SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Adversarial Search II

PROF. LIM KWAN HUI

50.021 Artificial Intelligence

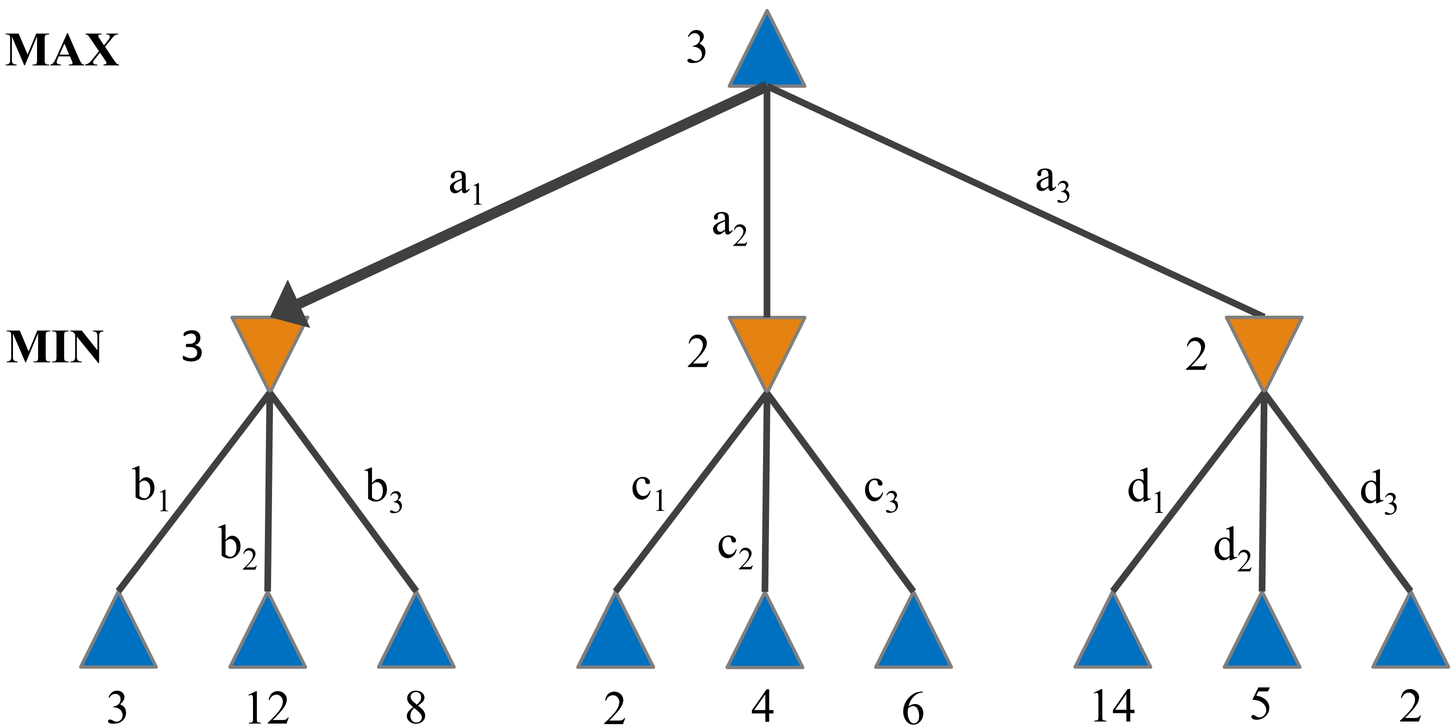
Recap: Representing a Game as a Search Problem

- We can formally define a strategic two-player game by:
  - Initial State
  - Actions
  - Terminal Test (Win / Lose / Draw)
  - Utility Function (numerical reward for the outcome)
    - Chess: +1, 0, -1
    - Poker: Cash won or lose

- In a zero-sum game with two players
  - each player's utility for a state are equal and opposite

Recap: Minimax Algorithm

- E.g., 2-ply game



Recap: Properties of Minimax

- Completeness: Yes, if tree is finite
- Optimality: Yes, against an optimal opponent.
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$ (depth-first exploration)

- What issues might there be in terms of time complexity?

Recap: Properties of Minimax

- Completeness: Yes, if tree is finite
- Optimality: Yes, against an optimal opponent.
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$ (depth-first exploration)

- What issues might there be in terms of time complexity?
  - For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  - $\Rightarrow$ Exact solution completely infeasible

Resource Limits

- Suppose we have 100 seconds, able to explore 10^4 nodes/second
  - $\Rightarrow 10^6$ nodes per move

- Standard Approach:
  - Cutoff test
    - e.g., depth limit
  - Evaluation function
    - = estimated desirability of position

Resource Limits

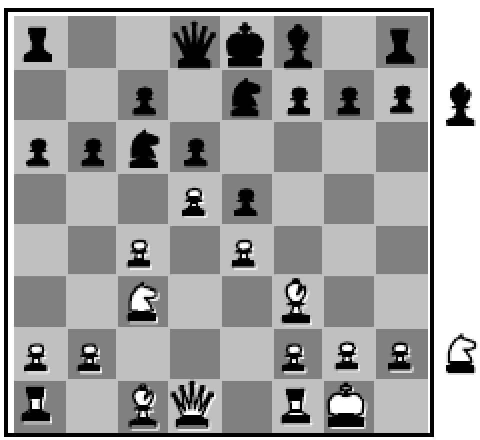
- Suppose we have 100 seconds, able to explore 10^4 nodes/second
  - $\Rightarrow 10^6$ nodes per move

- Standard Approach:
  - Cutoff test
    - e.g., depth limit
  - Evaluation function
    - = estimated desirability of position

- What type of evaluation function can you think of?
  - E.g., for a game of chess

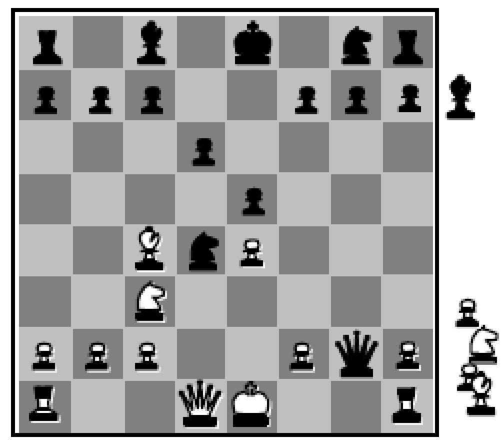
Evaluation Functions

- For chess, typically linear weighted sum of features
  - $Eval(s) = w_1f_1(s) + w_2f_2(s) + \ldots \ldots + w_nf_n(s)$
  - e.g., $w_1 = 9$ with $f_1(s)$ = (number of white queens) - (number of black queens)



