AI

Artificial Intelligence

8.2.4.
Adversarial Search (Ch 6)

Outline

- Games
- Optimal decisions
- Alpha-beta pruning

Outline

- Imperfect decisions
- Stochastic games
- Partially observable games
- State-of-the art game programs

Example Game

- Three bins
- You choose a bin. I choose a number from that bin
- You want to maximize your score
- Contents of the bins

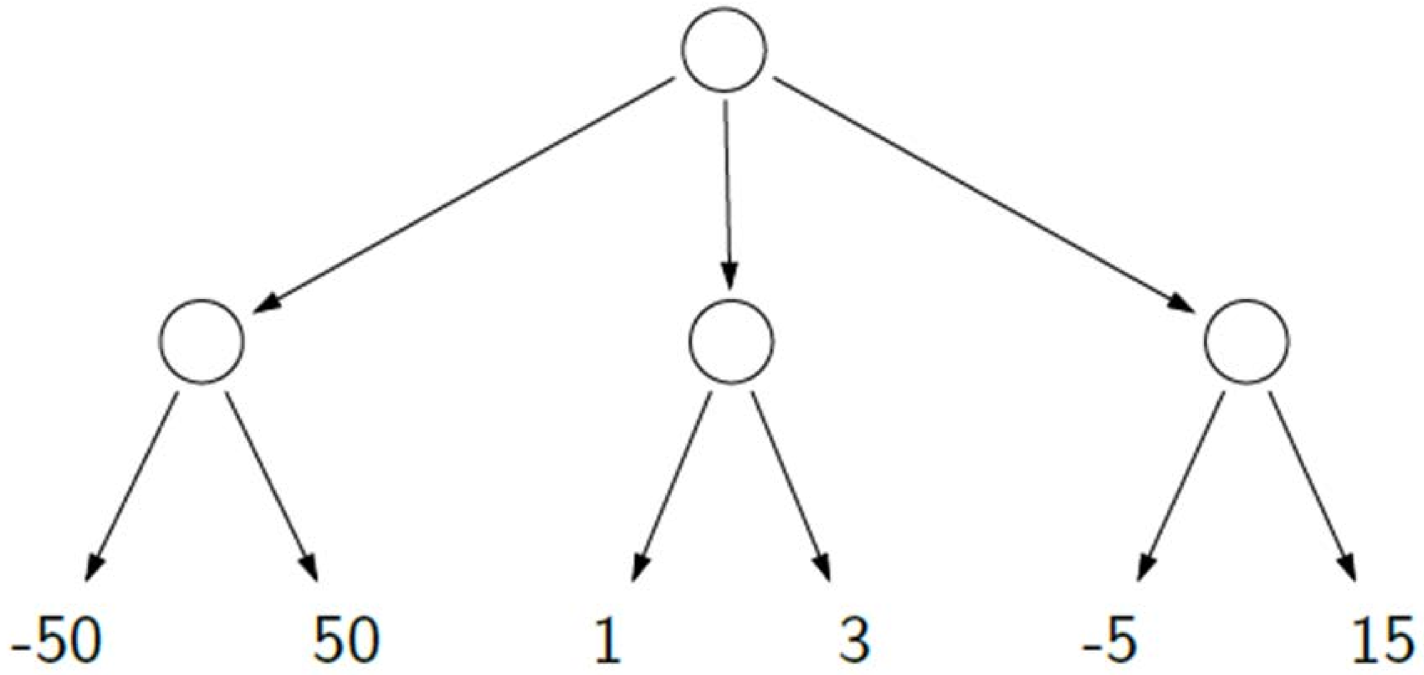
  A = -50 50

  B = 1 3

  C = -5 15
- Strategies
  - Cooperative: pick A
  - Competitive: pick B
  - Uniform Random: pick C

Example from Percy Liang

Game Tree

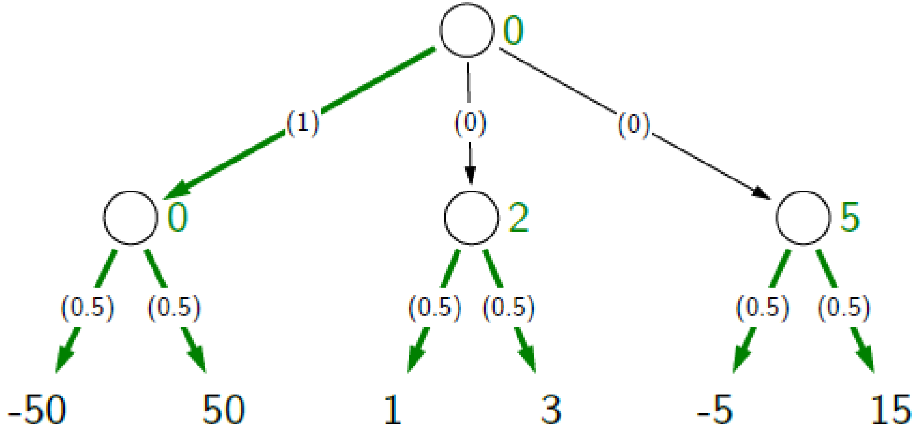


Game evaluation example

Example: game evaluation

$\pi_{\text{agent}}(s) = \text{A}$

$\pi_{\text{opp}}(s, a) = \frac{1}{2}$ for $a \in \text{Actions}(s)$



$$\boxed{V_{\text{agent,opp}}(s_{\text{start}}) = 0}$$

Example from Percy Liang

Types of games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

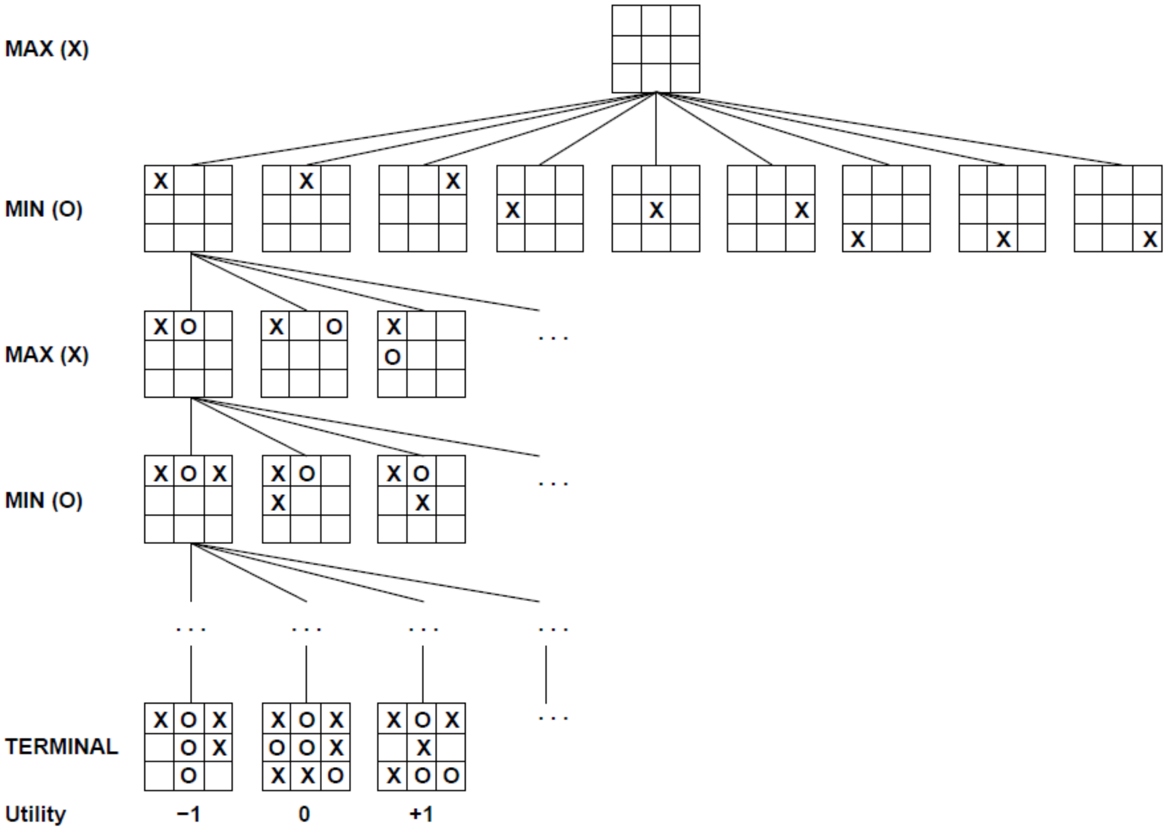
Games vs. search problems

- "Unpredictable" opponent → specifying a move for every possible opponent reply
- Time limits → unlikely to find goal, must approximate

Minimax Search

- Core of many computer games
- Pertains primarily to:
  - Turn based games
  - Two players
  - Players with “perfect knowledge”

Game tree (2-player, deterministic, turns)



Game Tree

- Nodes are states
- Edges are decisions
- Levels are called “plys”

