

Last update: February 3, 2009

GAME TREE SEARCH

CMSC 828N, GAME THEORY
DANA NAU

Finite sequential zero-sum games

Finite: finitely many agents, finitely many actions, finitely many states

Sequential: No simultaneous actions – players move one-at-a-time

Constant-sum: there's a constant k such that at every terminal state s ,
 $\sum\{\text{all the agents' utilities at } s\} = k$.

Every constant-sum game can be transformed into an equivalent **zero-sum** game (i.e., constant-sum with $k = 0$).

Thus “zero-sum” is often used to refer to all constant-sum games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon, monopoly, roulette
imperfect information	battleship, kriegspiel, rock-paper-scissors	bridge, poker, scrabble

Outline

- ◇ A brief history of work on this topic
- ◇ The minimax theorem
- ◇ Game trees
- ◇ The minimax algorithm
- ◇ α - β pruning
- ◇ Resource limits and approximate evaluation
- ◇ Games of chance (briefly)

A Brief History

- Computer considers possible lines of play (Babbage, 1846)
- Minimax theorem (von Neumann, 1928)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)
- Pruning to allow deeper search (McCarthy, 1956)
- Deep Blue wins 6-game chess match against Kasparov (Hsu *et al*, 1997)
- Checkers solved (Schaeffer *et al*, 2007)

The Minimax Theorem (von Neumann, 1928)

Recall that

- ◇ A *strategy* tells what an agent will do in every possible situation
- ◇ Strategies may be *pure* (deterministic) or *mixed* (probabilistic)

Minimax theorem: Let G be a two-person finite zero-sum game between agents A and B . Then there are strategies s and t , and a number V_G called G 's *minimax value*, such that

- If B uses t , then A 's best possible expected utility is V_G
- If A uses s , then B 's best possible expected utility is $-V_G$

Corollary 1: If A and B use s and t , respectively, then A 's expected utility is V_G and B 's expected utility is $-V_G$.

Corollary 2: If G is a perfect-information game, then there are pure strategies s and t that satisfy the theorem.

Relation to Decision Criteria

Let $U_A(s, t)$ and $U_B(s, t)$ be A 's and B 's expected utilities when A uses s and B uses t

Maximin decision criterion: maximize your minimum payoff

A would choose $s^* \in S$ that maximizes $\min_{t \in T} U_A(s^*, t)$

B would choose $t^* \in T$ that maximizes $\min_{s \in S} U_B(s, t^*)$

Minimax decision criterion: minimize the other agent's maximum payoff

e.g., A would choose $s^\dagger \in S$ that minimizes $\max_{t \in T} U_B(s^\dagger, t)$

e.g., B would choose $t^\dagger \in T$ that minimizes $\max_{s \in S} U_A(s, t^\dagger)$

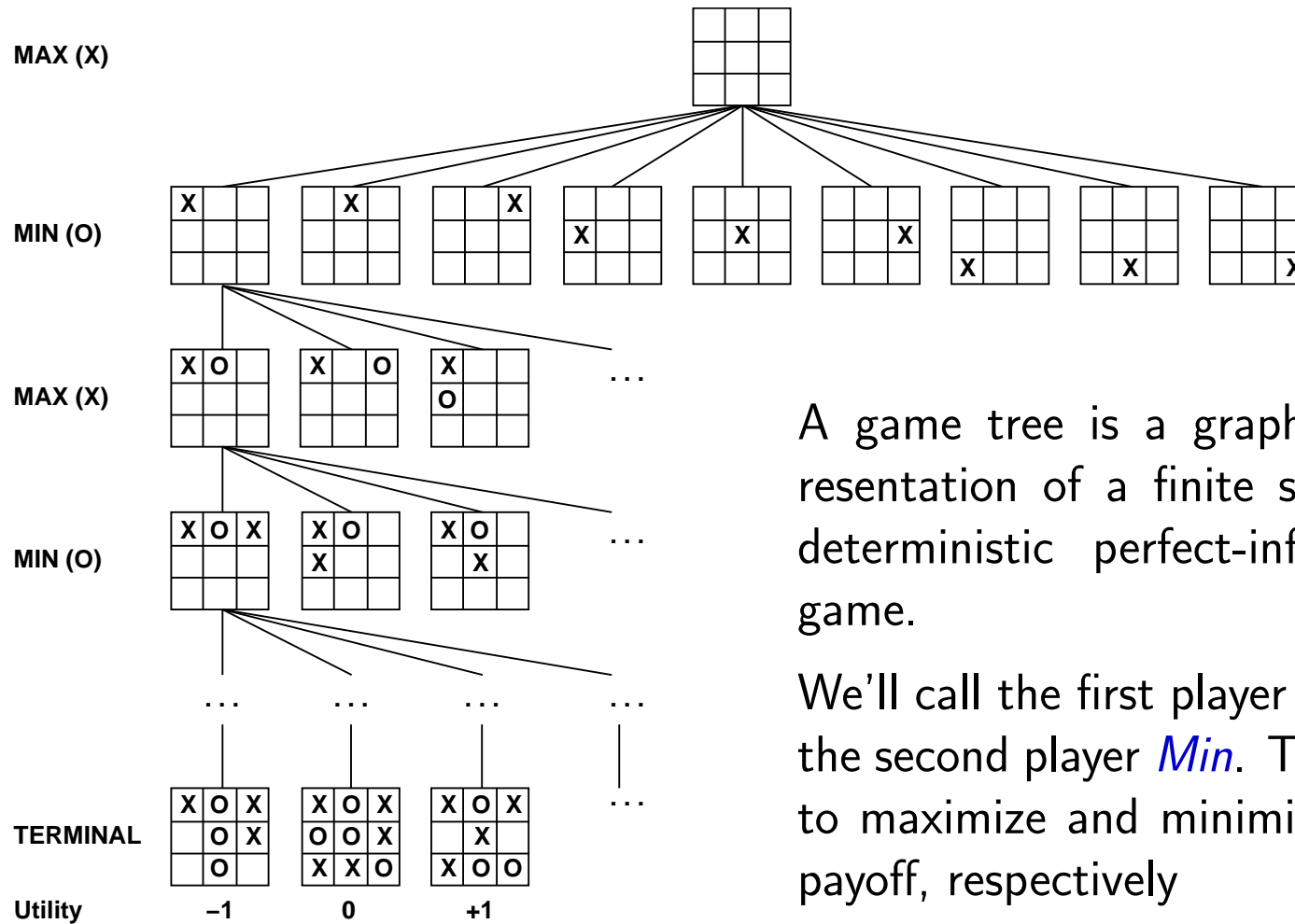
In a zero-sum game, the two decision criteria produce identical results

i.e., $s^* = s^\dagger$ and $t^* = t^\dagger$

Thus if G is a two-person finite zero-sum game between agents A and B ,

$$\begin{aligned} V_G &= U_A(s^*, t^*) = \max_{s \in S} U_A(s, t^*) = \min_{t \in T} U_A(s^*, t) \\ &= -U_B(s^*, t^*) = -\min_{s \in S} U_B(s, t^*) = -\max_{t \in T} U_B(s^*, t) \end{aligned}$$

