Applied AI

Lecture 4

Dr Artie Basukoski

[slides adapted from Artificial Intelligence: A Modern Approach, Russel and Norvigl]

Agenda

- Games – Adversarial Search
- The AI Behind Deep Blue (Beat World Chess champion ’97)
- Minimax algorithm
- Alpha beta pruning (α–β *pruning*)
- Limiting depth
- Games of Chance
- Imperfect Information

Types of games

- Deterministic and stochastic
- One, two, or more players
- Zero sum or cooperative?
- Perfect information games (you can see the whole state)
- Need an Algorithm to calculate next move at each state.

The AI Behind Deep Blue

Deep Blue vs. Kasparov

Deep Blue

Garry Kasparov

Second match (rematch)

- May 3–11, 1997: held in New York City, New York
- Result: **Deep Blue**–Kasparov (3½–2½)
- Record set: First computer program to defeat a world champion in a *match* under tournament regulations

Games vs. search problems

"Unpredictable" opponent ⇒ solution is a strategy
specifying a move for every possible opponent reply

Time limits ⇒ unlikely to find goal, must approximate

History:

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)

Shannon, 1950)

- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)
- Pruning to allow deeper search (McCarthy, 1956)

Types of games

	Deterministic	**Chance**
Perfect Information	Chess,checkers, go, Othello	Backgammon, monopoly
Imperfect Information	Battleships, blind tictactoe	Bridge, poker, scrabble, nuclear war

2 player zero-sum games

- The games most commonly studied within AI
- Deterministic
- 2 player (we call the two players Max and Min)
- Turn taking
- Perfect information
- Zero-sum games
  - (means what is good for one player is bad for the other)
- Max and Min each compete to find a sequence of actions leading to a win.
- Such as TicTacToe, Chess and Go

Game tree (2-player, deterministic, turns)

