LOCAL SEARCH

AIMA Chapter 4.1

Previously…

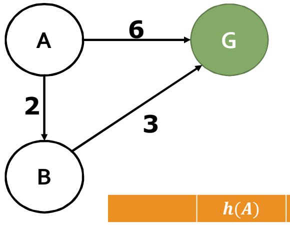
Use Heuristics to Guide Search

- Greedy best-first search
- A^* search

A^* Search Heuristic

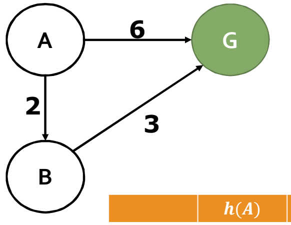
- If $h(n)$ is admissible, then A^* w. Tree-Search is optimal
- If $h(n)$ is consistent, then A^* w. Graph-Search is optimal
- A heuristic that dominates another incurs lower search cost

Consider the search problem



	$h(A)$	$h(B)$	$h(G)$
H-1	4	3	0
H-2	5	2	0

Consider the search problem



Consistent?

$$h(A) - h(B) \leq pathCost$$

Admissible?

$$h(A) \leq pathToGoal^*$$

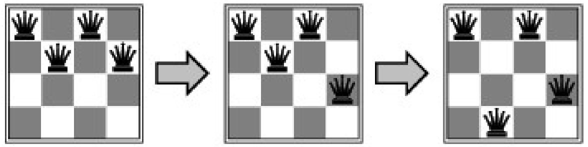
	$h(A)$	$h(B)$	$h(G)$
H-1	4	3	0
H-2	5	2	0

Local Search Algorithms

- The path to goal is irrelevant; the goal state itself is the solution
- State space = set of "complete" configurations
- Find final configuration satisfying constraints, e.g., n-queens
- Local search algorithms: maintain single "current best" state and try to improve it
- Advantages:
  - very little/constant memory
  - find reasonable solutions in large state space

