CSC520 - Artificial Intelligence

Lecture 8

Dr. Scott N. Gerard

North Carolina State University

February 1, 2025

Agenda

Multiagent Environments
Adversarial agents with conflicting goals

- Adversarial search
- MinMax algorithm
- Alpha-Beta pruning

Games

- Games are competitive or cooperative environments with two or more agents (players)
- Different types of games
    - Deterministic vs stochastic
    - 1, 2, . . . , n players. Large n $\Rightarrow$ economy
    - Zero sum vs win-win
    - Perfect information (fully observable) vs imperfect information (partially observable)
- Study adversarial search algorithms on games
- Intent is to compute a strategy or policy which recommends a move from each state

Two-player, Zero-sum Game Problem Formulation

Deterministic, two-players, turn-taking, perfect information, zero-sum

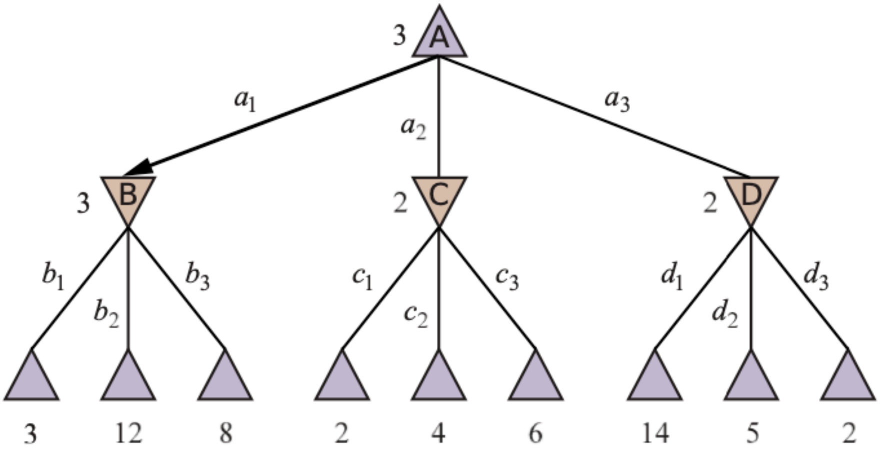
- S_0: initial state
- $\text{TO-MOVE}(s)$: the player to move in state s
- $\text{ACTIONS}(s)$: legal moves in state s
- $\text{RESULT}(s, a)$: transition model; state resulting from taking action a in s
- $\text{IS-TERMINAL}(s)$: if true, game is over in s
- $\text{UTILITY}(s, p)$: value of s to player p

Minimax Search

- State-space search tree
- Players alternate turns
- Compute each node's minimax value
  - Minimax value is the best achievable utility assuming both players play optimally, that is both are rational

Minimax Search Example

Two player, deterministic, zero-sum, and perfect information game



Minimax Search Algorithm

function MINIMAX-DECISION(*state*) **return** an *action*

$v \leftarrow$ MAX-VALUE(*state*)

return the *action* in ACTIONS(*state*) which leads to a state with value v

function MAX-VALUE(*state*) **return** a *utility* value

if IS-TERMINAL(*state*) **return** the UTILITY(*state*)

$v \leftarrow -\infty$

for a in ACTIONS(*state*) **do**

$v \leftarrow$ MAX(v, MIN-VALUE(RESULT(*state*, a)))

return v

function MIN-VALUE(*state*) **return** a *utility* value

if IS-TERMINAL(*state*) **return** the UTILITY(*state*)

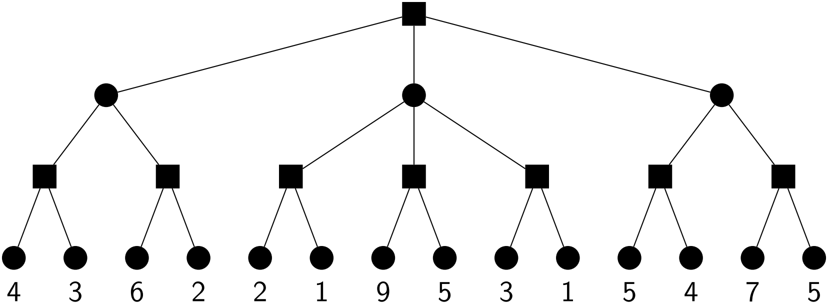
$v \leftarrow \infty$

for a in ACTIONS(*state*) **do**

$v \leftarrow$ MIN(v, MAX-VALUE(RESULT(*state*, a)))

return v

Minimax Algorithm Example



function MAX-VALUE(*state*) **return** a *utility* value

  if IS-TERMINAL(*state*) **return** the UTILITY(*state*)

