
Game Playing

1

Outline

- **Game playing**
 - The minimax algorithm
 - Resource limitations
 - alpha-beta pruning
 - Elements of chance

2

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
- Plan of attack:
 - Computer considers possible lines of play (Babbage, 1846)
 - 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)

3

Types of games

	deterministic	chance
perfect Information	chess, checkers, go, othello, rock-paper-scissors	backgammon monopoly
Imperfect Information	battleships, kriegspiel, stratego	bridge, poker, scrabble nuclear war

4

What kind of games?

- **Abstraction:** To describe a game we must capture every relevant aspect of the game. Such as :
 - Chess
 - Tic-tac-toe
 - ...
- **Accessible environments:** Such games are characterized by perfect information
- **Search:** game-playing then consists of a search through possible game positions
- **Unpredictable opponent:** introduces uncertainty thus game-playing must deal with contingency problems

5

Searching for the next move

- **Complexity:** many games have a huge search space
 - **Chess:** $b = 35, m=100 \Rightarrow \text{nodes} = 100^{35}$
if each node takes about 1 ns to explore
then each move will take about **10^{50} millennia**s to calculate.
- **Resource (e.g., time, memory) limit:** optimal solution not feasible/possible, thus must approximate
- **Pruning:** makes the search more efficient by discarding portions of the search tree that *obviously* cannot improve quality of result.
- **Evaluation functions:** heuristics to evaluate utility of a state without exhaustive search.

6

Two-player games

- A game formulated as a search problem:
 - Initial state: ?
 - Operators: ?
 - Terminal state: ?
 - Utility function: ?

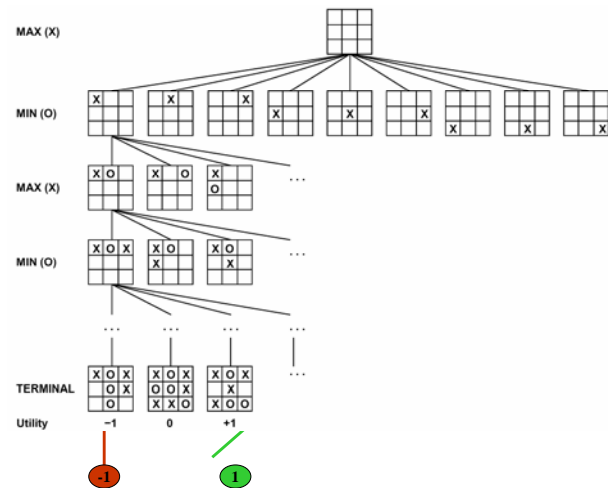
7

Two-player games

- A game formulated as a search problem:
 - **Initial state:** board position and turn
 - **Operators:** definition of legal moves
 - **Terminal state:** conditions for when game is over
 - **Utility function:** a numeric value that describes the outcome of the game.
e.g., -1, 0, 1 for loss, draw, win.

8

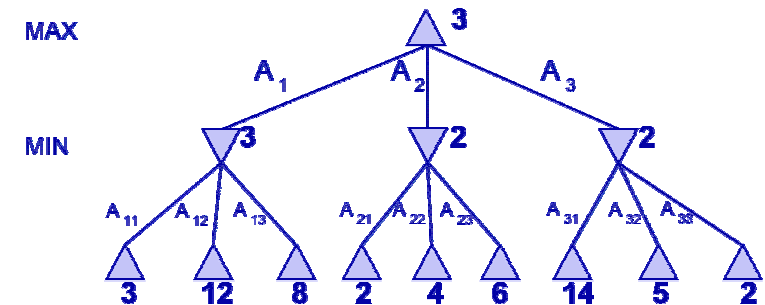
Example: Tic-Tac-Toe



9

The minimax algorithm

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



10

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)
   $v \leftarrow -\infty$ 
  for a, s in SUCCESSORS(state) do  $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$ 
  return v

