Chapter 5:
Adversarial Agents

CS-4365 Artificial Intelligence

Chris Irwin Davis, Ph.D.

Email: chrisirwindavis@utdallas.edu
Phone: (972) 883-3574
Office: ECSS 4.603

Overview

- Games
- Optimal Decisions in Games
- Imperfect Real-time Decisions
- Stochastic Games
- Partially Observable Games
- State-of-the-art Game Programs
- Alternate Approaches

Games

Games

- Competitive environments
  - Agents' goals are in conflict (i.e. mutually exclusive goals)
  - Adversarial search problems (games)
- Most common games in AI are a special subclass
  - which are:
    - deterministic, turn-taking, two-player, zero-sum, perfect information
  - e.g. Chess
  - Utility values are equal and opposite
  - If one person wins, the other necessarily loses

Game Complexity

- Games (unlike toy problems) are interesting because they are hard to solve
- Chess
  - Average branching factor = 35
  - Often 50 moves by each player
  - Thus, search tree has 35^{100} (i.e. 10^{154}) nodes
    - "Shannon Number" – named for MIT mathematician Claude Shannon[1]

1. Claude Shannon (1950). "Programming a Computer for Playing Chess". Philosophical Magazine #41 (p.314).

Games

- Optimal move
- Pruning
- Evaluation functions
- Games with elements of chance
- Imperfect Information

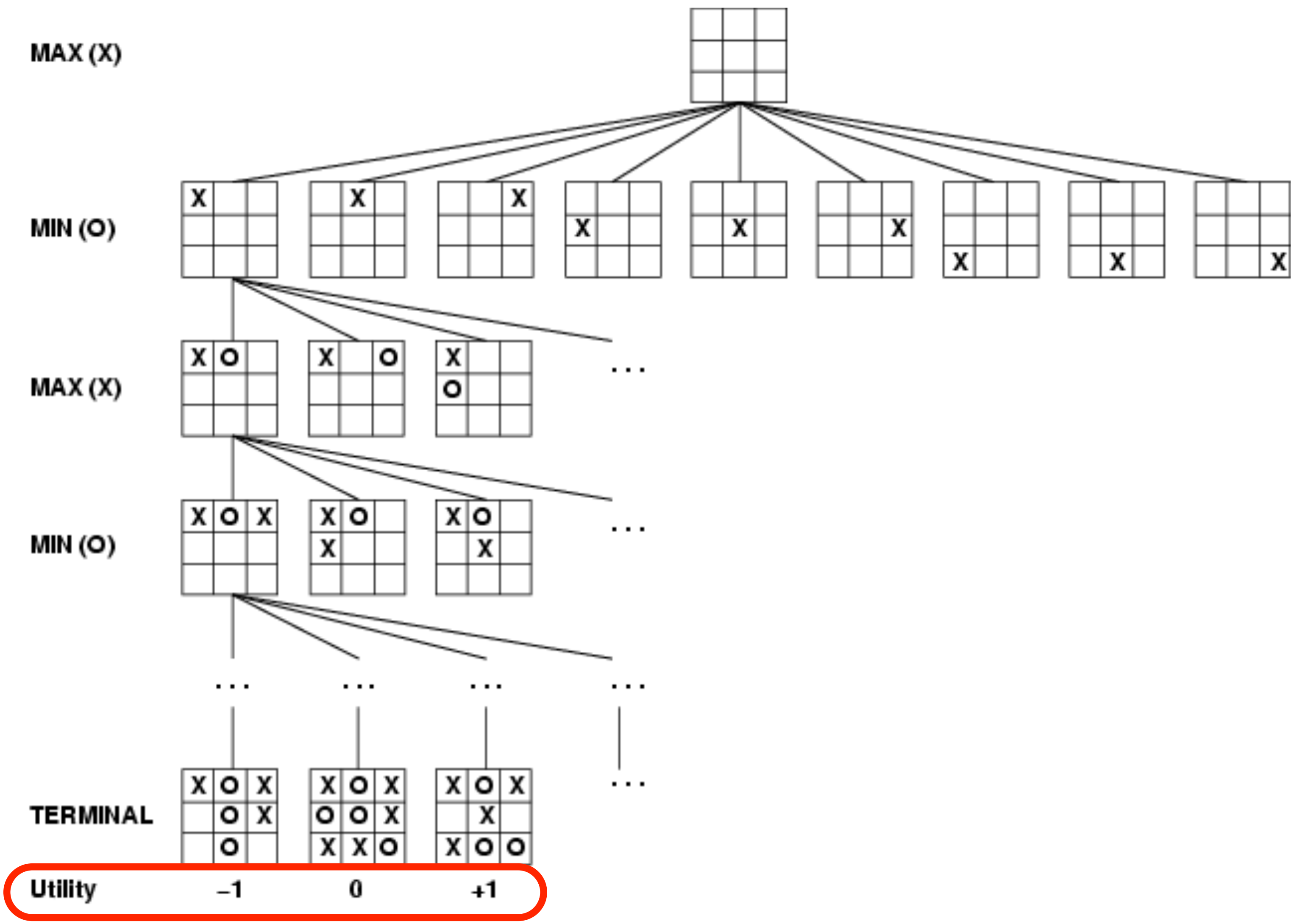
Games (Formal Definition)

- **S_0**: The initial state (how the game is set up at the start)
- **Player(*s*)**: Defines which player has the move in a state
- **Actions(*s*)**: Returns the set of legal moves in a state
- **Result(*s*, *a*)**: The **transition model**, which defines the result of a move
- **Terminal-Test(*s*)**: A **terminal test** – true when the game is over (i.e. in a **terminal state**) and false otherwise
- **Utility(*s*, *p*)**: Defines the final numeric value for a game that ends in a terminal state *s* for player *p*
  - A **zero-sum** game – the *total payoff* to all players is the same for every instance of the game

Games (Game Tree)

- Game tree (vs. graph)
- Nodes are game states and edges are moves
- Initially *x* possible moves (i.e. “max” player)
  - Tic-Tac-Toe has 9 possible initial moves
  - Checkers?
  - Chess?
- Turn taking
  - Alternating min-max

Tic-Tac-Toe Game Tree



Tic-Tac-Toe Analysis

- Fewer than 9! = 362,880 possible games
- 255,168 terminal nodes (possible games)
  - 131,184 finished games are won by X
  - 77,904 finished games are won by O
  - 46,080 finished games are drawn
- With rotations
  - 26,830 games

