Previously…

- Informed search strategies use heuristics to guide search
  - Greedy best-first search
  - A* search
- If $h(n)$ is admissible, then A* using TREE-SEARCH is optimal
- If $h(n)$ is consistent, then A* using GRAPH-SEARCH is optimal
- A heuristic that dominates another incurs lower search cost

A* using GRAPH-SEARCH

UCS: $g(n)$ GBFS: $h(n)$ A*: $f(n) = g(n) + h(n)$

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
      if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              frontier ← INSERT(child, frontier)
          else if child.STATE is in frontier with higher PATH-COST then
              replace that frontier node with child
```

Previously…

- Local search strategies
  - Hill-climbing search: use of heuristic function to improve "current" state
  - Simulated annealing: introduce some "downhill" moves to avoid getting trapped in local maxima
  - Local beam search: keep track of *k* best successor states
  - Genetic algorithms: mimic nature

ADVERSARIAL SEARCH

AIMA Chapter 5.1 – 5.5

Deep Blue vs. Garry Kasparov (1997)

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.
- Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
- 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.

Outline

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

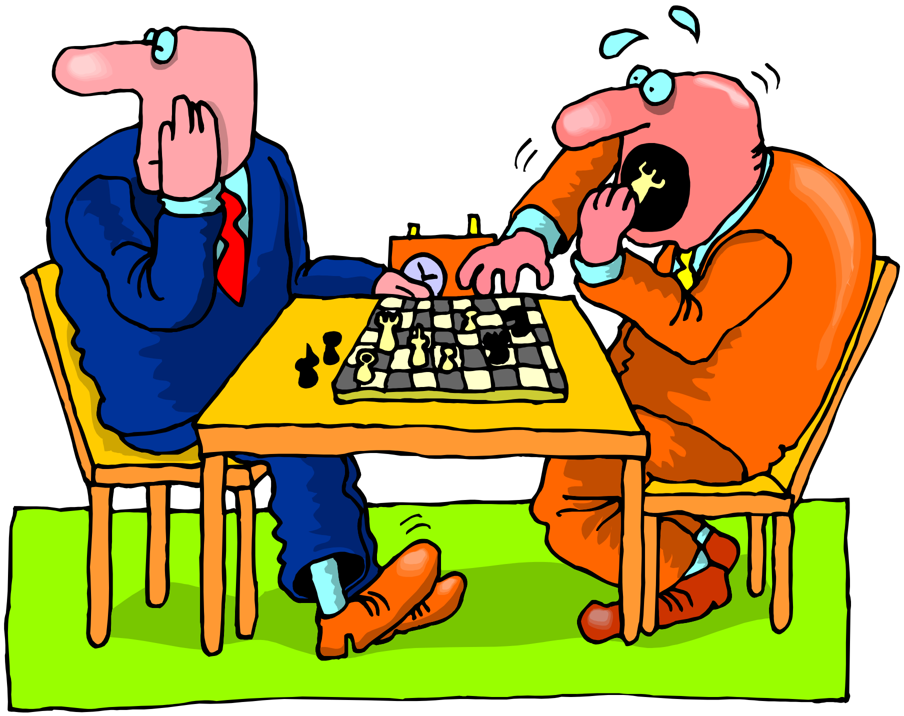
Games vs. Search Problems

- "Unpredictable" opponent → solution is a strategy specifying a move for every possible opponent reply/response
- Time limit → unlikely to find goal, must approximate

Let’s Play!

- Two players:

MAX



MIN

Game: Problem Formulation

A game is defined by 7 components:

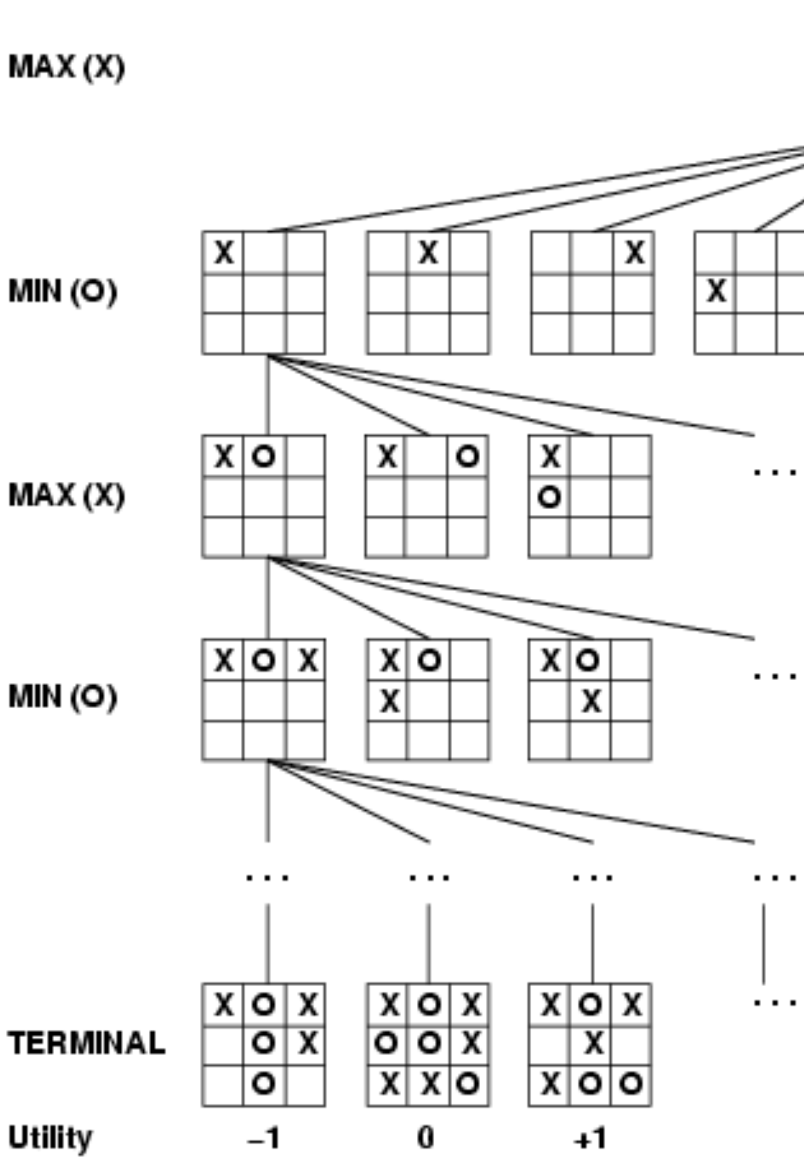
1. Initial state S_0
2. States
3. Players : PLAYER(s) defines which player has the move in state s
4. Actions : ACTIONS(s) returns set of legal moves in state s
5. Transition model : RESULT(s, a) returns state that results from the move a in state s

Game: Problem Formulation

A game is defined by 7 components:

6. Terminal test TERMINAL-TEST(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
   - Example: for chess, win +1, loss -1, draw 0

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



Key property: **zero-sum** game

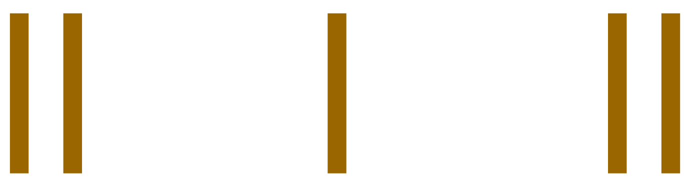
- Loosely speaking, it means that there's a loser for every winner
- Total utility score over all agents sum to zero (or constant)
- Aka constant-sum game
- Makes the game adversarial

To think about: not zero sum?