Game trees

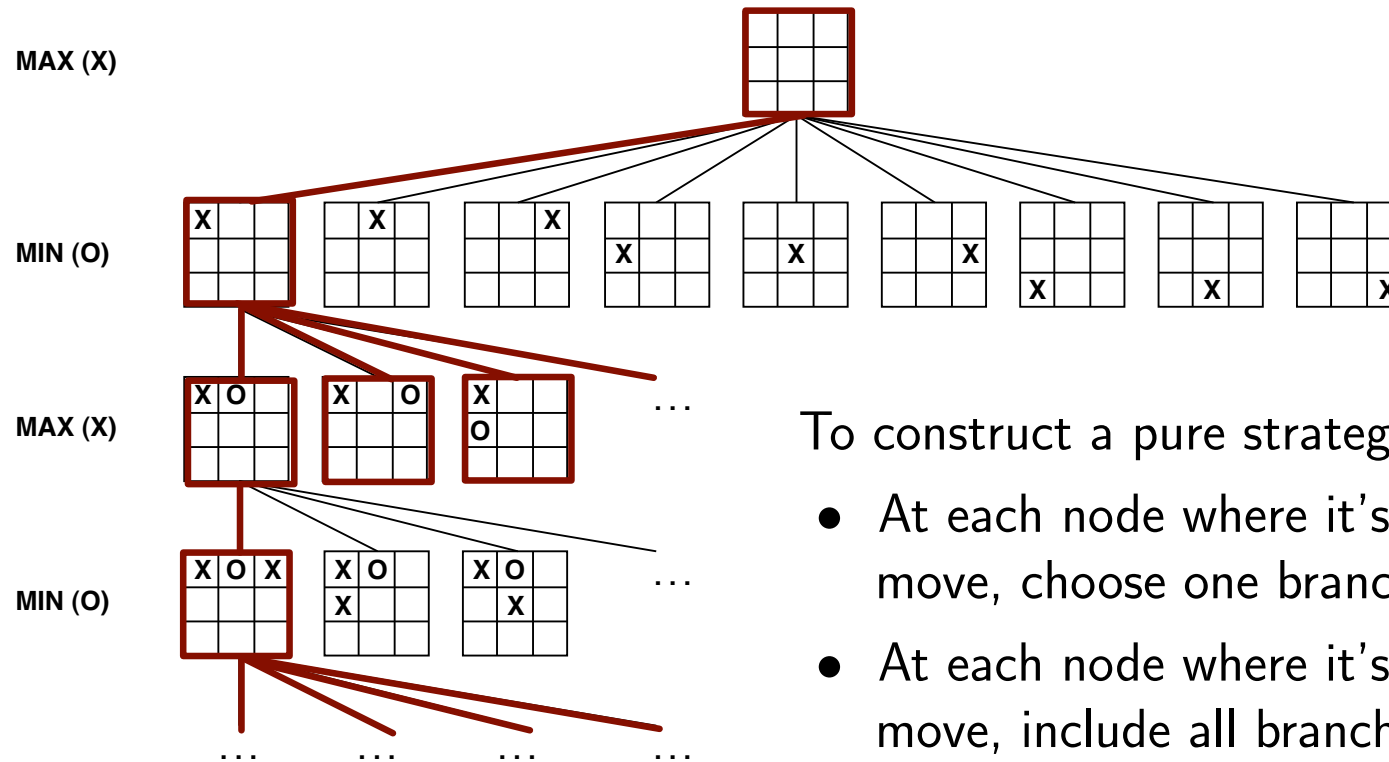


A game tree is a graphical representation of a finite sequential deterministic perfect-information game.

We'll call the first player *Max* and the second player *Min*. They want to maximize and minimize Max's payoff, respectively

Instead of “*game tree*,” game theorists would say “*extensive-form game*.” For each state s , they would call the subtree rooted at s a *subgame*.

Strategies on game trees



To construct a pure strategy for Max:

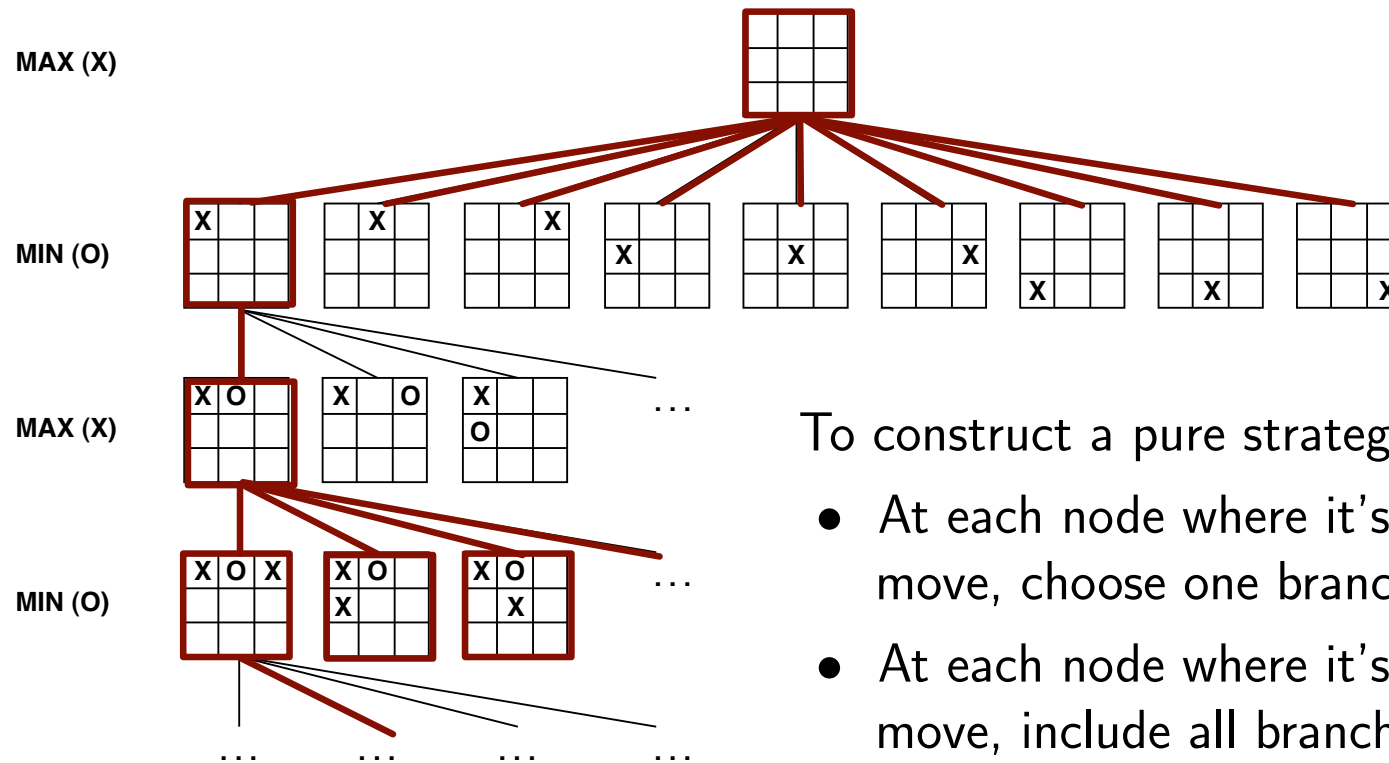
- At each node where it's Max's move, choose one branch
- At each node where it's Min's move, include all branches

Let b = the *branching factor* (max. number of children of any node)

h = the tree's *height* (max. depth of any node)

Then the number of pure strategies for Max is $\leq b^{\lceil h/2 \rceil}$, with equality if every node has b children and every terminal node has depth h

Strategies on game trees



To construct a pure strategy for Min:

- At each node where it's Min's move, choose one branch
- At each node where it's Max's move, include all branches

Number of pure strategies for Min is $\leq b^{\lfloor h/2 \rfloor}$, with equality if every node has b children and every terminal node has depth h

Finding the best strategy

Brute-force way to find Max's and Min's best strategies:

Construct the sets S and T of all of Max's and Min's pure strategies, then choose

$$s^* = \operatorname{argmax}_{s \in S} \min_{t \in T} U_{\text{Max}}(s, t)$$

$$t^* = \operatorname{argmin}_{t \in T} \max_{s \in S} U_{\text{Max}}(s, t)$$

Complexity analysis:

Need to construct and store $O(b^{h/2} + b^{h/2}) = O(b^{h/2})$ strategies

Each strategy is a tree that has $O(b^{h/2})$ nodes

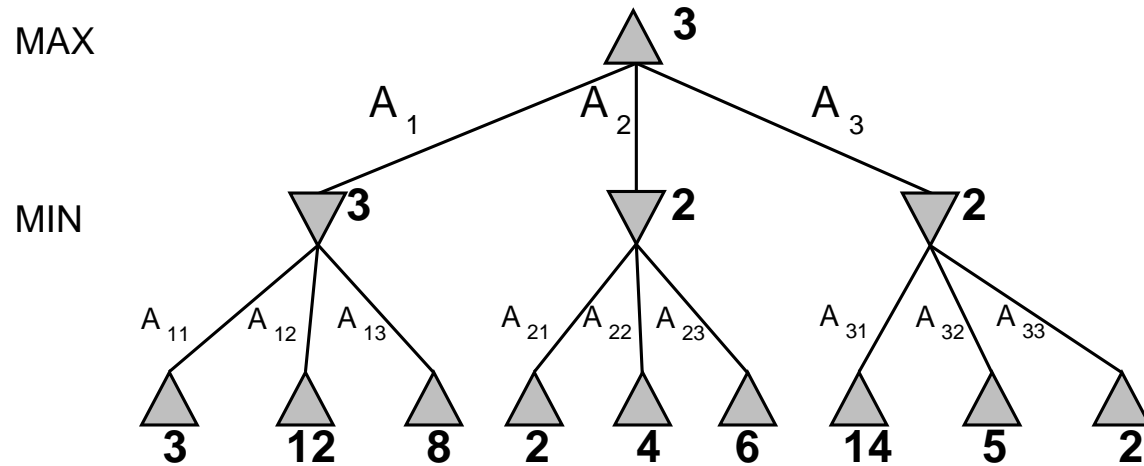
Thus space complexity is $O(b^{h/2}b^{h/2}) = O(b^h)$

Time complexity is slightly worse

But there's an easier way to find the strategies

Minimax Algorithm (Shannon, 1950)

Compute a game's minimax value recursively from the minimax values of its subgames:



```
function MINIMAX(s) returns a utility value
  if s is a terminal state then return Max's payoff at s
  else if it is Max's move in s then
    return max{MINIMAX(result(a, s)) : a is applicable to s}
  else return min{MINIMAX(result(a, s)) : a is applicable to s}
```

To get the next action, return *argmax* and *argmin* instead of *max* and *min*

Properties of the minimax algorithm

Is it sound?