Optimal Decisions in Games

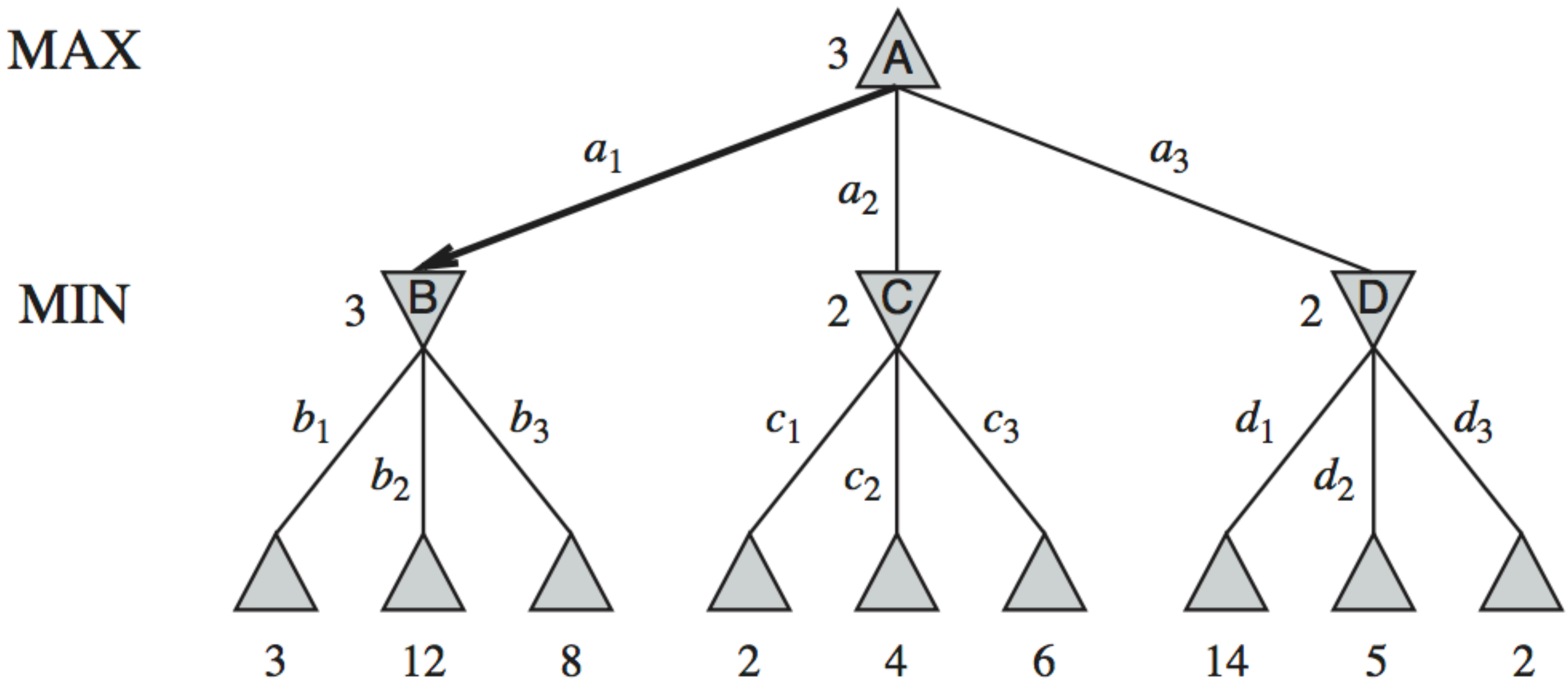
Min-Max Search

- "Normal" Search – An optimal solution is a sequence of actions leading to a **goal state** – a terminal state that is a *win*
- In adversarial search, MIN has something to say
- Execution
  - MAX moves in initial state
  - MIN responds
  - MAX must find a **contingent strategy**
  - MAX moves in each state resulting from every possible move by MIN

Min-Max Search

- Perfect play for deterministic games
- Choose move to position with highest minimax value
- Each player assumes their opponent will attempt to maximize their value