Example : Game of NIM

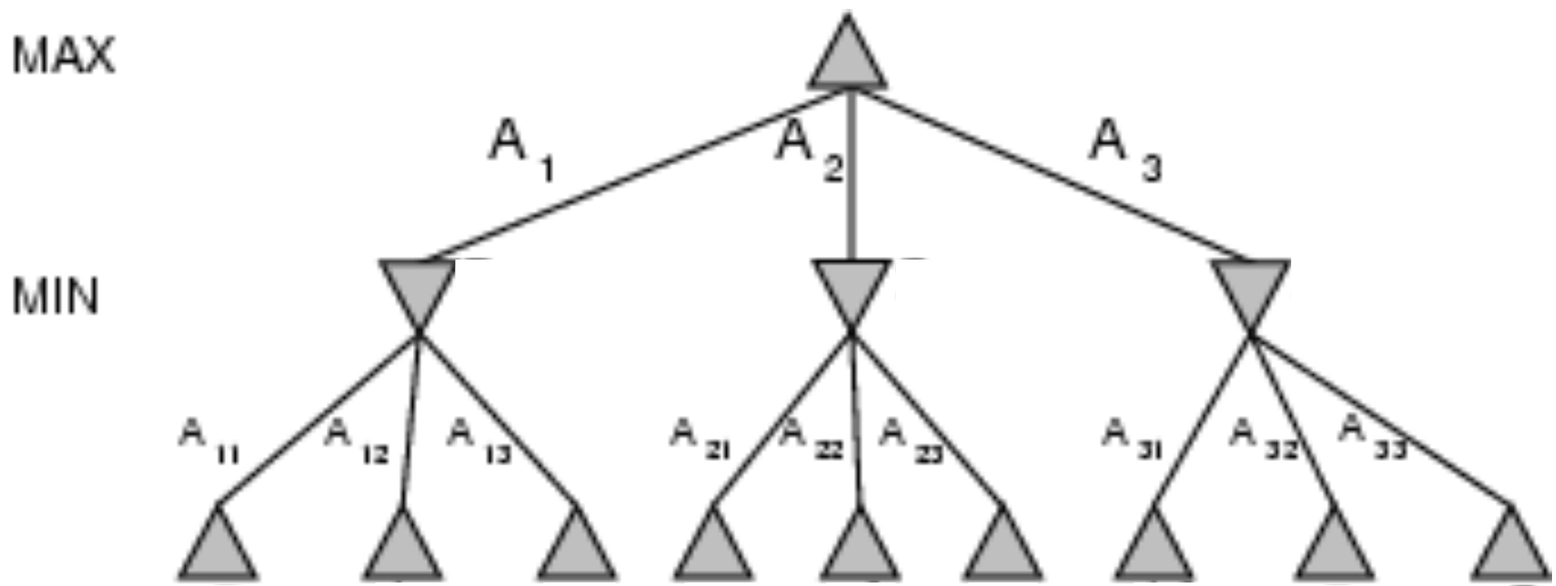
Several piles of sticks are given. We represent the configuration of the piles by a monotone sequence of integers, such as (1,3,5). A player may remove, in one turn, any number of sticks from one pile. Thus, (1,3,5) would become (1,1,3) if the player were to remove 4 sticks from the last pile. The player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

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



Minimax

- Optimal play for deterministic games
- Idea: choose move to state with highest minimax value
  = best achievable payoff (for MAX) against MIN's optimal play
- E.g., 2-ply game:



Minimax Value

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

Intuitively,

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

Minimax Algorithm

- Mistake in textbook (not reflected in errata): $s = state$

```
function MINIMAX-DECISION(state) returns an action
   return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for each a in ACTIONS(state) do
      v ← MAX(v, MIN-VALUE(RESULT(s, a)))
   return v

function MIN-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← ∞
   for each a in ACTIONS(state) do
      v ← MIN(v, MAX-VALUE(RESULT(s, a)))
   return v
```

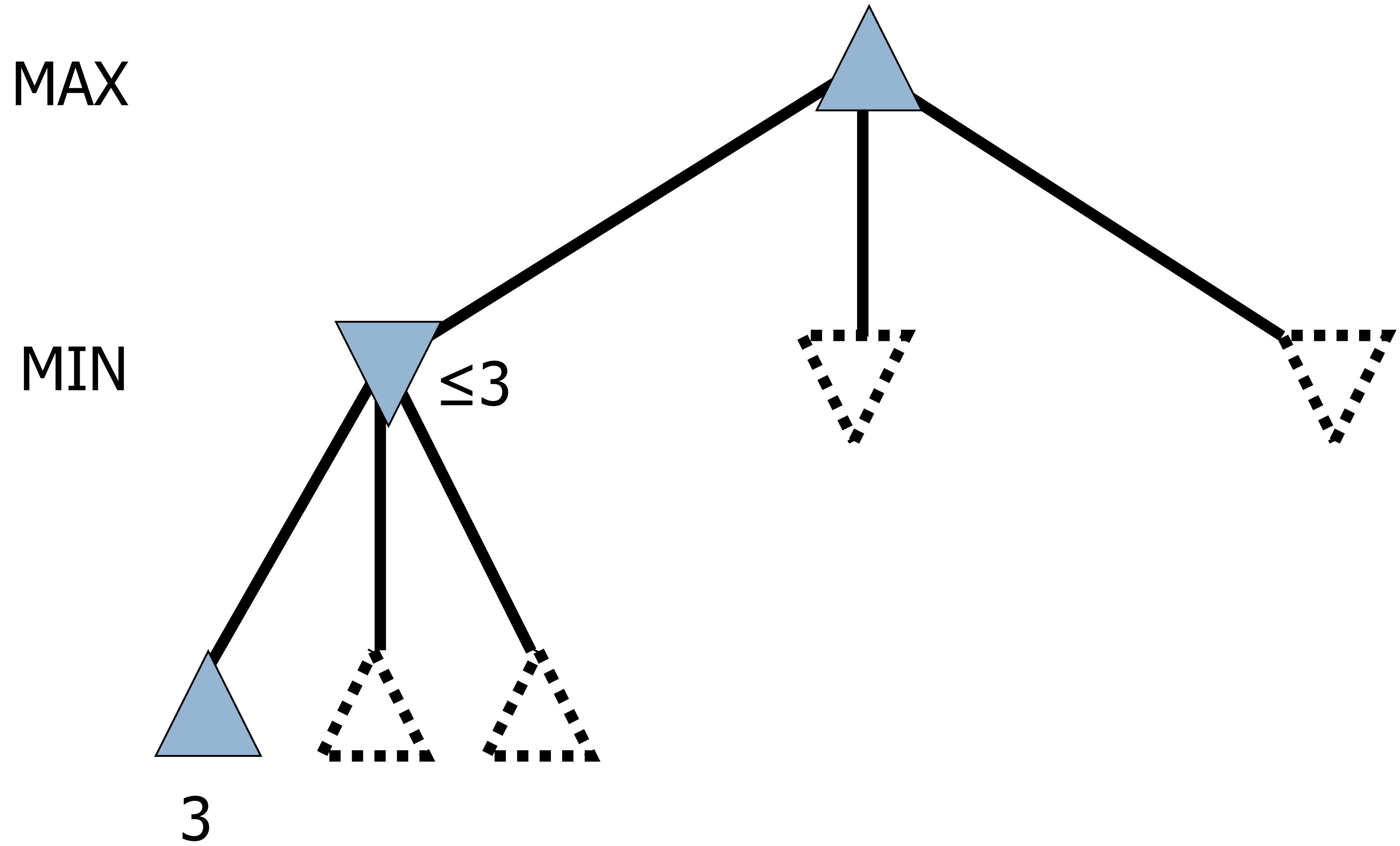
Properties of Minimax

- 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 → time cost to find exact solution impractical
- Do we need to explore every path? Pruning!