Black to move

White slightly better

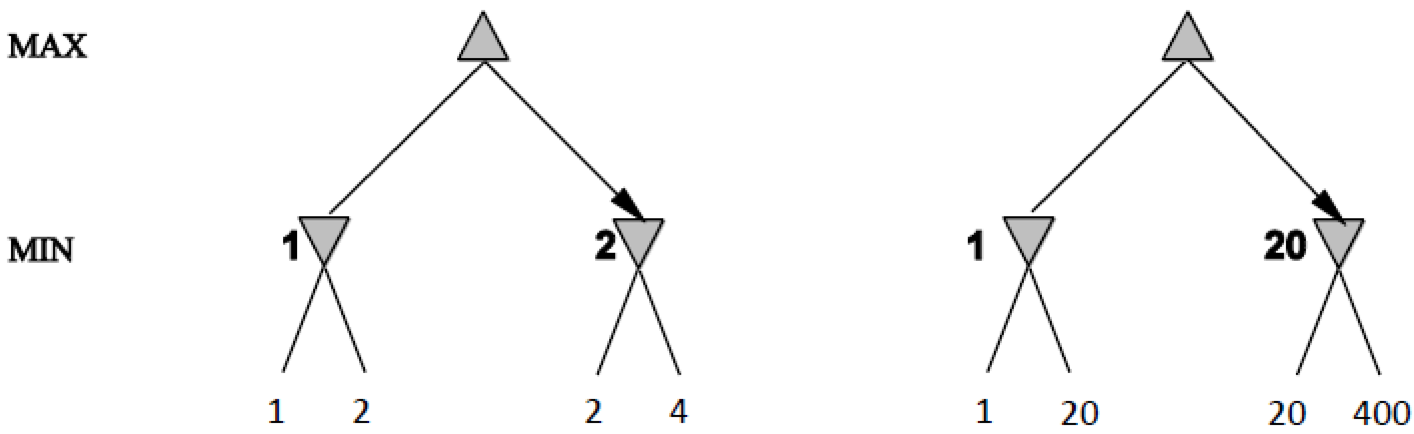


White to move

Black winning

Digression: Exact values don't matter

- Behaviour is preserved under any *monotonic* transformation of Eval
- Only the order matters:
  - payoff in deterministic games acts as an *ordinal utility* function



Cutting off search

- MinimaxCutoff is identical to MinimaxValue except
  - Terminal? is replaced by Cutoff?
  - Utility is replaced by Eval

Operator = Action or Move

```
function MINIMAX-DECISION(game) returns an operator

    for each op in OPERATORS[game] do
        VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
    end
    return the op with the highest VALUE[op]
```