Properties of the minimax algorithm

Is it sound? I.e., when it returns answers, are they correct?

Yes (can prove this by induction)

Is it complete?

Properties of the minimax algorithm

Is it sound? I.e., when it returns answers, are they correct?

Yes (can prove this by induction)

Is it complete? I.e., does it always return an answer when one exists?

Yes on **finite** trees (e.g., chess has specific rules for this).

Space complexity?

Properties of the minimax algorithm

Is it sound? I.e., when it returns answers, are they correct?

Yes (can prove this by induction)

Is it complete? I.e., does it always return an answer when one exists?

Yes on **finite** trees (e.g., chess has specific rules for this).

Space complexity? $O(bh)$, where b and h are as defined earlier

Time complexity?

Properties of the minimax algorithm

Is it sound? I.e., when it returns answers, are they correct?

Yes (can prove this by induction)

Is it complete? I.e., does it always return an answer when one exists?

Yes on **finite** trees (e.g., chess has specific rules for this).

Space complexity? $O(bh)$, where b and h are as defined earlier

Time complexity? $O(b^h)$

For chess, $b \approx 35$, $h \approx 100$ for “reasonable” games

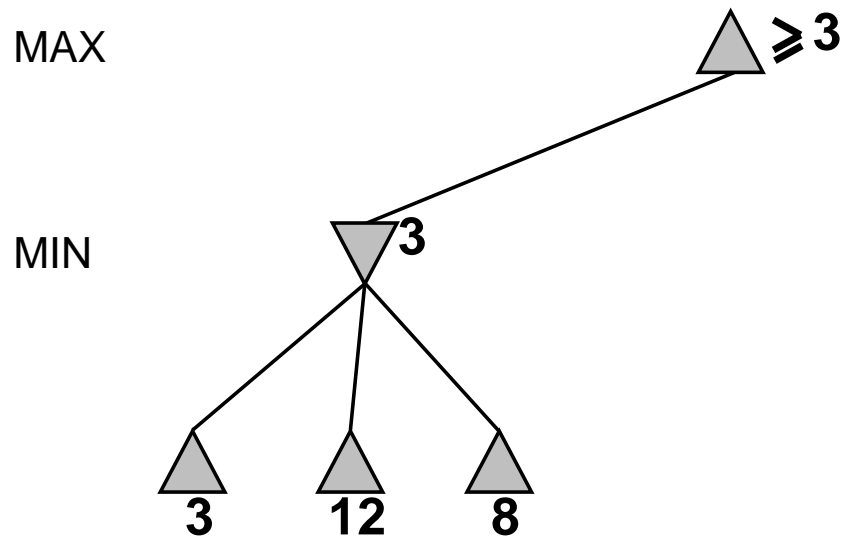
$35^{100} \approx 10^{135}$ nodes

This is about 10^{55} times the number of particles in the universe (about 10^{87})

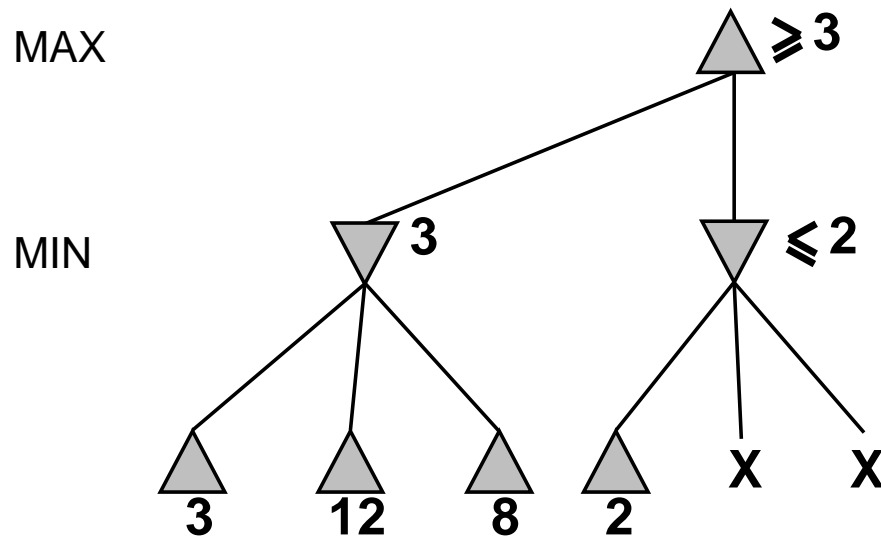
⇒ no way to examine every node!

But do we really need to examine every node?

α - β pruning example 1



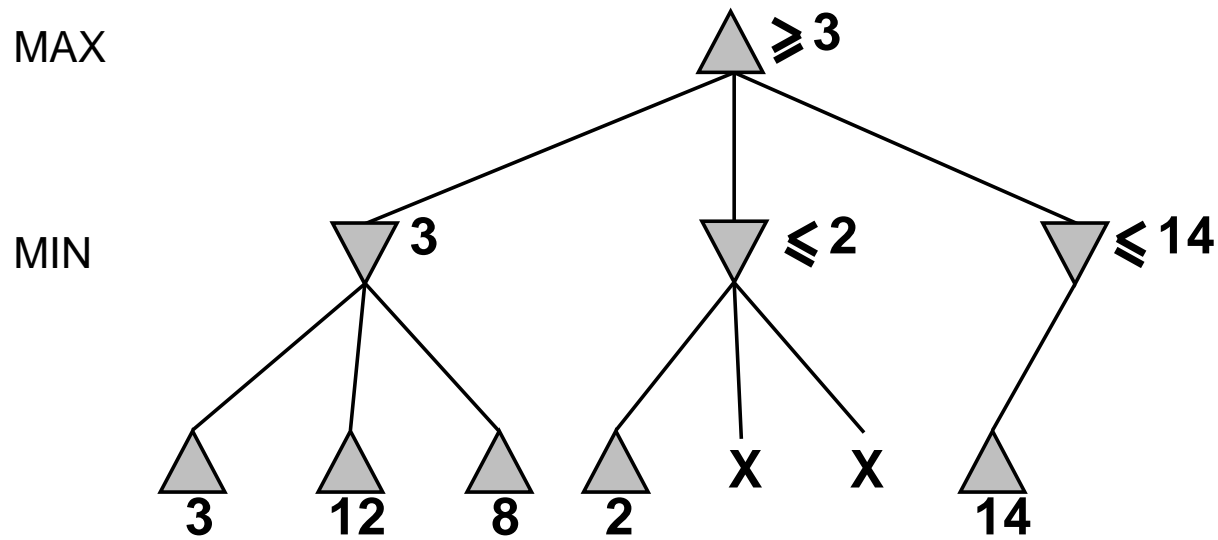
α - β pruning example 1



Max will never move to this node, because Max can do better by moving to the first one

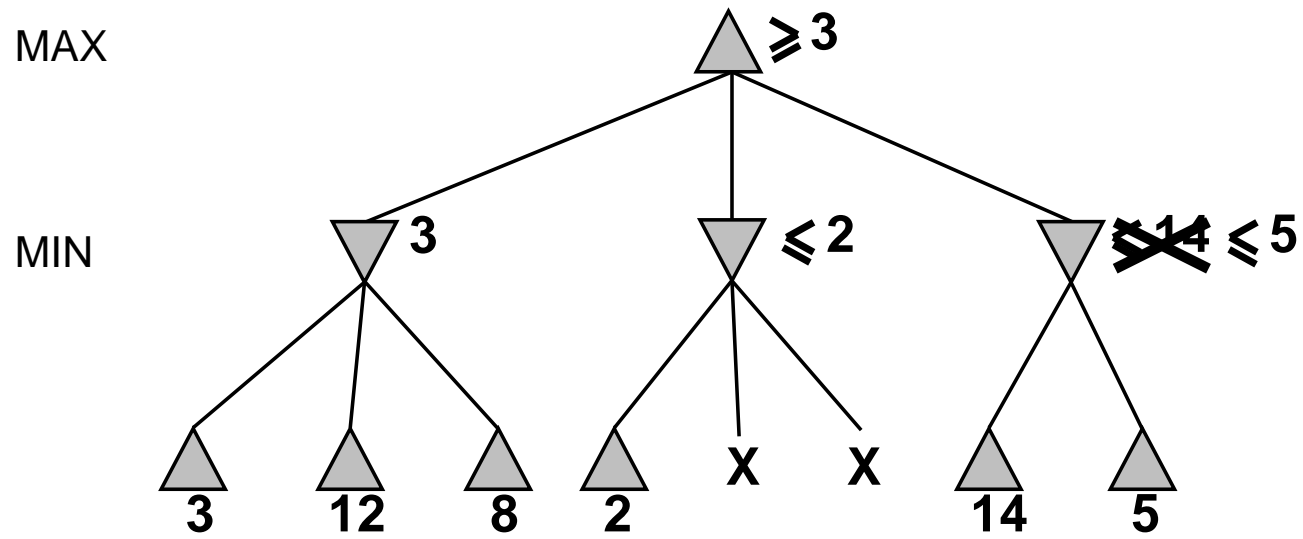
Thus we don't need to figure out this node's minimax value