Min-Max 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
```

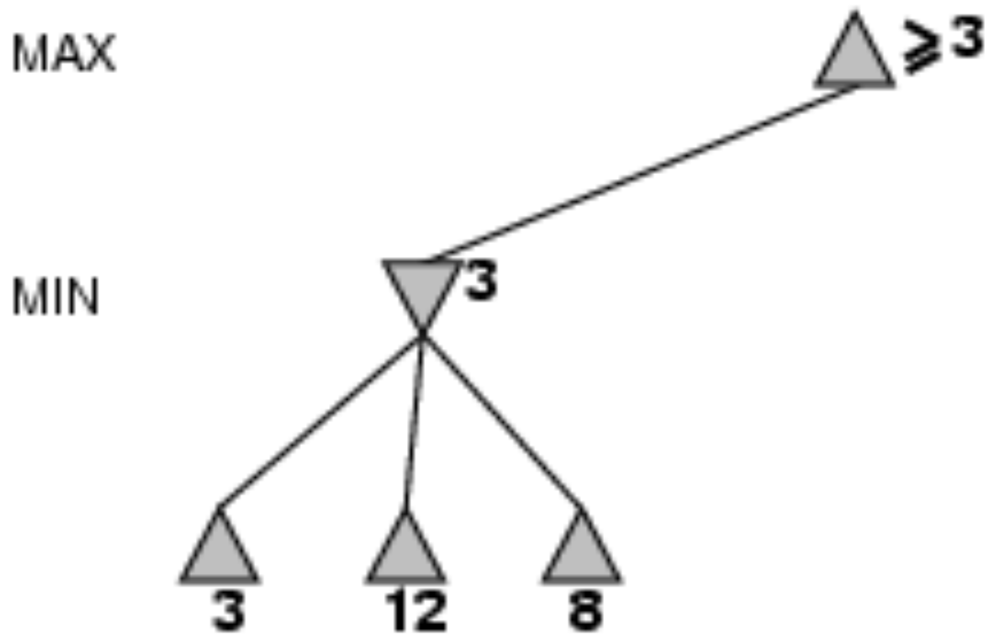
Properties of Min-Max

- Complete?
  - Yes (if tree is finite)
- Optimal?
  - Yes (against an optimal opponent)
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - $O(b^m)$
- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  exact solution completely infeasible

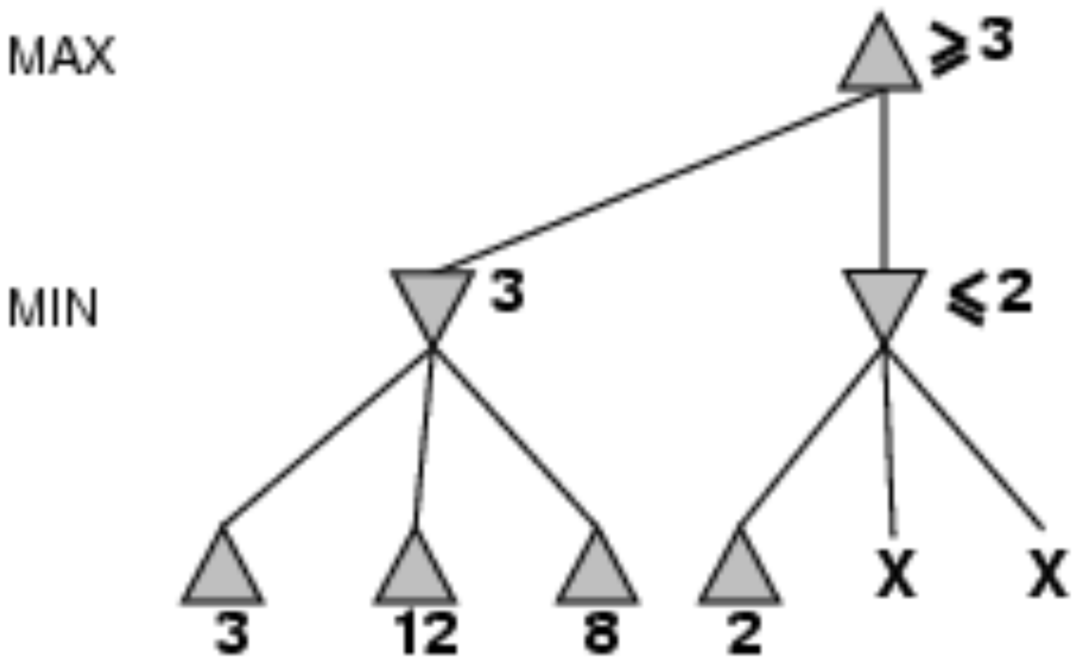
Alpha-Beta Pruning

- **Problem:** Number of minimax search games states is exponential in the tree depth
- Exponent can't be eliminated, but can effectively cut in half