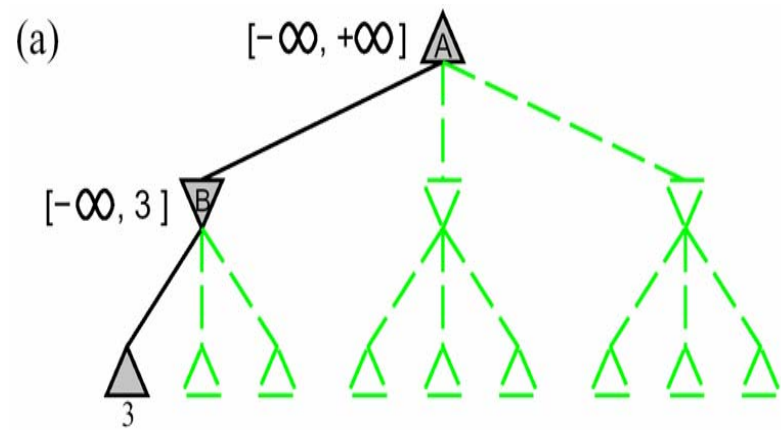
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow \infty$ 
  for a, s in SUCCESSORS(state) do  $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$ 
  return v
    
```

11

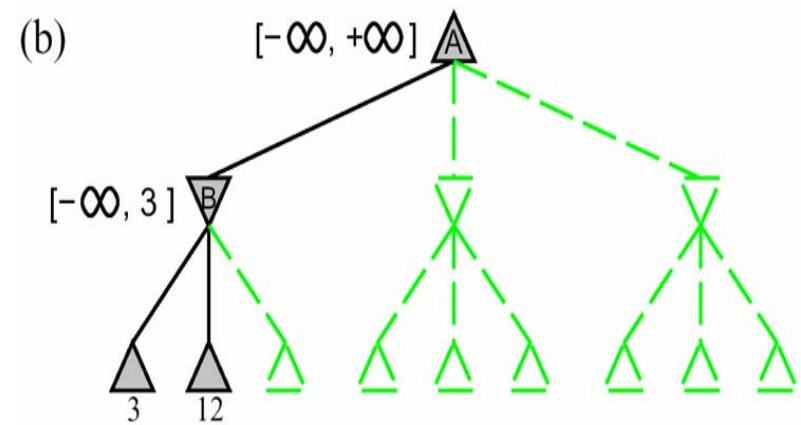
Properties of Minimax

- Completeness: if tree is finite (chess has specific rules for this)
- Optimality: 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
 \therefore exact solution completely infeasible
- But do we need to explore every path?

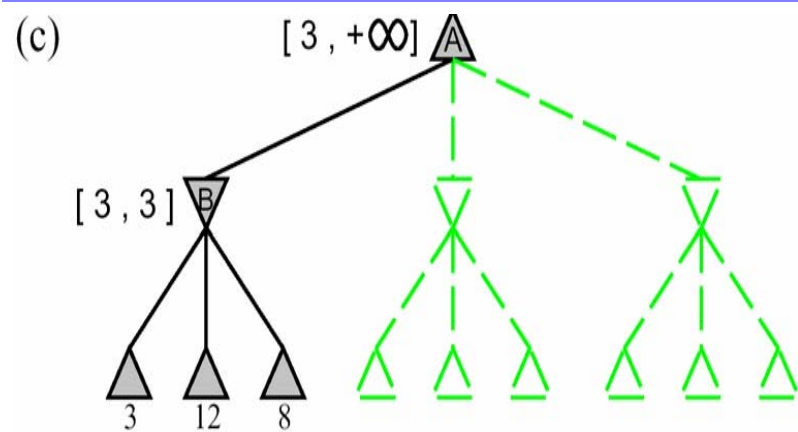
12



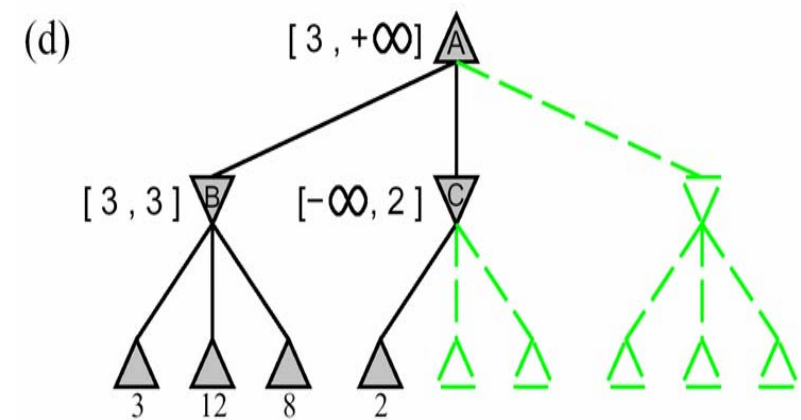
13



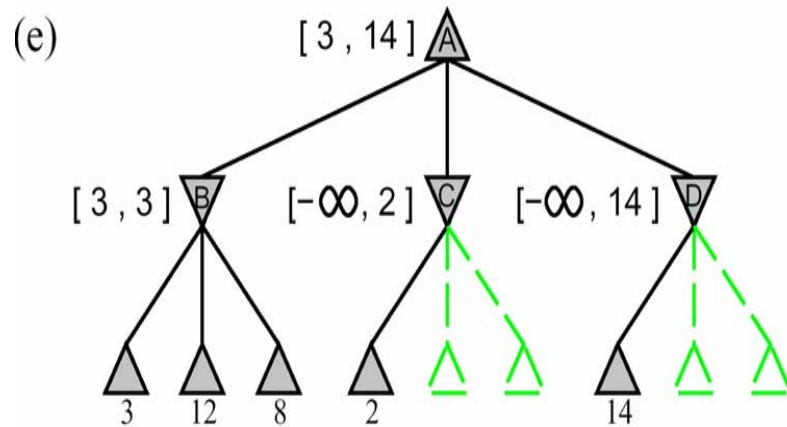
14



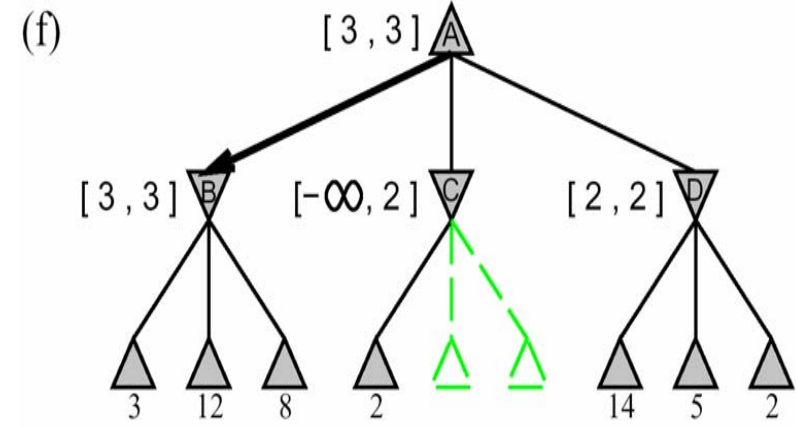
15



16



17



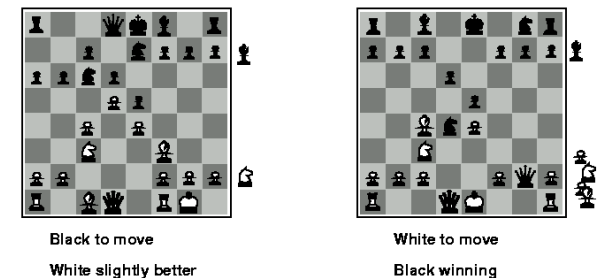
18

Move evaluation without complete search

- Complete search is too complex and impractical
- Evaluation function: evaluates value of state using heuristics and cuts off search
- New MINIMAX:
 - CUTOFF-TEST: cutoff test to replace the termination condition (e.g., deadline, depth-limit, etc.)
 - EVAL: evaluation function to replace utility function (e.g., number of chess pieces taken)

19

Evaluation functions



- **Weighted linear evaluation function:** to combine n heuristics