Example: n-queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal



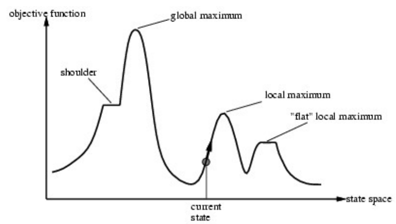
Hill-Climbing Search

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current ← MAKE-NODE(problem.INITIAL-STATE)
  loop do
      neighbor ← a highest-valued successor of current
      if neighbor.VALUE ≤ current.VALUE then return current.STATE
      current ← neighbor
```

"Like climbing Mt. Everest in thick fog with amnesia"

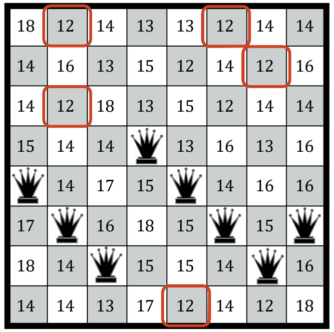
Hill-Climbing Search

- Problem: depending on initial state, can get stuck in local maxima



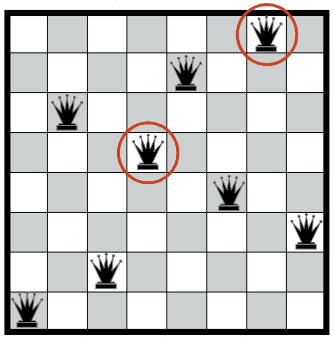
- Non-guaranteed fixes: sideway moves, random restarts

Hill-Climbing Search: 8-Queens



- h = number of pairs of queens that are attacking each other, either directly or indirectly
- $h = 17$ for the above state

Hill-Climbing Search: 8-Queens

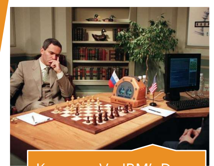


Local Minimum with $h = 1$

Adversarial Search

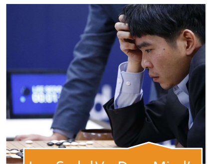
AIMA Chapter 5.1 – 5.5

AI vs. Human Players: the State of the Art



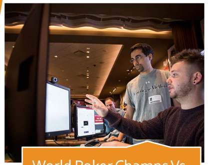
Kasparov Vs. IBM's Deep Blue (1997)

World Poker Champs Vs. Libratus (2017)

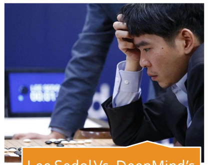


Lee Sedol Vs. DeepMind's AlphaGo (2016)

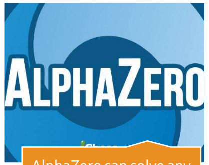
AI vs. Human Players: the State of the Art



World Poker Champs Vs. Libratus (2017)



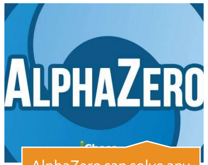
Lee Sedol Vs. DeepMind's AlphaGo (2016)



AlphaZero can solve any zero-sum game! (2018)

AI vs. Human Players: the State of the Art

Lee Sedol Vs. DeepMind's AlphaGo (2016)



AlphaZero can solve any zero-sum game! (2018)

OpenAI Five

OpenAI 5 vs world champs—DOTA2 (2019)

Deterministic Games in Practice

- Chinook (Checkers, 1994). Precomputed endgame database ⇒ perfect play for all positions with ≤ 8 pieces on the board (total of 444 billion positions).
- Deep Blue (Chess, 1997). Searches 200 million positions/sec + evaluation functions + secret sauce.
- Logistello (Othello, 1997). Human champions refuse to play against AI.