  - Possible to compute correct minimax decision w/o looking at every node → **Pruning**
  - **Alpha-Beta Pruning:** returns the same move, but prunes branches that can't possibly influence the final decision

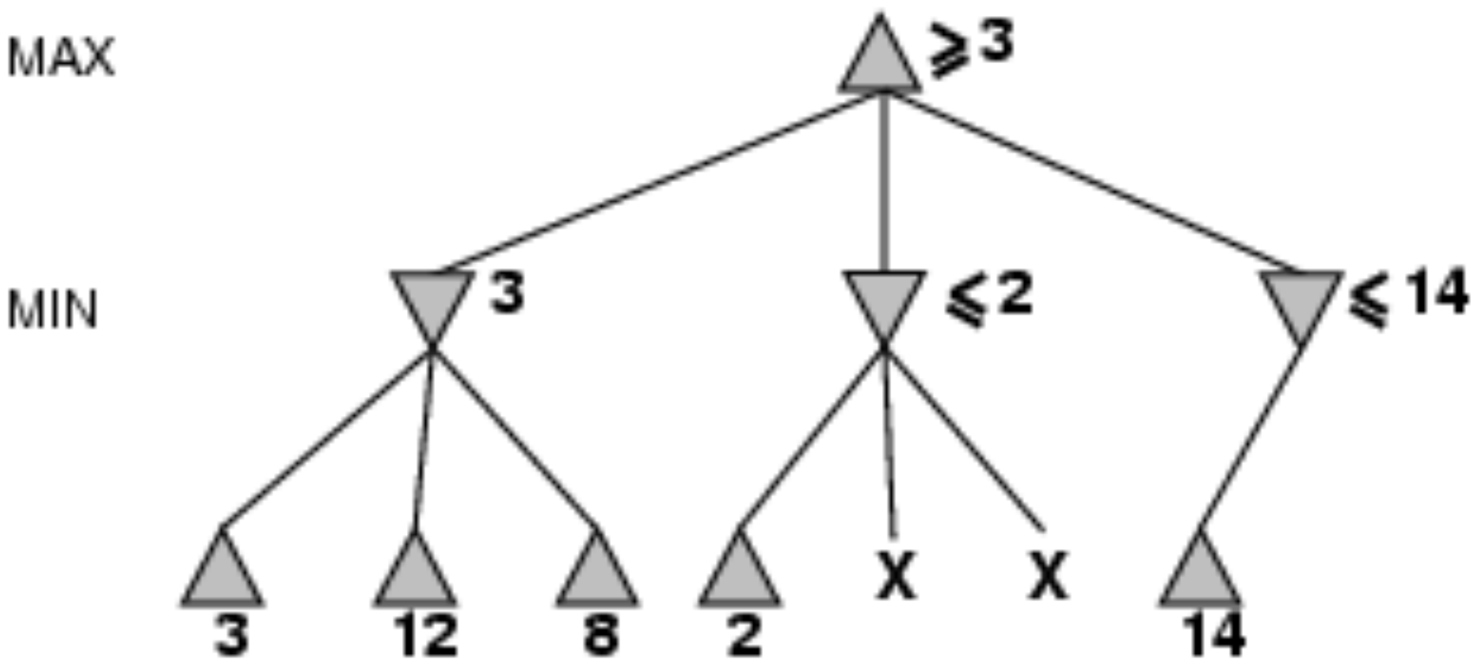
Alpha-Beta Pruning



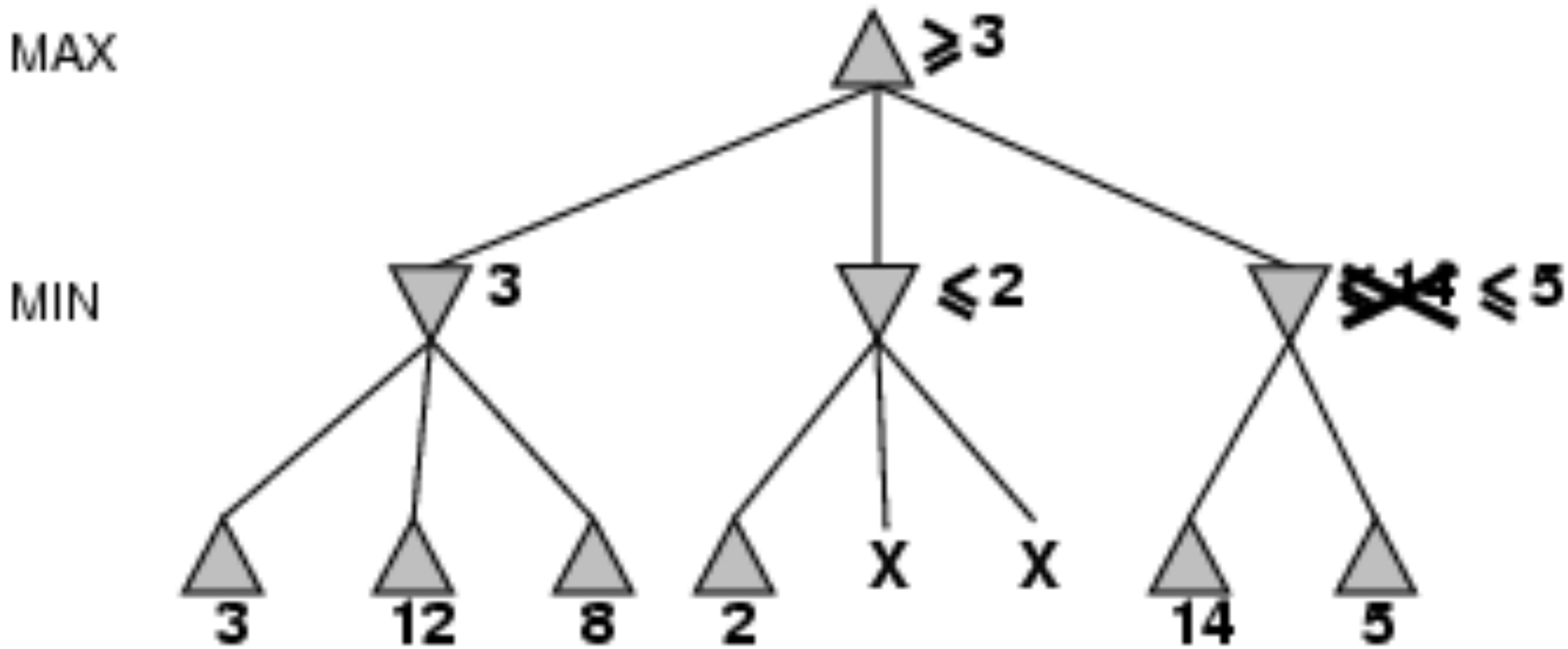
Alpha-Beta Pruning



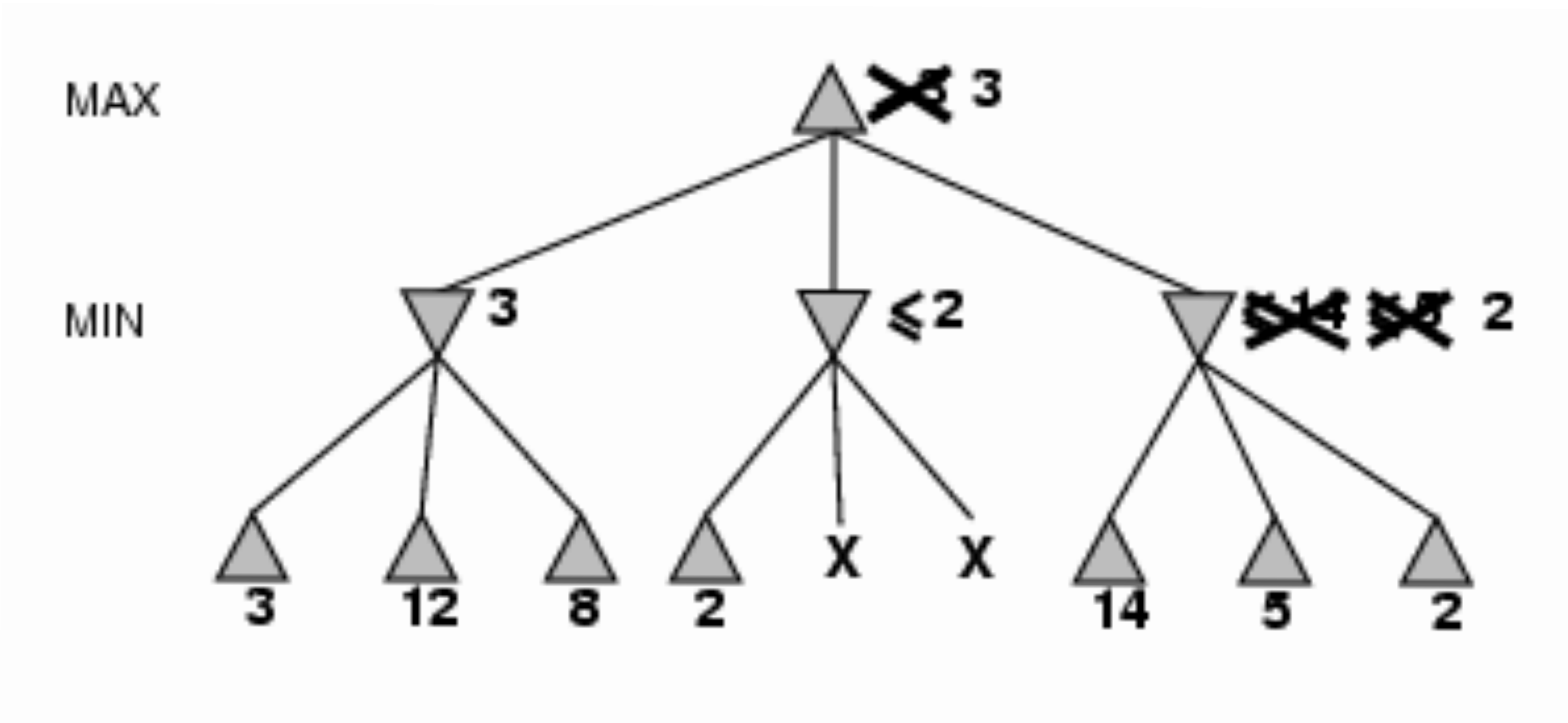
Alpha-Beta Pruning



Alpha-Beta Pruning



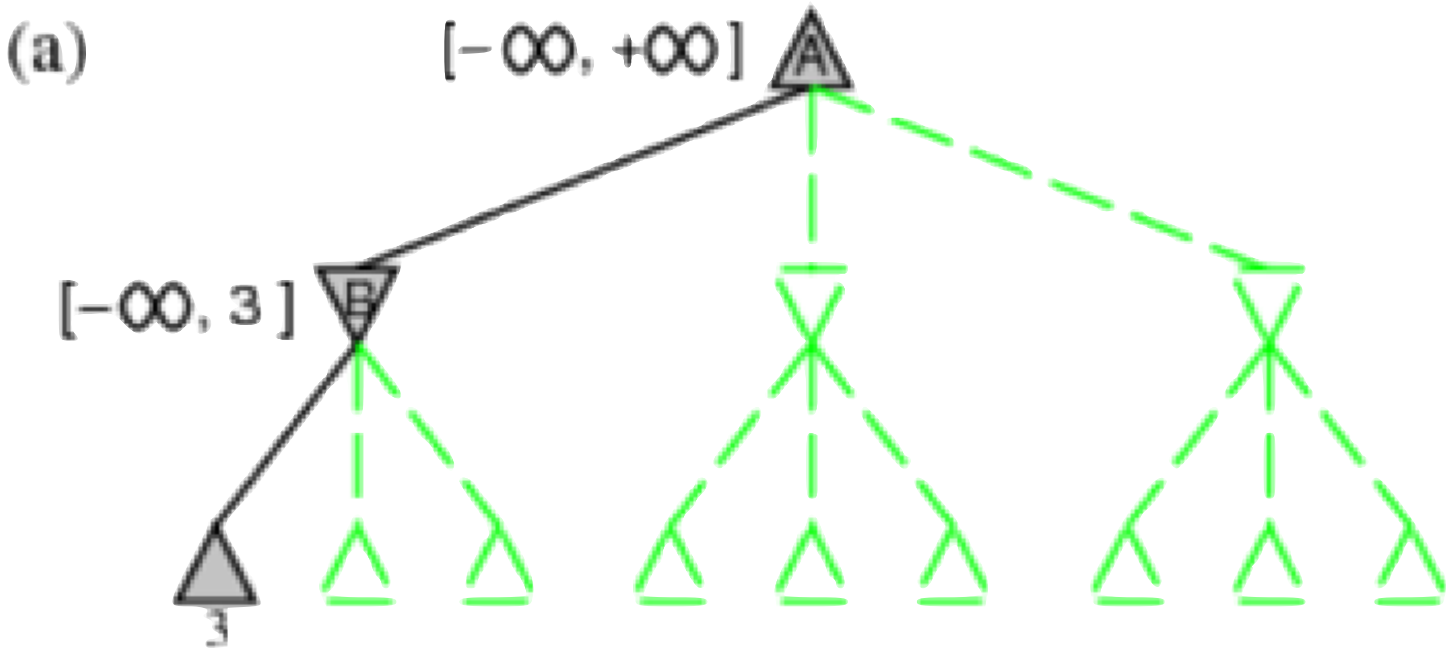
Alpha-Beta Pruning



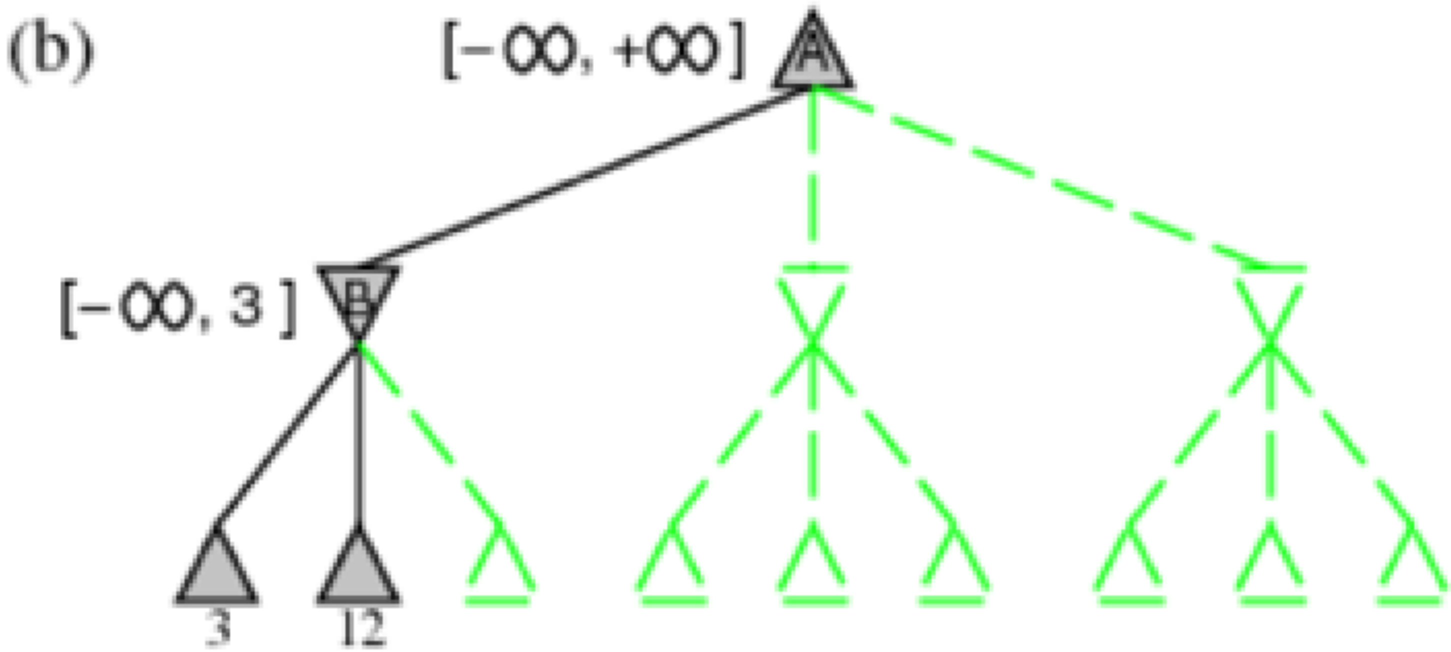
Alpha-Beta Pruning

- Start with the root node