  $v \leftarrow -\infty$

  for a in ACTIONS(*state*) **do**

    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(*state*, a)))

  return v

function MIN-VALUE(*state*) **return** a *utility* value

  if IS-TERMINAL(*state*) **return** the UTILITY(*state*)

  $v \leftarrow \infty$

  for a in ACTIONS(*state*) **do**

    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(*state*, a)))

  return v

Minimax Properties

- Similar to depth first search
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$

Minimax Properties

- Similar to depth first search
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$

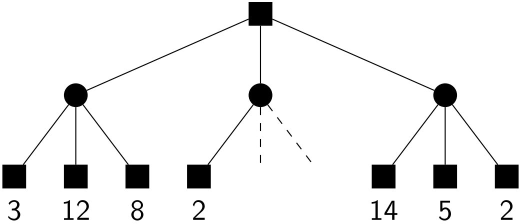
- For example, chess has $b \approx 35$, $m \approx 100$
  - Too large to compute exact solution

Minimax Properties

- Similar to depth first search
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$

- For example, chess has $b \approx 35$, $m \approx 100$
  - Too large to compute exact solution
- How to reduce the complexity?
  - Dynamic pruning of the search tree
  - Early cutoff of the search tree using a heuristic evaluation function

Tree Pruning



Alpha-Beta Pruning

- α = best choice found so far for MAX (at least)
- β = best choice found so far for MIN (at most)

function ALPHA-BETA-SEARCH(*state*) **return** an *action*

  v, *move* $\leftarrow$ MAX-VALUE(*state*, $-\infty$, $+\infty$) **return** *move*

function MAX-VALUE(*state*, α, β) **return** (*utility*, *move*) pair

  if IS-TERMINAL(*state*) **then return** UTILITY(*state*)

  $v \leftarrow -\infty$

  for a in ACTIONS(*state*) **do**

    $v2$, $a2$ $\leftarrow$ MIN-VALUE(RESULT(*state*, a), α, β)

    if $v2 > v$ **then**

      v, *move* $\leftarrow$ $v2$, $a2$

      $\alpha \leftarrow$ MAX(α, v)

    if $v \geq \beta$ **then return** v, *move*

  return v, *move*

function MIN-VALUE(*state*, α, β) **return** (*utility*, *move*) pair

  if IS-TERMINAL(*state*) **then return** UTILITY(*state*)

  $v \leftarrow +\infty$

  for a in ACTIONS(*state*) **do**

    $v2$, $a2$ $\leftarrow$ MAX-VALUE(RESULT(*state*, a), α, β)

    if $v2 < v$ **then**

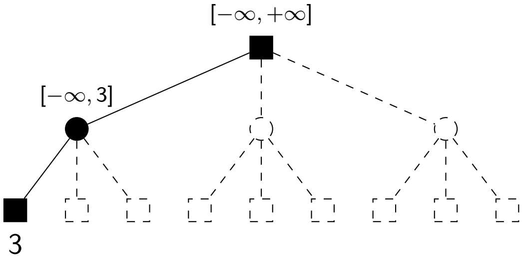
      v, *move* $\leftarrow$ $v2$, $a2$

      $\beta \leftarrow$ MIN(β, v)

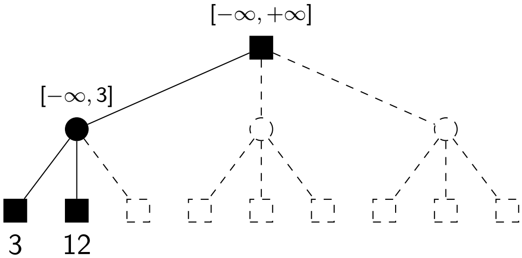
    if $v \leq \alpha$ **then return** v, *move*