Replaced by Cutoff-Test

```
function MINIMAX-VALUE(state, game) returns a utility value

    if TERMINAL-TEST[game](state) then
        return UTILITY[game](state)
    else if  MAX is to move in state then
        return the highest MINIMAX-VALUE of SUCCESSORS(state)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

Replaced by Eval score

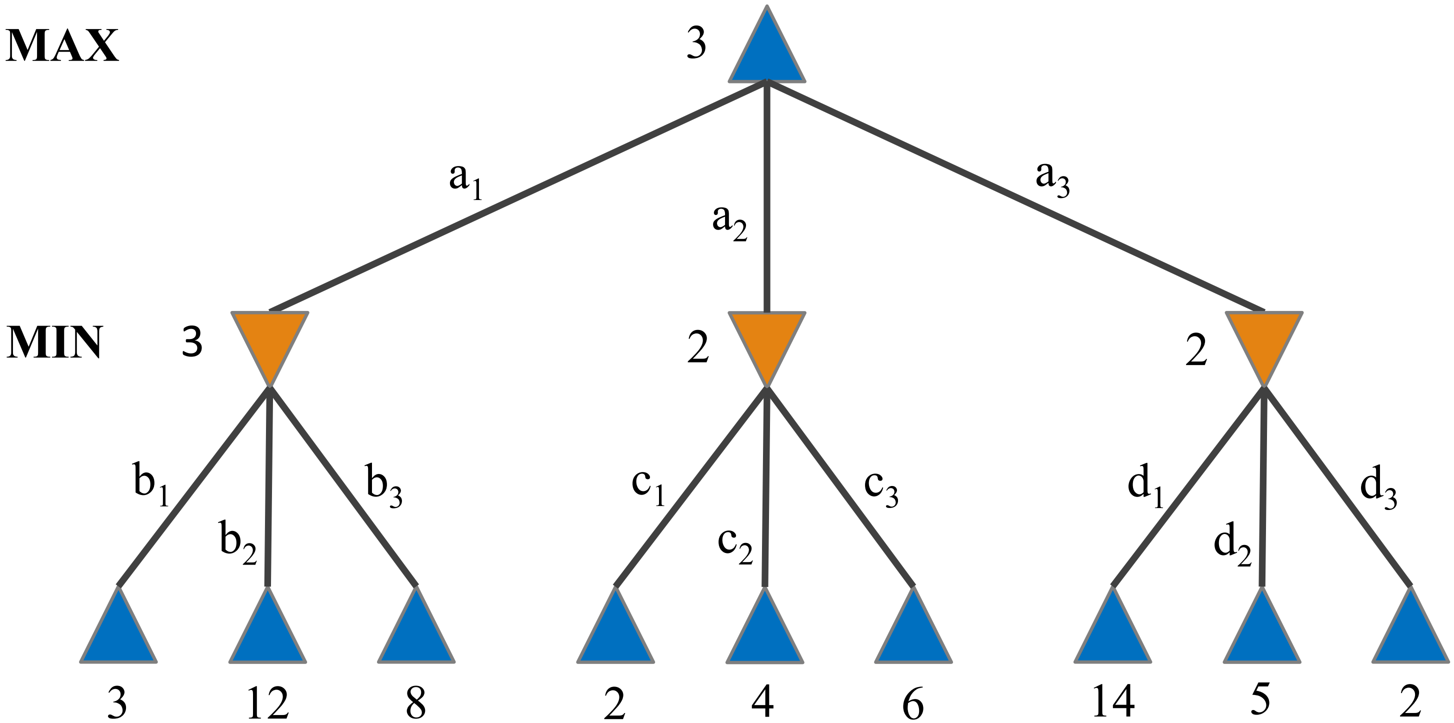
Cutting off search

- MinimaxCutoff is identical to MinimaxValue except
  - Terminal? is replaced by Cutoff?
  - Utility is replaced by Eval

- Does it work in practice?
  - $b^m = 10^6$, $b = 35 \Rightarrow m = 4$
  - 4-ply lookahead is a novice chess player!

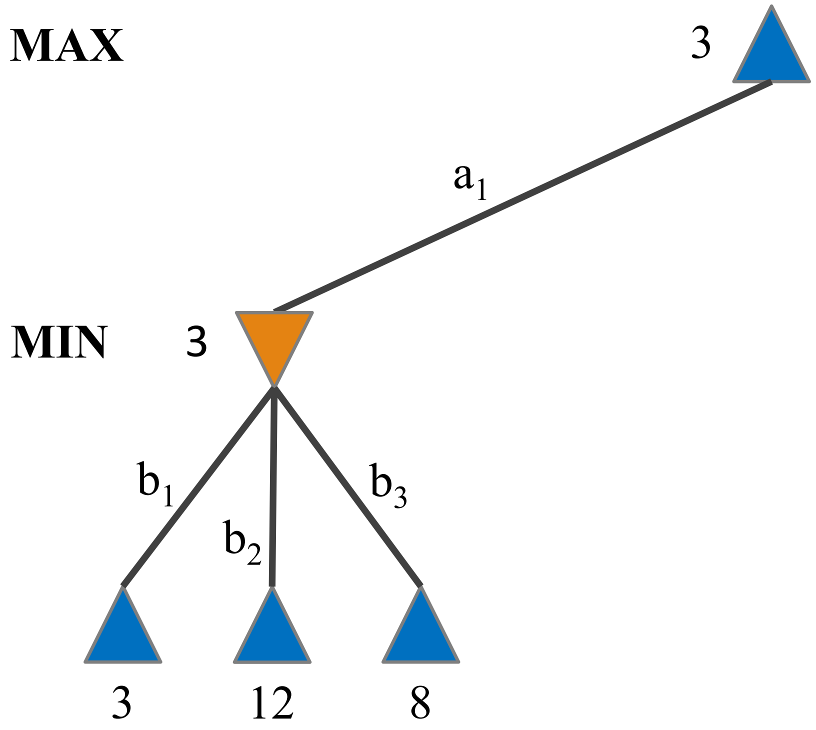
α - β Pruning Example

- Original Minimax



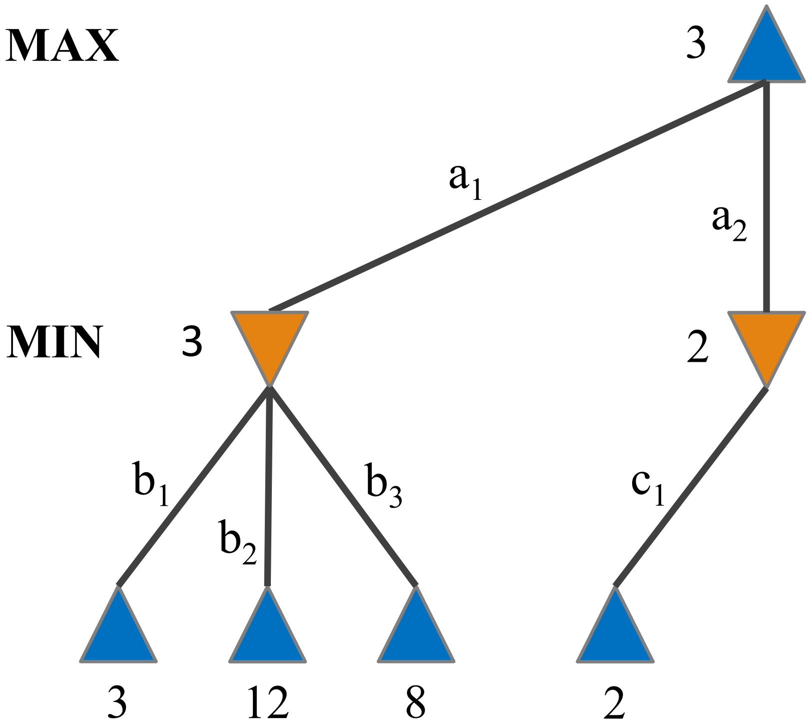
α - β Pruning Example