Naïve Approach

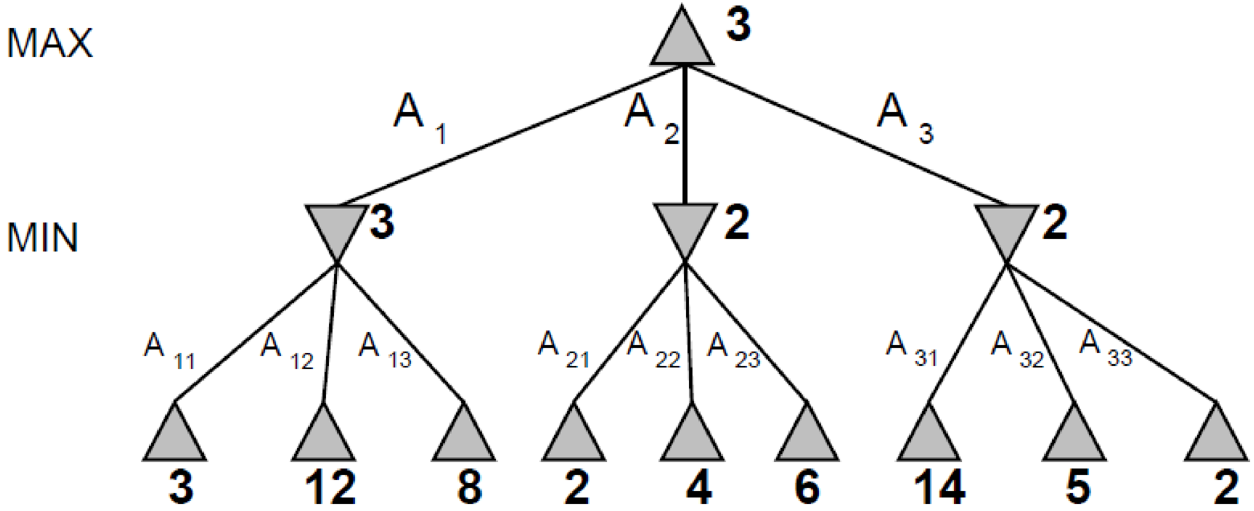
- Given a game tree, what would be the most straightforward playing approach?
- Any potential problems?

Minimax

- **Mini**mizing the **max**imum possible loss
- Choose move which results in best state
  - Select highest expected score for you
- Assume opponent is playing optimally too
  - Will choose lowest expected score for you

Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game:



Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game

    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

```
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 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
  → exact solution completely infeasible

Resource limits

Suppose we have 100 secs, explore 10^4 nodes/sec
→ 10^6 nodes per move

Standard approach:

- cutoff test:
  e.g., depth limit (perhaps add quiescence search)
- evaluation function
  = estimated desirability of position

Demo

- http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html

Evaluation Functions

- Assign a utility score to a state
  - Different for players?
- Usually a range of integers
  - [-1000,+1000]
- +infinity for win
- -infinity for loss

Approximating the Evaluation Function

- Use machine learning to learn the weights of the features
- This is still intractable in the space of all policies
- Monte Carlo simulation
- Go – 361 possible moves, depth also 361

By Goban1 - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=15223468

Cutting Off Search

- How to score a game before it ends?
  - You have to fudge it!
- Use a **heuristic** function to approximate state’s utility

Cutting Off Search

MinimaxCutoff is identical to *MinimaxValue* except

1. *Terminal?* is replaced by *Cutoff?*
2. *Utility* is replaced by *Eval*

Does it work in practice?

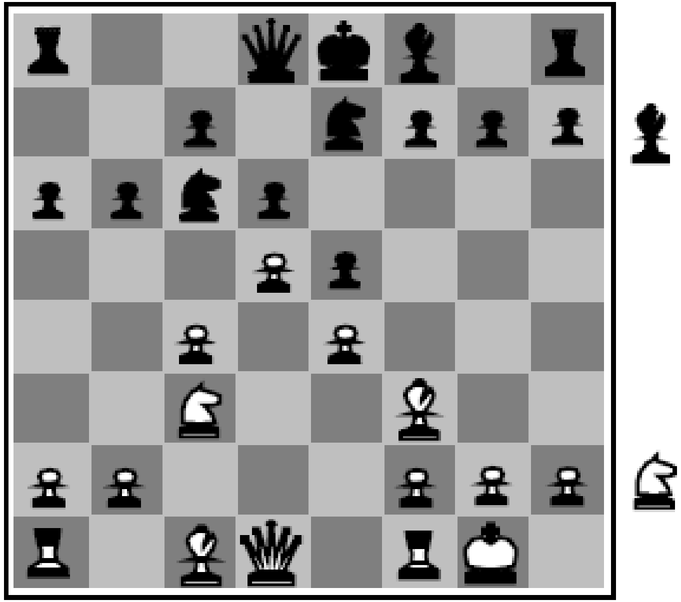
$b^m = 10^6$, $b=35 \rightarrow m=4$

4-ply lookahead is a hopeless chess player!

– 4-ply ≈ human novice
– 8-ply ≈ typical PC, human master
– 12-ply ≈ Deep Blue, Kasparov

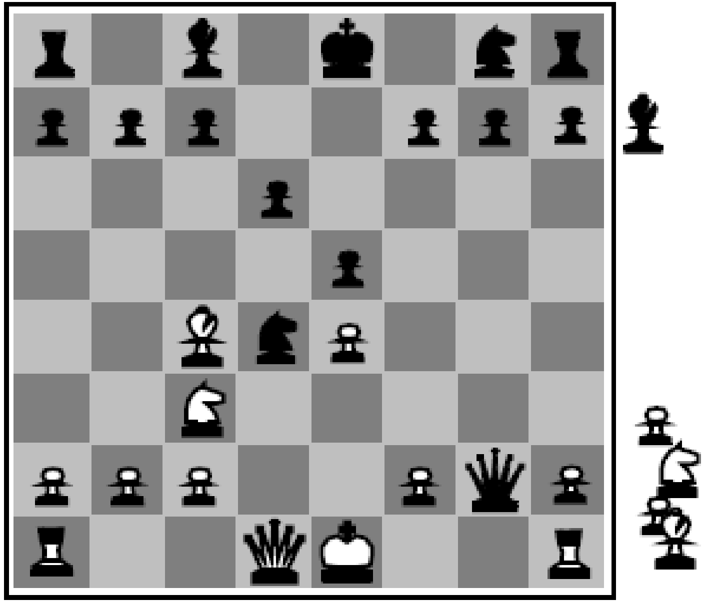
(A computer program which evaluates no further than its own legal moves plus the legal responses to those moves is searching to a depth of two-ply.)

Evaluation Function



Black to move

White slightly better



White to move

Black winning

Example Evaluation Function

- For chess, typically linear weighted sum of features

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

- $w_1 = 10^{10}$, $f_1(s) =$ (number of white kings) – (number of black kings), etc.
- $w_2 = 9$, $f_2(s) =$ (number of white queens) – (number of black queens), etc.
- 5 for rooks, 3 for knights, 2 for bishops, 1 for pawns

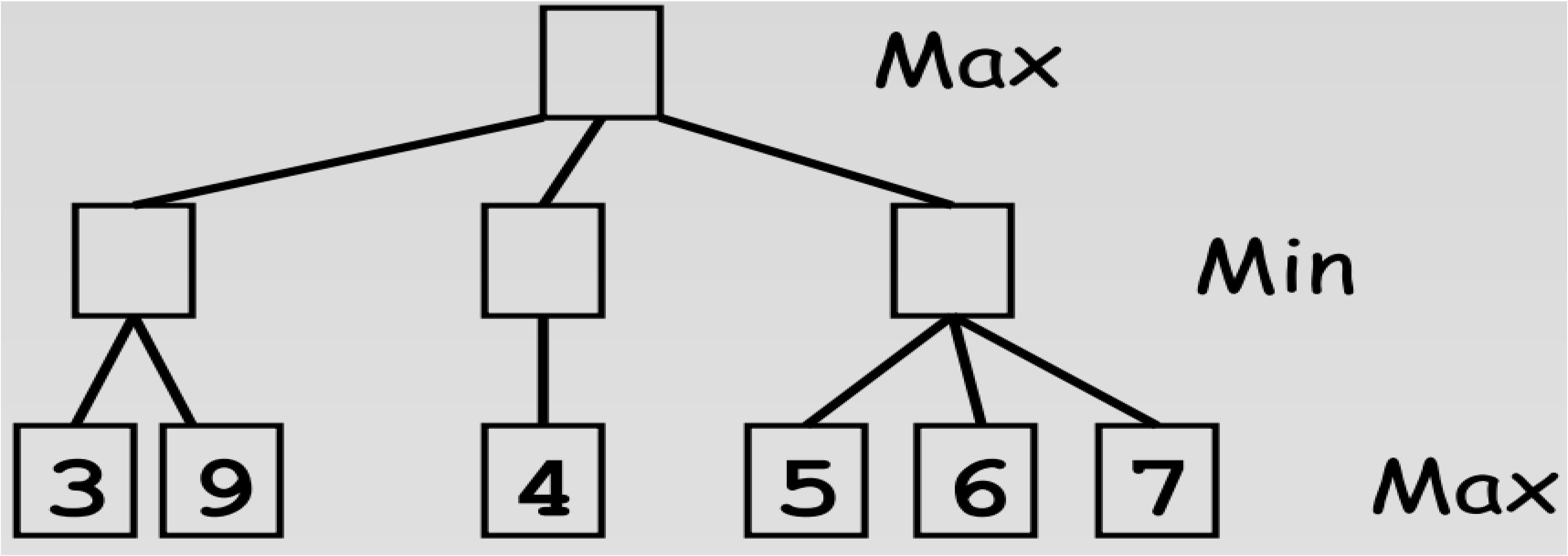
Evaluating States

- Assuming an ideal evaluation function, how would you make a move?
- Is this a good strategy with a bad function?

Look Ahead

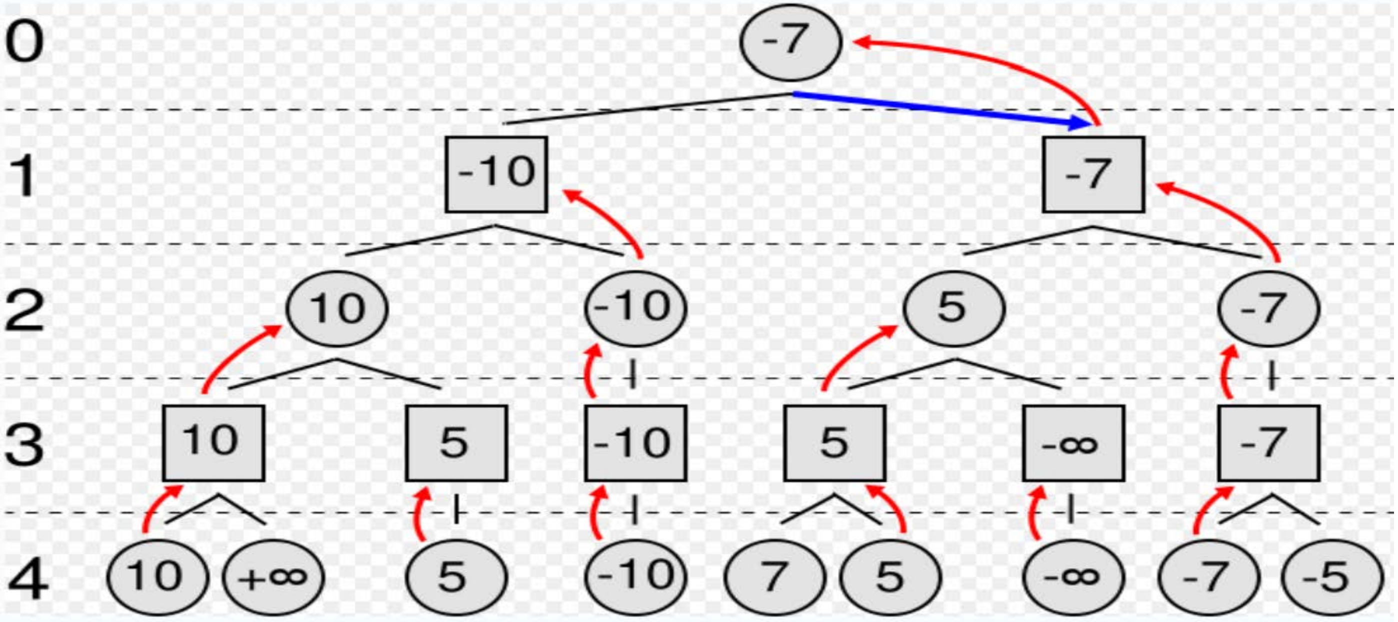
- Instead of only evaluating immediate future, look as far ahead as possible

Look Ahead

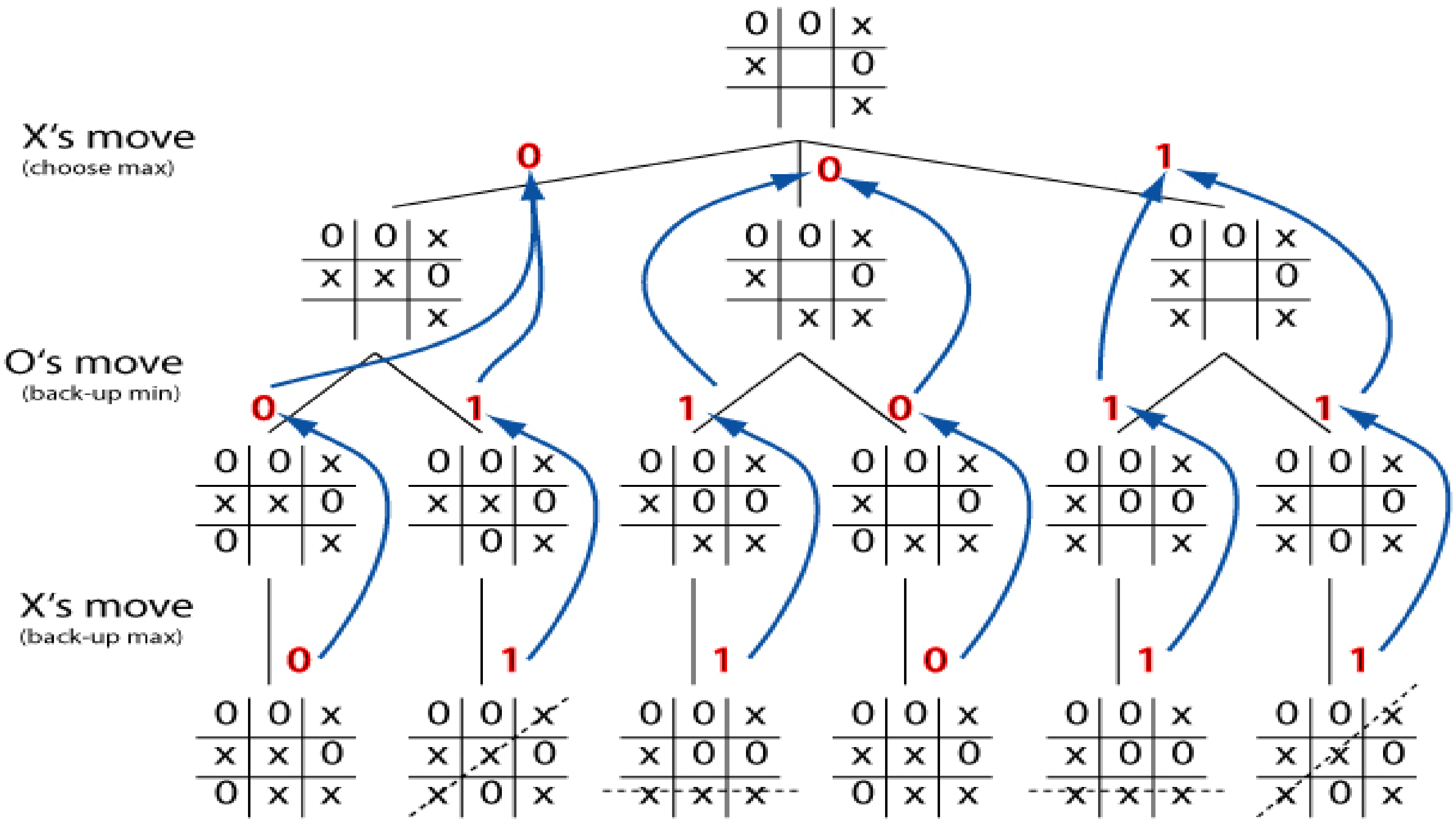


Bubbling Up

- Looking ahead allows utility values to “bubble up” to the root of the search tree



Tic-tac-toe Example



Recap

- What is a zero sum game?
- What is a game tree?
- What is Minimax?
  - Why is it called that?
- What is its space complexity?
- How can the Minimax algorithm be simplified?
  - Will this work for all games?

Next Up

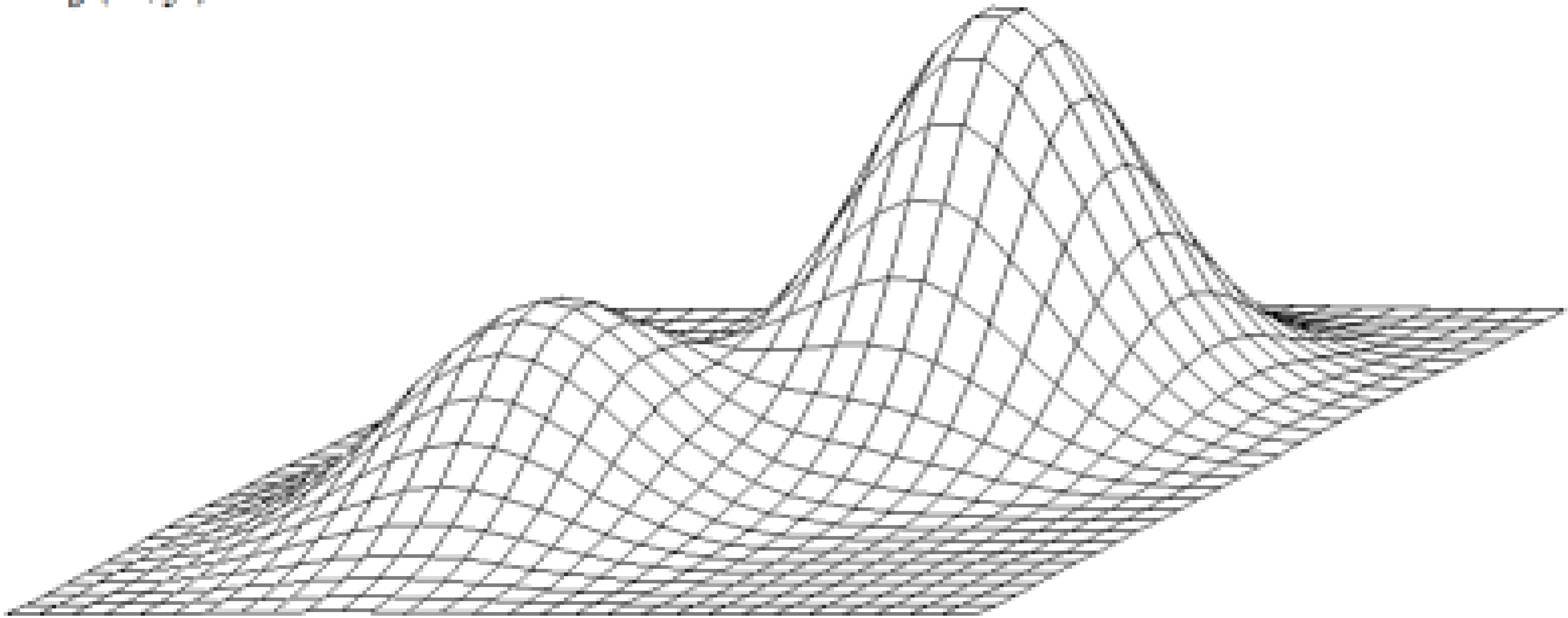
- Recall that minimax will produce optimal play against an optimal opponent if entire tree is searched