- Tag with initial Min-max values ($-\infty$, $+\infty$)
- Update values when expanding each node

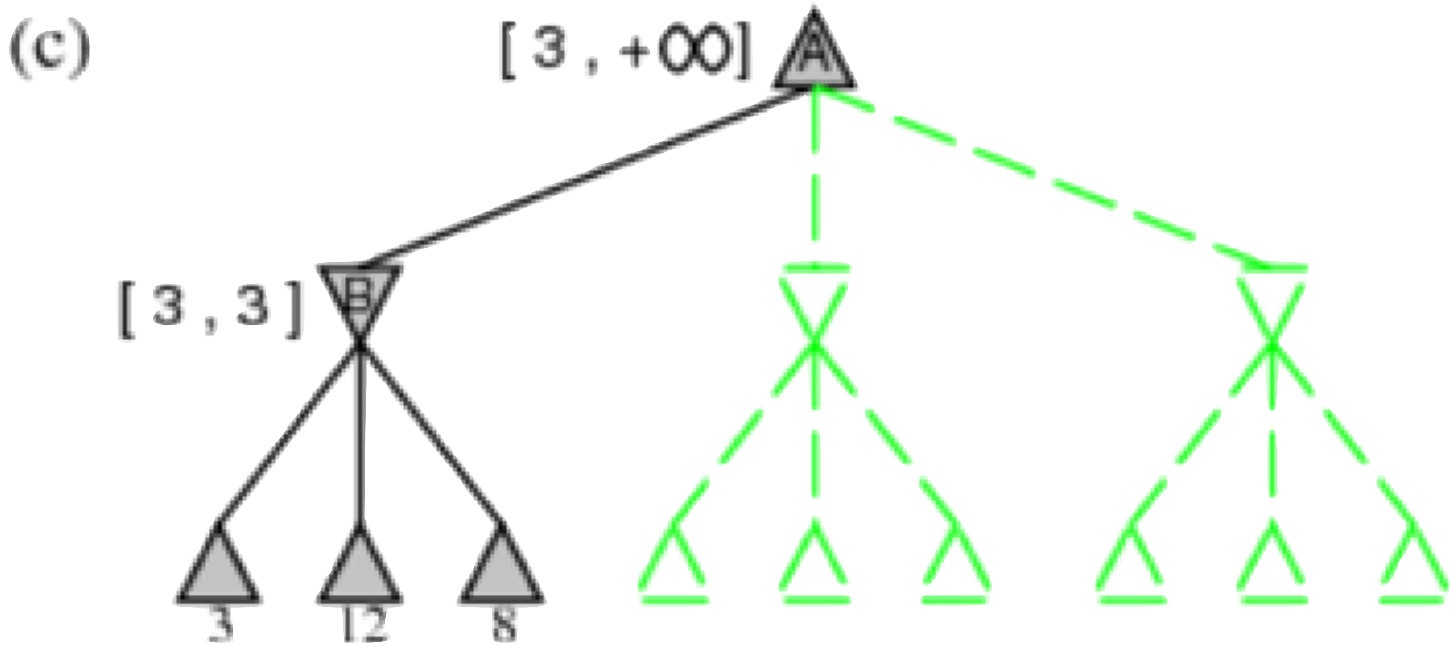
Alpha-Beta Pruning



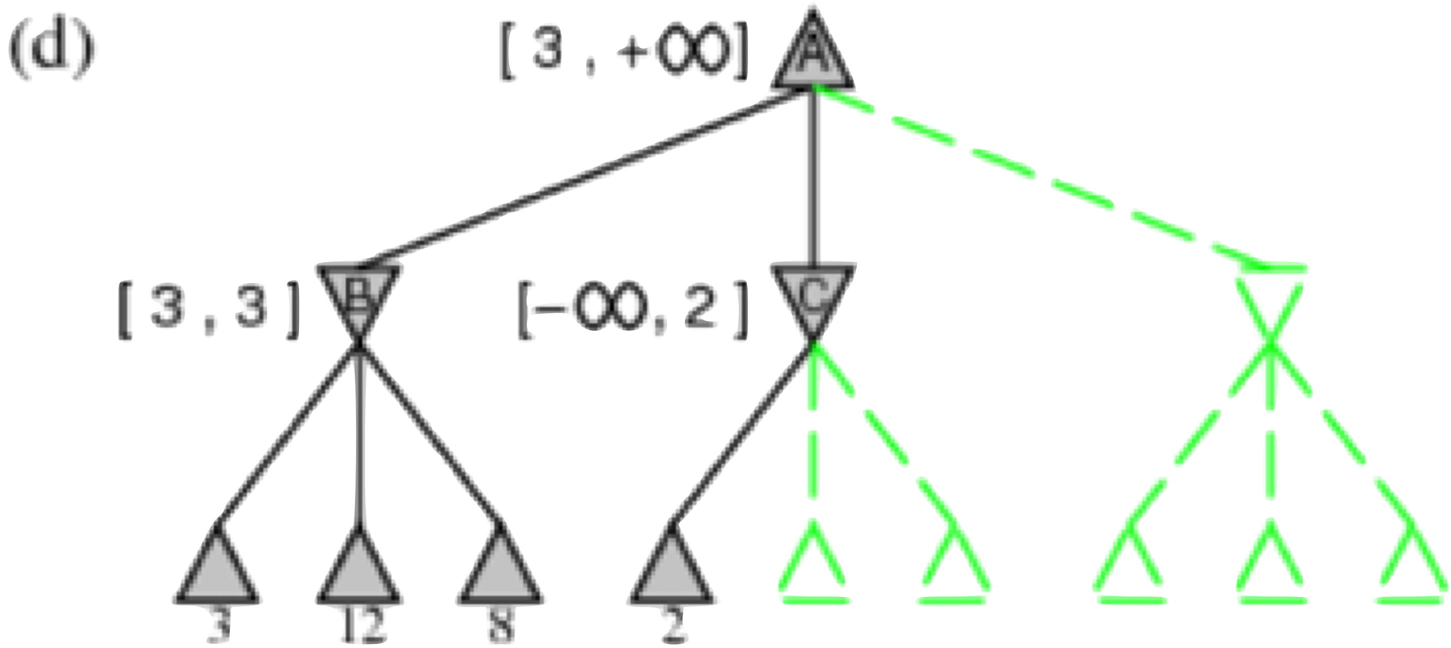
Alpha-Beta Pruning



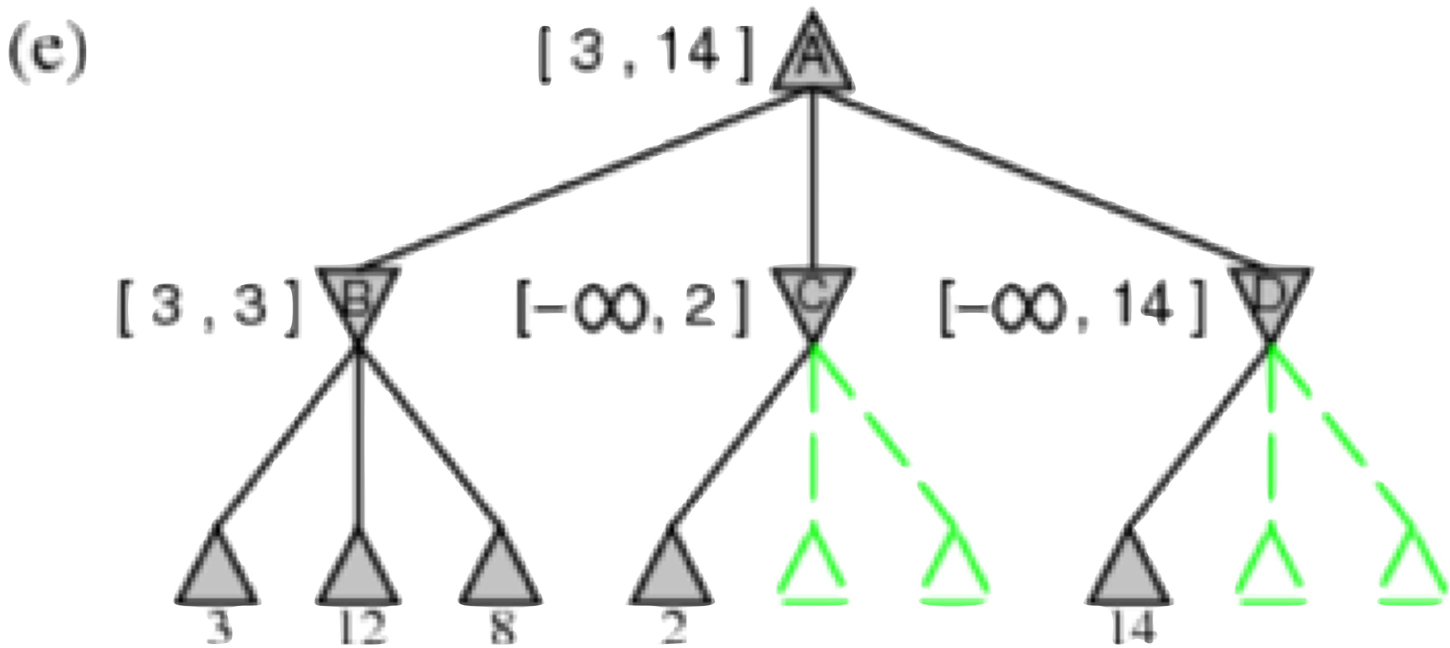
Alpha-Beta Pruning



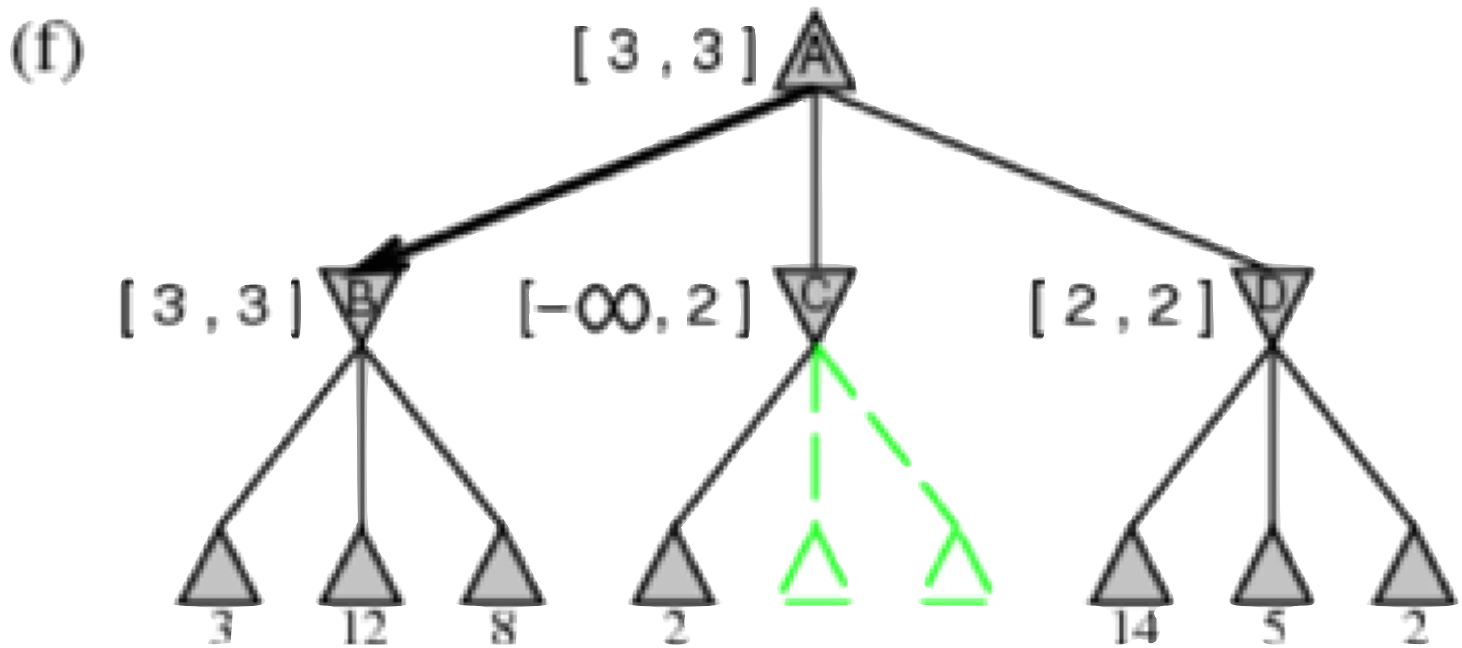
Alpha-Beta Pruning



Alpha-Beta Pruning

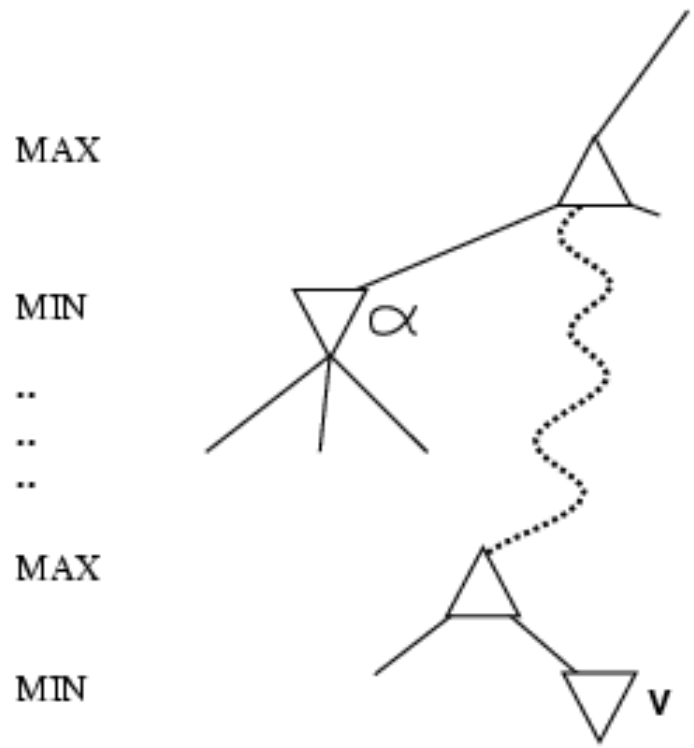


Alpha-Beta Pruning



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



Alpha-Beta Algorithm

function ALPHA-BETA-SEARCH($state$) **returns** *an action*
    inputs: $state$, current state in game

    $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
    return the $action$ in SUCCESSORS($state$) with value v

---

function MAX-VALUE($state, \alpha, \beta$) **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($state$)
    $v \leftarrow -\infty$
    for a, s in SUCCESSORS($state$) **do**
        $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
        if $v \geq \beta$ **then return** v
        $\alpha \leftarrow$ MAX(α, v)
    return v

Alpha-Beta 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
```

