Adversarial Search

Chapter 6
Sections 1 – 4
Outline

- Optimal decisions
- $\alpha$-$\beta$ pruning
- Imperfect, real-time decisions
Games vs. search problems

- "Unpredictable" opponent $\rightarrow$ specifying a move for every possible opponent reply

- Time limits $\rightarrow$ unlikely to find goal, must approximate

- Hmm: Is hex a game or a search problem by this definition?
Let’s play!

- Two players:
  - Max
  - Min

**Formal Description:**
- An initial state
- Successor function
- Terminal Test
- Utility Function
Game tree (2-player, deterministic, turns)

Key property: we have a zero-sum game.
- Loosely, it means that there’s a loser for every winner.
- Total utility score over all agents sum to zero.
- Makes the game adversarial.

To think about: not zero sum?
Example: Game of NIM

Several piles of sticks are given. We represent the configuration of the piles by a monotone sequence of integers, such as $(1, 3, 5)$. A player may remove, in one turn, any number of sticks from one pile. Thus, $(1, 3, 5)$ would become $(1, 1, 3)$ if the player were to remove 4 sticks from the last pile. The player who takes the last stick loses.

- Represent the NIM game $(1, 2, 2)$ as a game tree.
Minimax

- Perfect play for deterministic games

- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play

- E.g., 2-ply game:
Minimax algorithm

function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

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
Properties of minimax

- **Complete?** Yes (if tree is finite)
- **Optimal?** Yes (against an optimal opponent)
- **Time complexity?** $O(b^m)$
- **Space complexity?** $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games
$\rightarrow$ exact solution completely infeasible

What can we do?

**Pruning!**
α-β pruning example
α-β pruning example

```
MAX

MIN

3  12  8  2

≥3

3  2

≤2

X  X
```
α-β pruning example
α-β pruning example
Properties of α-β

- Pruning does not affect final result

- Good move ordering improves effectiveness of pruning

- With “perfect ordering”, time complexity = 
  - doubles depth of search
    - What’s the worse and average case time complexity?
    - Does it make sense then to have good heuristics for which nodes to expand first?

- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)
Why is it called α-β?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for max

- If v is worse than α, max will avoid it
  → prune that branch

- Define β similarly for min
The \( \alpha - \beta \) algorithm

**function** \texttt{ALPHA-BETA-SEARCH}(\textit{state}) \textbf{returns} an action

\textbf{inputs:} \textit{state}, current state in game

\( \alpha \leftarrow \texttt{MAX-VALUE}(\textit{state}, -\infty, +\infty) \)

\textbf{return} the action in \texttt{SUCCESSORS}(\textit{state}) with value \( v \)

**function** \texttt{MAX-VALUE}(\textit{state}, \alpha, \beta) \textbf{returns} a utility value

\textbf{inputs:} \textit{state}, current state in game

\( \alpha \), the value of the best alternative for \texttt{MAX} along the path to \textit{state}

\( \beta \), the value of the best alternative for \texttt{MIN} along the path to \textit{state}

\textbf{if} \texttt{TERMINAL-TEST}(\textit{state}) \textbf{then} \textbf{return} \texttt{UTILITY}(\textit{state})

\( v \leftarrow -\infty \)

\textbf{for} \( a, s \) in \texttt{SUCCESSORS}(\textit{state}) \textbf{do}

\( v \leftarrow \texttt{MAX}(v, \texttt{MIN-VALUE}(s, \alpha, \beta)) \)

\textbf{if} \( v \geq \beta \) \textbf{then} \textbf{return} \( v \)

\( \alpha \leftarrow \texttt{MAX}(\alpha, v) \)

\textbf{return} \( v \)
The α-β algorithm

```
function Min-Value(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state
    if Terminal-Test(state) then return Utility(state)
    v ← +∞
    for a, s in Successors(state) do
        v ← Min(v, Max-Value(s, α, β))
        if v ≤ α then return v
        β ← Min(β, v)
    return v
```
Resource limits

The big problem is that the search space in typical games is very large.

Suppose we have 100 secs, explore $10^4$ nodes/sec → $10^6$ nodes per move

Standard approach:

- **cutoff test:**
  e.g., depth limit (perhaps add *quiescence search*)

- **evaluation function**
  = estimated desirability of position
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
  \( f_1(s) = (\text{number of white queens}) - (\text{number of black queens}), \text{etc.} \)

- Caveat: assumes independence of the features
  - Bishops in chess better at endgame
  - Unmoved king and rook needed for castling

- Should model the *expected utility value* states with the same feature values lead to.
A utility value may map to many states, each of which may lead to different terminal states.

Want utility values to model likelihood of better utility states.
Cutting off search

`MinimaxCutoff` is identical to `MinimaxValue` except

1. `Terminal?` is replaced by `Cutoff?`
2. `Utility` is replaced by `Eval`

Does it work in practice?

\[ b^m = 10^6, \ b=35 \Rightarrow m=4 \]

4-ply lookahead is a hopeless chess player!

- 4-ply \( \approx \) human novice
- 8-ply \( \approx \) typical PC, human master
- 12-ply \( \approx \) Deep Blue, Kasparov
Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.


- Othello: human champions refuse to compete against computers, who are too good.

- Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.
Summary

- Games are fun to work on!
- They illustrate several important points about AI
- Perfection is unattainable → must approximate
- Good idea to think about what to think about
Min, go over Hex!

What does the web page say?

http://www.comp.nus.edu.sg/~cs3243/
What do you need to do

- Implement Minimax
- Implement Pruning (optional)
- Implement an evaluation function
  - Input: board, selected grid location
  - Output: continuous value
- (really optional) use state