- Is the same true if a cutoff is used?

Horizon Effect

- Your algorithm searches to depth n
- What happens if:
  - Evaluation(s) at depth n is very positive
  - Evaluation(s) at depth n+1 is very negative
- Or:
  - Evaluation(s) at depth n is very negative
  - Evaluation(s) at depth n+1 is very positive
- Will this ever happen in practice?

Local Maxima Problem

$$f(x,y)=e^{-(x^2+y^2)} + 2e^{-((x-1.7)^2+(y-1.7)^2)}$$



Search Limitation Mitigation

- Sometimes it is useful to look deeper into game tree
- We could peak past the horizon…
- But how can you decide what nodes to explore?
  - Quiescence search

Quiescence Search

- Human players have some intuition about move quality
  - “Interesting vs “boring”
  - “Promising” vs “dead end”
  - “Noisy” vs “quiet”
- Expand horizon for potential high impact moves
- Quiescence search adds this to Minimax

Quiescence Search

- Additional search performed on leaf nodes
- if looks_interesting(leaf_node):

    extend_search_depth(leaf_node)

  else:

    normal_evaluation(leaf_node)

Quiescence Search

- What constitutes an “interesting” state?
  - Moves that substantially alter game state
  - Moves that cause large fluctuations in evaluation function output
- Chess example: capture moves
- Must be careful to prevent indefinite extension of search depth
  - Chess: checks vs captures

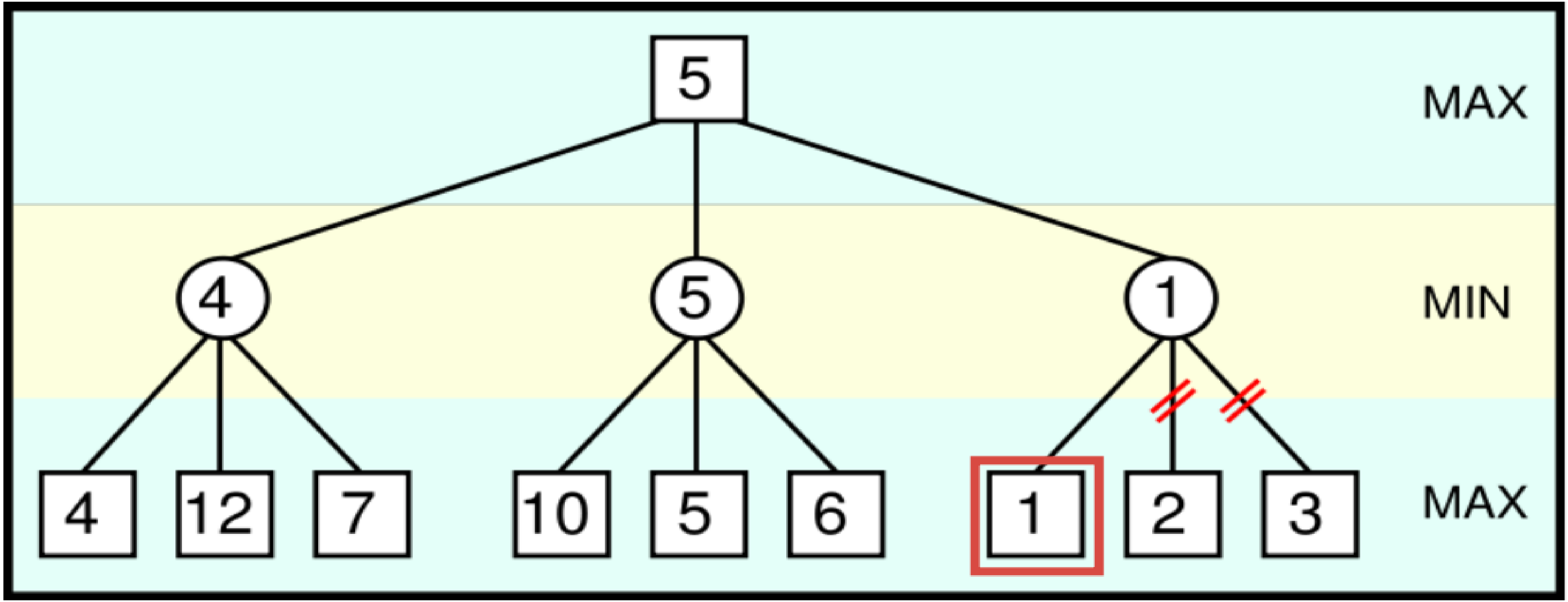
Search Limitation Mitigation

- Do you always need to search the entire tree?
  - No!
- Sometimes it is useful to look *less deeply* into tree
- But how can you decide what branches to ignore?
  - Tree pruning

Tree Pruning

- Moves chosen under assumption of optimal adversary
- You know the best move so far
- If you find a branch with a worse move, is there any point in looking further?
- Thought experiment: bag game

Pruning Example



Alpha-Beta Pruning

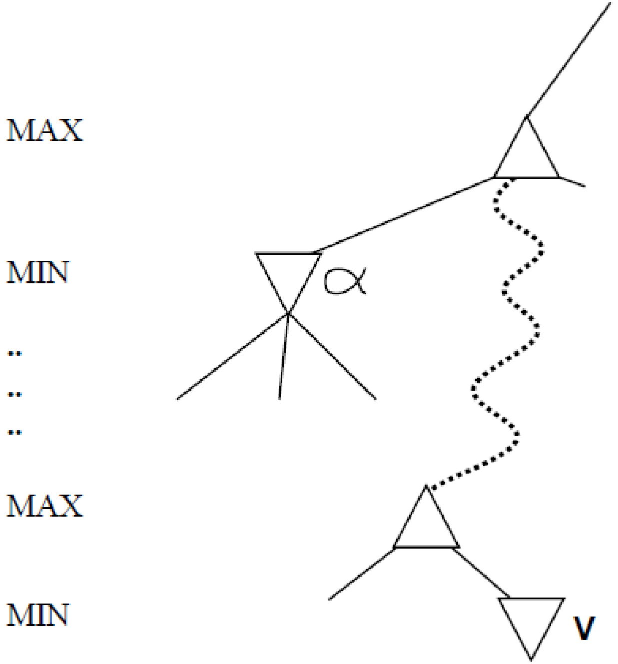
- During Minimax, keep track of two additional values
- Alpha
  – Your best score via any path
- Beta
  – Opponent’s best score via any path

Alpha-Beta Pruning

- Max player (you) will never make a move that could lead to a worse score for you
- Min player (opponent) will never make a move that could lead to a better score for you
- Stop evaluating a branch whenever:
  - A value greater than beta is found
  - A value less than alpha is found

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*

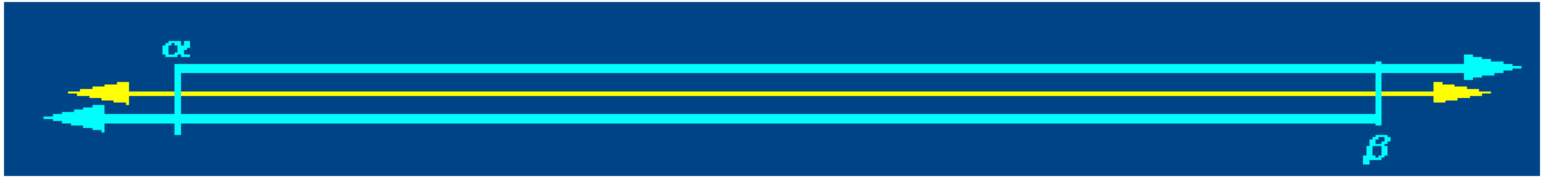


Alpha-Beta Pruning

- Based on observation that for all viable paths utility value n will be **$\alpha \le n \le \beta$**

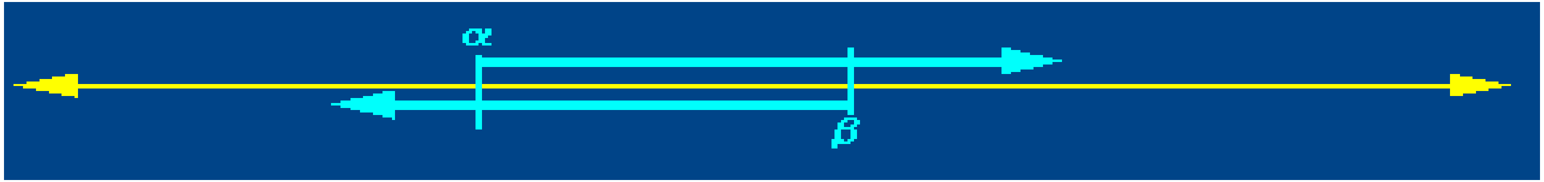
Alpha-Beta Pruning

- Initially, α = -infinity, β=infinity



Alpha-Beta Pruning

- As the search tree is traversed, the possible utility value window shrinks as
  - Alpha increases
  - Beta decreases



Alpha-Beta Pruning

- Once there is no longer any overlap in the possible ranges of alpha and beta, it is safe to conclude that the current node is a dead end

Minimax 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
```