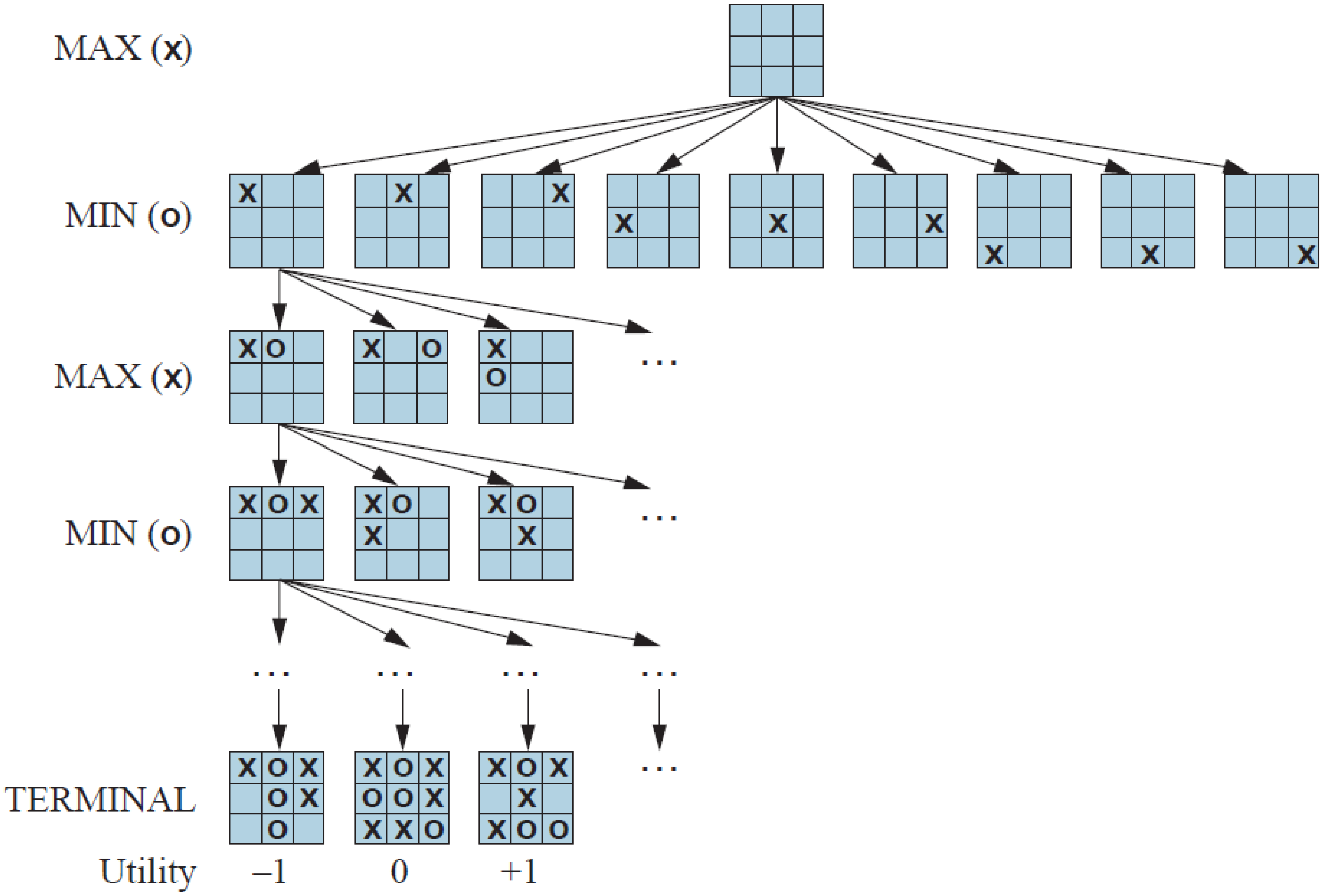


Figure 6.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square. We show part of the tree, giving alternating moves by MIN (O) and MAX (X), until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.

Minimax overview

- S0: The **initial state**, which specifies 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**, which is true when the game is over and false otherwise. States where the game has ended are called **terminal states**.
- UTILITY(s, p): A **utility function** (also called an objective function or payoff function), defines the final numeric value for a game that ends in terminal state s for a player p.

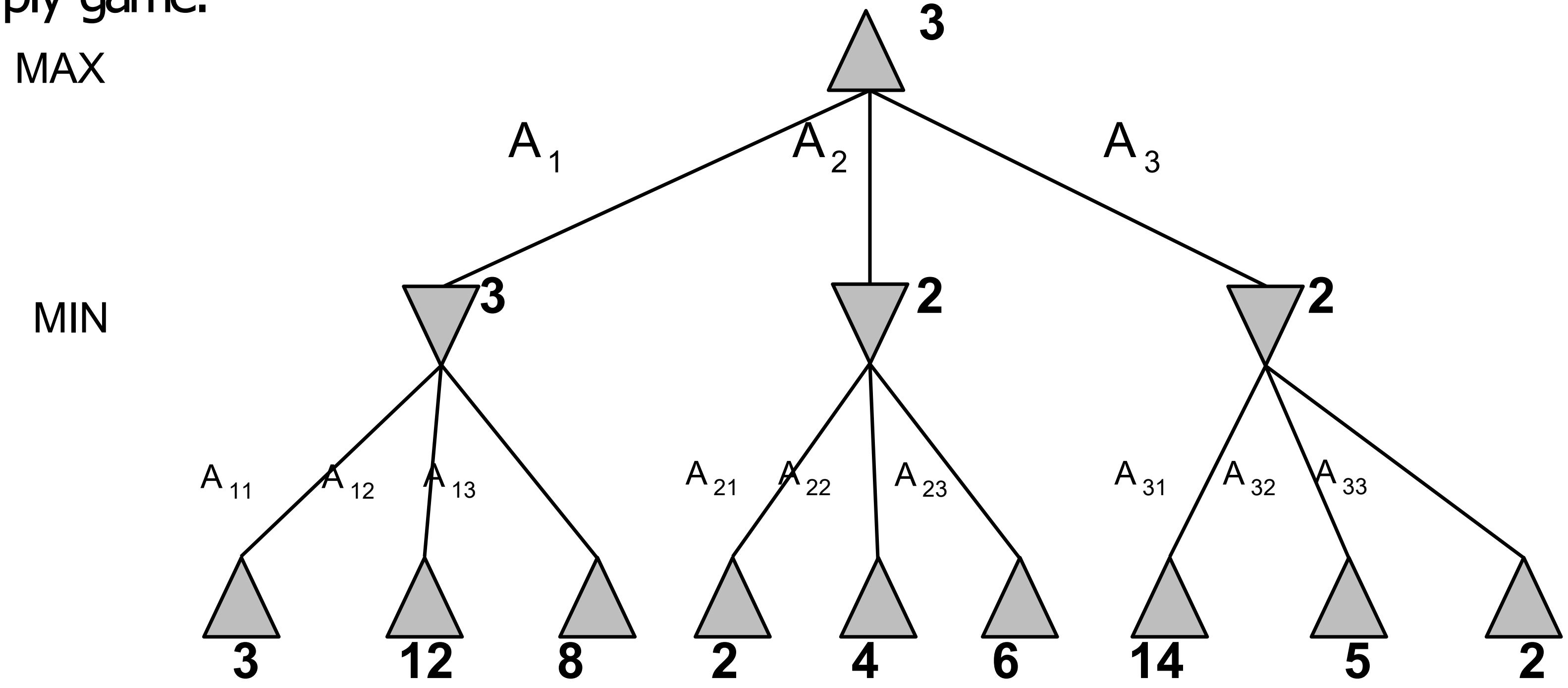
$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s)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

Minimax

Perfect play for deterministic, perfect-information 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 (chess has specific rules for this)

Optimal: Yes, against an optimal opponent. Otherwise??

