Games

Outline

- Games as search
- Optimal decisions in games
  - minimax decisions
  - α-β pruning
- Real-time decisions

2

Why study games?

- Examples: chess, checkers, Go, backgammon, bridge
- Why study games?
  - Fun; historically entertaining
  - Interesting subject of study because they are hard
  - Easy to represent and agents restricted to small number of actions
- Games are to AI as grand prix racing is to automobile design.

3

Games vs. Search

- Games are a form of *multi-agent environment*
  - What other agents do affect our success
- Competitive multi-agent environments give rise to *adversarial search*
- Solution is strategy (specifying a move for every possible opponent reply)

4

Types of Games

- Games of deterministic, perfect information: chess, checkers
- Stochastic games: backgammon
- Partially observable games: bridge, poker

5

Game setup

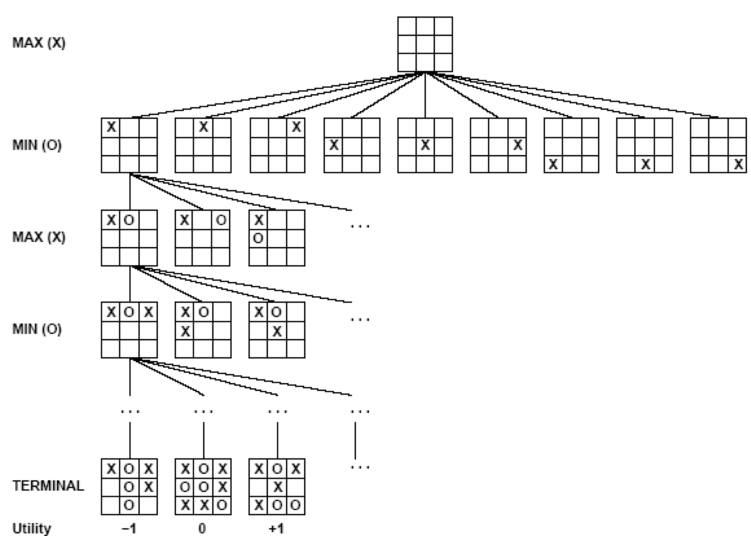
- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over.
- Winner is awarded points (payoff), loser gets penalties.
- ***Zero-sum game***: payoff values at the end of the game are equal and opposite (or the total payoff to all players is the same for every instance of the game)

6

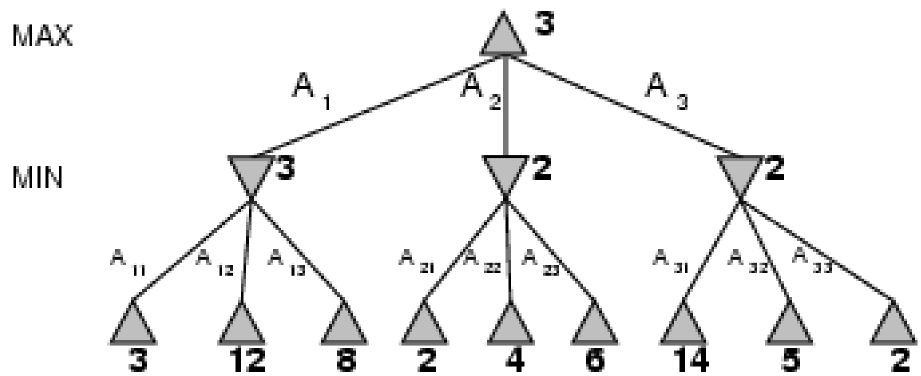
Game setup

- Games as search:
  - Initial state: e.g. board configuration of chess
  - PLAYER(s): which player has the move in a state
  - Actions and transition model (or successor function): list of (move, state) pairs specifying legal moves.
  - Terminal test: Is the game finished (in *terminal states*)?
  - ***Utility function*** or ***payoff function*** *UTILITY(s,p)*: Gives numerical value for terminal states. E.g. win (+1), loss (-1), and draw (0) in tic-tac-toe
- The initial state and the legal moves define the ***game tree***