- AlphaGo + Alphazero (Go/everything above, 2016-2017). Learning for evaluation functions + database and efficient search + secret sauce.

Outline

- Adversarial search problems (aka games)
- Optimal (i.e., Minimax) decisions
- α-β pruning
- Imperfect, real-time decisions

Games vs. Search Problems

Utility maximizing opponent

- solution is a strategy specifying a move for every possible opponent response.

Time limit

- unlikely to find goal, must approximate

Let's Play!

- Two players in a **zero-sum game**:
  - Winner gets paid and loser pays.
- Easy to think in terms of a **max player** and **min player**
  - Player 1 wants to maximize value (MAX player)
  - Player 2 wants to minimize value (MIN player)

Game: Problem Formulation

A game is defined by 7 components:

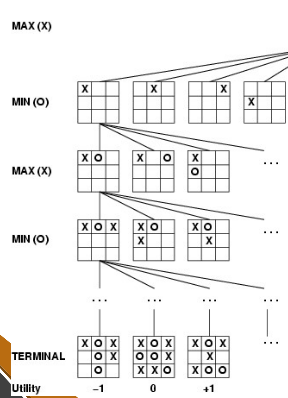
1. Initial state s_0
2. States
3. Players: $PLAYER(s)$ defines which player has the move in state s.
4. Actions : $ACTIONS(s)$ returns set of legal moves in state s
5. Transition model : $RESULT(s, a)$ returns state that results from the move a in state s.

Game: Problem Formulation

A game is defined by 7 components:

6. Terminal test $TERMINAL(s)$ returns true if game is over and false otherwise
   - Terminal states: states where the game has ended
7. Utility function $UTILITY(s, p)$ gives final numeric value for a game that ends in terminal state s for a player p
   - Chess: White wins $+1$; Black wins -1; draw 0.

Game Tree
(2-Player, Deterministic, Turn-Taking)



Key property: **zero-sum** game

- Loosely, it means that there's a loser for every winner
- Total utility score over all agents sum to zero (or constant)
- Aka constant-sum game
- Makes the game adversarial

Non Zero-Sum Games?

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.

Example : Game of NIM

<table>
<tr><th>Start</th><th>||| (A)</th><th>|||| (B)</th><th>||||| (C)</th></tr>
<tr><td>Bob take 2 from A</td><td>|</td><td>||||</td><td>|||||</td></tr>
<tr><td>Alice take 3 from C</td><td>|</td><td>||||</td><td>||</td></tr>
<tr><td>Bob take 1 from B</td><td>|</td><td>|||</td><td>||</td></tr>
<tr><td>Alice take 1 from B</td><td>|</td><td>||</td><td>||</td></tr>