Time complexity: $O(b^m)$

Space complexity: $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games

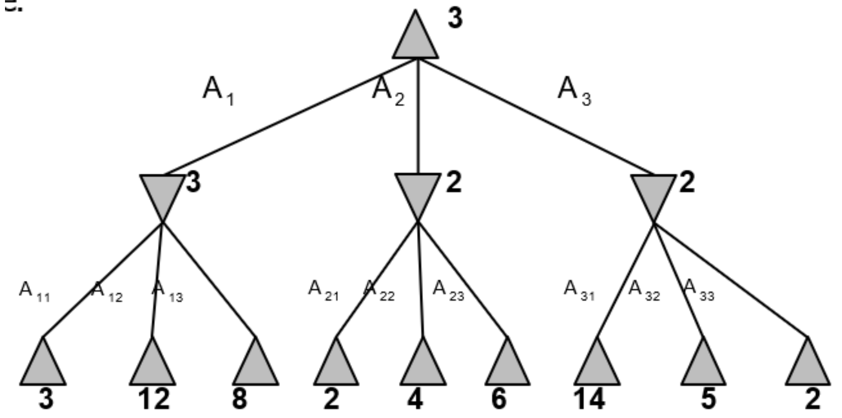
$\Rightarrow$ exact solution completely infeasible

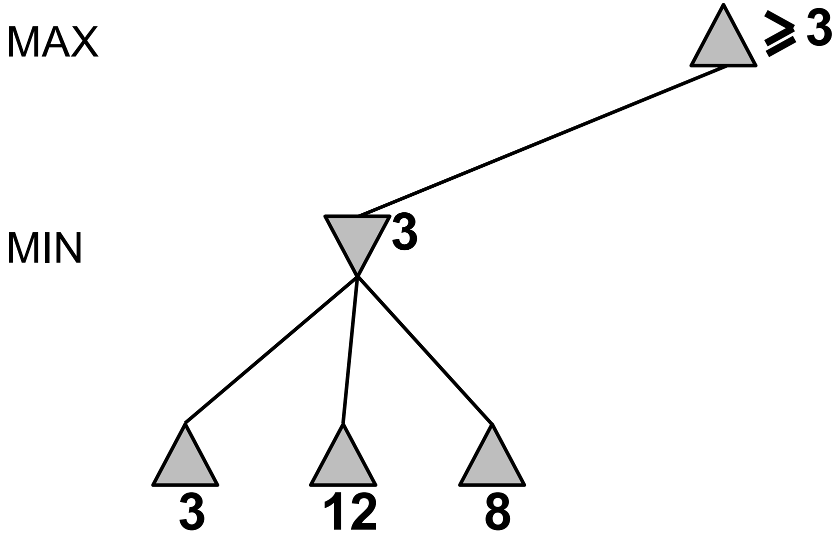
But do we need to explore every path?

α–β pruning

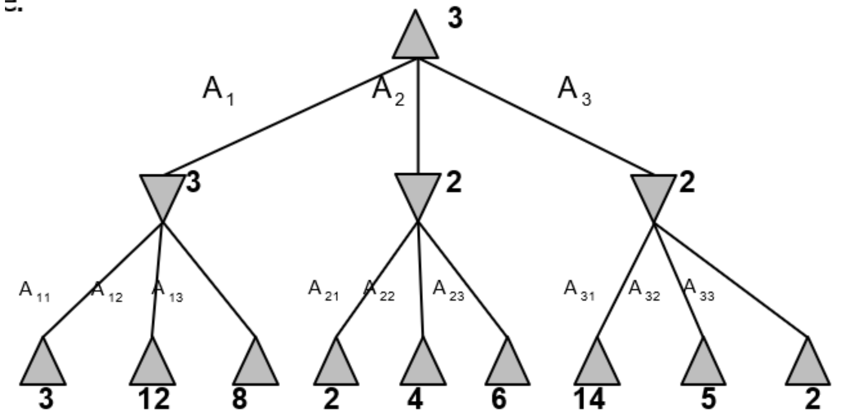
- The problem with minimax search is that the number of game states it has to examine is exponential and the depth of the tree.
- We can't eliminate the exponent, but we can reduce it by half (on average).
- α–β pruning can eliminate leaves and entire subtrees.
  - *α is Max's best value on path to root*
  - *β is Min's best value on path to root*
- Pruning has no effect on the minimax value computed for the root.
- Branches that were pruned would have wasted time without being relevant to the search.