The α-β algorithm

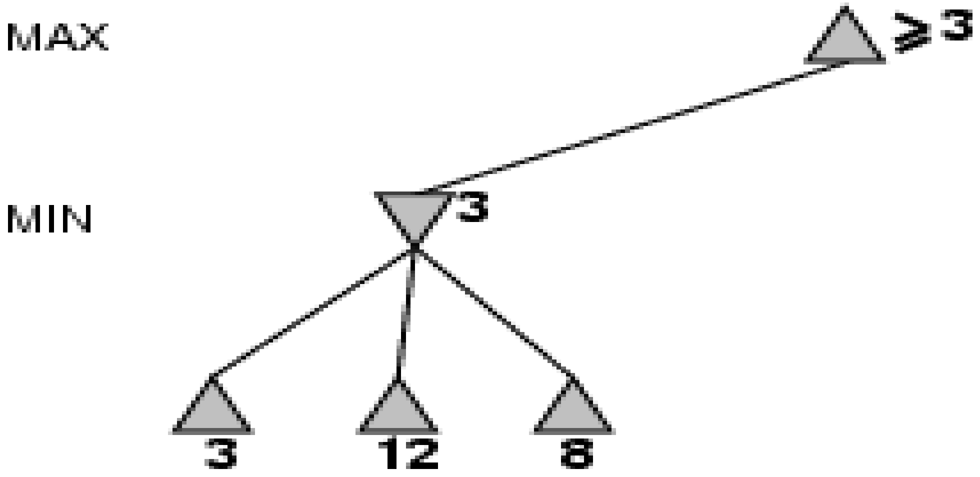
```
function ALPHA-BETA-SEARCH(state) returns an action
   inputs: state, current state in game

   v ← MAX-VALUE(state, −∞, +∞)
   return the action in SUCCESSORS(state) with value v
```

```
function MAX-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 ← MAX(v, MIN-VALUE(s, α, β))
      if v ≥ β then return v
      α ← MAX(α, v)
   return v
```

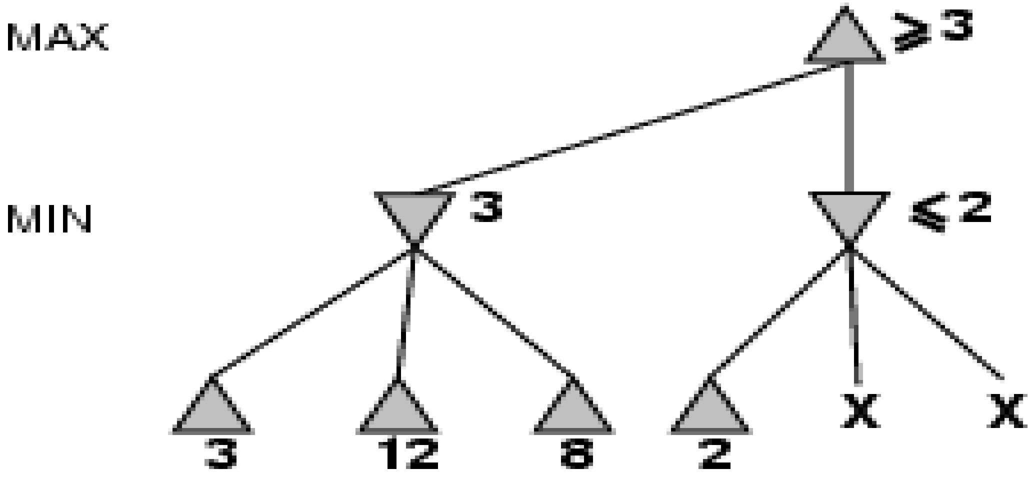
The α-β algorithm

```
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
```