α - β pruning example 1



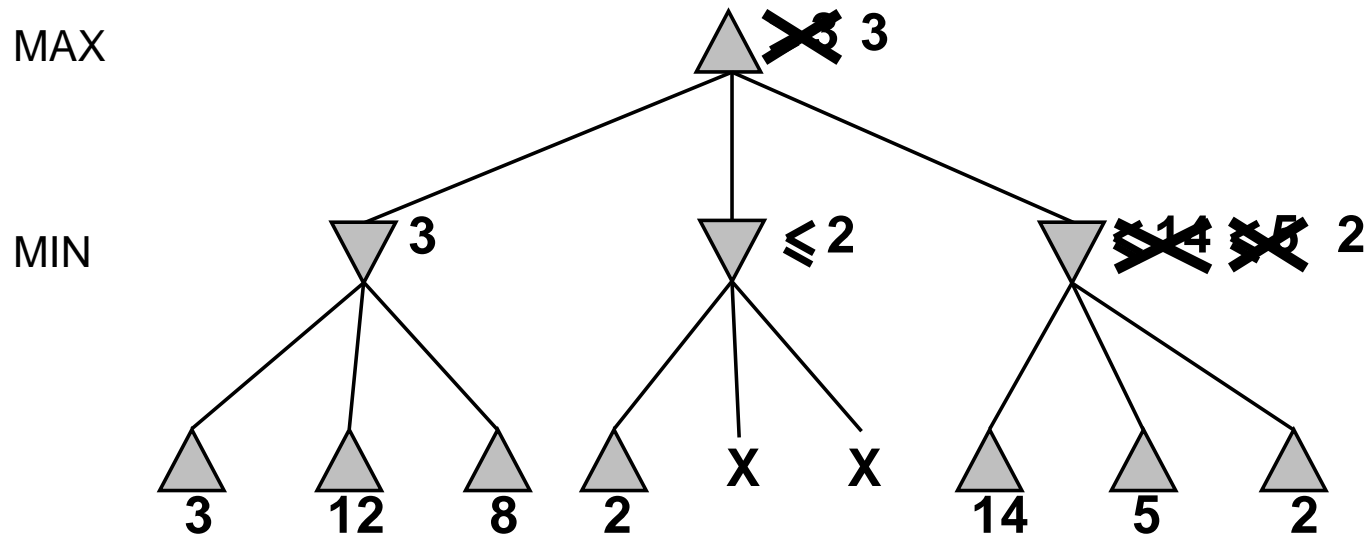
This node might be better than the first one