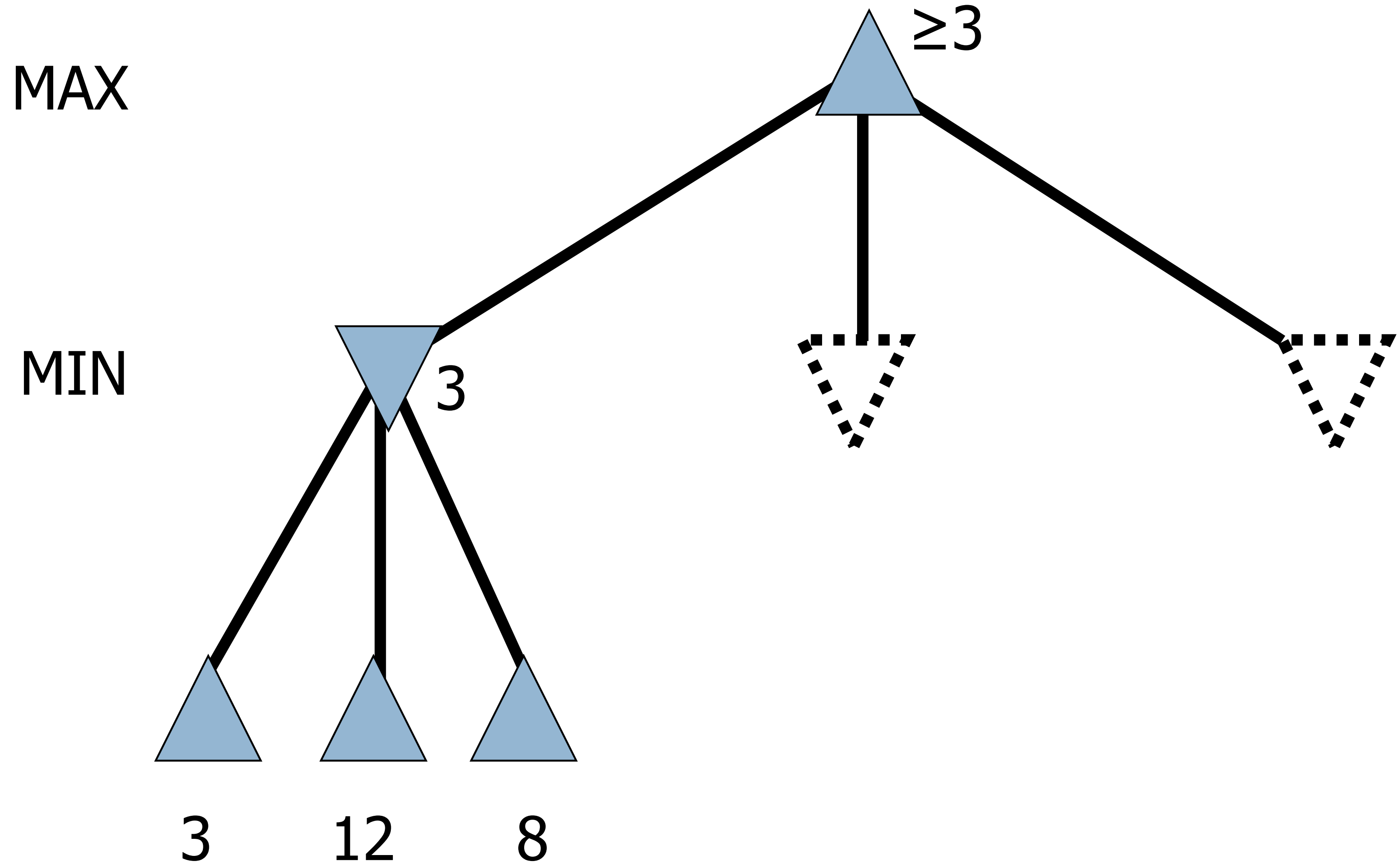
α-β Pruning

- Idea: If we maintain the lower bound α and upper bound β of the values of, respectively, MAX's and MIN's nodes seen thus far, then we can prune subtrees that will never affect minimax decision.