- Intuitive example



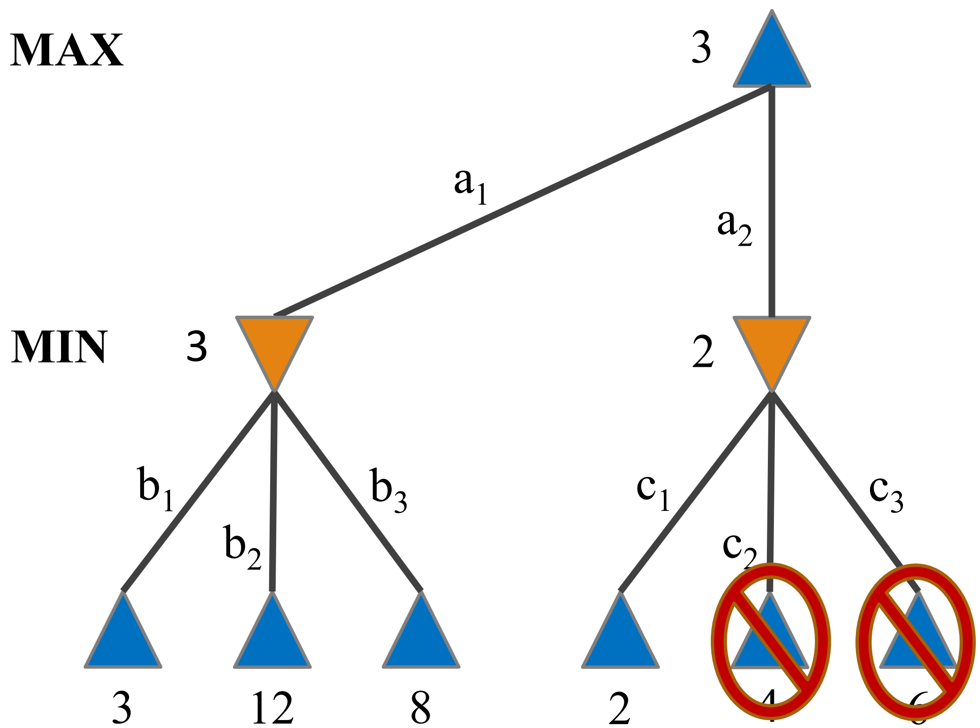
α - β Pruning Example

- Intuitive example



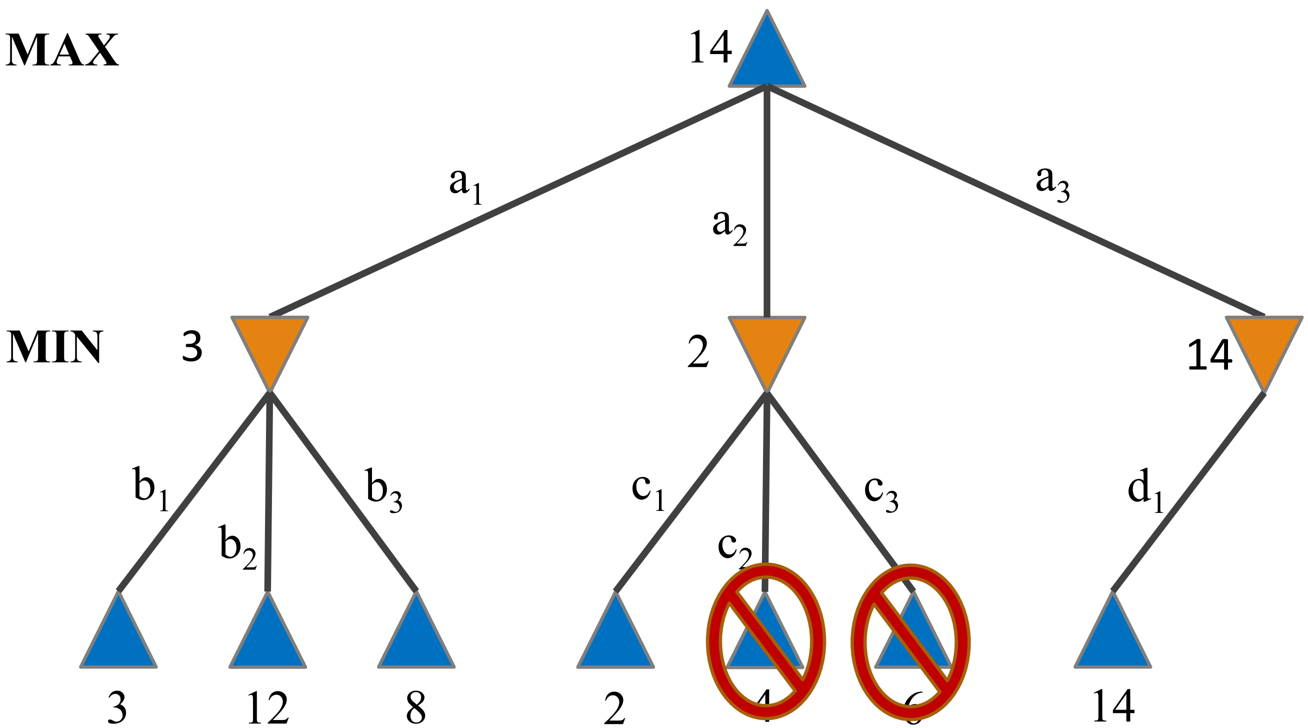
α - β Pruning Example

- Intuitive example



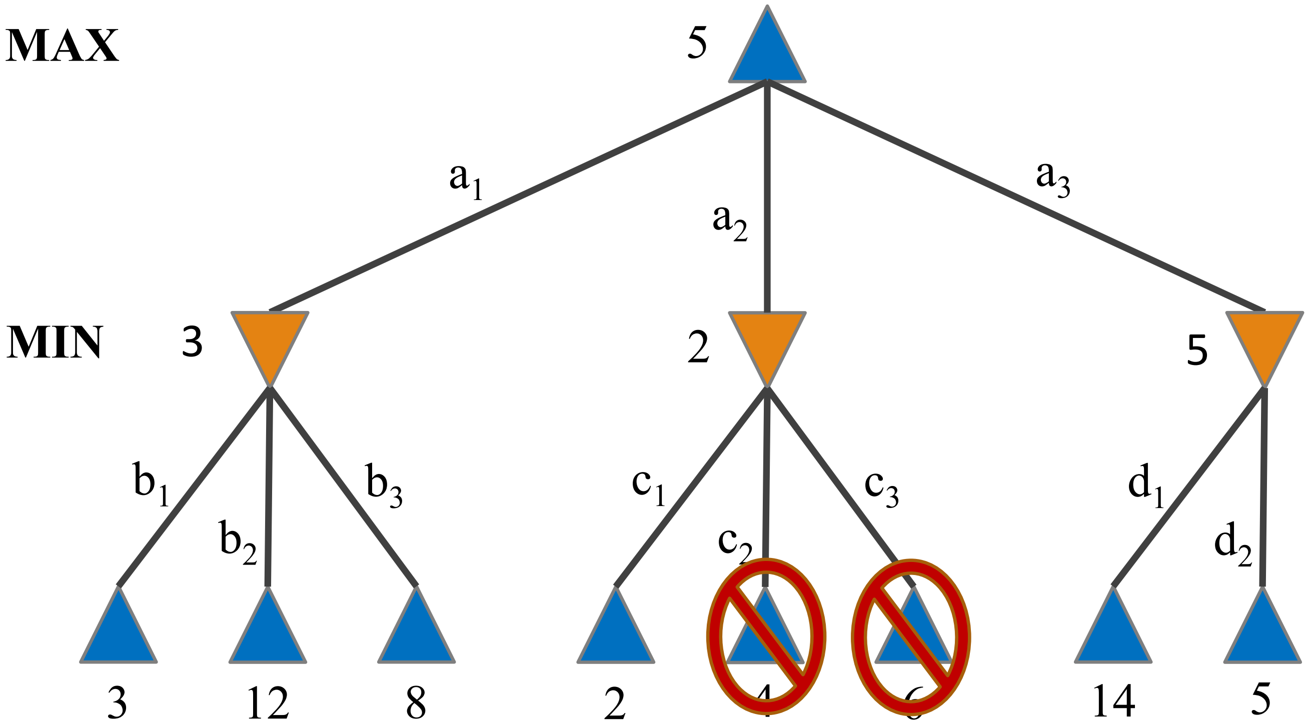
α - β Pruning Example

- Intuitive example



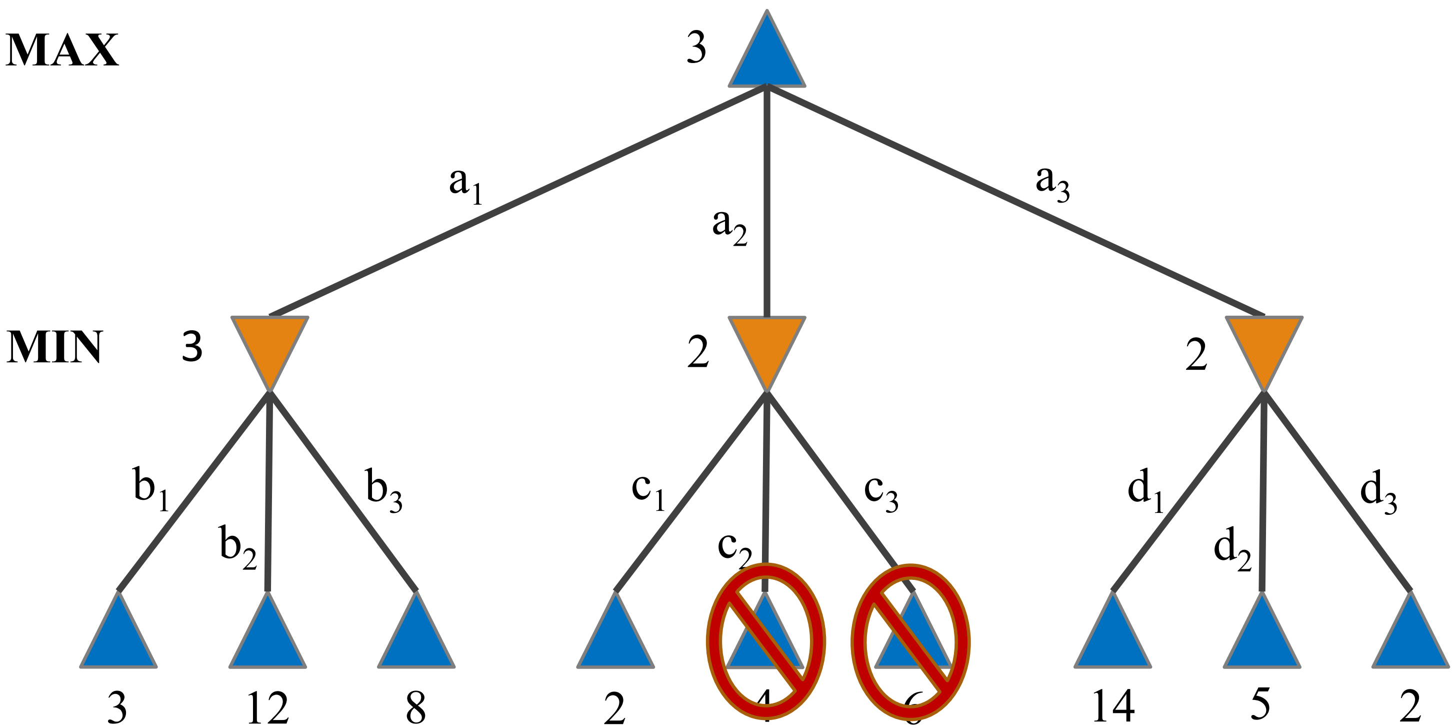
α - β Pruning Example

- Intuitive example



α - β Pruning Example

- Intuitive example



Properties of α - β Pruning

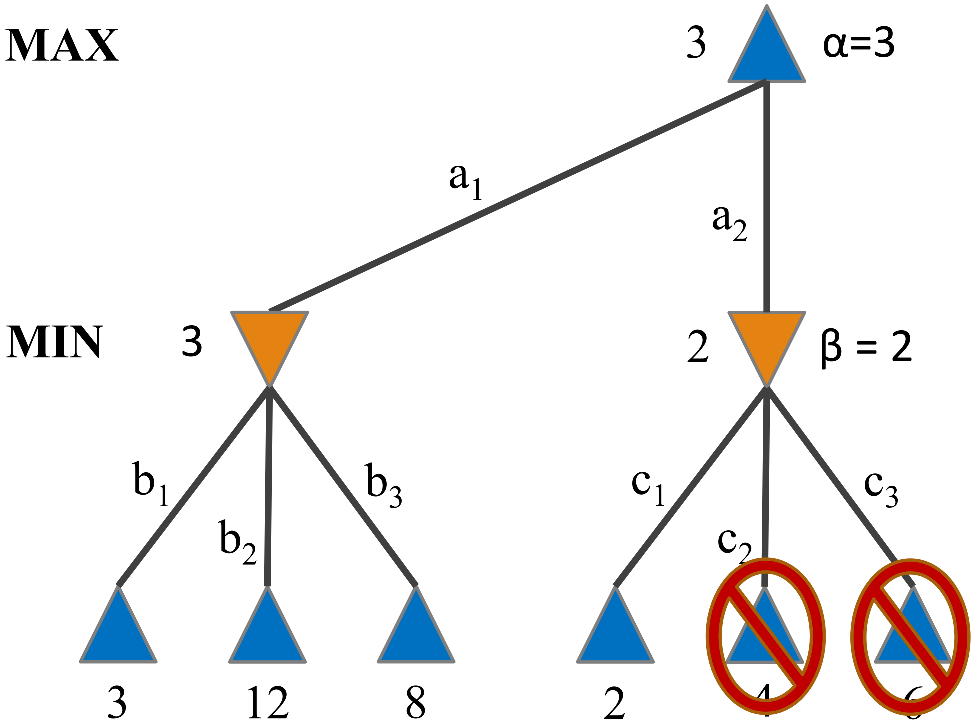
- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering", time complexity is $O(b^{m/2})$
  - ⇒ doubles depth of search
  - ⇒ can easily reach depth 8 and play good chess
- A simple example of the value of reasoning about which computations are relevant (a form of meta-reasoning)

$\alpha - \beta$ Algorithm