$$f = w_1f_1 + w_2f_2 + \dots + w_nf_n$$
- e.g, w 's could be the values of pieces (1 for pawn, 3 for bishop etc.) f 's could be the number of type of pieces on the board

20

α - β pruning: search cutoff

- **Pruning:** eliminating a branch of the search tree from consideration without exhaustive examination of each node
- **α - β pruning:** the basic idea is to prune portions of the search tree that cannot improve the utility value of the max or min node, by just considering the values of nodes seen so far.
- Does it work? Yes, it roughly cuts the branching factor from b to \sqrt{b} resulting in as double as far look-ahead than pure minimax

21

alpha-beta search

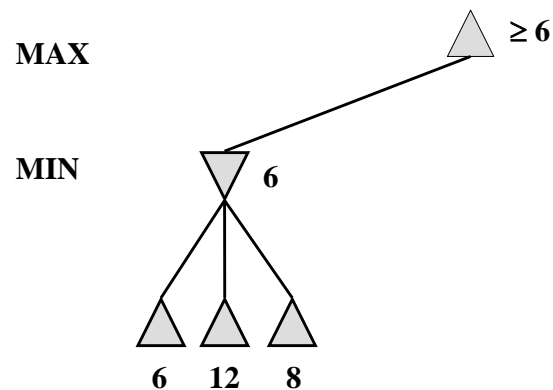
```
function ALPHA-BETA-DECISION(state) returns an action
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  inputs: state, current state in game
          $\alpha$ , the value of the best alternative for MAX along the path to state
          $\beta$ , 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
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$ 
    if  $v \geq \beta$  then return v
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return v