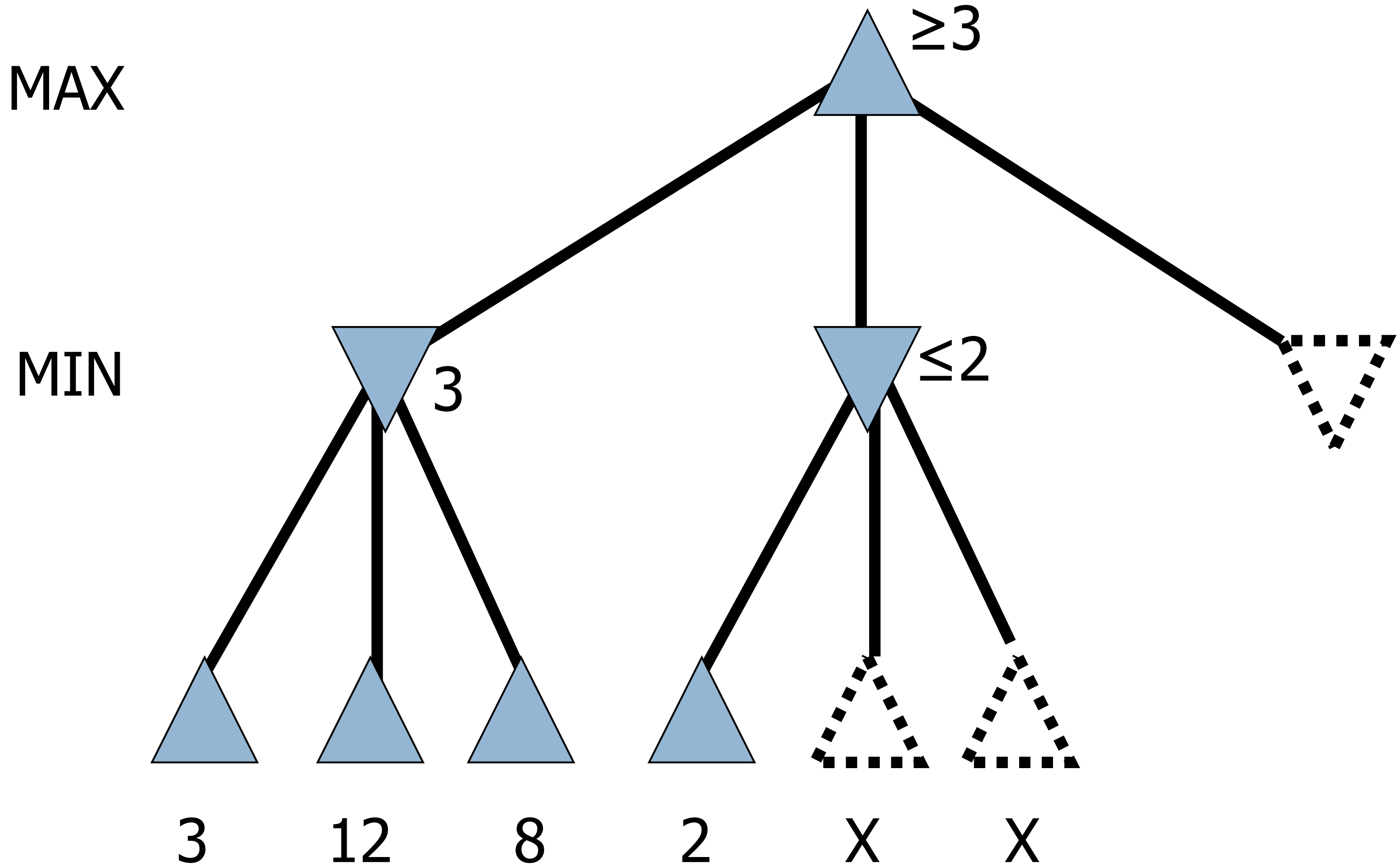
α-β Pruning Example



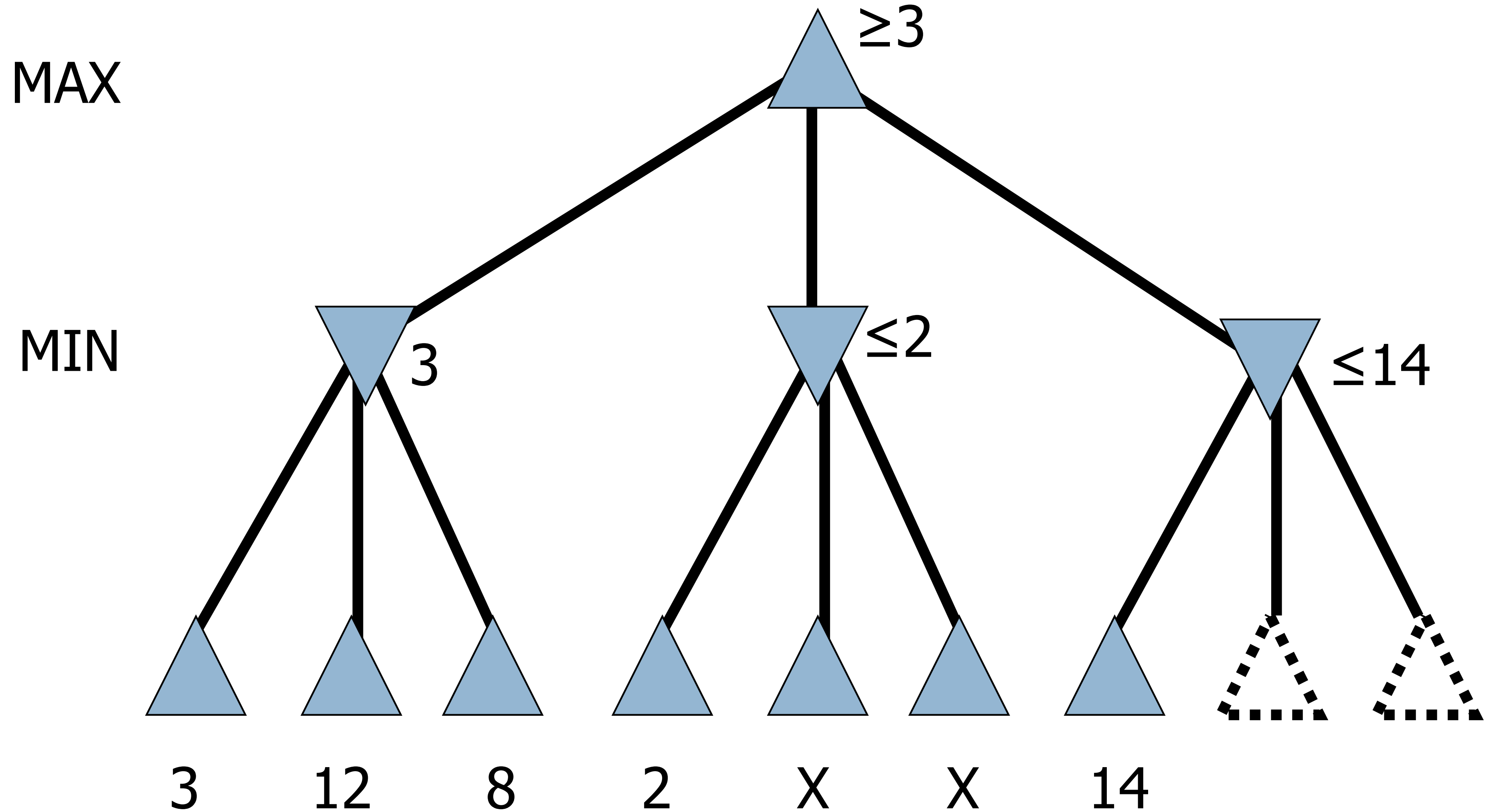
α-β Pruning Example



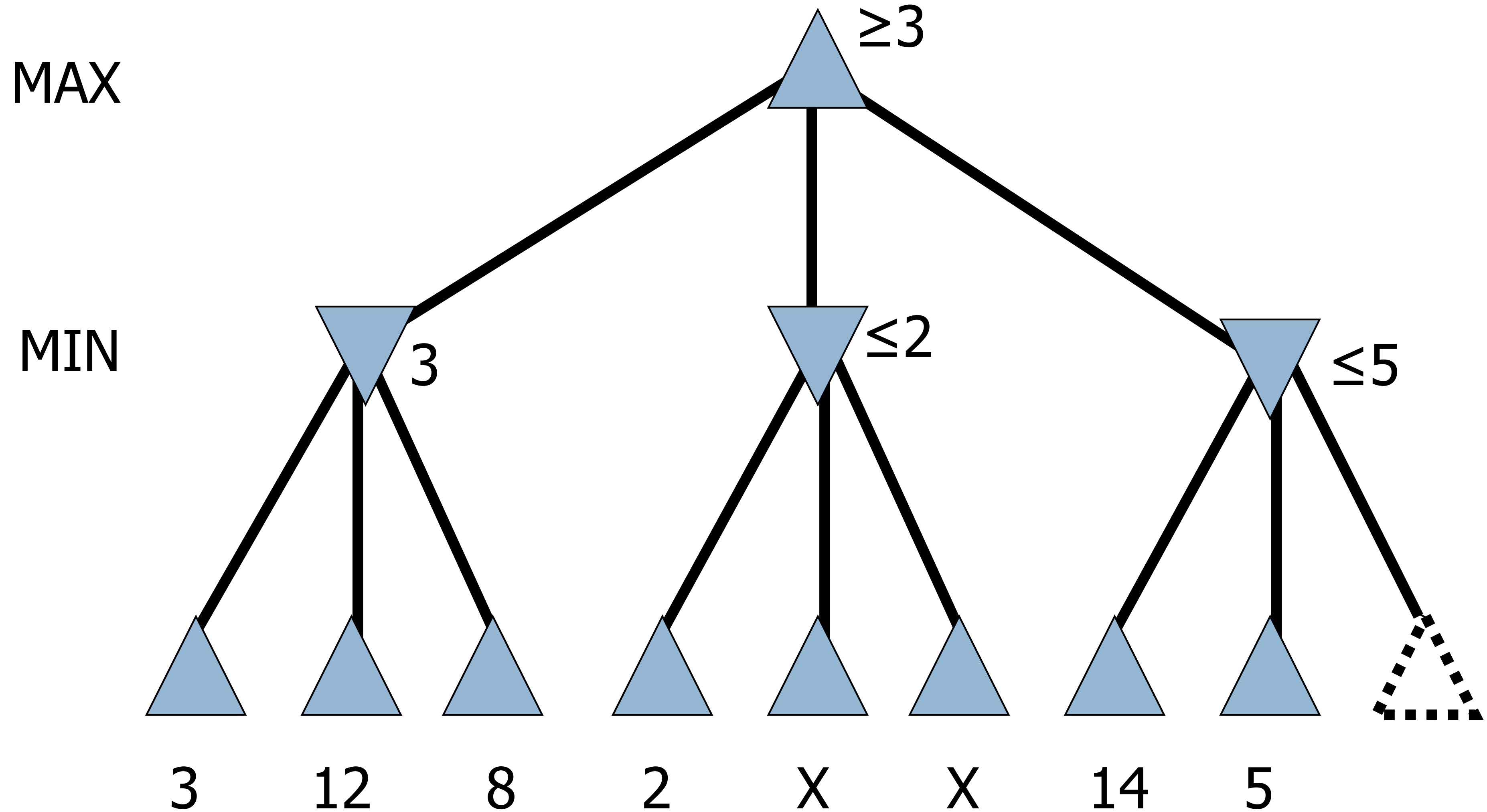
α-β Pruning Example



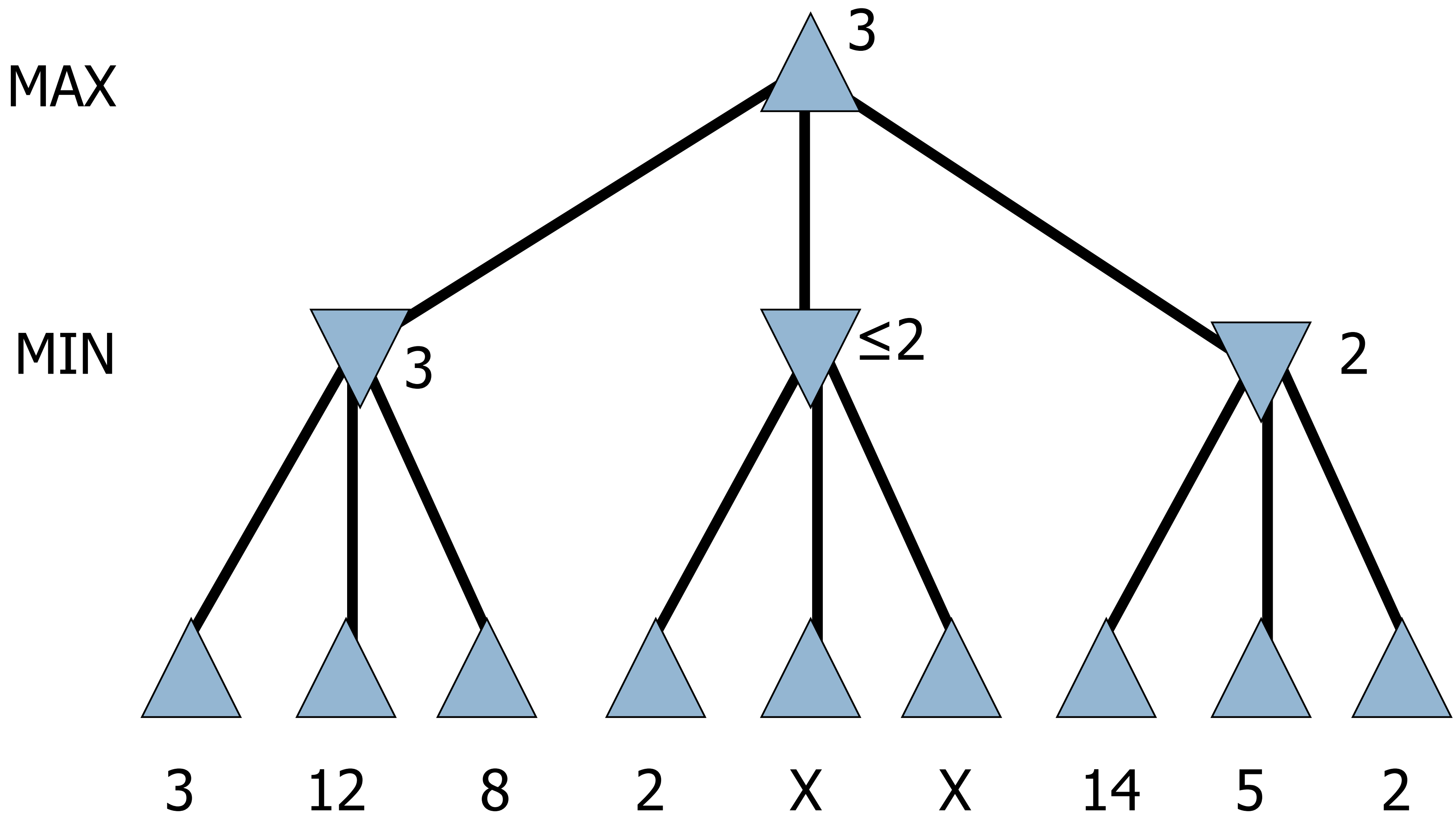
α-β Pruning Example



