Lecture 25

Minimax

Slides from AIMA – http://aima.cs.berkeley.edu
Announcement:

1. Final May 1, Comprehensive
2. April 24, 26, 28, Project presentation; order will be randomly generated; project progress;
3. Quiz 4
function **Minimax-Decision**(*state*) returns *an action*

inputs: *state*, current state in game

return the *a* in **Actions**(*state*) maximizing **Min-Value**(Result(*a*,
*state*))

function **Max-Value**(*state*) returns *a utility value*

if **Terminal-Test**(*state*) then return **Utility**(*state*)

*v* ← −∞

for *a*, *s* in **Successors**(*state*) do *v* ← **Max**(*v*, **Min-Value**(*s*))

return *v*

function **Min-Value**(*state*) returns *a utility value*

if **Terminal-Test**(*state*) then return **Utility**(*state*)

*v* ← ∞

for *a*, *s* in **Successors**(*state*) do *v* ← **Min**(*v*, **Max-Value**(*s*))

return *v*
minimax(Pos, BestNextPos, Val) :-
    bagof(NextPos, move(Pos, NextPos), NextPosList),
    best(NextPosList, BestNextPos, Val), !.

% Pos has successors

minimax(Pos, _, Val) :-
    utility(Pos, Val).

% Pos has no successors
best([Pos], Pos, Val) :-
    minimax(Pos, _, Val), !.

best([Pos1 | PosList], BestPos, BestVal) :-
    minimax(Pos1, _, Val1),
    best(PosList, Pos2, Val2),
    betterOf(Pos1, Val1, Pos2, Val2, BestPos, BestVal).

% There is no more position to compare
% There are other positions to compare
betterOf(Pos0, Val0, _, Val1, Pos0, Val0) :-
    min_to_move(Pos0),
    Val0 > Val1, !.
    % Pos0 better than Pos1
    % MIN to move in Pos0
    % MAX prefers the greater value

betterOf(Pos0, Val0, _, Val1, Pos0, Val0) :-
    max_to_move(Pos0),
    Val0 < Val1, !.
    % Pos0 better than Pos1
    % MAX to move in Pos0
    % MIN prefers the lesser value

betterOf(_, _, Pos1, Val1, Pos1, Val1).
    % Otherwise Pos1 better than Pos0
move(Pos, NextPos) : states that NextPos is a legal move from Pos
utility(Pos, Val) : states that Pos as a value equal to Val
min_to_move(Pos) : states that the current player in Pos is \textit{min}
max_to_move(Pos) : states that the current player in Pos is \textit{max}
The \(\alpha-\beta\) algorithm

```plaintext
function Alpha-Beta-Decision(state) returns an action
    return the \(a\) in Actions(state) maximizing Min-Value(Result(a, state))

function Max-Value(state, \(\alpha, \beta\)) returns a utility value
    inputs: state, current state in game
    \(\alpha\), the value of the best alternative for \(\text{max}\) along the path to state
    \(\beta\), the value of the best alternative for \(\text{min}\) along the path to state
    if Terminal-Test(state) then return Utility(state)
    \(v \leftarrow -\infty\)
    for \(a, s\) in Successors(state) do
        \(v \leftarrow \text{Max}(v, \text{Min-Value}(s, \alpha, \beta))\)
        if \(v \geq \beta\) then return \(v\)
        \(\alpha \leftarrow \text{Max}(\alpha, v)\)
    return \(v\)

function Min-Value(state, \(\alpha, \beta\)) returns a utility value
    same as Max-Value but with roles of \(\alpha, \beta\) reversed
```
Properties of $\alpha-\beta$

- Pruning **does not** affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O\left(b^{m/2}\right)$
  \[ \Rightarrow \textbf{doubles} \text{ solvable depth} \]
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)
- Unfortunately, $35^{50}$ is still impossible!
Resource limits

Standard approach:

- Use Cutoff-Test instead of Terminal-Test
e.g., depth limit (perhaps add quiescence search)
- Use Eval instead of Utility
i.e., evaluation function that estimates desirability of position

Suppose we have 100 seconds, explore $10^4$ nodes/second

$\Rightarrow$ $10^6$ nodes per move $\approx 35^{8/2} \Rightarrow \alpha-\beta$ reaches depth 8
$\Rightarrow$ pretty good chess program
Evaluation functions

For chess, typically linear weighted sum of features

\[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

e.g., \( w_1 = 9 \) with

\( f_1(s) = (\text{number of white queens}) - (\text{number of black queens}), \) etc.