Partial Game Tree for Tic-Tac-Toe



Two-Ply Game Tree



9

Optimal strategies

- Find the contingent *strategy* for MAX assuming an infallible MIN opponent.
- Assumption: Both players play optimally !!
- Given a game tree, the optimal strategy can be determined by using the **minimax value** of each node:

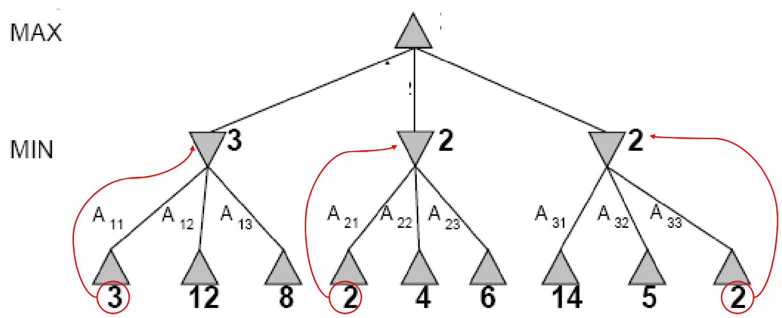
  MINIMAX-VALUE(n)=

UTILITY(n)	If n is a terminal
$\max_{s \in successors(n)}$ MINIMAX-VALUE(s)	If n is a max node
$\min_{s \in successors(n)}$ MINIMAX-VALUE(s)	If n is a min node

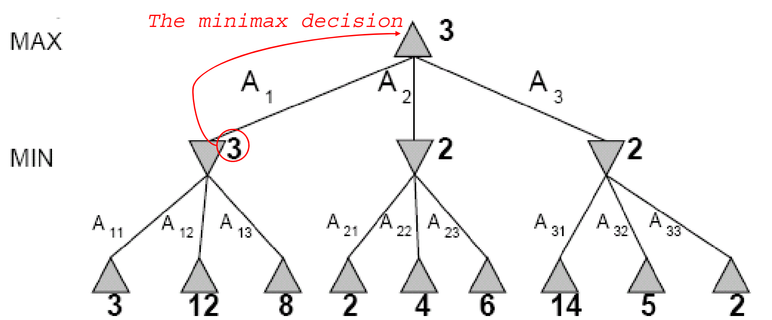
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play

10

Two-Ply Game Tree



Two-Ply Game Tree



Minimax maximizes the worst-case outcome for max.

Minimax Algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*
inputs: *state*, current state in game
return *argmax* $_{a \text{ in ACTIONS}(state)}$ MIN-VALUE(RESULT(*state,a*))

---

function MAX-VALUE(*state*) **returns** *a utility value*
if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
$v \leftarrow -\infty$
for each *a* in ACTIONS(*state*) do
$v \leftarrow$ MAX(v, MIN-VALUE(RESULT(*state,a*)))
return v

---

function MIN-VALUE(*state*) **returns** *a utility value*
if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
$v \leftarrow \infty$
for each *a* in ACTIONS(*state*) do
$v \leftarrow$ MIN(v,MAX-VALUE(RESULT(*state,a*)))
return v

13

Properties of minimax

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

14

What if MIN does not play optimally?

- Definition of optimal play for MAX assumes MIN plays optimally: maximizes worst-case outcome for MAX.
- But if MIN does not play optimally, MAX will do even better.
- There may be better strategies against suboptimal opponents, but they do worse against optimal opponents