<tr><td>Bob take 1 from A</td><td>0</td><td>||</td><td>||</td></tr>
<tr><td>Alice take 1 from B</td><td>0</td><td>|</td><td>||</td></tr>
<tr><td>Bob take 2 from C</td><td>0</td><td>|</td><td>0</td></tr>
</table>

Bob wins!!

|| | ||

Represent the NIM game (1,2,2) **as a game tree.**

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

Play against what kind of player?
Stupid? Smart?

Player Strategies

A strategy s for player i: what will player i do at every node of the tree that they make a move in?

Need to specify behavior in states that will never be reached!

Winning Strategy

A strategy s_1^* for player 1 is called **winning** if for any strategy s_2 by player 2, the game ends with player 1 as the winner.

A strategy t_1^* for player 1 is called **non-losing** if for any strategy s_2 by player 2, the game ends in either a tie or a win for player 1.

Nash Equilibrium

When players knows the strategies of all opponents: no one wants to change her strategy

Stronger form of Nash equilibrium – subgame perfect Nash equilibrium

Every subgame is a Nash equilibrium

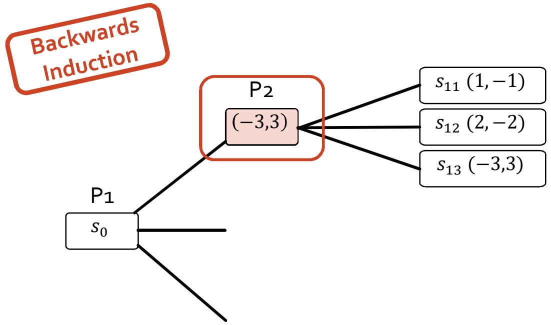
Optimal Strategy at Node - Minimax

$$Minimax(s) = \begin{cases} Utility(s) & \text{if TerminalTest}(s) \\ \max_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s,a)) & \text{if Player}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s,a)) & \text{if Player}(s) = \text{MIN} \end{cases}$$

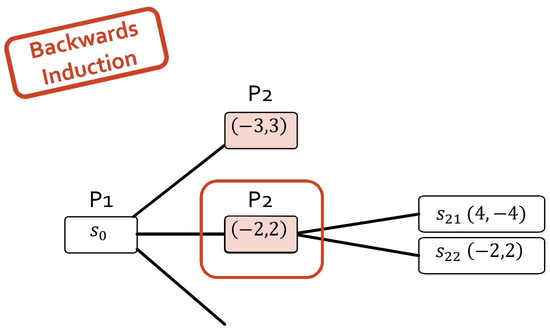
Intuitively,

- MAX chooses move to maximize the minimum payoff
- MIN chooses move to minimize the maximum payoff

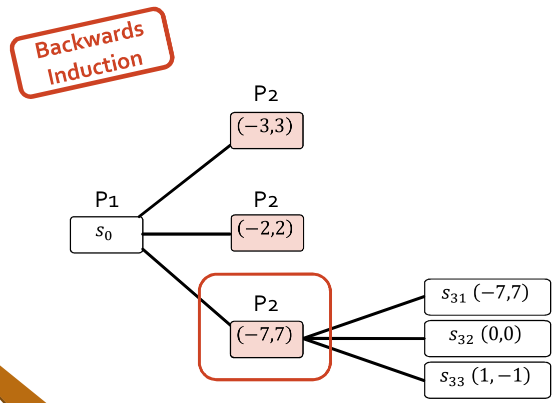
Minimax Play (Subperfect Nash Equilibrium)



Minimax Play (Subperfect Nash Equilibrium)

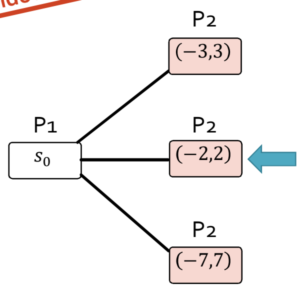


Minimax Play (Subperfect Nash Equilibrium)



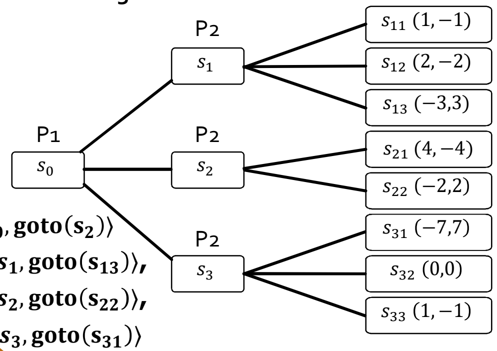
Minimax Play
(Subperfect Nash Equilibrium)

Backwards Induction



Minimax Play
(Subperfect Nash Equilibrium)

What are the optimal strategies in this game?



P1 $\rightarrow \langle s_0, \mathbf{goto}(s_2) \rangle$