  return v, *move*

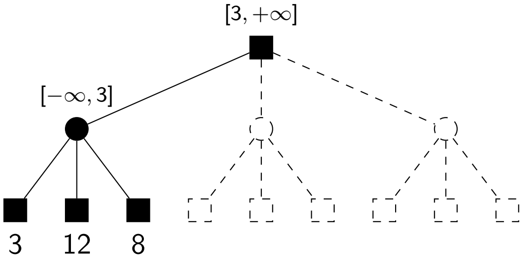
Alpha-Beta Pruning



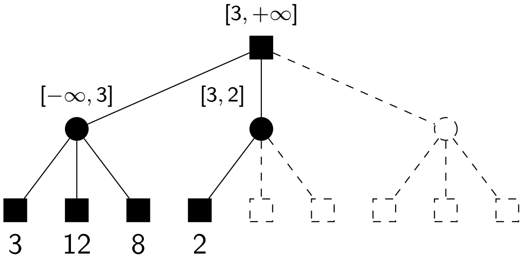
Alpha-Beta Pruning



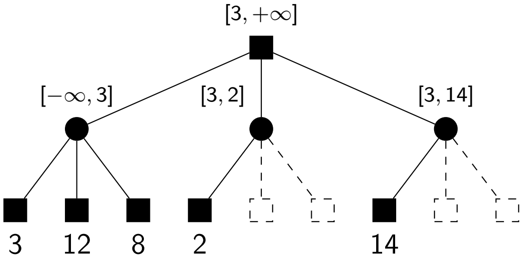
Alpha-Beta Pruning



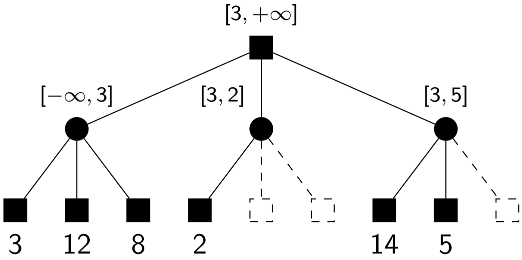
Alpha-Beta Pruning



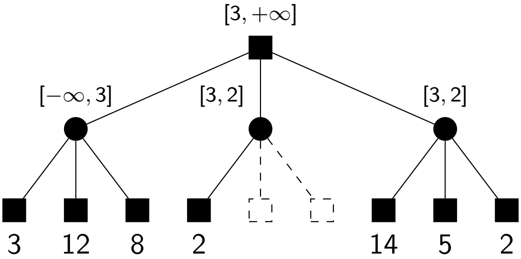
Alpha-Beta Pruning



Alpha-Beta Pruning



Alpha-Beta Pruning

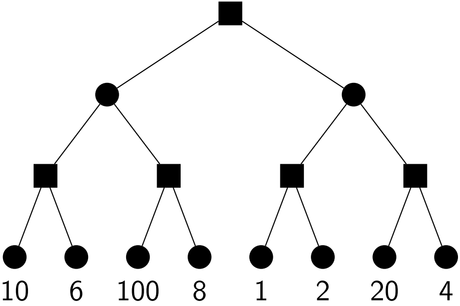


Alpha-Beta Pruning

- Pruning does not affect the final result
- Effectiveness of pruning depends upon the order of successors
  - More pruning is possible if successors likely to be best are examined first
- With *perfect ordering* time complexity drops to $O(b^{m/2})$
  - Can look ahead roughly twice as deep as minimax in the same amount of time

Class Exercise

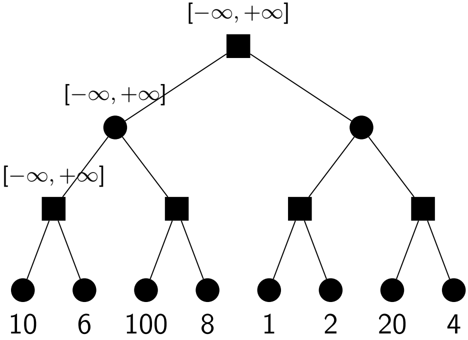
Apply Alpha-Beta search to the game tree below.



