3. Adversary Search

Jonathan Schaeffer
jonathan@cs.ualberta.ca
www.cs.ualberta.ca/~jonathan

Definitions - Nodes

- Root node
  - State (position) which is to be searched
- Terminal node
  - A node which has a fixed application-dependent value (e.g., win, loss, draw)
- Leaf node
  - A node which has been assigned a heuristic value
  - A heuristic is an "educated guess" to a approximate a terminal value
- Interior node
  - Nodes whose value is a function of the successors

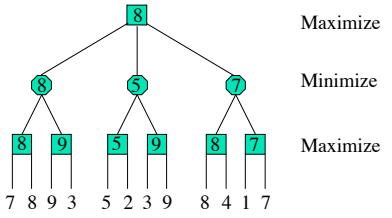
Definitions - Tree

- Search depth
  - Number of state transitions (moves) from the root of the search to the current state (position)
- Branching factor
  - Average number of successor nodes (moves)
- Tree/DAG
  - Most search trees are really DAGS
  - A node can have 1 parent (tree) or possibly more than 1 (DAG)

Adversary Search

- Two (or more) opponents, each trying to maximize their expectations
- Player 1 is called Max
  - Obtain the maximum result
  - Minimize that of the opponent
- Player 2 is called Min
  - Obtain the minimum result
  - Maximize that of the opponent

Minimax Search



Minimax Search

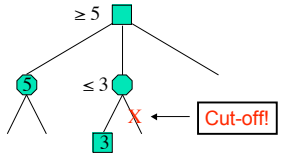
```
Call:     result = MiniMax( s, depth, MAX );

int MiniMax( state s, int depth, int type ) {
    if( terminal node || depth == 0 ) return( Evaluate( s ) );
    if( type == MAX ) {
        for( score = -∞, child = 1; child <= NumbSuccessors( s ); child++ ) {
            value = MiniMax( Successor( s, child ), depth-1, MIN );
            if( value > score ) score = value;
        }
    }
    else {
        for( score = +∞, child = 1; child <= NumbSuccessors( s ); child++ ) {
            value = MiniMax( Successor( s, child ), depth-1, MAX );
            if( value < score ) score = value;
        }
    }
    return( score );
}
```


Minimax Analysis

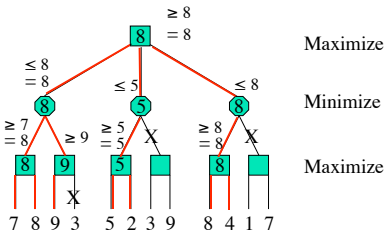
- Assume a fixed branching factor and a fixed depth
- Search complexity is b^d
- Can we do better?

Observation

- Some nodes in the search can be *proven* to be irrelevent to the outcome of the search



Alpha-Beta Algorithm



Alpha-Beta Algorithm

- Maintain two bounds:
  - Alpha (α): a lower bound on best that the player to move can achive
  - Beta (β): an upper bound on what the opponent can achieve
- Search, maintaining α and β
- Whenever $\alpha \geq \beta$, further search at this node is irrelevent

NegaMax Formulation

- Minimax formulation was awkward because the search alternated between MINs and MAXs
- The NegaMax formulation allows only a MAX to be used [1]
- When descending, negate and switch bounds
- When ascending, negate return value

Alpha-Beta Algorithm

```
Call:    result = AlphaBeta( s, depth, -∞, +∞);

int AlphaBeta( state s, int depth, int alpha, int beta ) {
    if( terminal node || depth == 0 ) return( Evaluate( s ) );
    score = -∞;
    for( child = 1; child <= NumbSuccessors( s ); child++ ) {
        value = -AlphaBeta( Successor( s, child ),
                            depth-1, -beta, -alpha );
        if( value > score ) score = value;
        if( score > alpha ) alpha = score;
        if( score >= beta ) break;
    }
    return( score );
}
```

Alpha-Beta Example

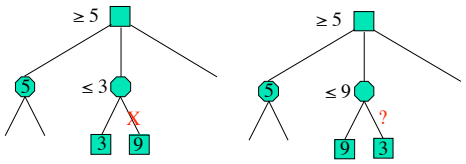
- << Done on white board >>
- Note the *principal variation*
  - Path from root to leaf node of best play by each side
- Note the values returned
  - Regular: with respect to the root player
  - NegaMax: with respect to whose "turn" it is

Warning!

- Alpha-beta is only a few lines of pseudo-code, but it is tricky!
- A bug can remain hidden for a long time
- You may only see a problem when a bad value min's and max's its way to the root, and the probability of this happening can be small!

Analysis

- What is the best case for Alpha-Beta?
- Consider two case:



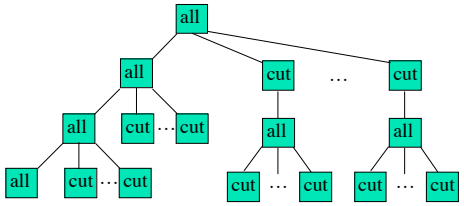
Successor Ordering

- Alpha-beta's performance depends on getting cut-offs as quickly as possible
- At a node where a cut-off is possible, ideally want to search (one of the) best move(s) first, and cut-off immediately

Alpha-Beta Node Types

- Define two node types (6 are possible [2])
- ALL -- all successors of a node must be considered
- CUT -- a cut-off can occur; one or more successors or a node must be considered

Minimal Alpha-beta Tree



- But… this requires an oracle!

Alpha-Beta Analysis

- Assume a fixed branching factor and a fixed depth
- Best case: $b^{\lceil d/2 \rceil} + b^{\lfloor d/2 \rfloor} - 1$ [1]
  - Approximate as $b^{d/2}$
- Impact?
  - Minimax: 10^9 = 1,000,000,000
  - Alpha-beta: $10^5 + 10^4$ = 110,000

Alpha-Beta Analysis

- But… best-case analysis depends on choosing the best move first at cut nodes (not always possible)
- The worst case? No cut-offs, and Alpha-Beta degrades to Minimax
- Exercise: can you construct a tree where no Alpha-Beta cut-offs occur?

Isn't This Good Enough?

- No!
- Ken Thompson showed that search depth was strongly correlated with performance in chess [3]
- Searching one move (or one *ply*) deeper made a huge difference in performance

Performance! Performance!

- Improve Alpha-Beta to guarantee near best-case results
- Improve the heuristic evaluation function to improve the predictive capabilities of the search
- Use parallelism to increase the search depth

Improvements

- Using storage
  - Cycle detection
  - Off-line and on-line computed values
- Exploratory searches
- Successor ordering
- Playing with the search windows
- Playing with the search depth

References

[1] D. Knuth and R. Moore. "An Analysis of Alpha-Beta Pruning", Artificial Intelligence, vol. 6, no. 4, pp. 293-326, 1975.

[2] A. Reinefeld and T.A. Marsland. "A Quantitative Analysis of Minimax Window Search", IJCAI, pp. 951-954, 1987.

[3] K. Thompson. "Computer Chess Strength", Advances in Computer Chess 3, M. Clarke (ed.), pp. 55-56, 1982.