P2 $\rightarrow \langle s_1, \mathbf{goto}(s_{13}) \rangle$, $\langle s_2, \mathbf{goto}(s_{22}) \rangle$, $\langle s_3, \mathbf{goto}(s_{31}) \rangle$

Properties of Minimax

Computes a Nash equilibrium of the game

Property	
Complete?	Yes (if game tree is finite)
Optimal	Yes (optimal gameplay)
Time	$\mathcal{O}(b^m)$
Space	Like DFS: $\mathcal{O}(bm)$

Minimax Algorithm

- Runs in time polynomial in tree size
- Returns a sub-perfect Nash equilibrium: the best action at every choice node.

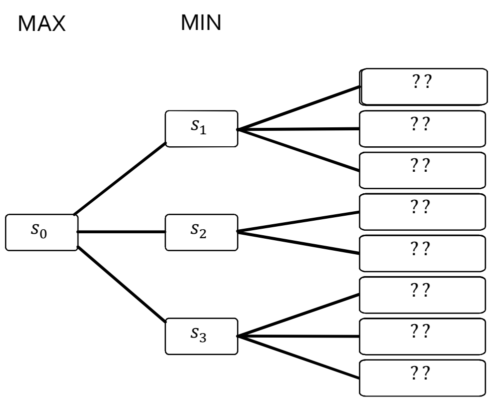
Are we done here?

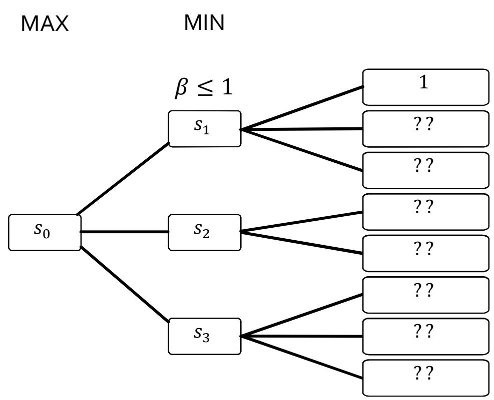
Backwards Induction

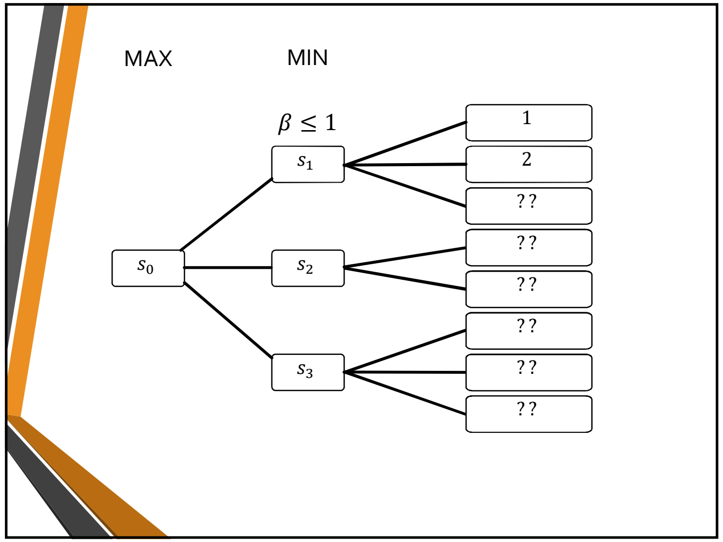
- Game trees are huge: chess has game tree with $\sim 10^{123}$ nodes (planet Earth has $\sim 10^{50}$ atoms)
- Impossible to expand the entire tree

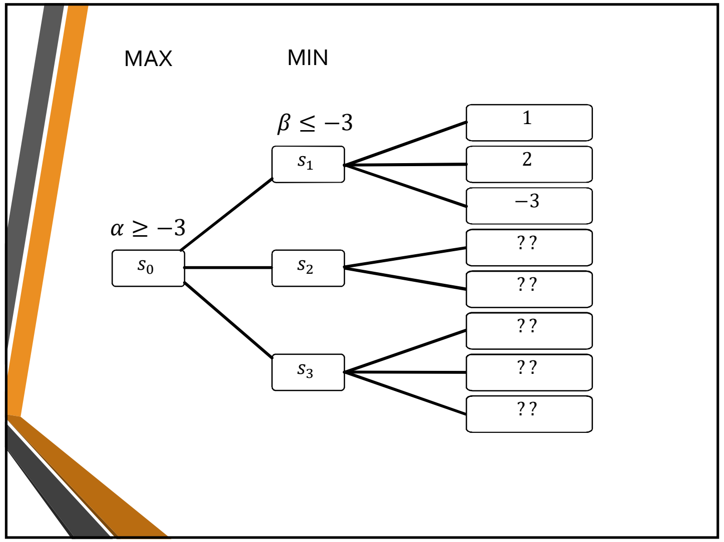
α-β Pruning

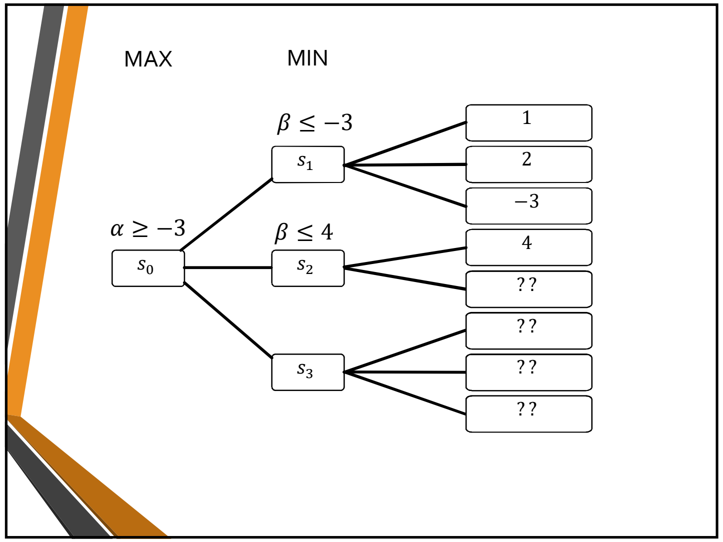
- **Basic idea:** *"If you have an idea that is surely bad, don't take the time to see how truly awful it is."* -- Pat Winston
- Maintain a **lower bound α** and **upper bound β** of the values of, respectively, MAX's and MIN's nodes seen thus far. We can prune subtrees that will never affect minimax decision.

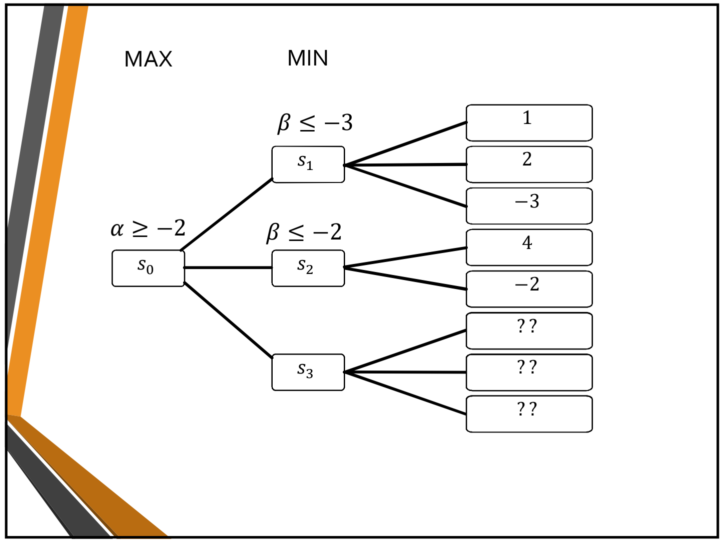


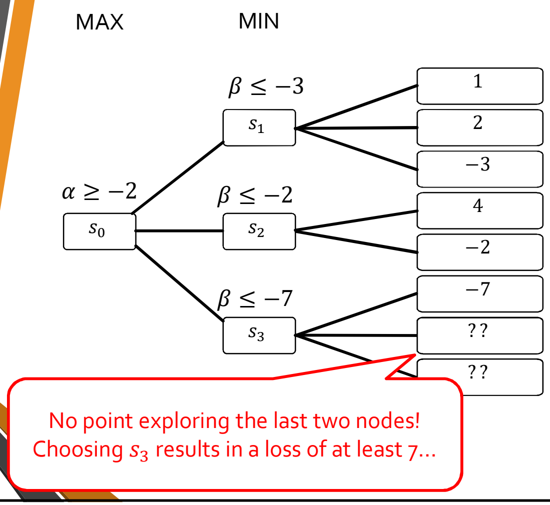



MAX
MIN
$\beta \leq 1$
s_1
s_0
s_2
s_3
1
2
??
??
??
??
??
??


MAX
MIN
$\beta \leq -3$
$\alpha \geq -3$
s_1
s_0
s_2
s_3
1
2
-3
??
??
??
??
??


MAX
MIN
$\beta \leq -3$
$\alpha \geq -3$
$\beta \leq 4$
s_1
s_0
s_2
s_3
1
2
-3
4
??
??
??
??


MAX
MIN
$\beta \leq -3$
$\alpha \geq -2$
$\beta \leq -2$
s_1
s_0
s_2
s_3
1
2
-3
4
-2
??
??
??



α-β Pruning

- MAX node n: $\alpha(n)$ = highest **observed** value found on path from n; initially $\alpha(n) = -\infty$
- MIN node n: $\beta(n)$ is the lowest observed value found on path from n; initially $\beta(n) = +\infty$
- Pruning:
  - given a MIN node n, stop searching below n if there is some MAX ancestor i of n with $\alpha(i) \ge \beta(n)$
  - given a MAX node n, stop searching below n if there is some MIN ancestor i of n with $\beta(i) \le \alpha(n)$

Analysis of α-β Pruning

- When we prune a branch, it **never** affects final outcome.