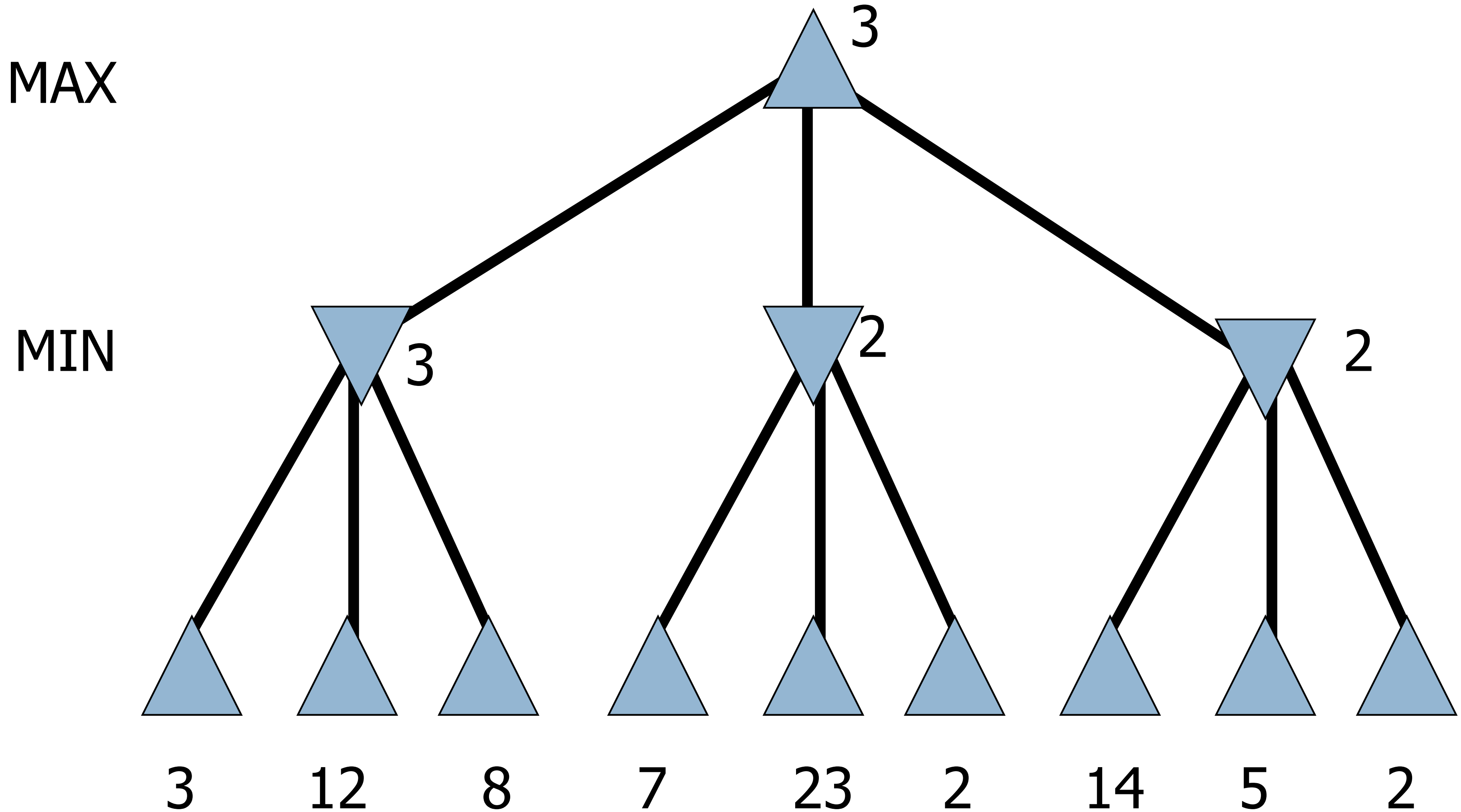
α-β Pruning Example



α-β Pruning Example



α-β Pruning Example: Cannot Prune!

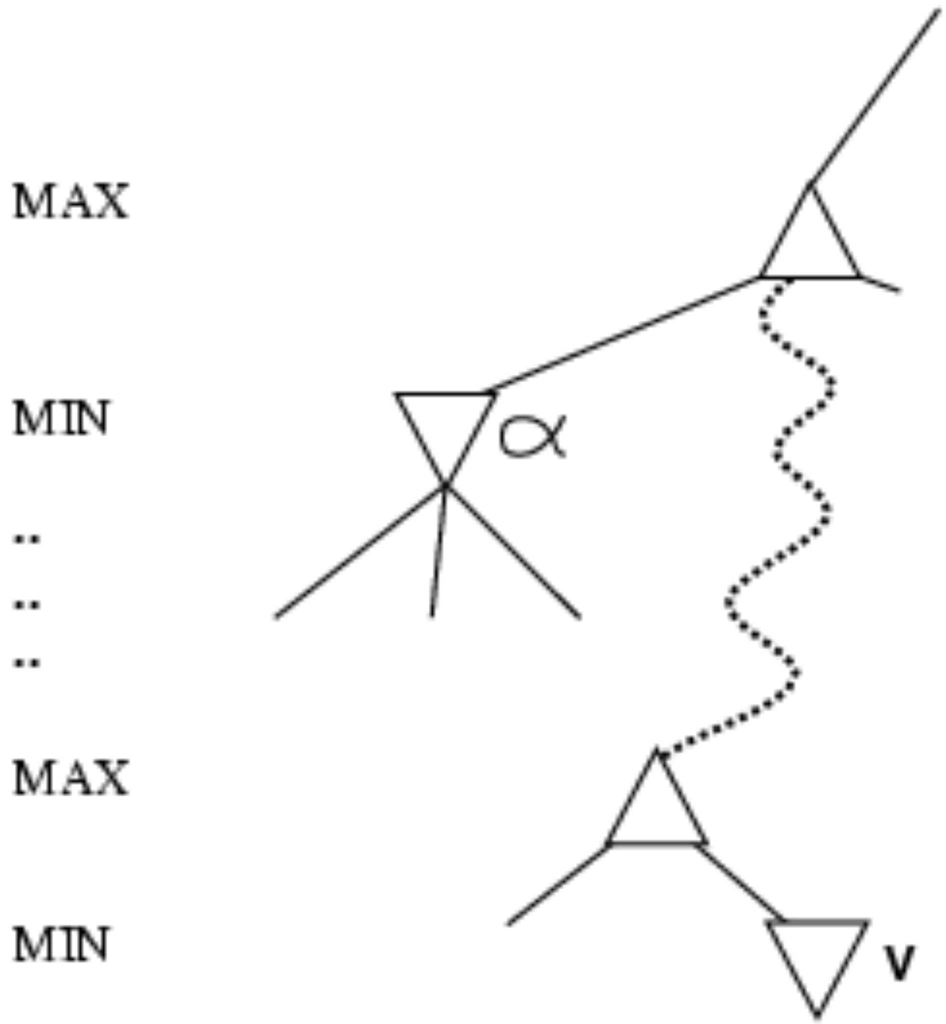


Properties of α-β Pruning

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- "Perfect" ordering: time complexity = $O(b^{m/2})$
  - doubles depth of search
  - E.g., for chess, simple ordering gets you close to this best-case result
  - Does it make sense then to have good heuristics for which nodes to expand first?