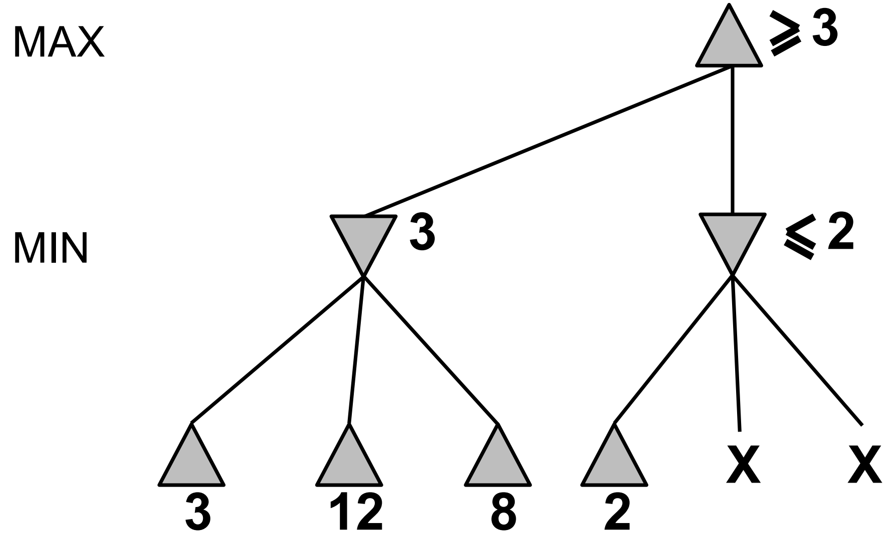
α–β pruning example



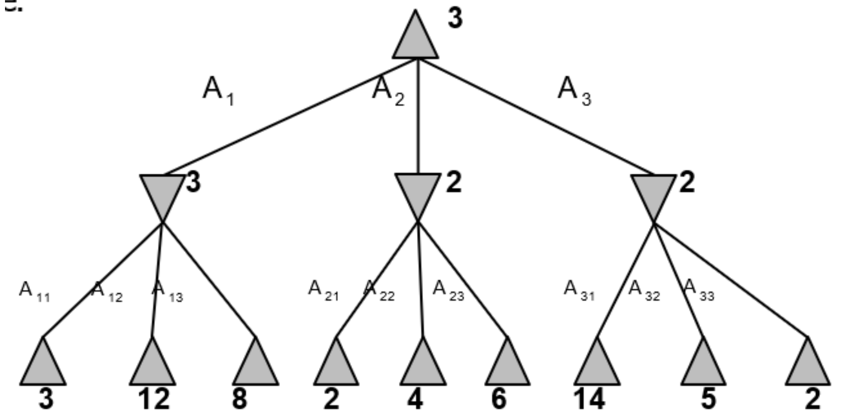


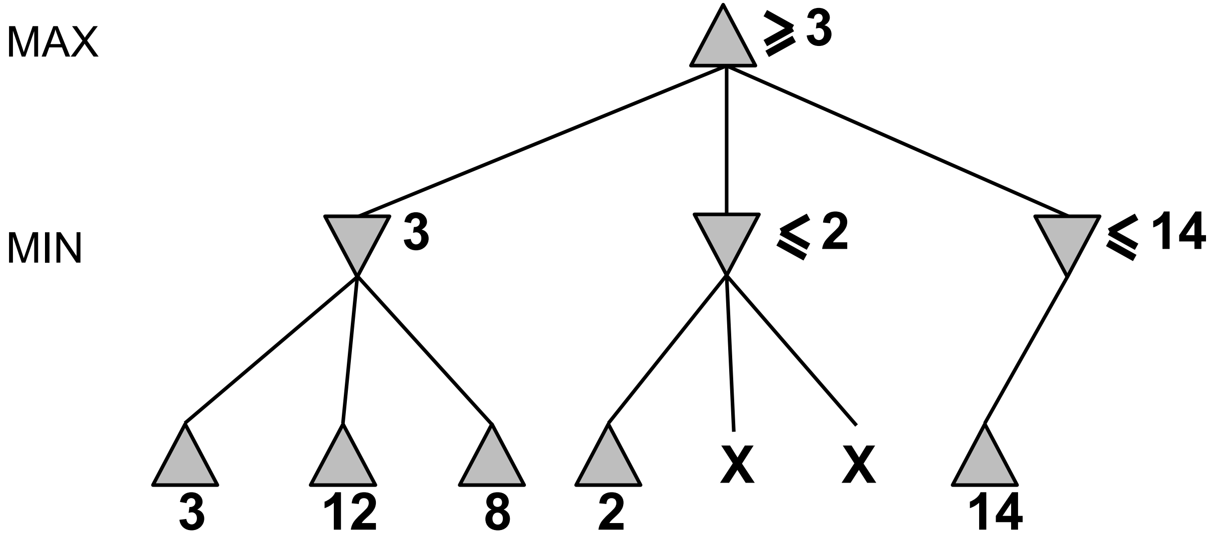
α–β pruning example



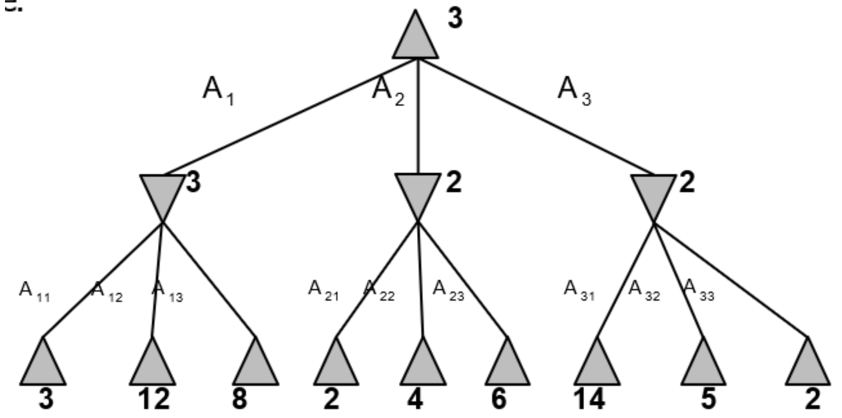


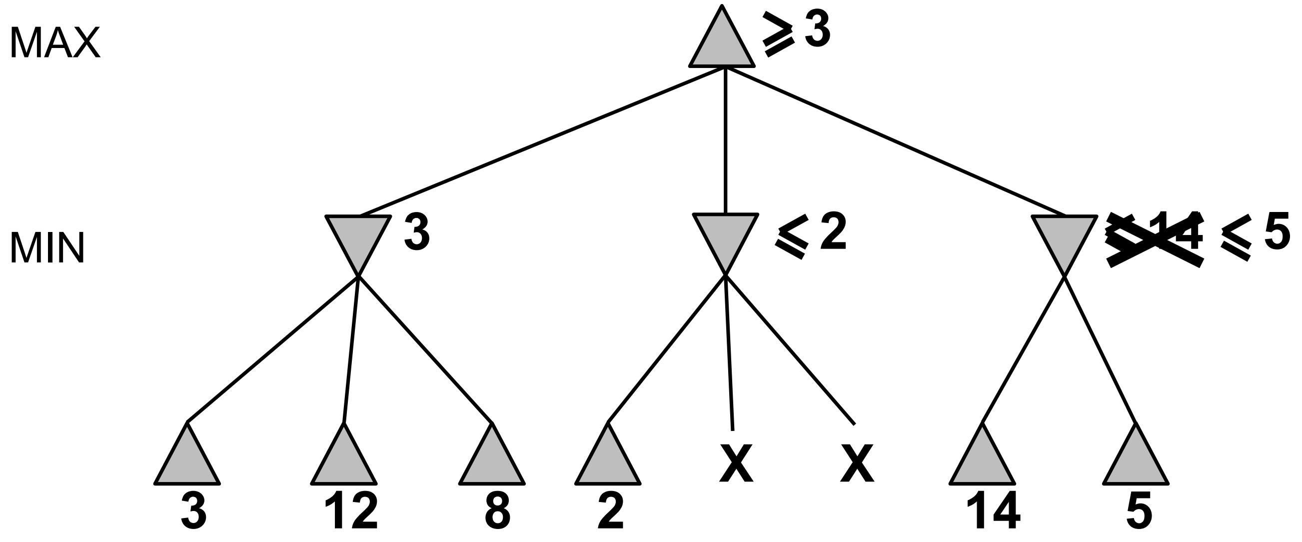
α–β pruning example



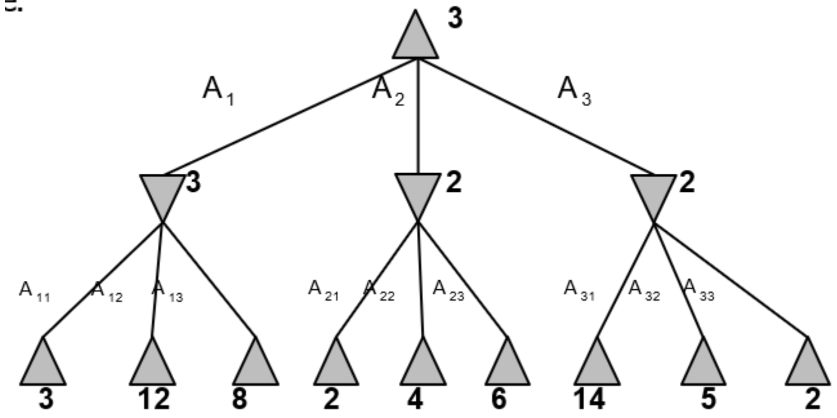


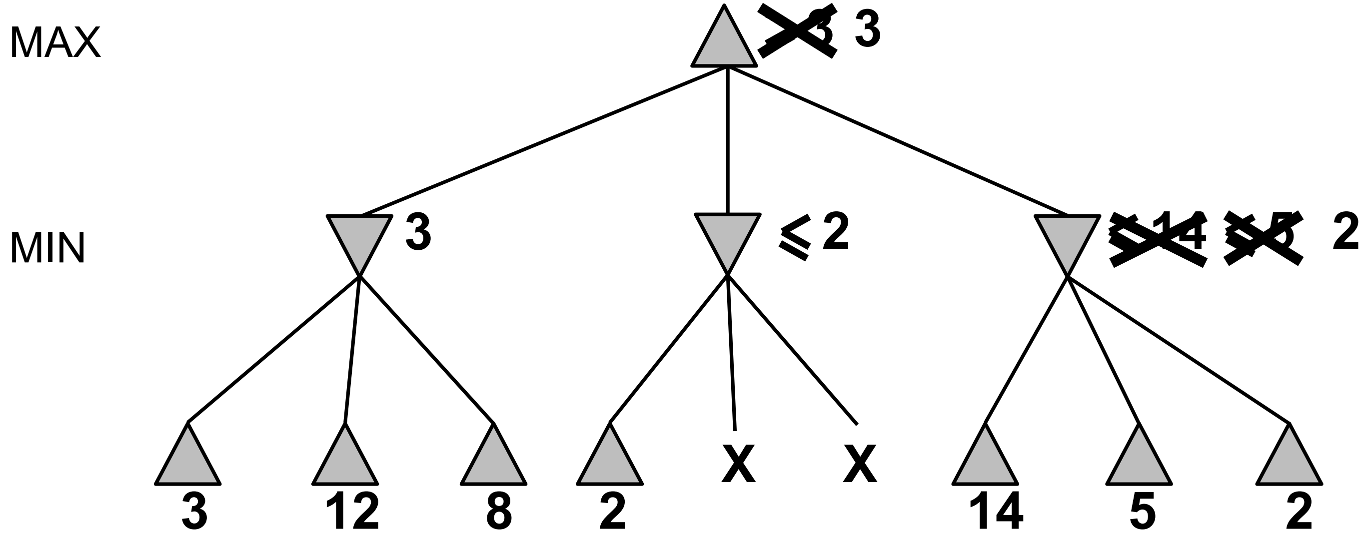
α–β pruning example





α–β pruning example





The α–β algorithm

```