α - β pruning example 1



It still might be better than the first one

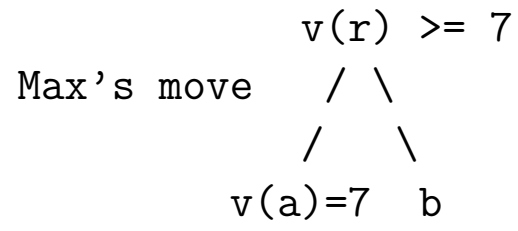
α - β pruning example 1



No, it isn't

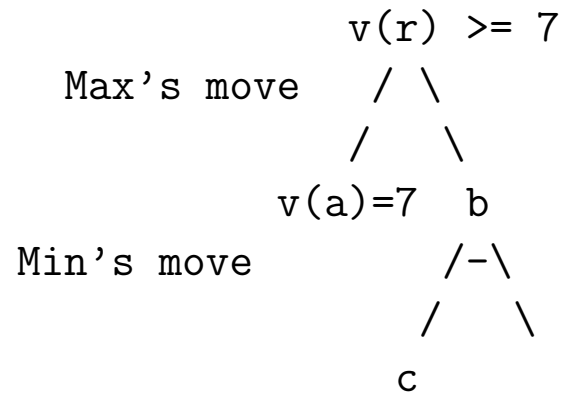
α - β pruning example 2

Same idea works farther down in the tree



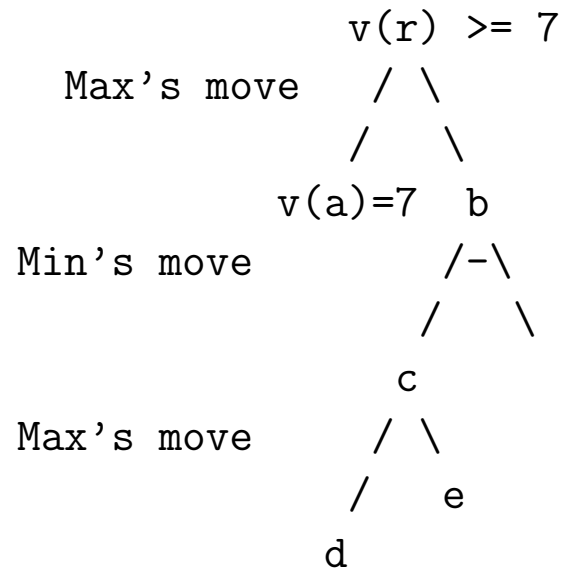
α - β pruning example 2

Same idea works farther down in the tree



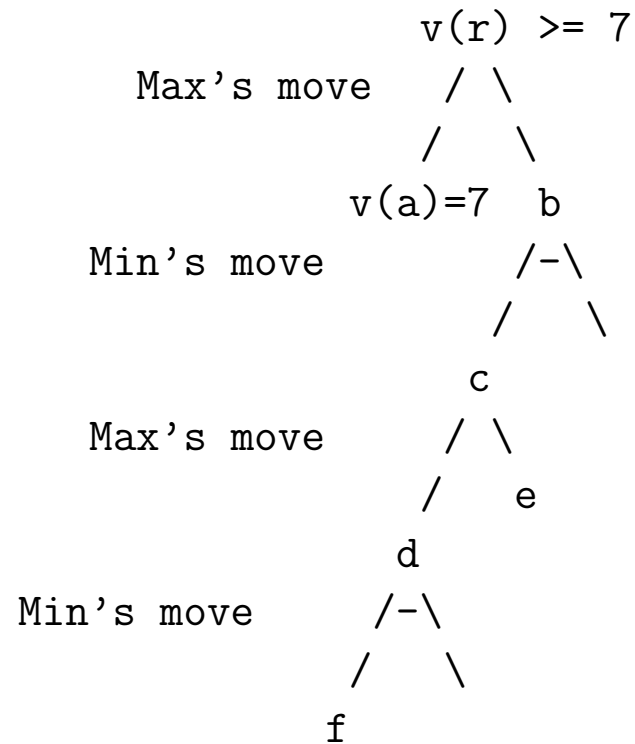
α - β pruning example 2

Same idea works farther down in the tree



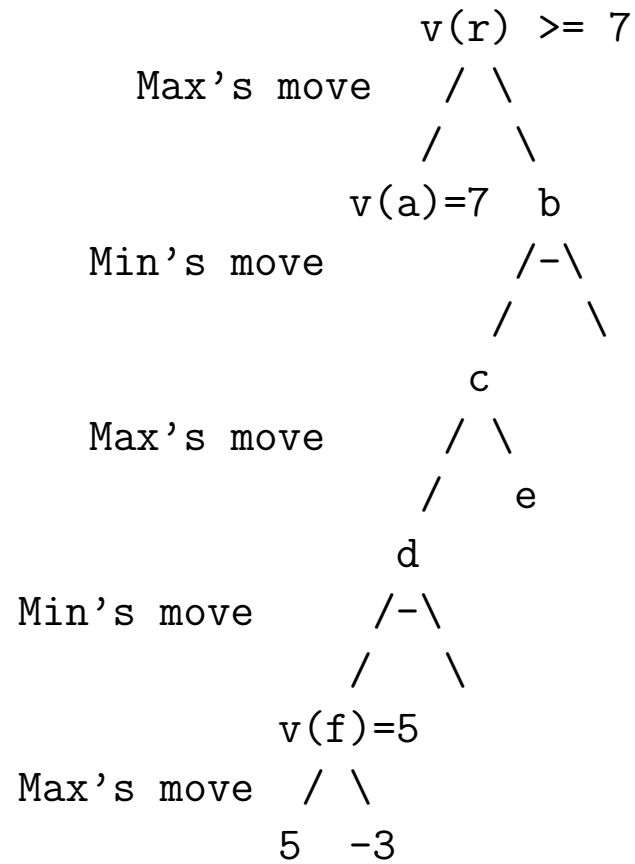
α - β pruning example 2

Same idea works farther down in the tree



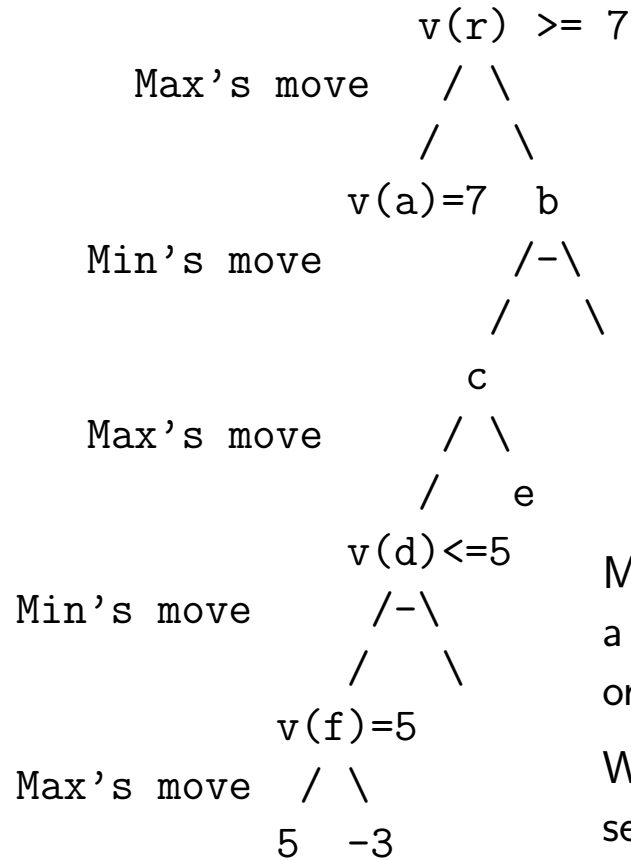
α - β pruning example 2

Same idea works farther down in the tree



α - β pruning example 2

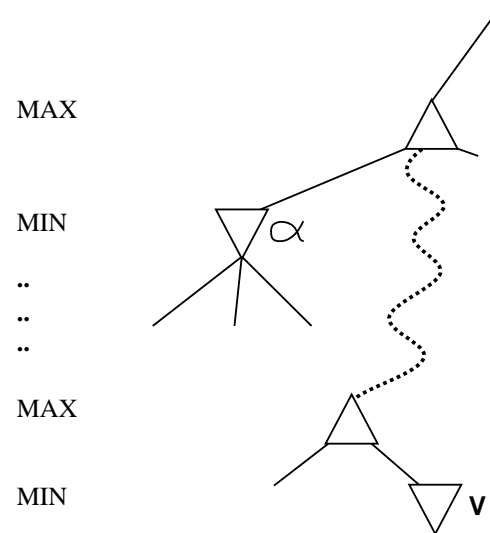
Same idea works farther down in the tree



Max will never move to d , because Max can get a better payoff elsewhere, either by moving to a or maybe by moving to e .

We need to examine e further, but we can stop searching below d .

Why is it called α - β ?



α is the biggest lower bound on the path to the current state s .

If s has $v \leq \alpha$, then Max can get a minimax value $\geq \alpha$ by moving elsewhere

β is the smallest upper bound on the path to the current state s

If s has $v \geq \beta$, then Min can get a minimax value $\leq \beta$ by moving elsewhere