- Random ordering: time complexity = $O(b^{3m/4})$ for moderate b

Why is it called α-β Pruning?

- α = 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, then MAX will avoid it
  → prune remaining branches at node with v
- Define β similarly for MIN



The α-β Pruning Algorithm


```
function ALPHA-BETA-SEARCH(state) returns an action
   v ← MAX-VALUE(state, −∞, +∞)
   return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for each a in ACTIONS(state) do
      v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
      if v ≥ β then return v
      α ← MAX(α, v)
   return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← +∞
   for each a in ACTIONS(state) do
      v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
      if v ≤ α then return v
      β ← MIN(β, v)
   return v
```

Summary: α-β Pruning Algorithm

- Initially, $\alpha = -\infty$, $\beta = +\infty$
- α is max along search path
- β is min along search path
- At MIN node, can stop if we find a node with value v smaller than or equal to α
- At MAX node, can stop if we find a node with value v larger than or equal to β

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

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

Heuristic Minimax Value

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

$$\text{H-}\textsc{Minimax}(s, d) = \begin{cases} \textsc{Eval}(s) & \text{if } \textsc{Cutoff-Test}(s, d) \\ \max_{a \in \textsc{Actions}(s)} \text{H-}\textsc{Minimax}(\textsc{Result}(s, a), d + 1) & \text{if } \textsc{Player}(s) = \text{MAX} \\ \min_{a \in \textsc{Actions}(s)} \text{H-}\textsc{Minimax}(\textsc{Result}(s, a), d + 1) & \text{if } \textsc{Player}(s) = \text{MIN} \end{cases}$$

Designing Good Evaluation Functions

- Order all the terminal states in the same way as the true utility function
- Fast to compute
- For non-terminal states, must be strongly correlated with actual chances of winning

Evaluation Functions

- For chess, typically linear weighted sum of features

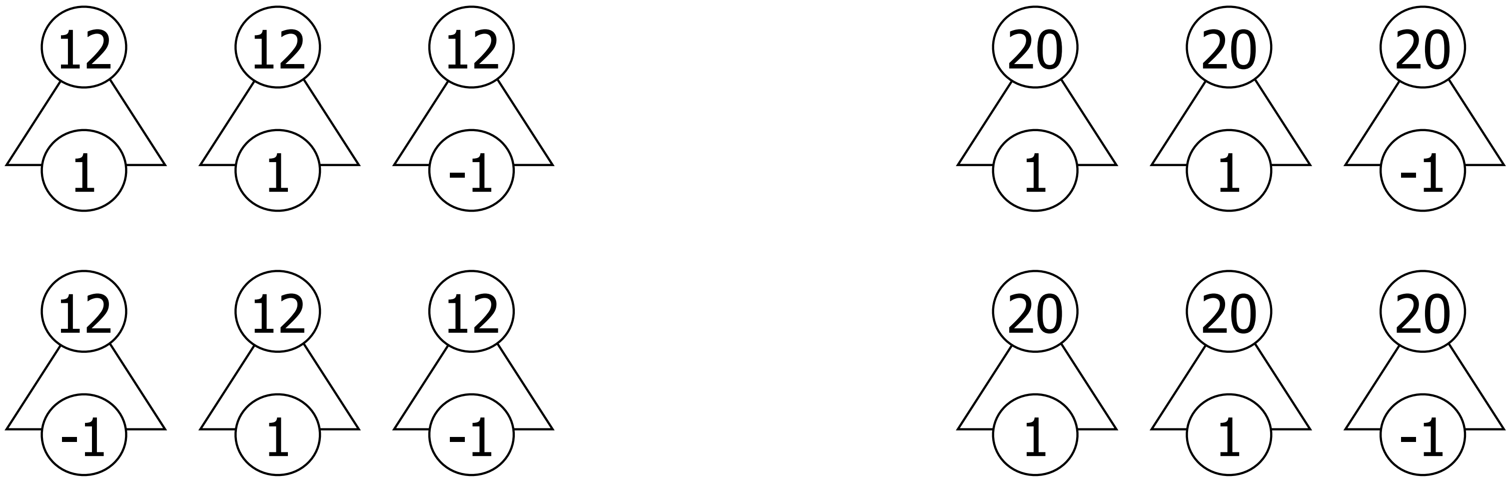
$$Eval(s) = w_1\, f_1(s) + w_2\, f_2(s) + \ldots + w_n\, f_n(s)$$

- e.g., $w_1 = 9$ with

  $f_1(s)$ = (number of white queens) – (number of black queens), etc.

