Alpha-Beta Pruning,
Incomplete Information

2-10-16
Backwards Induction

- For each action, recursively determine the eventual outcome.

- Pick the action that leads to the best outcome for the current player.

Backwards induction applies to all extensive-form games.

Minimax is a two-player, zero-sum, perfect information special case.
Minimax pseudocode

\[
\text{minimax}(\text{state}, \text{depth}=0, \text{maximizing}=\text{True})
\]
\[
\quad \text{if depth limit or end of game reached:}
\quad \quad \text{return value(state)}
\]
\[
\quad \text{values = []}
\]
\[
\quad \text{for action in possibleActions(state):}
\quad \quad \text{next = successor(state, action)}
\quad \quad \text{values.add(minimax(next, depth+1, !maximizing))}
\]
\[
\quad \text{if maximizing: \#at depth 0, return move instead of value}
\quad \quad \text{return max(values)}
\]
\[
\quad \text{else:}
\quad \quad \text{return min(values)}
\]
Minimax example
Minimax exercise
Pruning the Search Space

- For each action, recursively determine the eventual outcome.
  - Keep track of the best outcome that each player can achieve along the path.
  - Don’t recurse if you’re sure the branch won’t change the best outcome.
- Pick the action that leads to the best outcome for the current player.

This sort of pruning is only possible in two-player, zero-sum games.
AB(state, alpha=-inf, beta=inf, depth=0, maximizing=True)

    if depth limit or end of game reached:
        return value(state)

    v = -inf if maximizing else inf

    for action in possibleActions(state):
        next = successor(state, action)

        if maximizing:
            v = max(v, AB(next, alpha, beta, depth+1, !maximizing)
            if v >= beta: return v

            alpha = max(v, alpha)
        else:
            v = min(v, AB(next, alpha, beta, depth+1, !maximizing)
            if v =< alpha: return v

            beta = min(v, beta)

    return v

#at depth 0, return move instead of value
α-β example
$\alpha$-$\beta$ exercise
Poll

Do you want to hold a hex tournament?

a) Yes
b) No
What can we model so far?

With minimax, we can solve:

- two-player, zero-sum, complete information, sequential move games
  - lots of classic board games: chess, checkers, go, hex
  - not much else, and the search space for the above is often too big

With backwards induction, so far, we can solve:

- complete information, sequential move games
  - simple models of corporate competition, a few other economic applications
What can’t we model so far?

Random outcomes

Simultaneous moves

Incomplete information
Incomplete Information

Key idea: information sets
Randomness

Key idea: moves by nature
Backwards Induction Revisited

- For each information set, recursively determine the eventual outcome.

- Pick the action that leads to the best outcome in expectation for the current player.

This can get sticky … the best action at an information set may be to randomize.