In both cases, s 's minimax value can't affect the value of the root node

\Rightarrow stop searching below s

The α - β algorithm

function ALPHA-BETA(s, α, β) **returns** a utility value

inputs: s , current state in game

α , the value of the best alternative for MAX along the path to s

β , the value of the best alternative for MIN along the path to s

if s is a terminal state **then return** Max's payoff at s

else if it is Max's move at s **then**

$v \leftarrow -\infty$

for every action a applicable to s **do**

$v \leftarrow \max(v, \text{ALPHA-BETA}(\text{result}(a, s), \alpha, \beta))$

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

$\alpha \leftarrow \max(\alpha, v)$

else

$v \leftarrow \infty$

for every action a applicable to s **do**

$v \leftarrow \min(v, \text{ALPHA-BETA}(\text{result}(a, s), \alpha, \beta))$

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

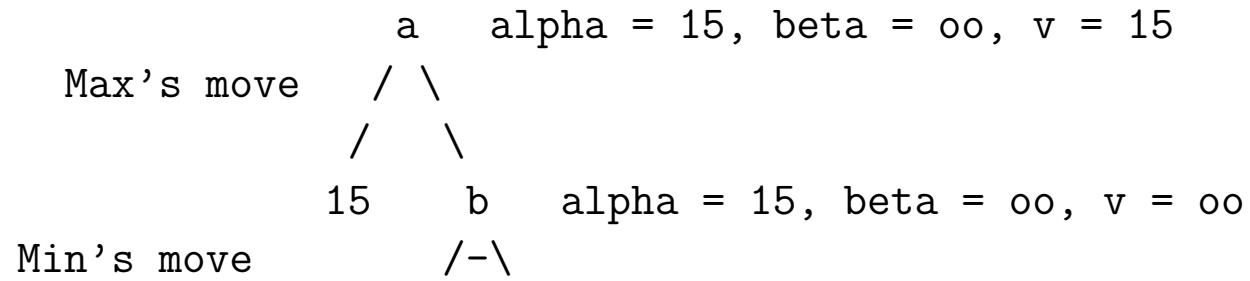
$\beta \leftarrow \min(\beta, v)$

return v

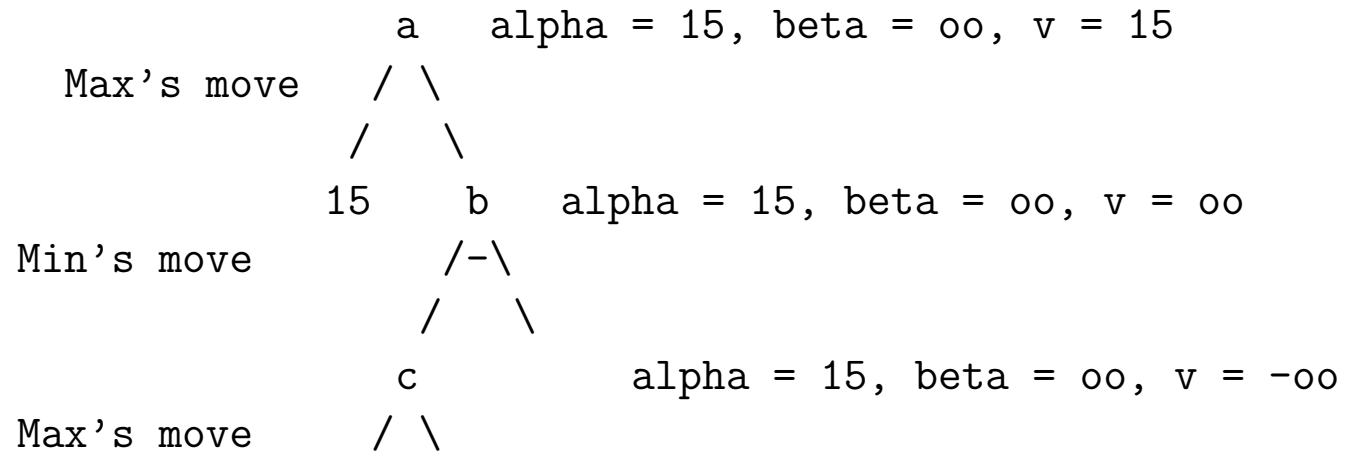
α - β pruning example

Max's move a alpha = $-\infty$, beta = ∞ , v = $-\infty$
 / \

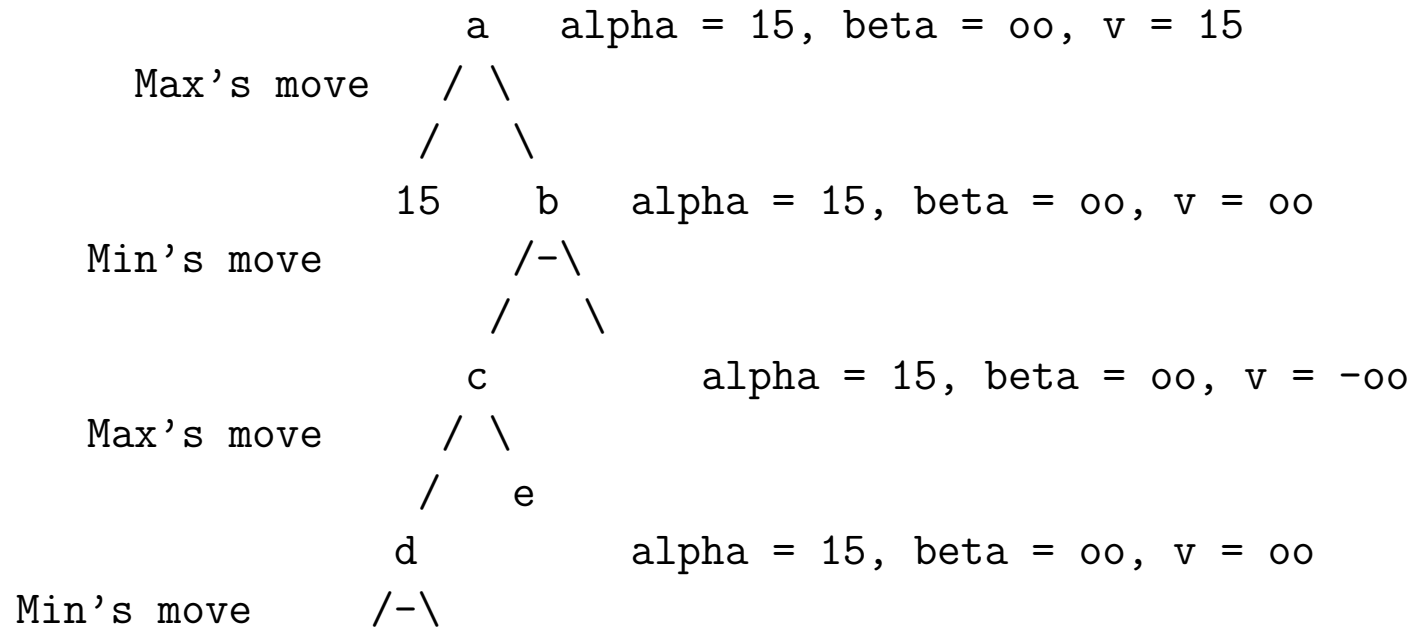
α - β pruning example



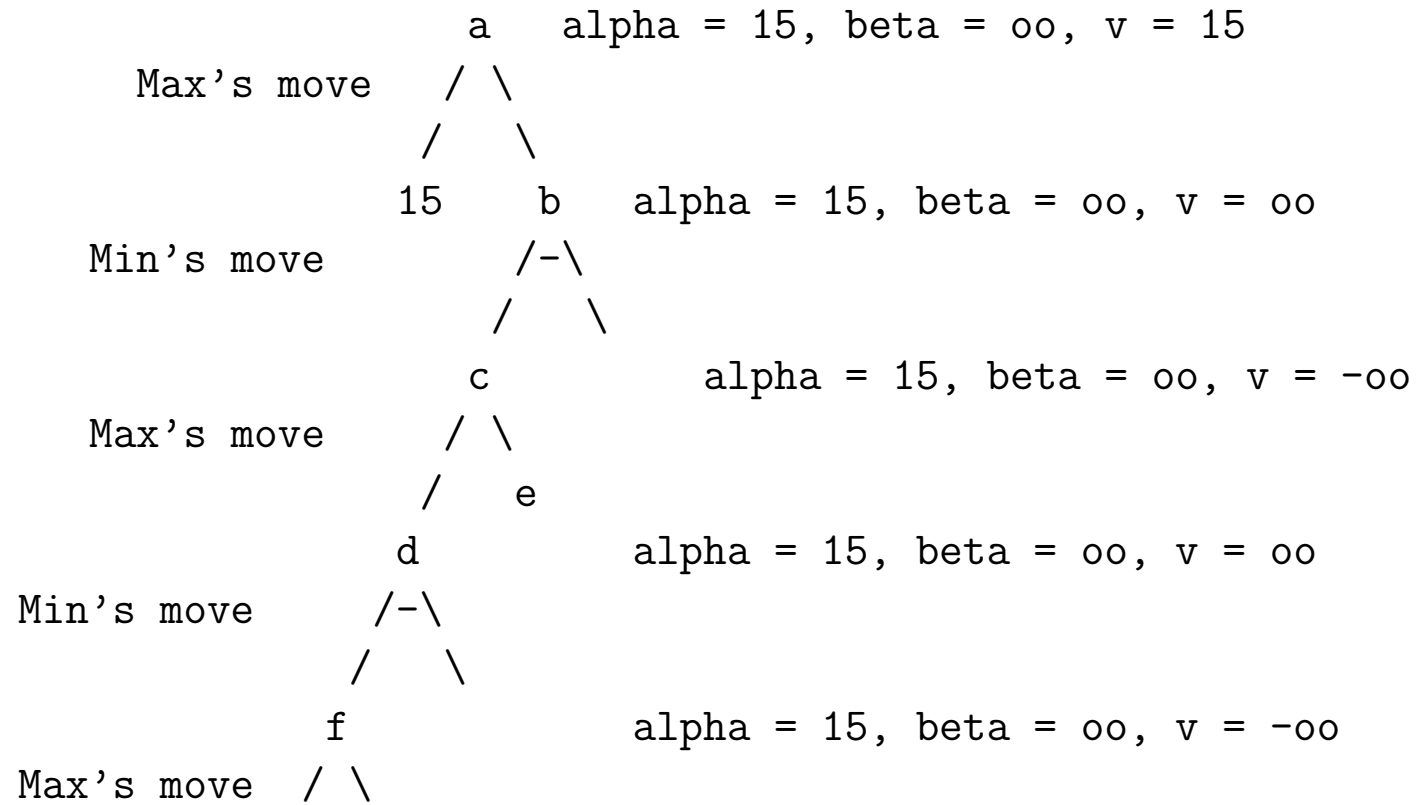
α - β pruning example



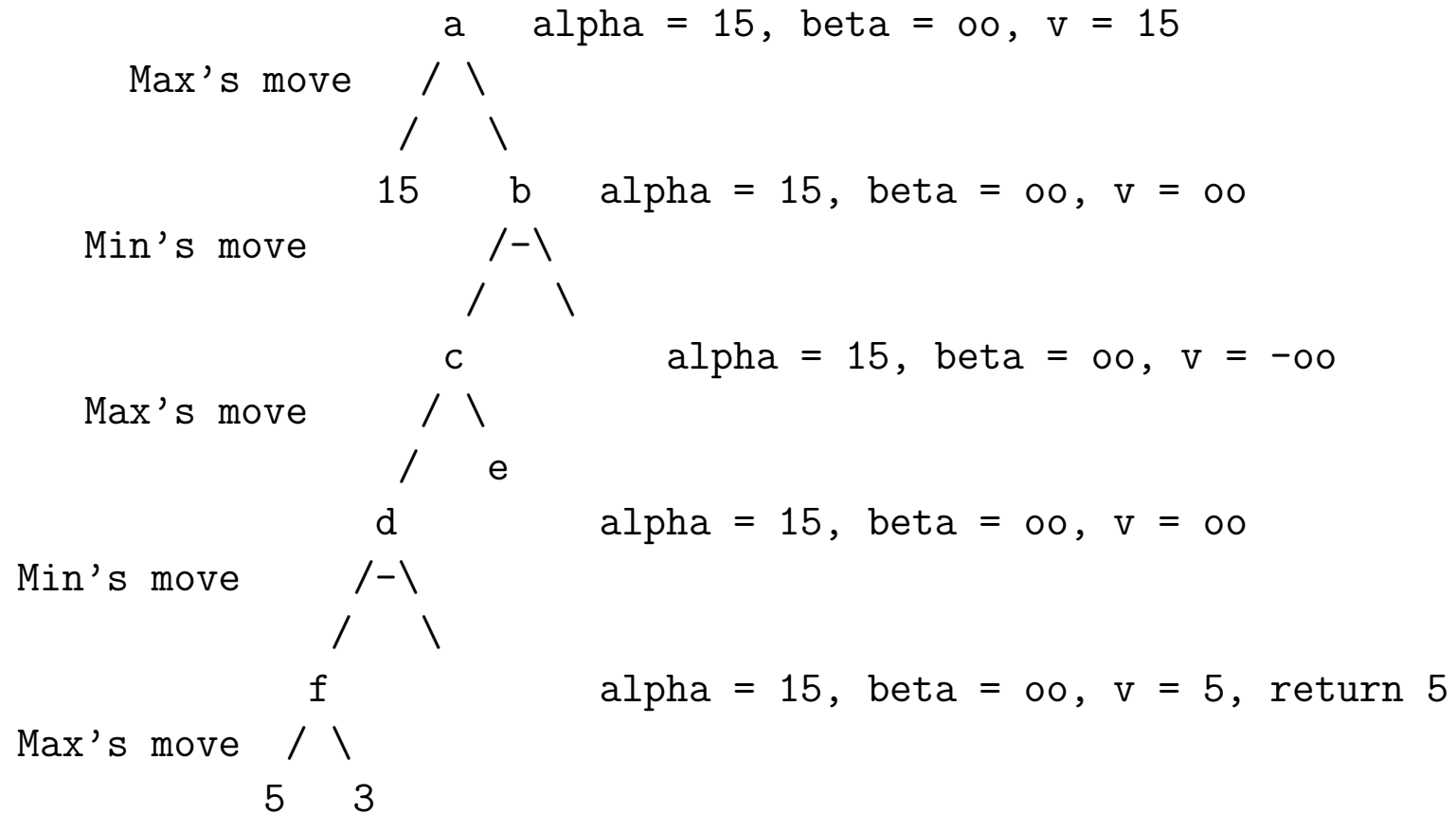
α - β pruning example



α - β pruning example



α - β pruning example



Properties of α - β

α - β is a simple example of the value of reasoning about which computations are relevant (a form of **metareasoning**)

- ◇ if $\alpha \leq \text{minimax}(s) \leq \beta$, then alpha-beta returns $\text{minimax}(s)$
- ◇ if $\text{minimax}(s) \leq \alpha$, then alpha-beta returns a value $\leq \alpha$
- ◇ if $\text{minimax}(s) \geq \beta$, then alpha-beta returns a value $\geq \beta$

If we start with $\alpha = -\infty$ and $\beta = \infty$, then
alpha-beta will always return $\text{minimax}(s)$

Good move ordering can enable us to prune more nodes.