```
function MAX-VALUE(state, game, α, β) returns the minimax value of state
    inputs: state, current state in game
            game, game description
            α, the best score for MAX along the path to state
            β, the best score for MIN along the path to state

    if CUTOFF-TEST(state) then return EVAL(state)
    for each s in SUCCESSORS(state) do
        α ← MAX(α, MIN-VALUE(s, game, α, β))
        if α ≥ β then return β
    end
    return α
```

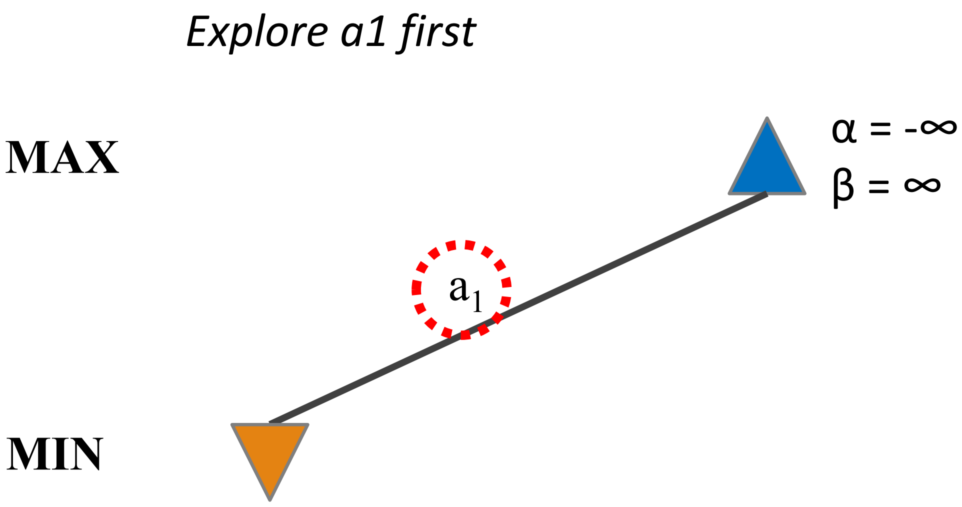
```
function MIN-VALUE(state, game, α, β) returns the minimax value of state