Class Exercise

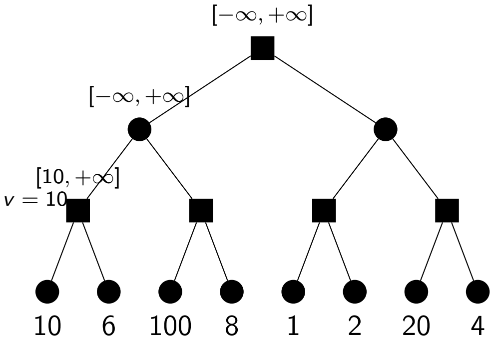
MAX-VALUE prunes if $v \geq \beta$ and updates $\alpha \leftarrow \text{MAX}(\alpha, v)$

MIN-VALUE prunes if $v \leq \alpha$ and updates $\beta \leftarrow \text{MIN}(\beta, v)$



Class Exercise

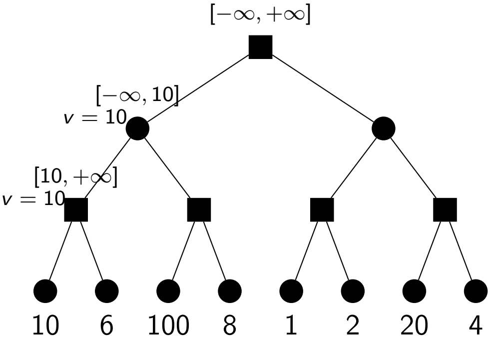
Apply Alpha-Beta search to the game tree below.



Class Exercise

MAX-VALUE prunes if $v \geq \beta$ and updates $\alpha \leftarrow \text{MAX}(\alpha, v)$

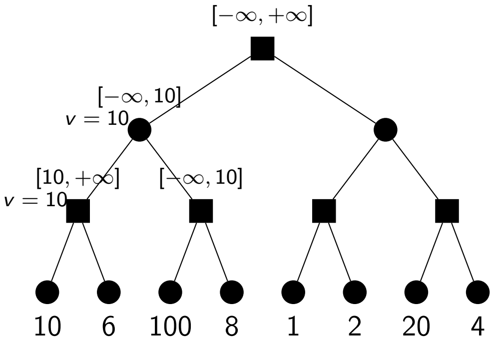
MIN-VALUE prunes if $v \leq \alpha$ and updates $\beta \leftarrow \text{MIN}(\beta, v)$



Class Exercise

MAX-VALUE prunes if $v \geq \beta$ and updates $\alpha \leftarrow \text{MAX}(\alpha, v)$

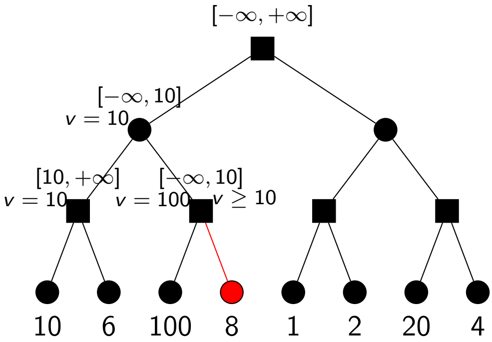
MIN-VALUE prunes if $v \leq \alpha$ and updates $\beta \leftarrow \text{MIN}(\beta, v)$



Class Exercise

MAX-VALUE prunes if $v \geq \beta$ and updates $\alpha \leftarrow \text{MAX}(\alpha, v)$

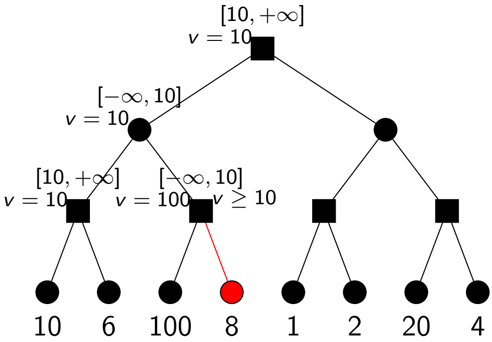
MIN-VALUE prunes if $v \leq \alpha$ and updates $\beta \leftarrow \text{MIN}(\beta, v)$



Class Exercise

MAX-VALUE prunes if $v \geq \beta$ and updates $\alpha \leftarrow \text{MAX}(\alpha, v)$

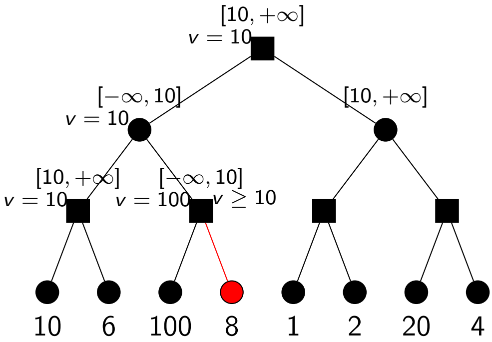
MIN-VALUE prunes if $v \leq \alpha$ and updates $\beta \leftarrow \text{MIN}(\beta, v)$