- Caveat: Contribution of each feature is independent of other feature values
  - Bishops in chess better at endgame
- Should model the *expected utility value* over states with the same feature values

Expected Utility Value



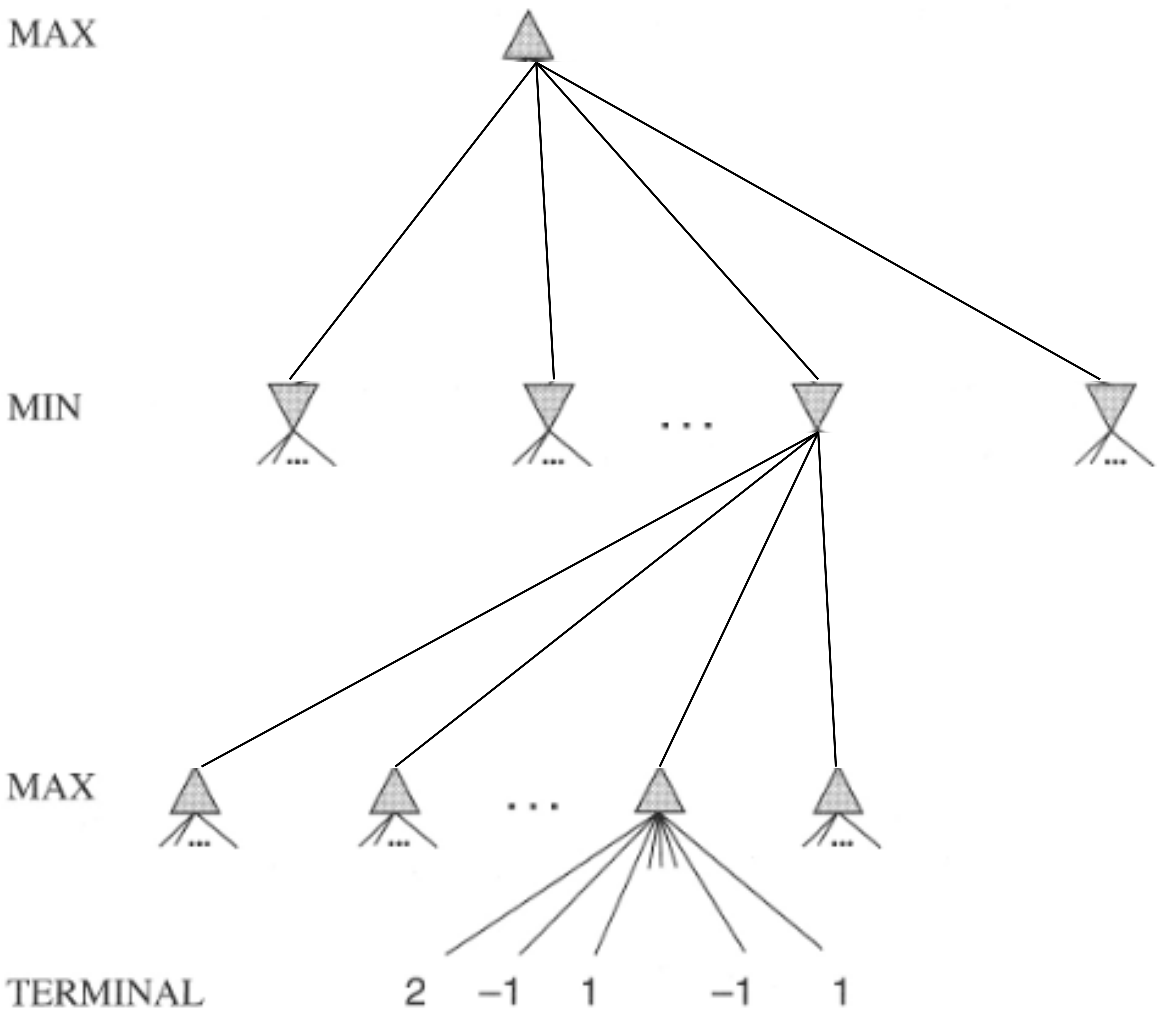
- An expected utility value may map to many states, each of which may lead to different terminal states
- Want expected utility values to model likelihood of “better-utility” states
- Evaluation function need not return actual expected values as long as ordering of states is the same

Stochastic Games

- What about an element of chance?
- How do we deal with uncertainty?

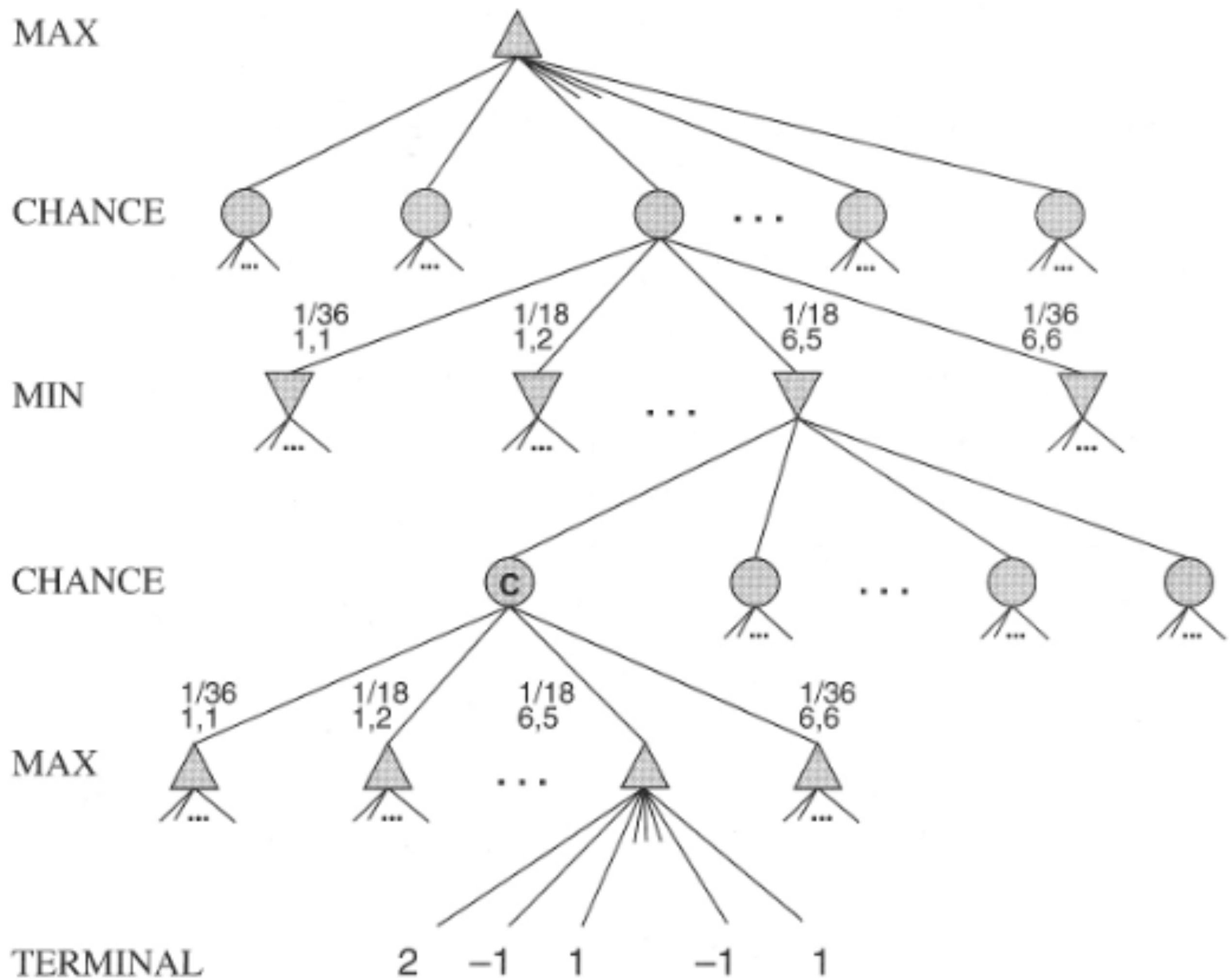
- Can we still use the minimax idea?

Adding Chance Layers



Calculate the expected value of a state

Adding Chance Layers



EXPECTIMINIMAX: Calculate the expected value of a state

Stochastic Games in Real World