    if CUTOFF-TEST(state) then return EVAL(state)
    for each s in SUCCESSORS(state) do
        β ← MIN(β, MAX-VALUE(s, game, α, β))
        if β ≤ α then return α
    end
    return β
```

α - β Pruning Example



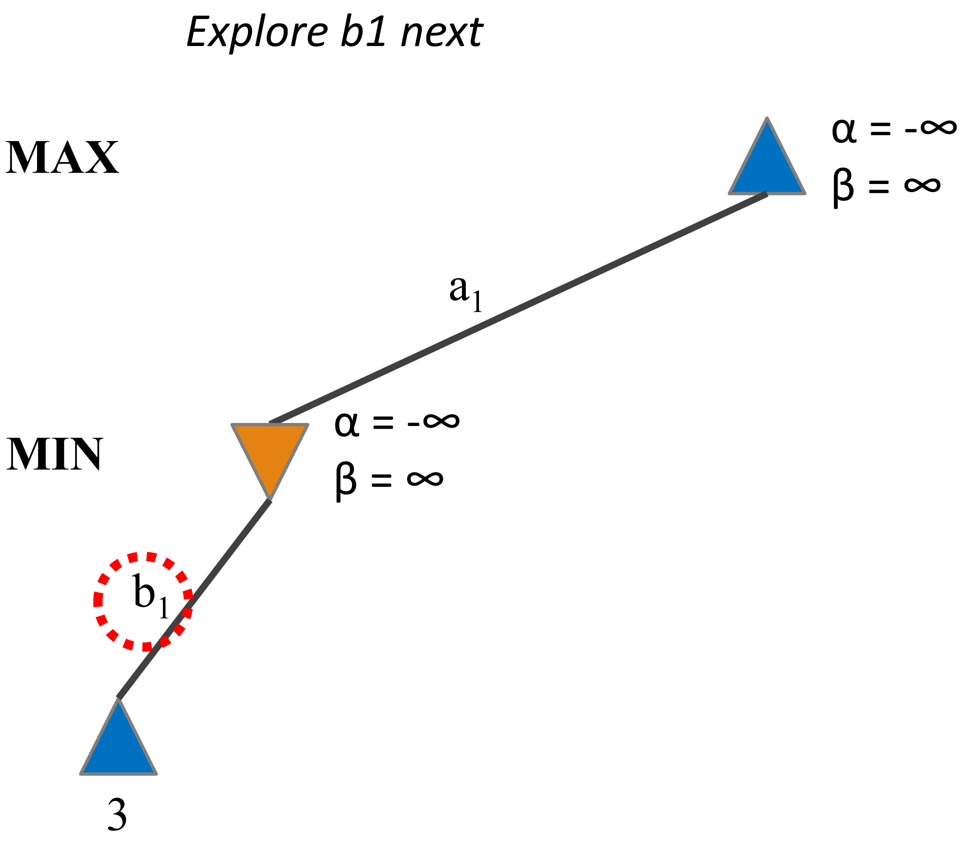
- α is the best value (to MAX) found so far off the current path
- β is the best value (to MIN) found so far off the current path
- Prune if α >= β

α - β Pruning Example



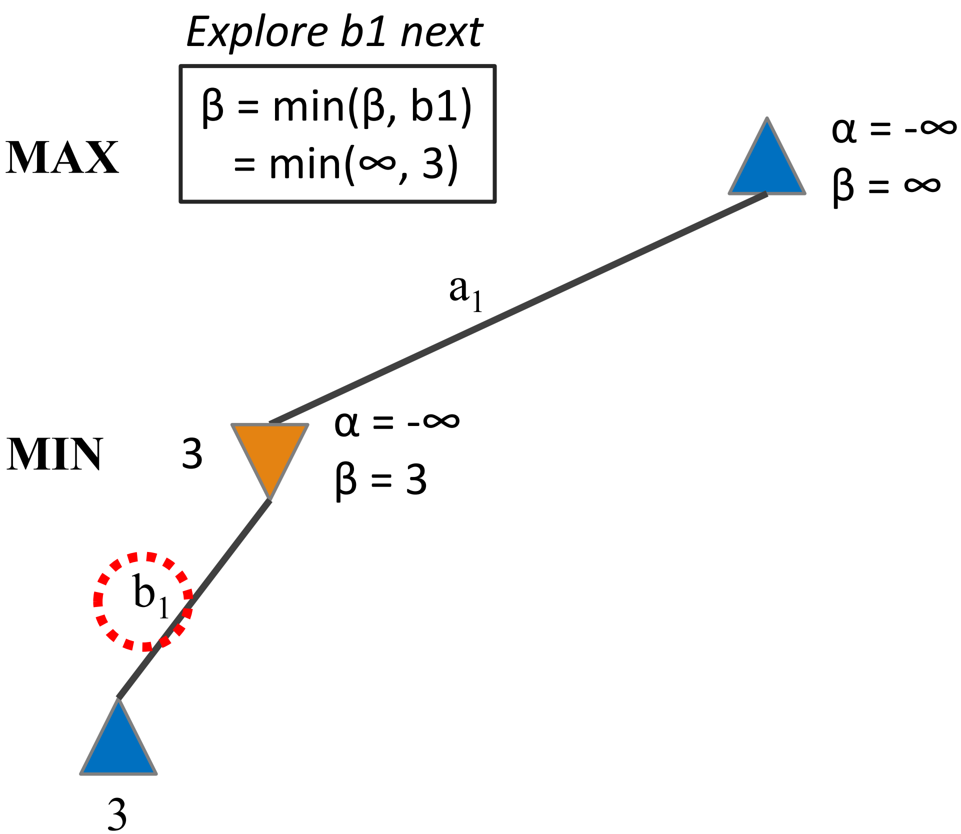
- α is the best value (to MAX) found so far off the current path
- β is the best value (to MIN) found so far off the current path
- Prune if $\alpha >= \beta$

α - β Pruning Example



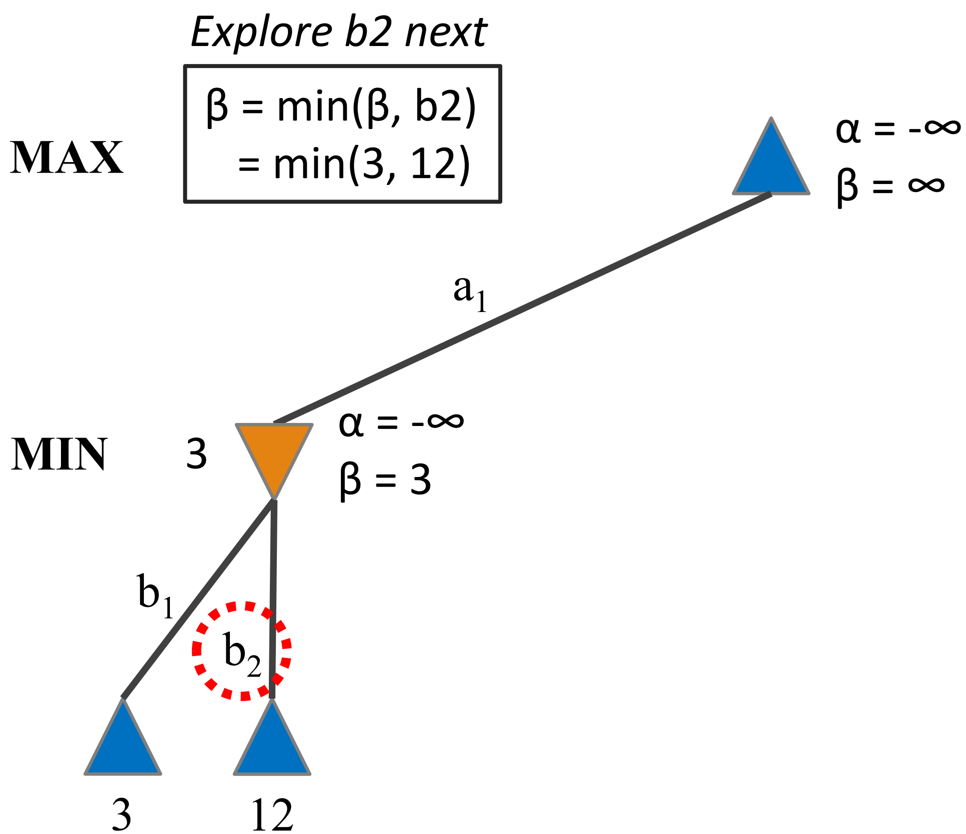
- α is the best value (to MAX) found so far off the current path
- β is the best value (to MIN) found so far off the current path
- Prune if α >= β

α - β Pruning Example



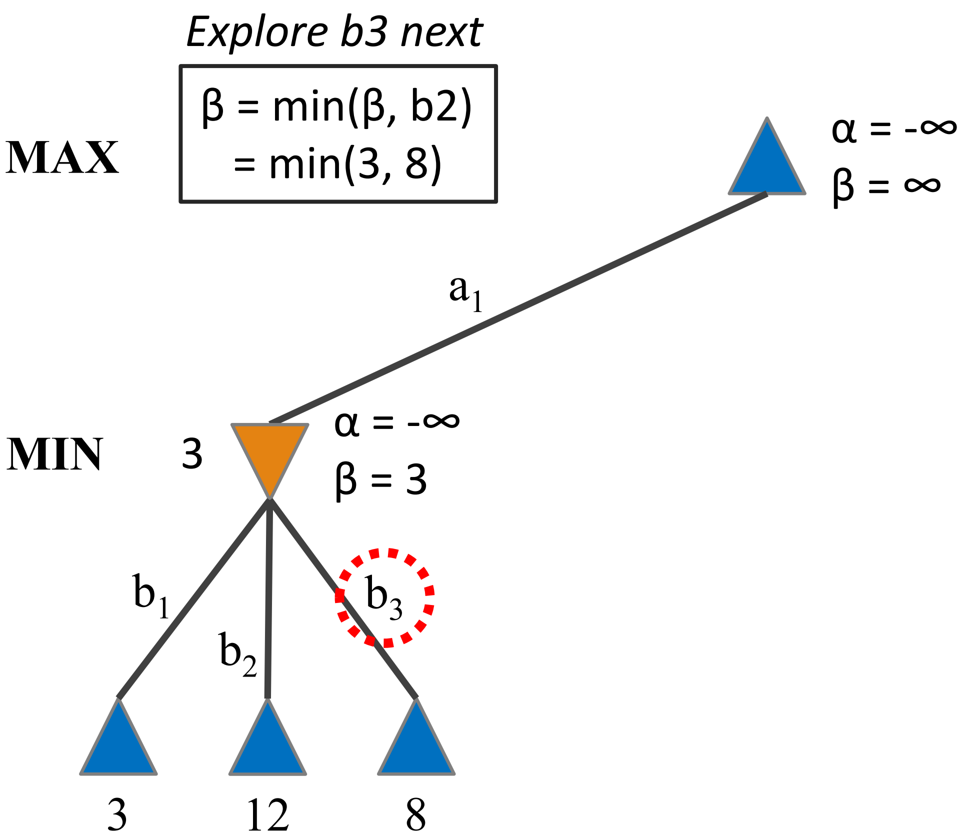
- α is the best value (to MAX) found so far off the current path
- β is the best value (to MIN) found so far off the current path
- Prune if α >= β

α - β Pruning Example



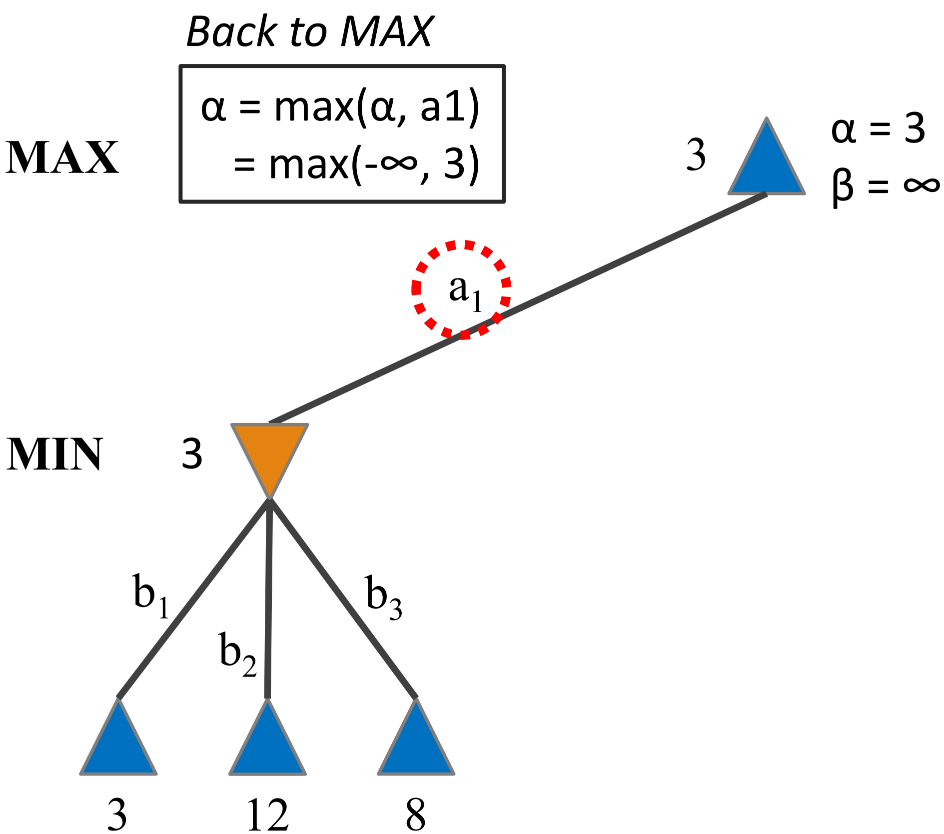
- α is the best value (to MAX) found so far off the current path