Improving minimax search

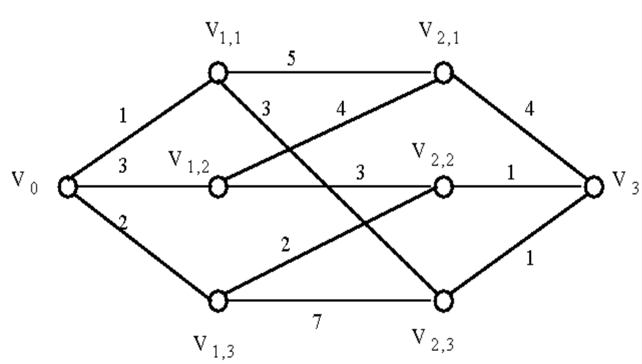
- Number of states is exponential to the number of moves.
  - Solution: Do not examine every node
  - ==> Alpha-beta pruning
    - Remove branches that do not influence final decision
  - (Related algorithm: DFBnB)

Diversion: Depth-First Branch & Bound (DFBnB)

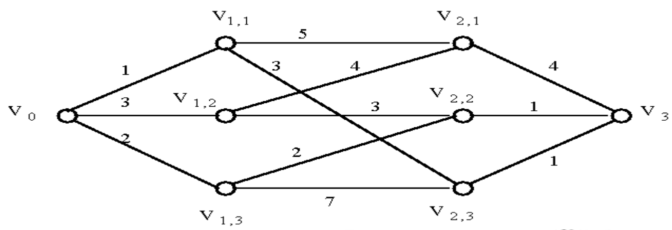
- Want to find the optimal solution among many solutions (or solving an optimization problem)
- DF search
- Keep track of best solution so far
- If f(n) = g(n)+h(n) ≥ cost(best-soln)
  - Then prune n