function Alpha-Beta-Decision(state) returns an action
   return the a in Actions(state) maximizing Min-Value(Result(a, state))
```

```
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)) ; Max of all min values
      if v ≥ β then return v   ; Prune: Min will not choose value > beta
      α ← Max(α, v)
   return v
```

```
function Min-Value(state, α, β) returns a utility value
   same as Max-Value but with roles of α, β reversed
```

Properties of α–β

Pruning does not affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity = $O(b^{m/2})$
$\Rightarrow$ doubles solvable depth

A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

Unfortunately, 35^{50} is still impossible!

Resource limits – Limiting depth

Standard approach:

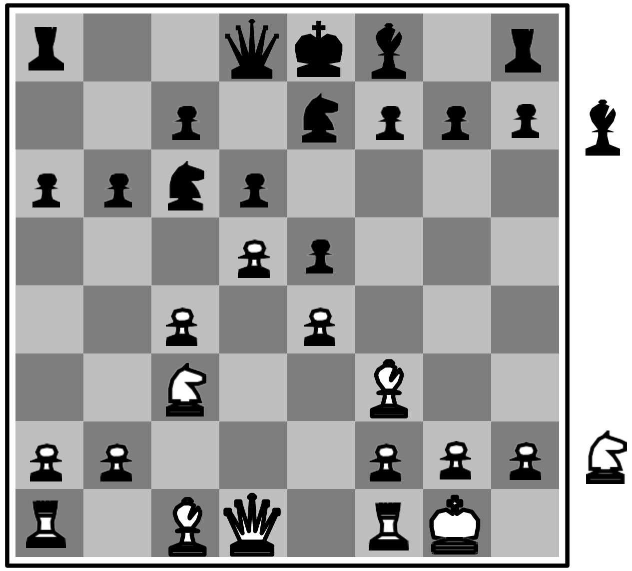
- Use Cutoff-Test instead of Terminal-Test
  e.g., depth limit
- Use Eval instead of Utility
  i.e., evaluation function that estimates desirability of position