Best case is if

- ◇ at nodes where it's Max's move, children are largest-value first
- ◇ at nodes where it's Min's move, children are smallest-value first

In this case time complexity = $O(b^{h/2}) \Rightarrow$ doubles the solvable depth

Worst case is the reverse

In this case, α - β will search every node

Resource limits

Even with alpha-beta, it can still be infeasible to search the entire game tree (e.g., recall chess has about 10^{135} nodes)

⇒ need to limit the depth of the search

Basic approach: let d be a positive integer

Whenever we reach a node of depth $> d$

- If we're at a terminal state, then return Max's payoff
- Otherwise return an *estimate* of the node's utility value, computed by a **static evaluation function**

α - β with a bound d on the search depth

```
function ALPHA-BETA( $s, \alpha, \beta, d$ ) returns a utility value
  inputs:  $s, \alpha, \beta$ , same as before
          $d$ , an upper bound on the search depth

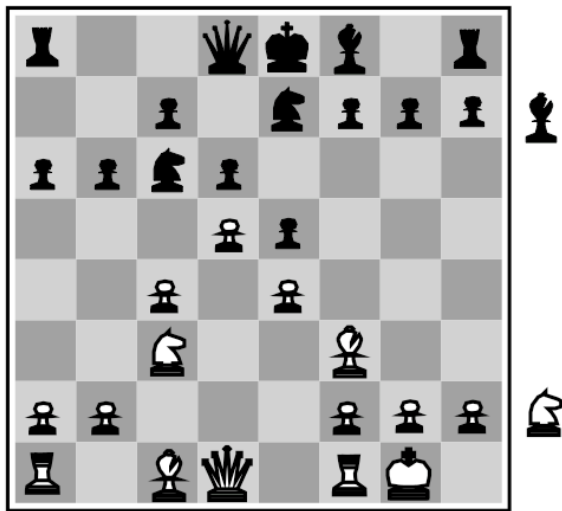
  if  $s$  is a terminal state then return Max's payoff at  $s$ 
  else if  $d = 0$  then return EVAL( $s$ )
  else if it is Max's move at  $s$  then
     $v \leftarrow -\infty$ 
    for every action  $a$  applicable to  $s$  do
       $v \leftarrow \max(v, \text{ALPHA-BETA}(\text{result}(a, s), \alpha, \beta, d - 1))$ 
      if  $v \geq \beta$  then return  $v$ 
       $\alpha \leftarrow \max(\alpha, v)$ 
  else
     $v \leftarrow \infty$ 
    for every action  $a$  applicable to  $s$  do
       $v \leftarrow \min(v, \text{ALPHA-BETA}(\text{result}(a, s), \alpha, \beta, d - 1))$ 
      if  $v \leq \alpha$  then return  $v$ 
       $\beta \leftarrow \min(\beta, v)$ 
  return  $v$ 
```

Evaluation functions

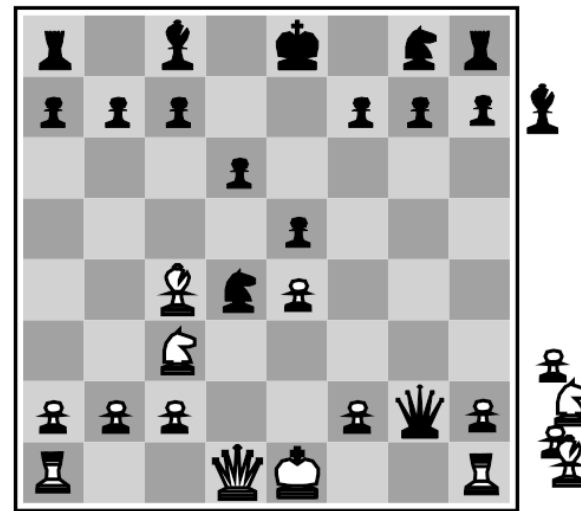
$EVAL(s)$ is supposed to return an approximation of s 's minimax value

$EVAL$ is often a **weighted sum** of *features*

$$EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$



Black to move
White slightly better

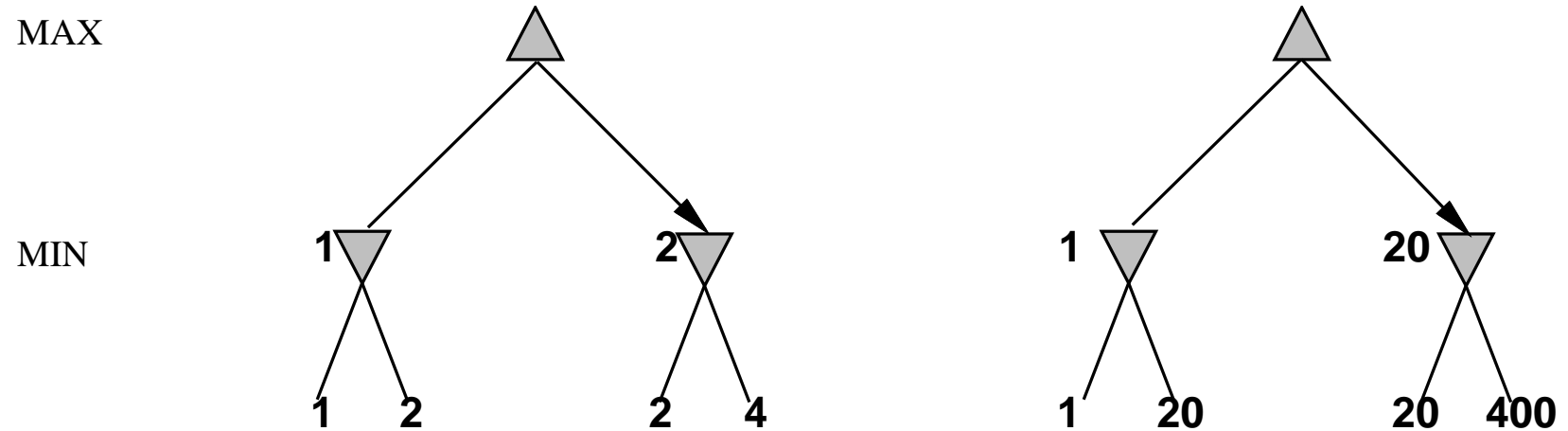


White to move
Black winning

E.g.,

$$\begin{aligned} & 1(\text{number of white pawns} - \text{number of black pawns}) \\ & + 3(\text{number of white knights} - \text{number of black knights}) \\ & + \dots \end{aligned}$$

Exact values for E_{VAL} don't matter



Behavior is preserved under any **monotonic** transformation of E_{VAL}

Only the order matters:

In deterministic games, payoff acts as an *ordinal utility* function

Discussion

Deeper lookahead (i.e., larger depth bound d) usually gives better decisions

Exceptions do exist

Main result in my PhD dissertation (30 years ago!):

“pathological” games in which deeper lookahead gives worse decisions

But such games hardly ever occur in practice

Suppose we have 100 seconds, explore 10^4 nodes/second

$\Rightarrow 10^6 \approx 35^{8/2}$ nodes per move

$\Rightarrow \alpha\text{-}\beta$ reaches depth 8 \Rightarrow pretty good chess program

Some modifications that can improve the accuracy or computation time:

node ordering (see next slide)

quiescence search

biasing

transposition tables

thinking on the opponent's time

...

Node ordering

Recall that I said:

Best case is if

- ◇ at nodes where it's Max's move, children are largest first
- ◇ at nodes where it's Min's move, children are smallest first

In this case time complexity = $O(b^{h/2}) \Rightarrow$ doubles the solvable depth

Worst case is the reverse

How to get closer to the best case:

- ◇ Every time you expand a state s , apply **EVAL** to its children
- ◇ When it's Max's move, sort the children in order of largest **EVAL** first
- ◇ When it's Min's move, sort the children in order of smallest **EVAL** first

Deterministic games in practice

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.

Checkers was **solved** in April 2007: with perfect play, both players can guarantee a draw. This took 10^{14} calculations over 18 years. Checkers has a search space of size 5×10^{20} .