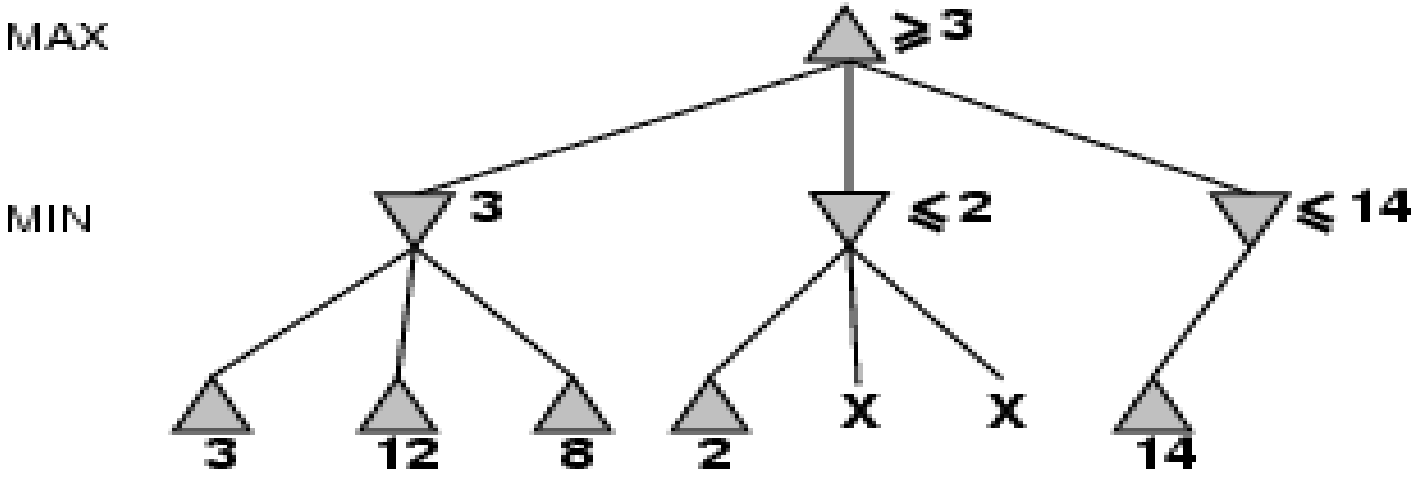
α-β pruning example



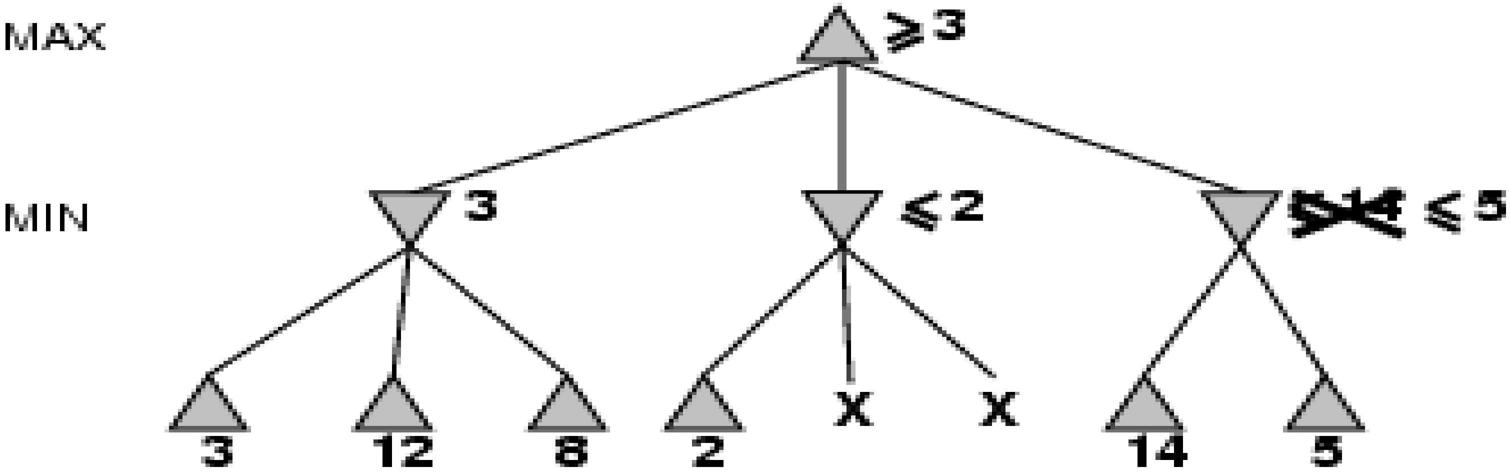
α-β pruning example



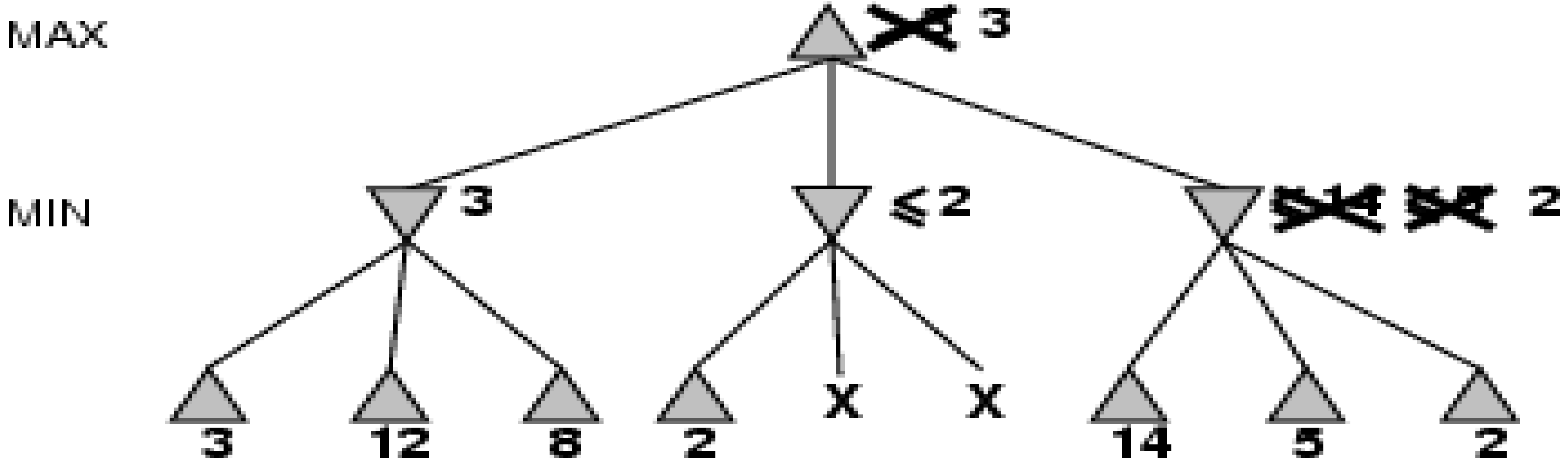
α-β pruning example



α-β pruning example



α-β pruning example



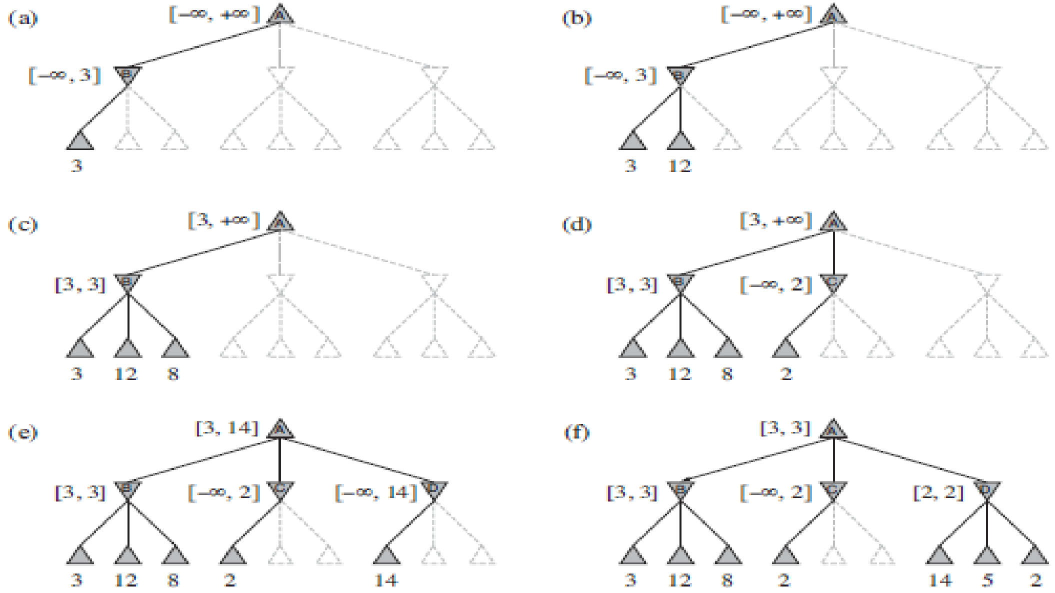
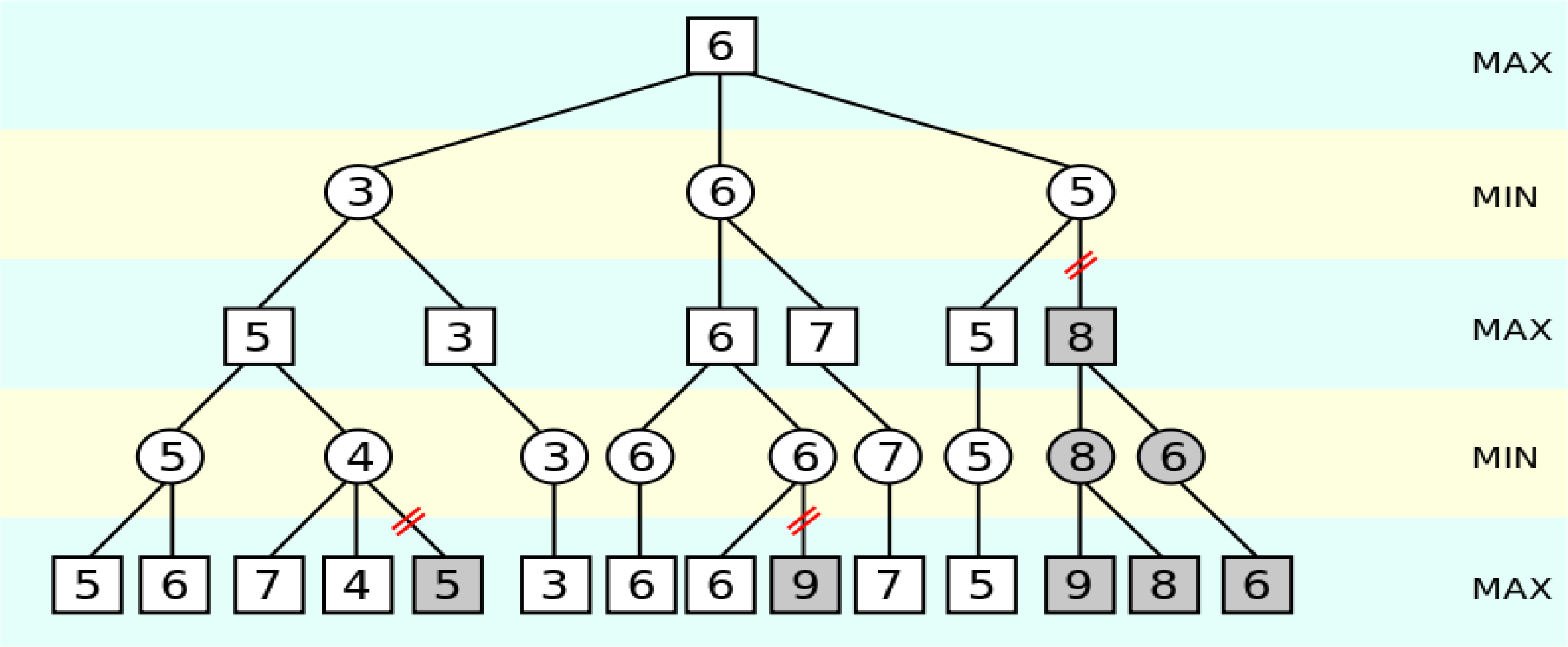


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2. At each point, we show the range of possible values for each node. (a) The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of *at most* 3. (b) The second leaf below B has a value of 12; MIN would avoid this move, so the value of B is still at most 3. (c) The third leaf below B has a value of 8; we have seen all B's successor states, so the value of B is exactly 3. Now, we can infer that the value of the root is *at least* 3, because MAX has a choice worth 3 at the root. (d) The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of *at most* 2. But we know that B is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successor states of C. This is an example of alpha–beta pruning. (e) The first leaf below D has the value 14, so D is worth *at most* 14. This is still higher than MAX's best alternative (i.e., 3), so we need to keep exploring D's successor states. Notice also that we now have bounds on all of the successors of the root, so the root's value is also at most 14. (f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX's decision at the root is to move to B, giving a value of 3.

