beyonce has one of the greatest videos of all time. Of all time!
- Won despite our handicap being the only group to not have an Andrew
- Other groups played well but we can't all be winners