function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  same as MAX-VALUE but with roles of  $\alpha$ ,  $\beta$  reversed
```

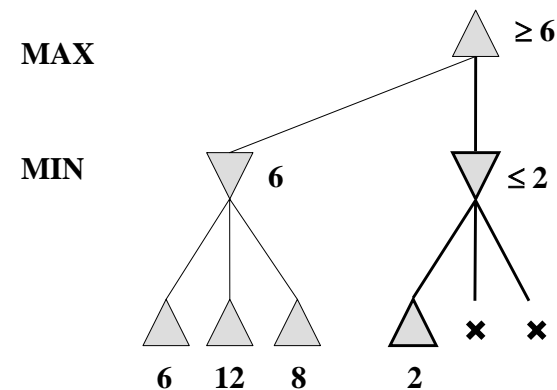
22

α - β pruning: example



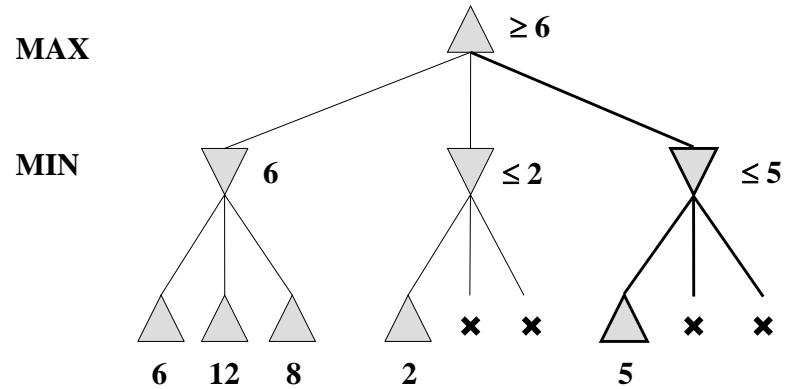
23

α - β pruning: example



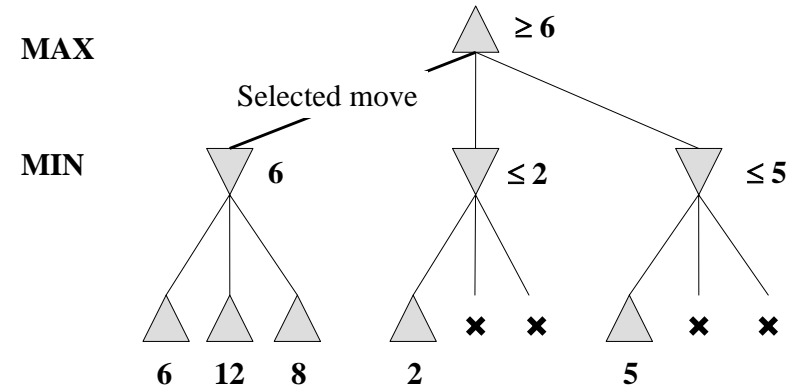
24

α - β pruning: example



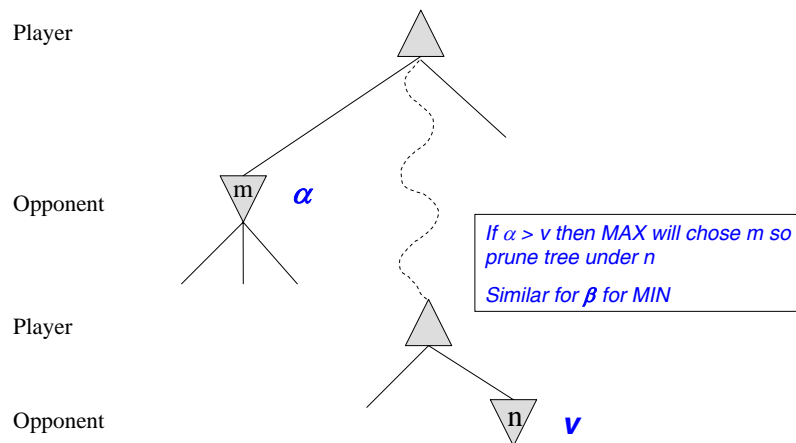
25

α - β pruning: example



26

α - β pruning: general principle



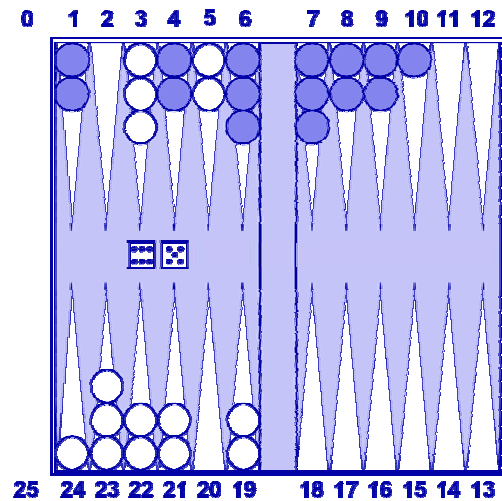
27

State-of-the-art for deterministic games

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

28

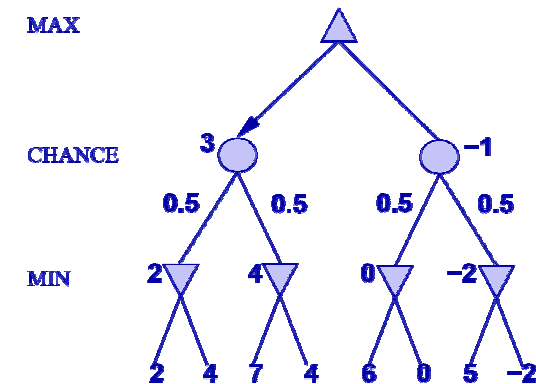
Nondeterministic Games



29

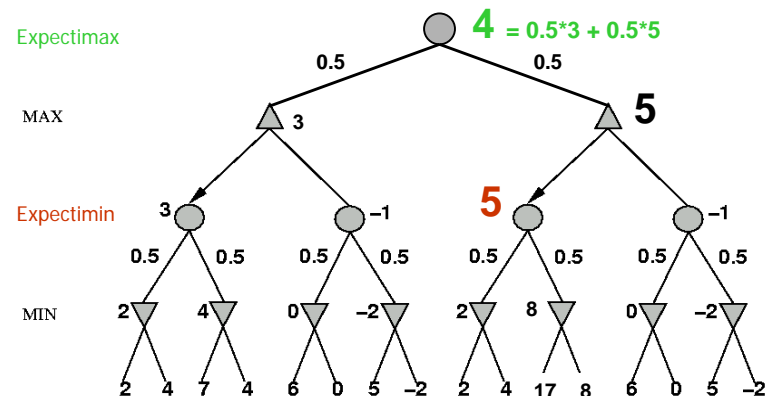
Nondeterministic games: the element of chance