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

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

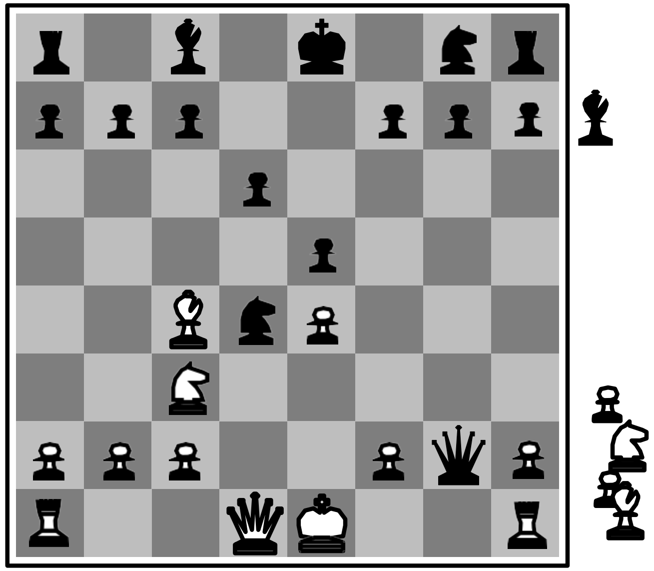
$\Rightarrow$ α–β reaches depth 8 $\Rightarrow$ pretty good chess program

Evaluation functions



Black to move

White slightly better



White to move

Black winning

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),

Deterministic games in practice

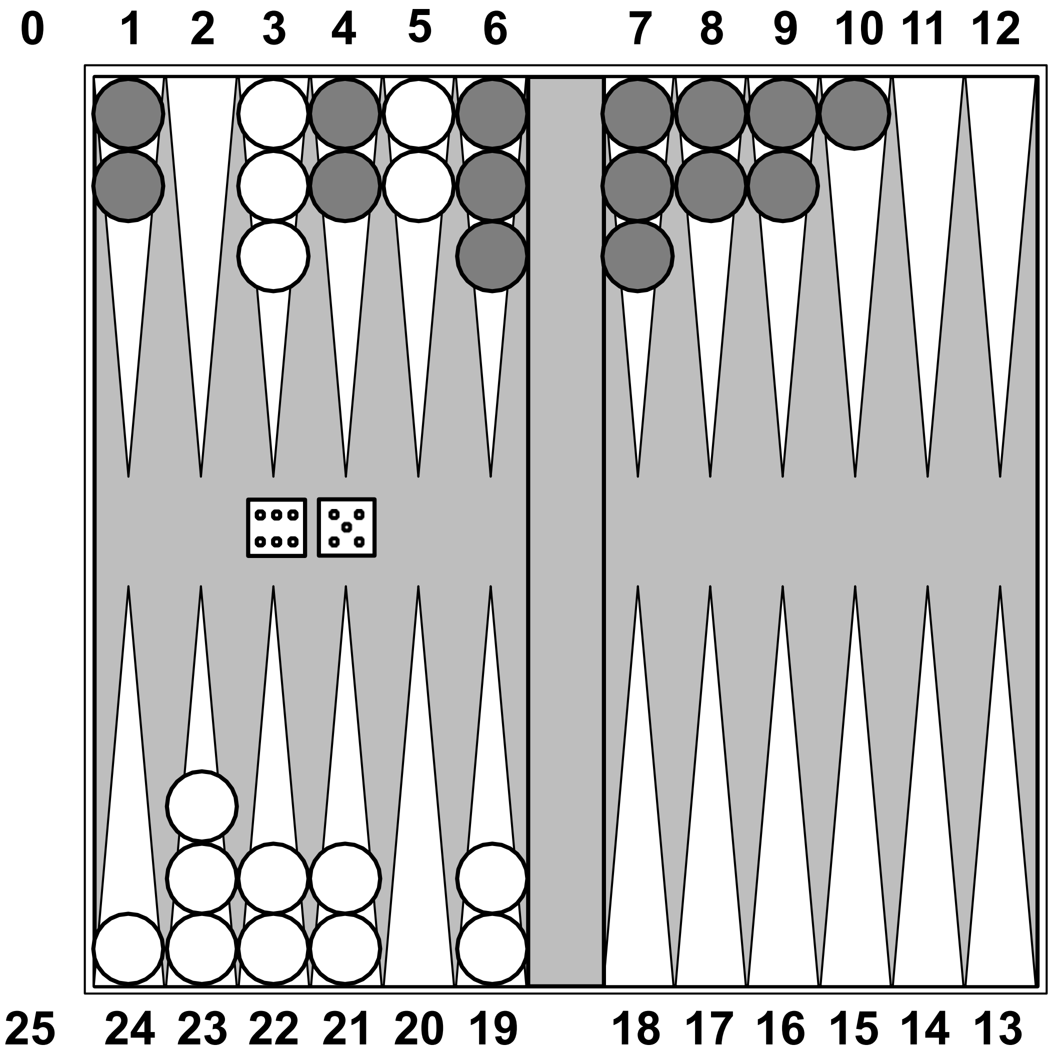
Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

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, who are too good.