- β is the best value (to MIN) found so far off the current path
- Prune if α >= β

α - β Pruning Example



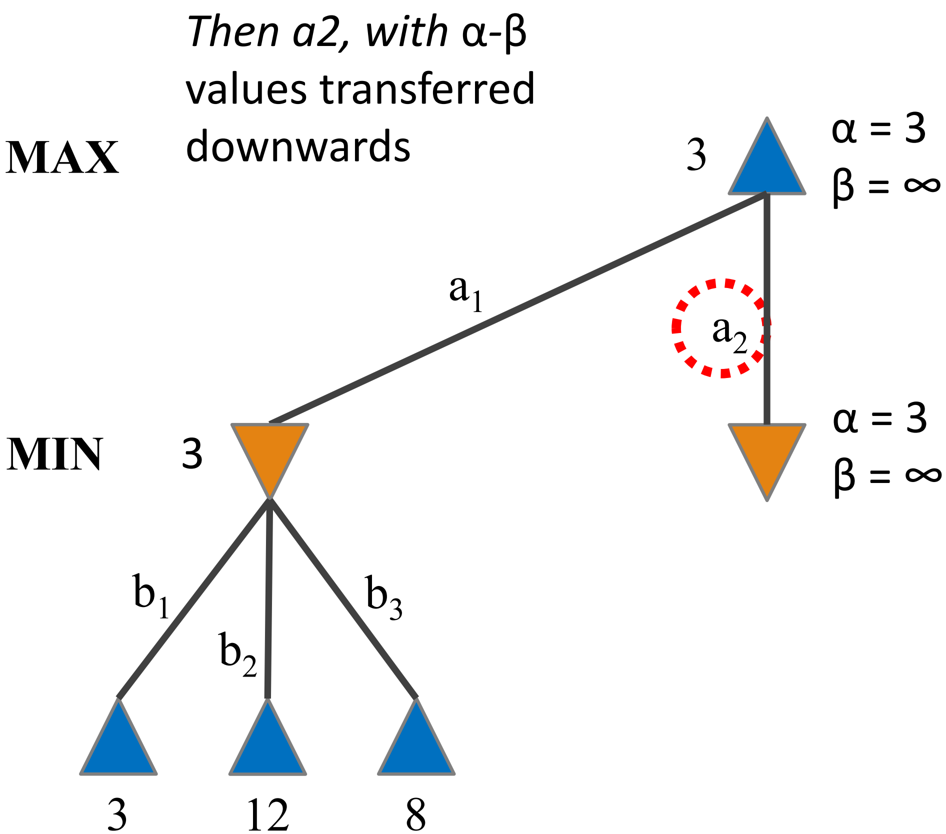
- α is the best value (to MAX) found so far off the current path
- β is the best value (to MIN) found so far off the current path
- Prune if α >= β

α - β Pruning Example



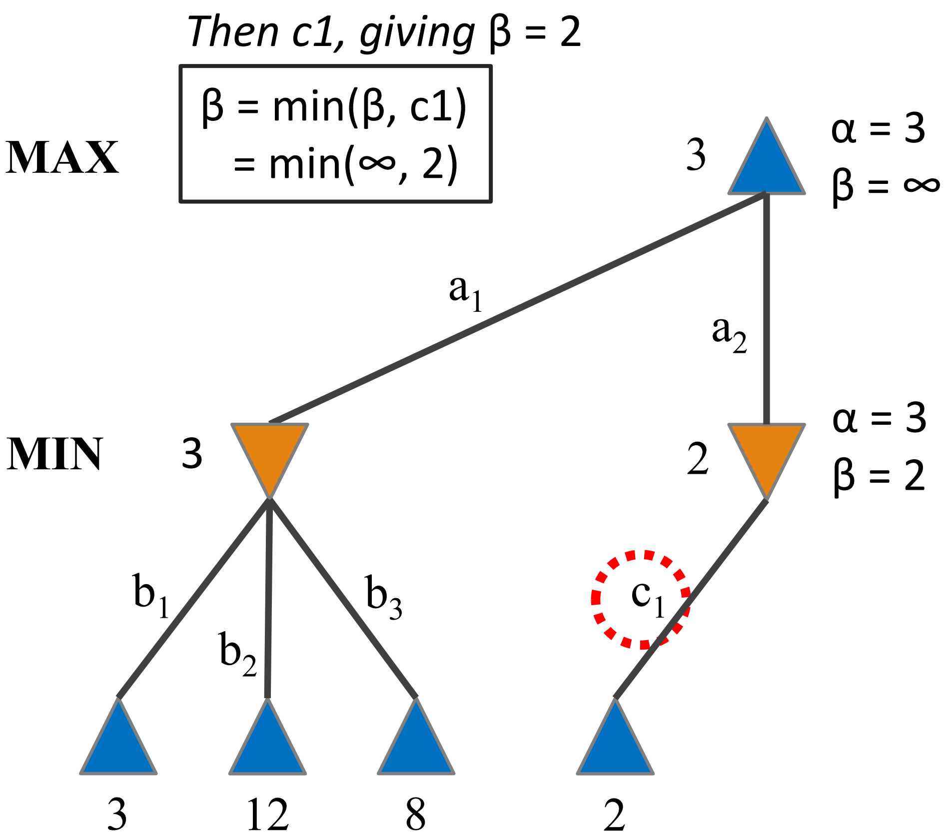
- α is the best value (to MAX) found so far off the current path
- β is the best value (to MIN) found so far off the current path
- Prune if $\alpha >= \beta$

α - β Pruning Example



- α is the best value (to MAX) found so far off the current path
- β is the best value (to MIN) found so far off the current path
- Prune if α >= β

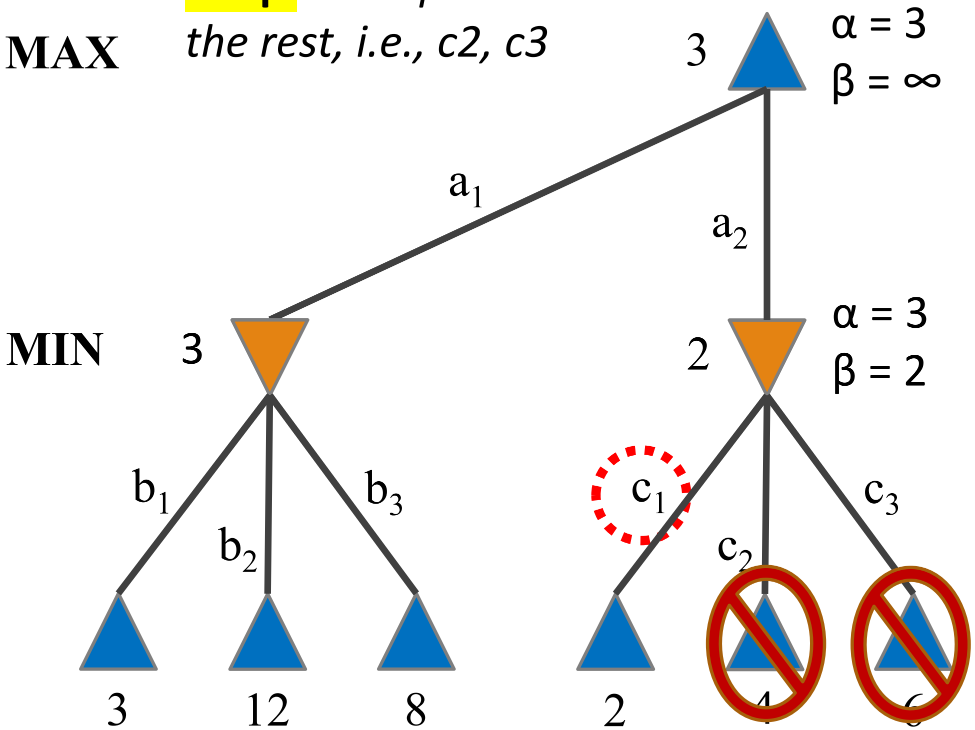
α - β Pruning Example



- α is the best value (to MAX) found so far off the current path
- β is the best value (to MIN) found so far off the current path
- Prune if α >= β

α - β Pruning Example

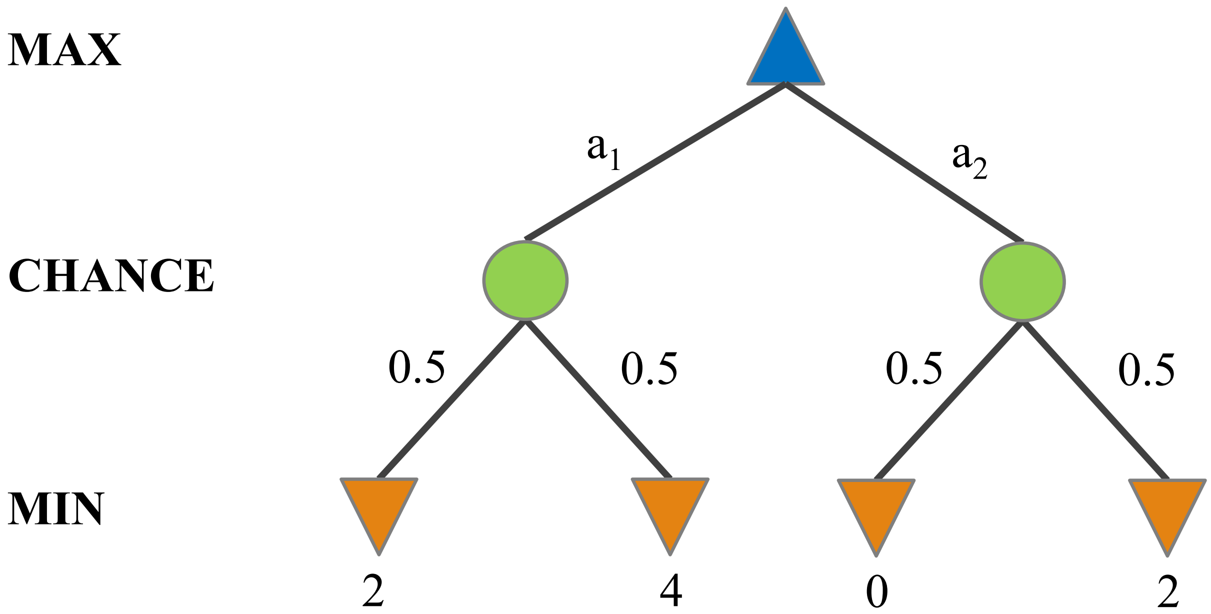
After exploring C1, **α ≥ β** *so we prune the rest, i.e., c2, c3*



- α is the best value (to MAX) found so far off the current path
- β is the best value (to MIN) found so far off the current path
- Prune if α >= β

Non-deterministic Games