Class Exercise

MAX-VALUE prunes if $v \geq \beta$ and updates $\alpha \leftarrow \text{MAX}(\alpha, v)$

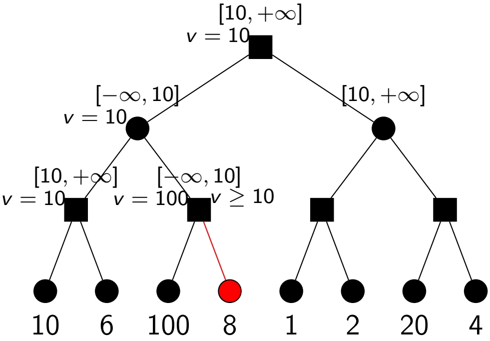
MIN-VALUE prunes if $v \leq \alpha$ and updates $\beta \leftarrow \text{MIN}(\beta, v)$



Class Exercise

MAX-VALUE prunes if $v \geq \beta$ and updates $\alpha \leftarrow$ MAX(α, v)

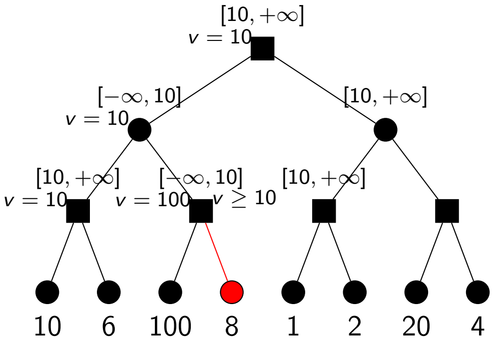
MIN-VALUE prunes if $v \leq \alpha$ and updates $\beta \leftarrow$ MIN(β, v)



Class Exercise

MAX-VALUE prunes if $v \geq \beta$ and updates $\alpha \leftarrow \text{MAX}(\alpha, v)$

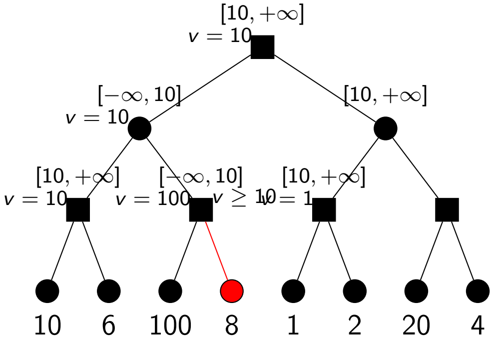
MIN-VALUE prunes if $v \leq \alpha$ and updates $\beta \leftarrow \text{MIN}(\beta, v)$



Class Exercise

MAX-VALUE prunes if $v \geq \beta$ and updates $\alpha \leftarrow$ MAX(α, v)

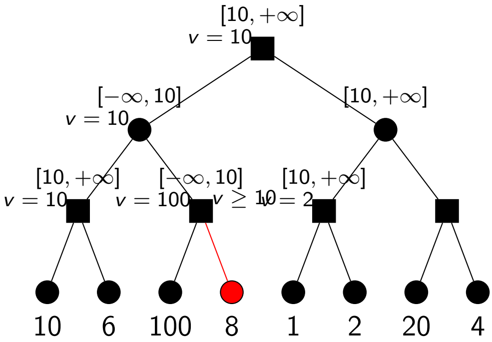
MIN-VALUE prunes if $v \leq \alpha$ and updates $\beta \leftarrow$ MIN(β, v)



Class Exercise

MAX-VALUE prunes if $v \geq \beta$ and updates $\alpha \leftarrow \text{MAX}(\alpha, v)$

MIN-VALUE prunes if $v \leq \alpha$ and updates $\beta \leftarrow \text{MIN}(\beta, v)$



Class Exercise

MAX-VALUE prunes if $v \geq \beta$ and updates $\alpha \leftarrow \text{MAX}(\alpha, v)$

MIN-VALUE prunes if $v \leq \alpha$ and updates $\beta \leftarrow \text{MIN}(\beta, v)$