In nondeterministic games, chance introduced by dice, card-shuffling
Simplified example with coin-flipping



30

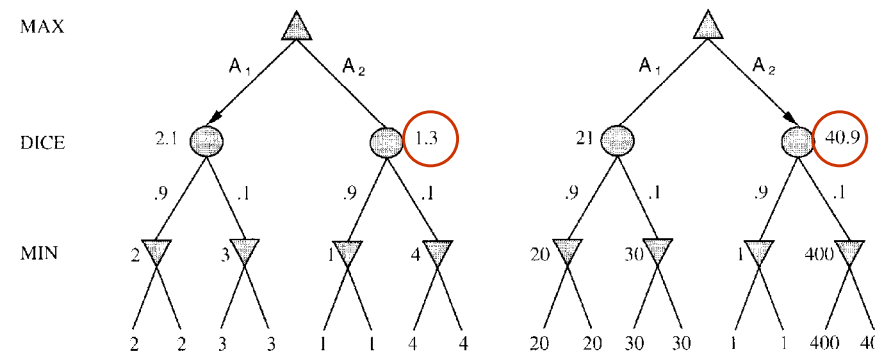
Nondeterministic games: the element of chance



31

Evaluation functions

Order-preserving transformation do not necessarily behave the same



32

State-of-the-art for nondeterministic games

- Dice rolls increase b : 21 possible rolls with 2 dice
- Backgammon ≈ 20 legal moves (can be 6,000 with 1-1 roll)
- depth 4 = $20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$
- As depth increases, probability of reaching a given node shrinks
 - value of look-ahead is diminished
- α - β pruning is much less effective
- TDGammon uses depth-2 search + very good Eval
 - world-champion level

33

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

34