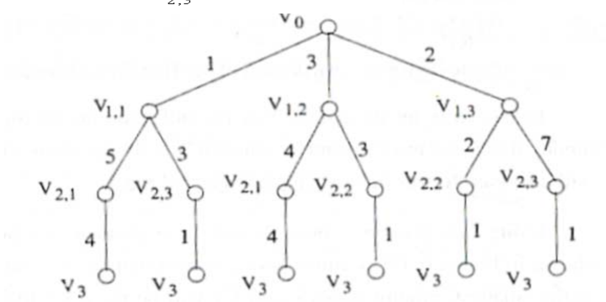
Diversion: DFBnB

Find the shortest path from V_0 to V_3



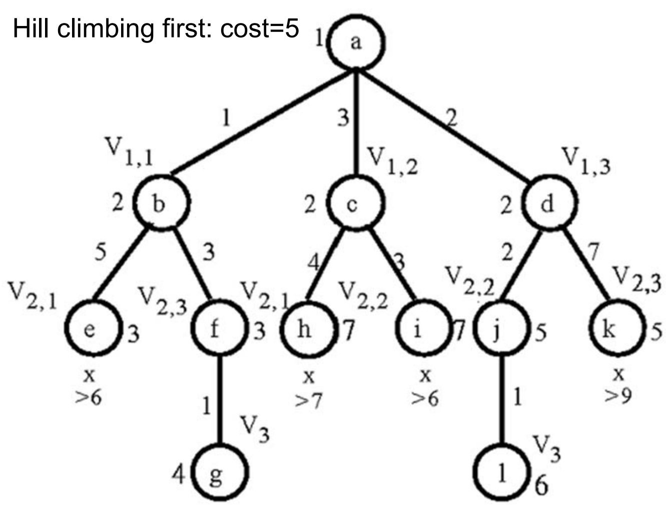
Diversion: DFBnB





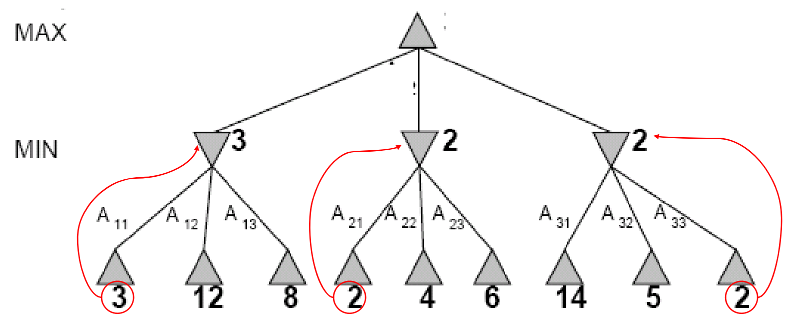
Diversion: DFBnB

Hill climbing first: cost=5



20

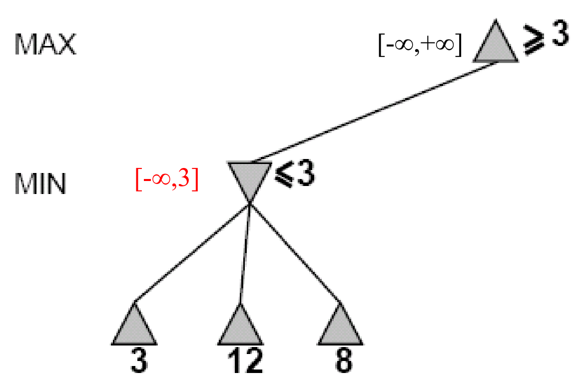
Pruning example



21

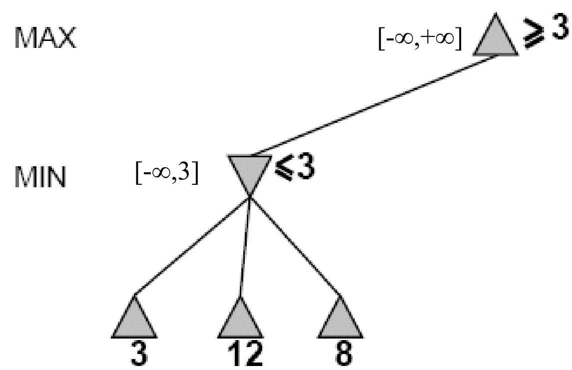
Alpha-Beta Example

Do DF-search until first leaf



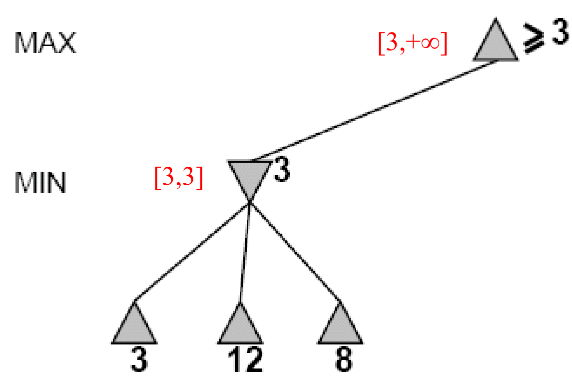
22

Alpha-Beta Example (continued)



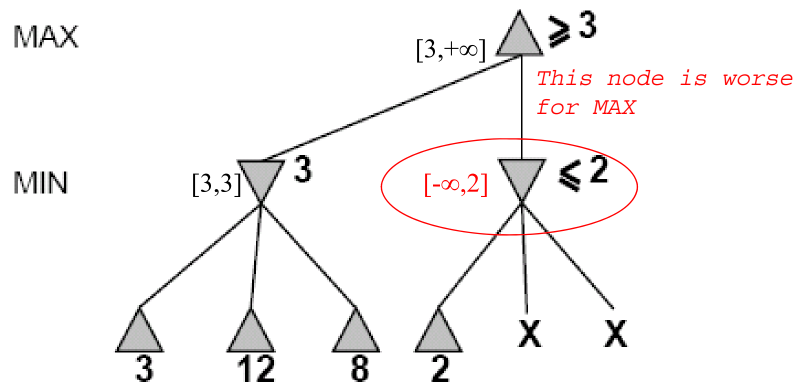
23

Alpha-Beta Example (continued)