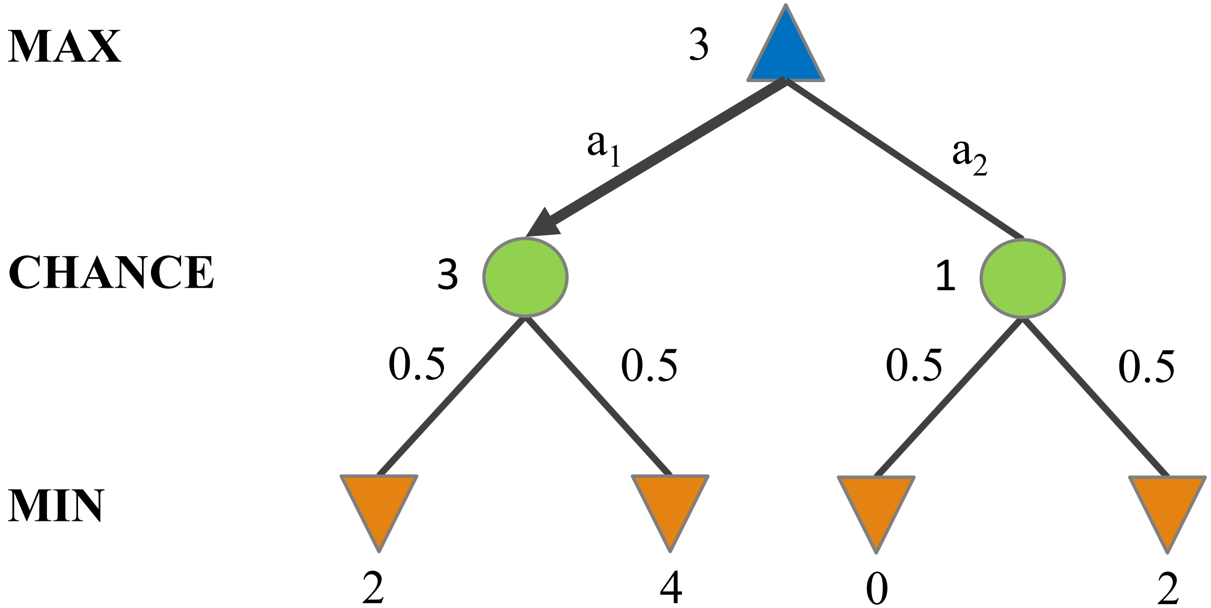
- Adversarial search where the actions/transitions are non-deterministic
  - E.g., in backgammon, the dice rolls determine the legal moves
- Simplified example with coin-flipping instead of dice rolling



Non-deterministic Games

- **ExpectiMiniMax**, similar to MiniMax but accounts for chance nodes

if state is a chance node **then**   **return** weighted average of ExpectiMinimax-Value of Successors(state)



Summary & Objectives

- Understand the differences between standard search problems and adversarial search problems
- Understand the workings behind the Minimax algorithm and α - β pruning
- Able to use Minimax algorithm to solve an adversarial/game search problem
- Able to use α - β pruning to speed up adversarial/game search

Reminders

- AI Quiz 2
  - Time/Venue: Tue 2 Apr 2024, 3.30pm @ LT5
  - Topics Covered: Weeks 5 to 10
  - Format: MCQs and open-ended questions. Completed within 60min
  - 1 x A4 cheatsheet (double-sided) allowed, calculators allowed. Cheatsheet can be printed or handwritten.
  - You will not be asked to produce codes but may be given partial pseudo codes and asked to do something with it, e.g., complete, explain, correct, etc.