Another α-β Pruning Example

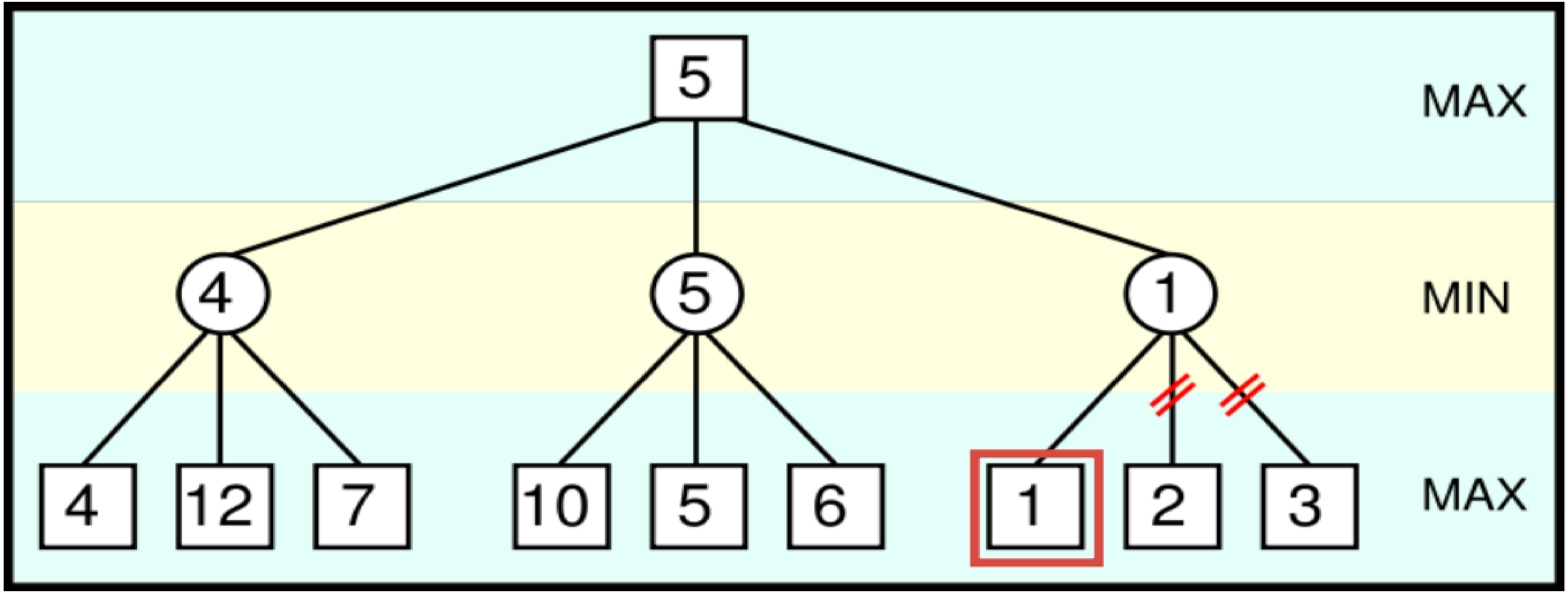


Tree Pruning vs Heuristics

- Search depth cut off may affect outcome of algorithm
- How about pruning?

Move Ordering

- Does the order in which moves are listed have any impact of alpha-beta?



Move Ordering

- Techniques for improving move ordering
- Apply evaluation function to nodes prior to expanding children
  - Search in descending order
  - But sacrifices search depth
- Cache results of previous algorithm

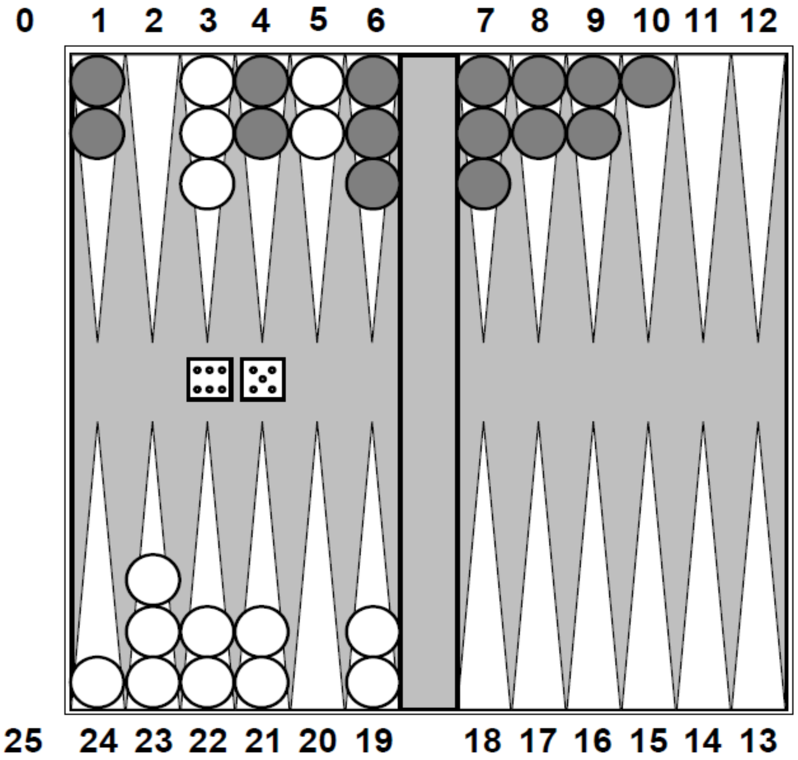
Properties of α-β

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  → doubles depth of search
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

Deterministic games in practice

- Checkers
  - Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a pre-computed 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
  - AlphaGo recently beat the best players in the world.

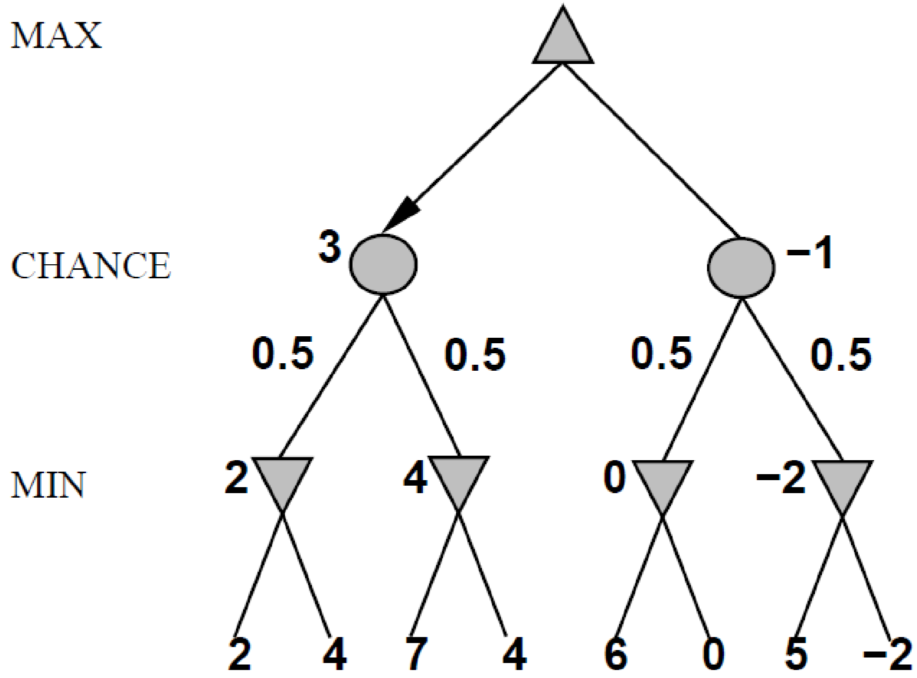
Non-deterministic: backgammon



Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:



Algorithm for nondeterministic games

EXPECTIMINIMAX gives perfect play

Just like MINIMAX, except we must also handle chance nodes:

```
. . .
if state is a MAX node then
    return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a MIN node then
    return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a chance node then
    return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
. . .
```

Nondeterministic games in practice

Dice rolls increase b: 21 possible rolls with 2 dice
Backgammon $\approx$ 20 legal moves (can be 6,000 with 1-1 roll)

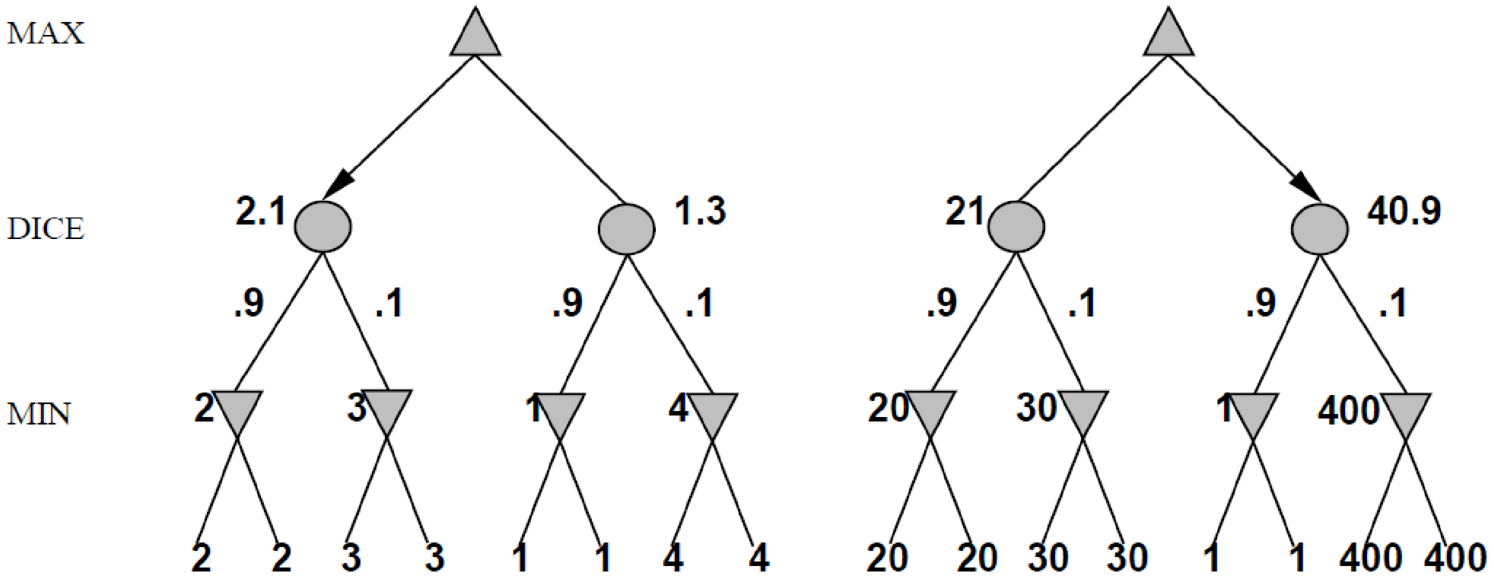
$$\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