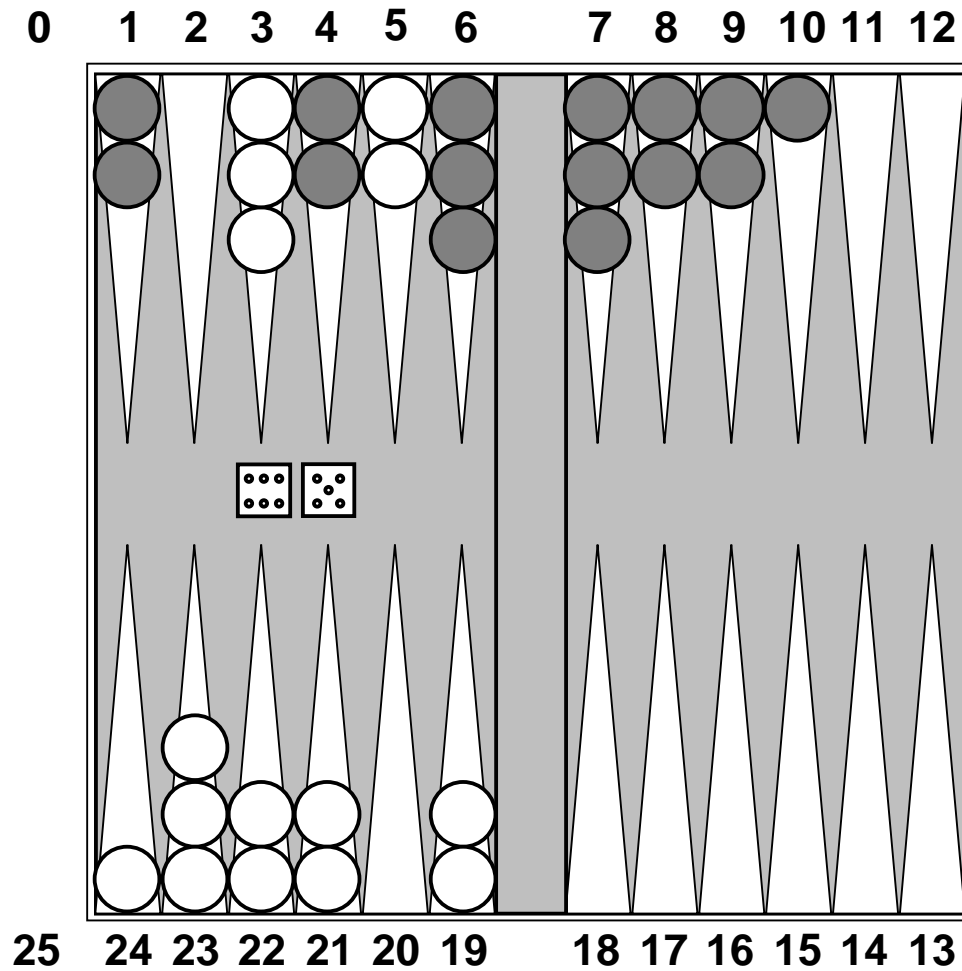
Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

Othello: human champions refuse to compete against computers, which are too good.

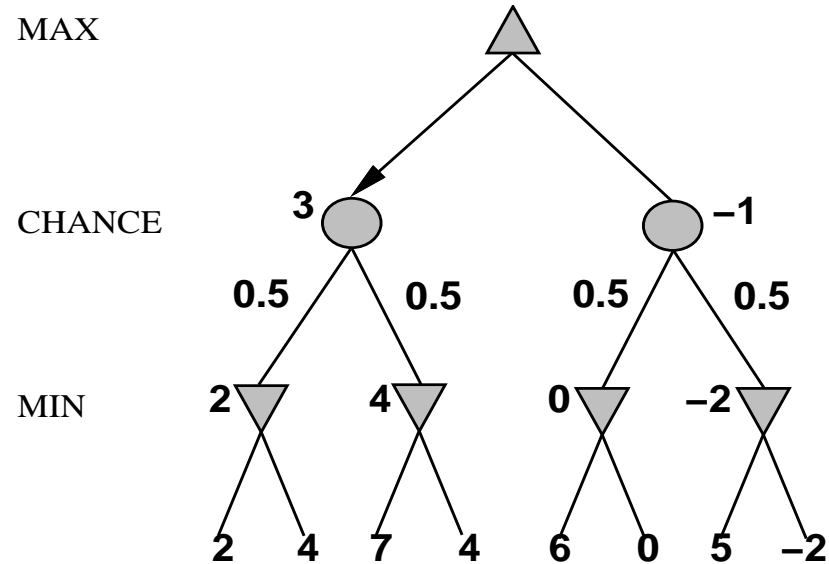
Go: human champions refuse to compete against computers, which are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

Perfect-information nondeterministic games

Backgammon: chance is introduced by dice



Expectiminimax

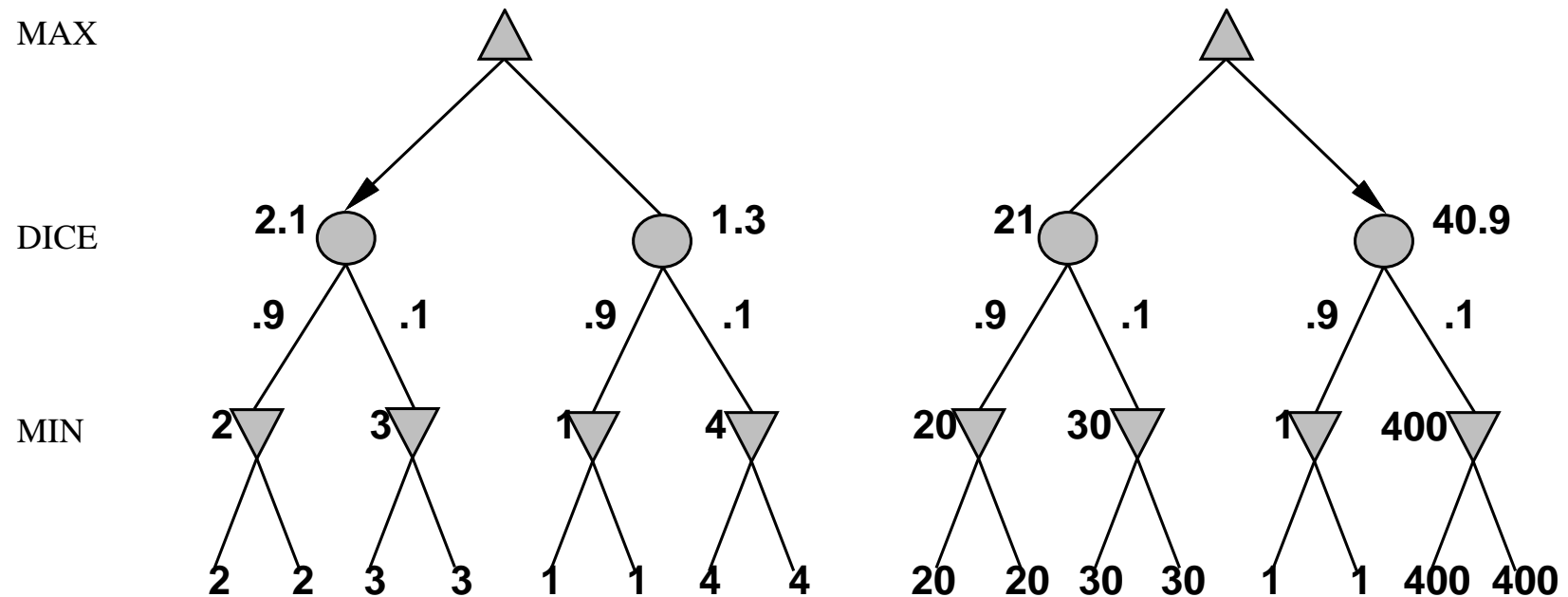


```

function EXPECTIMINIMAX( $s$ ) returns an expected utility
  if  $s$  is a terminal state then return Max's payoff at  $s$ 
  if  $s$  is a "chance" node then
    return  $\sum_{s'} P(s'|s) \text{EXPECTIMINIMAX}(s')$ 
  else if it is Max's move at  $s$  then
    return  $\max\{\text{EXPECTIMINIMAX}(\text{result}(a, s)) : a \text{ is applicable to } s\}$ 
  else return  $\min\{\text{EXPECTIMINIMAX}(\text{result}(a, s)) : a \text{ is applicable to } s\}$ 
  
```

This gives optimal play (i.e., highest expected utility)

With nondeterminism, exact values do matter



At “chance” nodes, we need to compute weighted averages

Behavior is preserved only by **positive linear** transformations of EV_{AL}
Hence EV_{AL} should be proportional to the expected payoff

In practice

Dice rolls increase b : 21 possible rolls with 2 dice
Given the dice roll, ≈ 20 legal moves on average
(for some dice roles, can be much higher)

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

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

Summary

If a game is two-player zero-sum, then maximin and minimax are the same

If the game also is perfect-information, only need to look at pure strategies

If the game also is sequential, deterministic, and finite, then can do a game-tree search

minimax values, alpha-beta pruning

In sufficiently complicated games, perfection is unattainable

⇒ must approximate: limited search depth, static evaluation function

For nondeterminism, incorporate “chance” nodes into the game tree

I didn't discuss card games because they have both chance and imperfect information

To get game tree search to work there, would need some changes

For next time

Next time, we'll start discussing non-zero-sum games, equilibria, repeated games

Homework assignment

Before noon on Thursday, initiate an email discussion with me (if I haven't done so already) about

- ◇ A topic on which you'd like to lead a class discussion
- ◇ Some possible source materials (papers, textbook chapter)

By next Tuesday, I'd like to have an agreement with you about what the topic will be, and a rough idea of what the source materials will be