Move Ordering

- Effectiveness of Alpha-Beta pruning is highly dependent on the order the states are examined
  - Revisit example
  - Which successors are likely to be the best?
  - If this can be done, $O(b^n)$, where $n = m/2$, instead of $O(b^m)$ for minimax
- Alpha-Beta can solve a tree roughly twice as deep as minimax in the same amount of time
  - Why?

Properties of α-β

- Pruning does not affect final result
- With "perfect ordering" time complexity = $O(b^{m/2})$
- Good move ordering improves effectiveness of pruning
  - Why?

Imperfect Real-time Decisions

Evaluation Functions

- An **evaluation function** returns an *estimate* of the expected utility of the game from a given position
  - Similar to heuristic function from Informed Search
  - Turns some non-terminal nodes into terminal leaves
- Modify minimax or alpha-beta
  - Replace utility function with EVAL (estimates utility)
  - Replace terminal test with CUTOFF test

Evaluation Functions

- Properties
  - 1) Order the terminal states (like a true utility function)
  - 2) Reasonable computation time (point is to search faster)
  - 3) For non-terminal states, should be strongly correlated with actually chance of winning

Cutting off Search

- The minimax algorithm generates the entire game search space, whereas the alpha–beta algorithm allows us to prune large parts of it.
- However, alpha–beta still has to search all the way to terminal states for at least a portion of the search space.
- This depth is usually not practical, because moves must be made in a reasonable amount of time—typically a few minutes at most.

Cutting off Search

- Claude Shannon's paper *Programming a Computer for Playing Chess* (1950) proposed instead that programs should cut off the search earlier and apply a heuristic ***evaluation function*** to states in the search, effectively turning nonterminal nodes into terminal leaves.
- In other words, the suggestion is to alter minimax or alpha–beta in two ways:
  - replace the utility function by a heuristic evaluation function EVAL, which estimates the position's utility, and
  - replace the terminal test by a ***cutoff test*** that decides when to apply EVAL.

Cutting off Search

- That gives us the following for heuristic minimax for state s and maximum depth d:

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

Cutting off Search

- CUTOFF function
- What is a good CUTOFF function in Chess?
- CUTOFF-TEST(*state*, *depth*)
  - returns true for all depth greater than some fixed depth
  - returns true for all terminal states

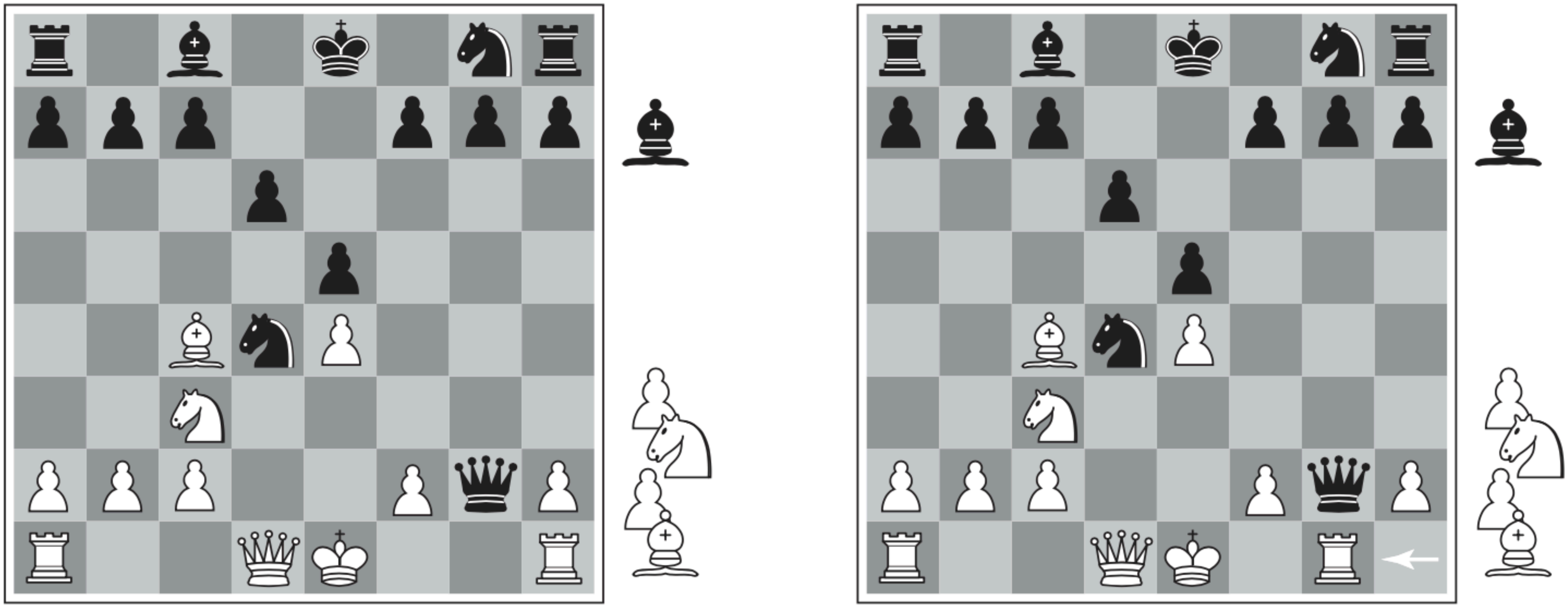
Cutting off Search

- These simple approaches can lead to errors due to the approximate nature of the evaluation function.
- Consider again the simple evaluation function for chess based on material advantage.
- Suppose the program searches to the depth limit, reaching the position in Figure 5.8(b), where Black is ahead by a knight and two pawns.

Cutting off Search

- It would report this as the heuristic value of the state, thereby declaring that the state is a probable win by Black.
- But White’s next move captures Black’s queen with no compensation.
- Hence, the position is really won for White, but this can be seen only by looking ahead one more ply.

Cutting off Search



(a) White to move

(b) White to move

Figure 5.8 Two chess positions that differ only in the position of the rook at lower right. In (a), Black has an advantage of a knight and two pawns, which should be enough to win the game. In (b), White will capture the queen, giving it an advantage that should be strong enough to win.

Forward Pruning

- Some nodes may be pruned without further consideration
- Beam search – consider only a “beam” of best moves
  - No guarantee that the best move will not be pruned
- PROBCUT algorithm (Buro, 1995) is a forward pruning version of alpha-beta that uses statistics gained from prior experience to lessen the chance that the best choice will be pruned.