- Good move ordering improves effectiveness of pruning
- "Perfect" ordering: time complexity $= O\left(b^{\frac{m}{2}}\right)$
  - → Good pruning strategies allow us to search twice as deep!
  - Chess: simple ordering (checks, then take pieces, then forward moves, then backwards moves) gets you close to best-case result.
  - It makes sense to have good expansion order heuristics.
- Random ordering: time complexity = $O\left(b^{\frac{3m}{4}}\right)$ for $b < 1000$

Summary: α-β Pruning Algorithm

- Initially, $\alpha(n) = -\infty$, $\beta(n) = +\infty$
- $\alpha(n)$ is max along search path containing n
- $\beta(n)$ is min along search path containing n
- If a MIN node has value $v \le \alpha(n)$, no need to explore further.
- If a MAX node has value $v \ge \beta(n)$, no need to explore further.

Time Limit

- **Problem:** very large search space in typical games
- **Solution:** α-β pruning removes large parts of search space
- Unresolved problem: Maximum depth of tree
- Standard solutions:
  - evaluation function = estimated expected utility of state
  - cutoff test: e.g., depth limit

Heuristic Minimax Value

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \boxed{\text{if TERMINAL-TEST}(s)} \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s) & \boxed{\text{if CUTOFF-TEST}(s,d)} \\ \max_{a \in \text{ACTIONS}(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Run minimax until depth d; then start using the evaluation function to choose nodes.

Evaluation Functions

- An evaluation function is a mapping from game states to real values: $f: \mathcal{S} \rightarrow \mathbb{R}$
  - So far:
    $$f(s) = \begin{cases} UTILITY(s) \text{ if } TERMINAL(s) \\ 0 \; otherwise \end{cases}$$
- Should be cheap to compute
- For non-terminal states, must be strongly correlated with actual chances of winning
- Tic-Tac-Toe:
  $f(n) = [\text{\#of open lines for X}] - [\text{\# of open lines for O}]$

Evaluation Functions

- Chess:
  - Alan Turing's evaluation function:
    $f(n) = \frac{w(n)}{b(n)}$ where $w(n)$ is the point value of white's pieces and $b(n)$ is the point value of black's pieces.