Go: human champions refused to compete against computers before 2017, because they were too bad!! In go, $b > 300$, so most programs used pattern knowledge bases to suggest plausible moves before deep learning changed the game.

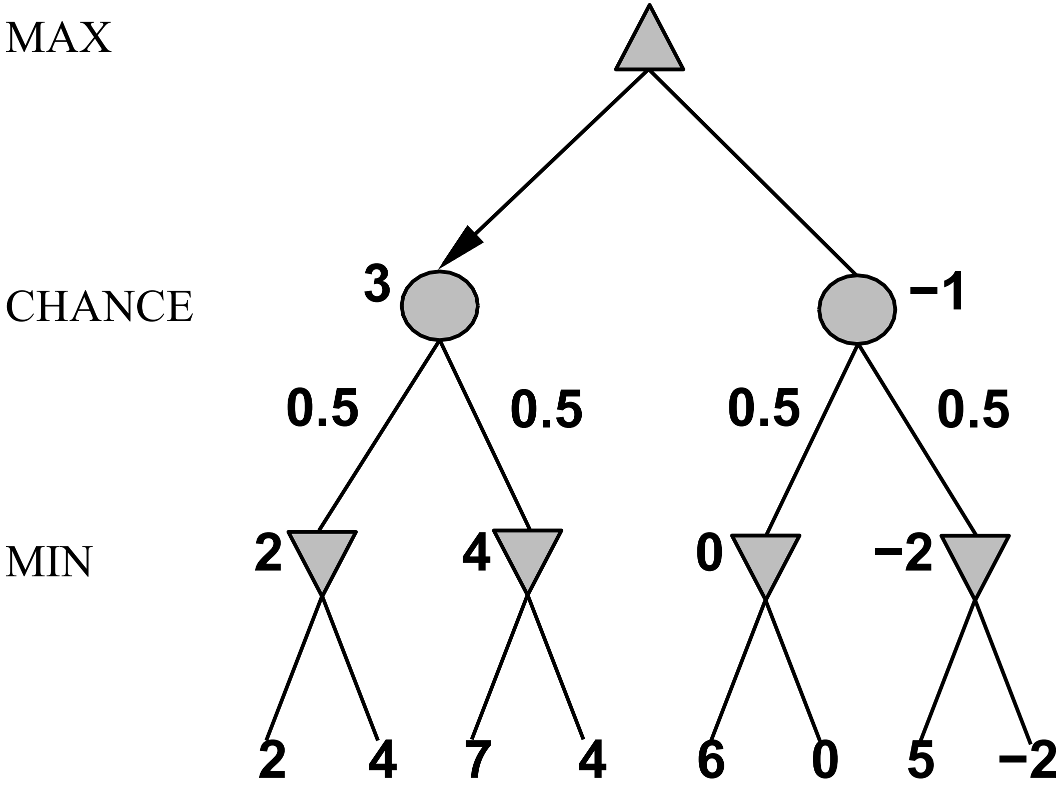
Nondeterministic games: backgammon



Nondeterministic games in general

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

Simplified example with equal chance of two outcomes:



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*)
...

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

Point is that each game needs to have carefully considered

heuristics and techniques

Bridge programs approximate this idea by

1) generating 100 deals consistent with bidding information
2) picking the action that wins most tricks on average

Games in practice

Computers have reached world standards in many games

QUACKLE program defeated the former world champion in Scrabble.

Libratus took on four of the top poker players in the world in a 20-day match of no limit Texas hold ’em and decisively beat them all. Pluribus beat 5 simultaneous players Texas hold ‘em.

TDGammon uses depth-2 search + very good Eval is world-champion level. Led to advances in the theory of correct backgammon play.

Even Video Games such as Starcraft II. In 2019, Vinyals and DeepMind team AlphaStar program based on deep learning and reinforcement learning defeated expert gamers 10-1.

Summary

Games are fun to work on! (and dangerous, eg MAD)

They illustrate several important points about AI

- ✦ perfection is unattainable $\Rightarrow$ must approximate
- ✦ good idea to think about what to think about
- ✦ uncertainty constrains the assignment of values to states
- ✦ optimal decisions depend on information state, not real state

Games are to AI as grand prix racing is to automobile design

Questions or comments