The Horizon Effect

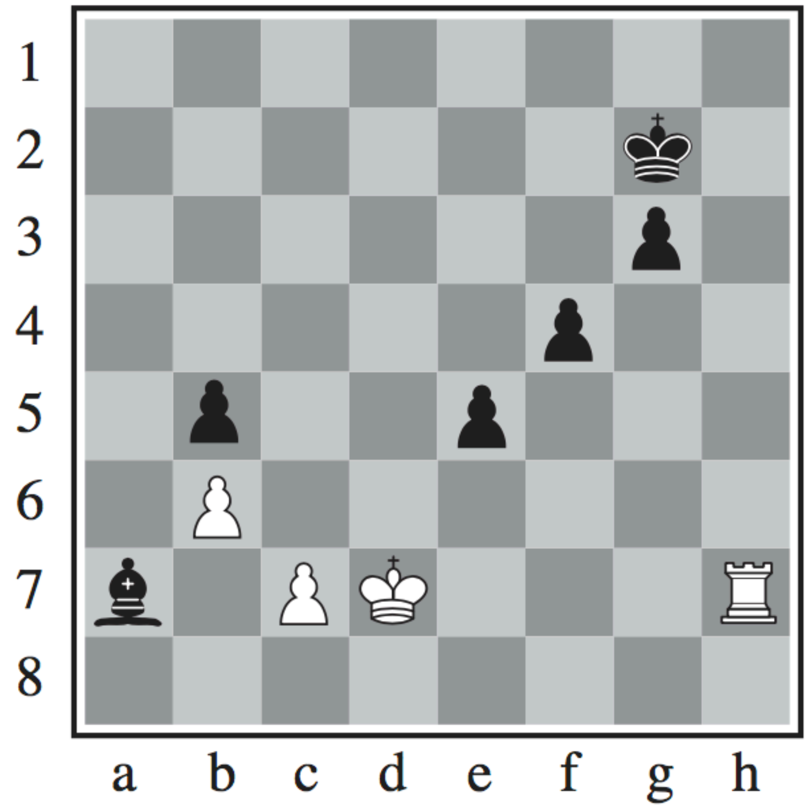


Figure 5.9 The horizon effect. With Black to move, the black bishop is surely doomed. But Black can forestall that event by checking the white king with its pawns, forcing the king to capture the pawns. This pushes the inevitable loss of the bishop over the horizon, and thus the pawn sacrifices are seen by the search algorithm as good moves rather than bad ones.

Search vs. Lookup

- Search
- Lookup table of patterns
  - Beginning game
  - End game

Stochastic Games

Stochastic Games

- In real life, unpredictable external events can cause unforeseen situations
- Many games mirror this unpredictability by introducing a random element (dice, spinner, etc.)
  - Linear probability distribution
  - Non-linear probability distribution
- How could we construct a tree?

Stochastic Game Tree

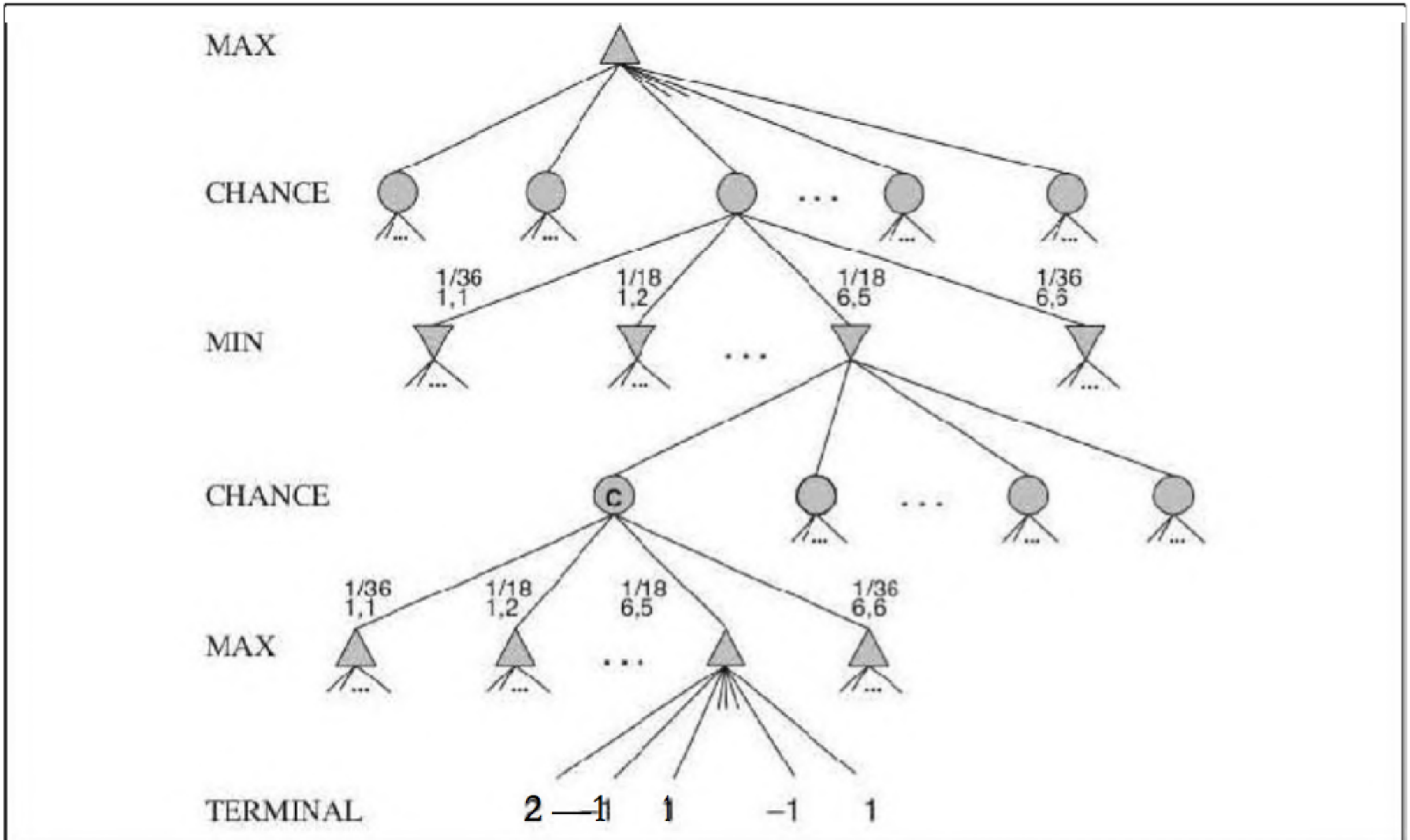


Figure 5.11 Schematic game tree for a backgammon position

Partially Observable Games

Card Games

- How to construct a game tree?
  - Blackjack
  - Poker
- At first sight, it might seem that these card games are just like dice games: the cards are dealt randomly and determine the moves available to each player, but all the “dice” are rolled at the beginning!
- Even though this analogy turns out to be incorrect, it suggests an effective algorithm: consider all possible deals of the invisible cards; solve each one as if it were a fully observable game; and then choose the move that has the best outcome averaged over all the deals.

Card Games

- Suppose that each deal s occurs with probability P(s); then the move we want is

$$\operatorname*{argmax}_{a} \sum_{s} P(s)\, \textsc{Minimax}(\textsc{Result}(s, a)) \; .$$

- Here, we run exact MINIMAX if computationally feasible; otherwise, we run H-MINIMAX.

State-of-the-art Game Programs

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

Deterministic games in practice

- Reversi / 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.