  - Modern evaluation functions: weighted sum of position features
    $f(n) = w_1 F_1(n) + w_2 F_2(n) + \cdots + w_k F_k(n)$
  - Example features: piece count, piece placement, controlled squares etc.
  - Deep Blue has about 6000 features.
  - How do we determine weights? Do they change dynamically?

Evaluation Functions

- Suppose that $f(n) = \sum_{j=1}^{k} w_j F_j(n)$
- If for $n, n' \in \mathcal{S}$ we have $F_j(n) = F_j(n')$ for all j, then they're indistinguishable.
- … evaluates all states with same value.

- Let $\mathcal{S}(\vec{q}) = \{n \in \mathcal{S}: \forall j \in [k], F_j(n) = q_j\}$: all states whose feature values are as specified in the vector $\vec{q}$.
- "All states where white has two pawns but black has a bishop."
- Suppose we know that in this case,
  - Black wins 61% of games
  - White wins 15% of games
  - 24% of games end in a draw
- Then expected utility for black is
  $0.61 \times 1 + 0.15 \times (-1) + 0.24 \times 0 = 0.46$
- Evaluation function need not return actual expected values, just maintain relative order of states.

Evaluation alone is not enough!

- Opening move search doesn't result in useful utility estimates
- Applying evaluation functions on end game scenarios may not solve the game
  - Western chess KRK end game
- Use a policy or lookup table (taken from expert knowledge or previous game history)

Cutting Off Search

- Modify minimax or α-β pruning algorithms by replacing
  - TERMINAL-TEST(*state*) with CUTOFF-TEST(*state*, *depth*)
  - UTILITY(*state*) is replaced by EVAL(*state*)
- Can also be combined with iterative deepening

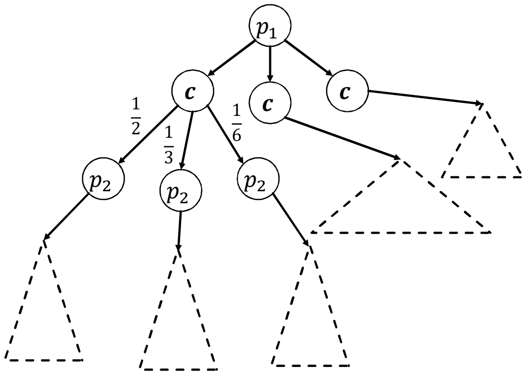
Cutting Off Search

- Why is it useful?
  - For chess, $b^m = 10^6, b = 35 \Rightarrow m = 4$
  - Ply = 4 – Novice chess player
  - Ply = 8 – Typical PC engine
  - Ply = 12~14 – Grandmaster chess (Deep blue)

Stochastic Games

- Many games have randomization:
  - Backgammon
  - Settlers of Catan
  - Poker
- How do we deal with uncertainty?
- Can we still use minimax? Yes, but search space is much bigger

Adding Chance Layers



Calculate the expected value of a state
(MUCH harder than deterministic games)