As depth increases, probability of reaching a given node shrinks
$\Rightarrow$ value of lookahead is diminished

α–β pruning is much less effective

TDGammon uses depth-2 search + very good Eval
$\approx$ world-champion level

Digression: Exact values DO matter



Behaviour is preserved only by positive linear transformation of EVAL

Hence EVAL should be proportional to the expected payoff

Games of imperfect information

E.g., card games, where opponent's initial cards are unknown

Typically we can calculate a probability for each possible deal

Seems just like having one big dice roll at the beginning of the game*

Idea: compute the minimax value of each action in each deal,
then choose the action with highest expected value over all deals*

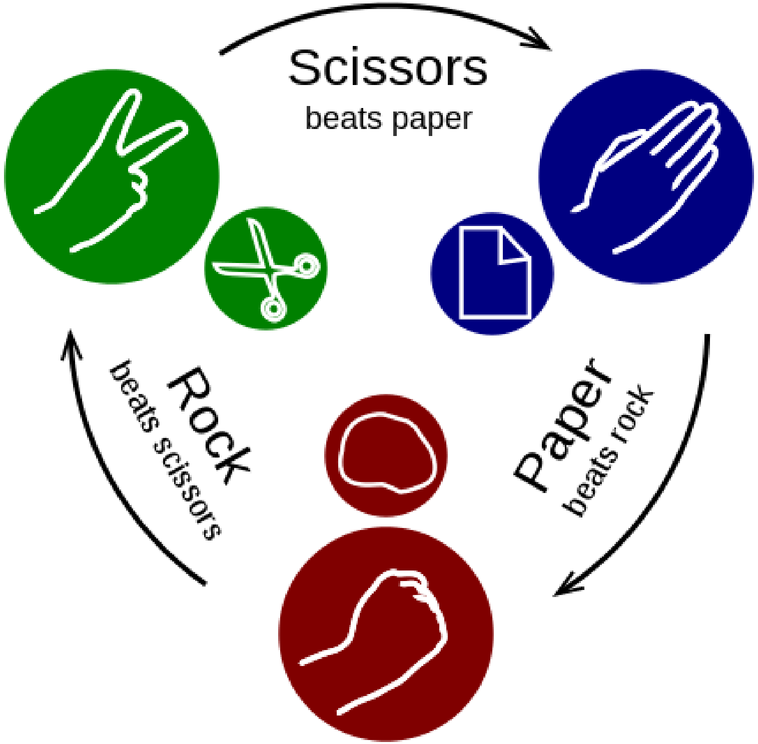
Special case: if an action is optimal for all deals, it's optimal.*

GIB, current best bridge program, approximates this idea by
1) generating 100 deals consistent with bidding information
2) picking the action that wins most tricks on average

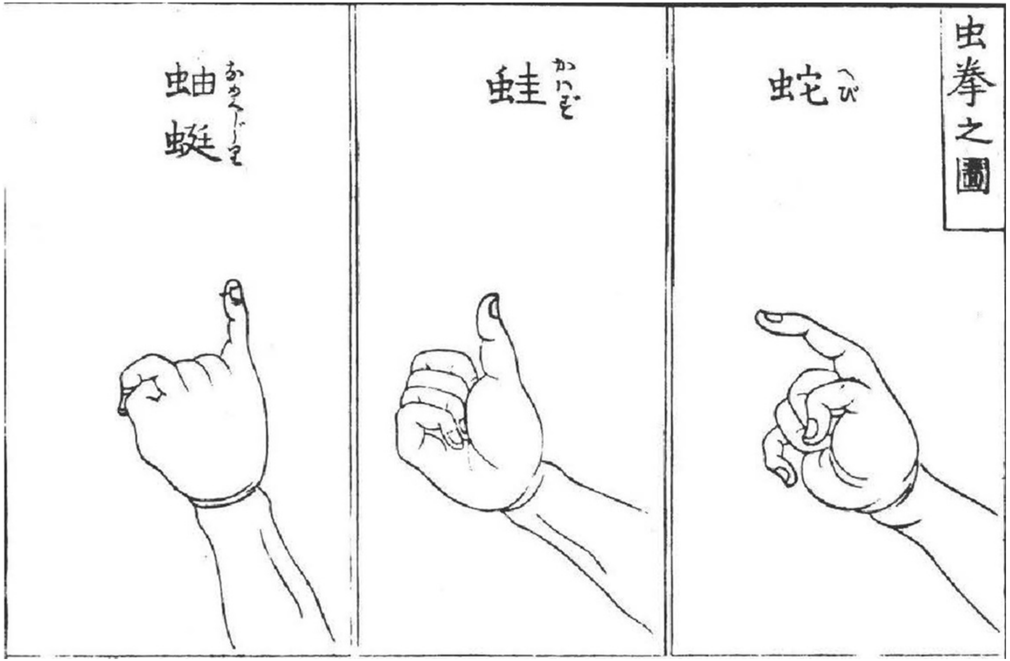
Notes

- Arthur Samuel (1959) – Checkers
  - linear combination of features, alpha-beta pruning
- Gerry Tesauro (1995) – Backgammon
  - TD-learning (combination of dynamic programming and Monte Carlo) – method invented by Sutton (1988)

Simultaneous Games

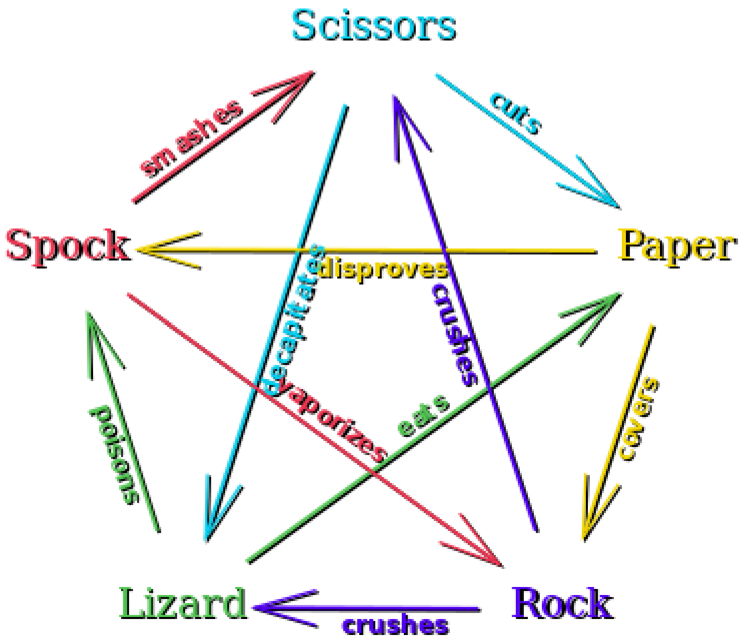


Mushi-ken



By Linhart, Sepp. "Die Repräsentation Von Tieren Im Japanischen Ken-Spiel: Versuch Einer Interpretation." Asiatische Studien: Zeitschrift Der Schweizerischen Asiengesellschaft 65.2 (2011): 541-61.Yoshinami and Gojaku. 1809. Kensarae sumai zue (拳會角力圖會). 2 vols. Edo: Murataya, Jirobe, Osaka: Kawachiya Taisuke, Bunka 6., Public Domain, https://commons.wikimedia.org/w/index.php?curid=37139430

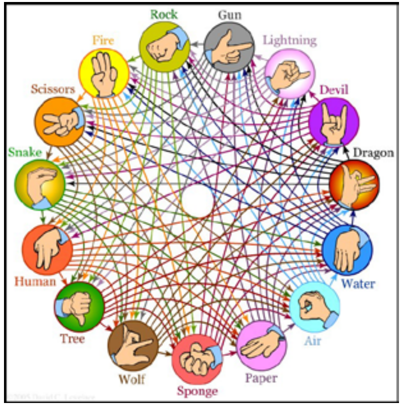
Rock-Paper-Scissors-Lizard-Spock



By VidTheKid - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=6560997

Versions with up to 101 Weapons ☺

- http://www.umop.com/rps7.htm
- http://www.umop.com/rps15.htm



- http://www.umop.com/rps101/rps101chart.html

Strategies for zero-sum games

- Pure Strategies
- Mixed Strategies

Non-zero sum games

- Collaborative Games

Prisoner B / Prisoner A	**Prisoner B stays silent (*cooperates*)**	**Prisoner B betrays (*defects*)**
Prisoner A stays silent (*cooperates*)	Each serves 1 year	Prisoner A: 3 years Prisoner B: goes free
Prisoner A betrays (*defects*)	Prisoner A: goes free Prisoner B: 3 years	Each serves 2 years

- Prisoner's Dilemma
- Nash Equilibrium
  - when no participant can gain by a unilateral change of strategy

Iterated Prisoner's Dilemma

- Nash equilibrium is again to defect at all turns
  - Proof by induction starting from the last turn
  - This applies even if the number of turns is unknown but is limited
- Evolution of cooperation
  - If the number of turns is random (Aumann 1959)
  - Best strategy – tit-for-tat with small probability of switch (Rapoport, Axelrod, early 1980s)

Summary

- Games are fun to work on!
- They illustrate several important points about AI
- Perfection is unattainable → must approximate
- AlphaGo uses supervised learning, reinforcement learning, as well as neural networks (for the evaluation function)
- Good idea to think about what to think about

AI