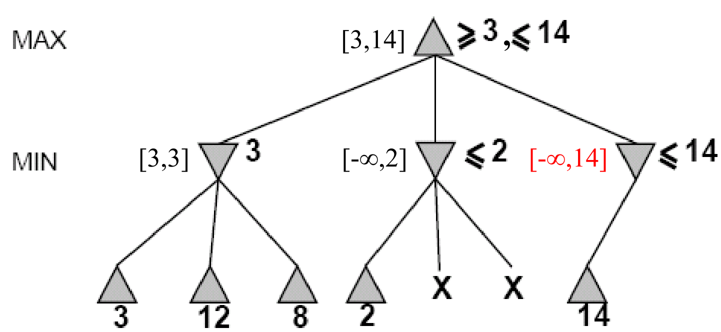
24

Alpha-Beta Example (continued)



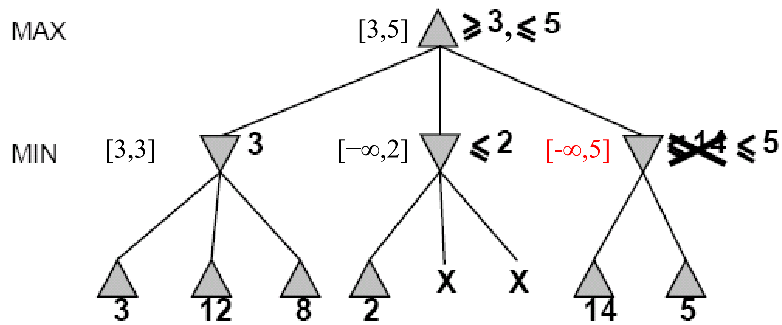
25

Alpha-Beta Example (continued)



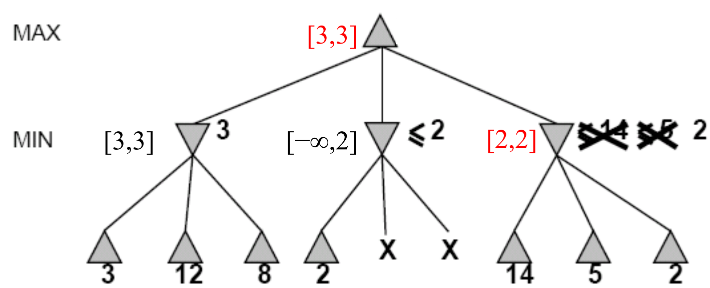
26

Alpha-Beta Example (continued)



27

Alpha-Beta Example (continued)



28

Why is it called α-β?

- Similar for MIN nodes
- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
- β is the value of the best (i.e., lowest-value) choice found so far at any choice point along the path for *min*

29

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 ACTIONS(state) with value v

function MAX-VALUE(state,α , β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← - ∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(Result(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α ,v)
  return v
```

30

The α-β algorithm

function MIN-VALUE(*state*, α , β) returns *a utility value*
  if TERMINAL-TEST(*state*) then return UTILITY(*state*)
  $v \leftarrow +\infty$
  for each *a* in ACTIONS(*state*) do
    $v \leftarrow$ MIN(v, MAX-VALUE(Result(*s*,a), α, β))
    if $v \leq \alpha$ then return v
    $\beta \leftarrow$ MIN(β ,v)
  return v

31

Properties of Alpha-Beta Pruning

- Pruning does not affect final results
- The effectiveness of pruning is highly dependent on the order in which the successors are checked
  - Try to examine first the successors that are likely to be best
  - ordering heuristics
- With "perfect ordering," time complexity is O($b^{m/2}$)
  - Branching factor of sqrt(b) !!
  - Alpha-beta pruning can look twice as far as minimax in the same amount of time

32

Real-time Decisions

- Minimax and alpha-beta pruning require leaf-node evaluations.
- May be impractical within a reasonable amount of time.
- Suppose we have 100 secs, explore 10^6 nodes/sec
  → 10^8 nodes per move ~35^5
  → Alpha-beta pruning reaches depth 10

33

Real-time Decisions

Standard approach

- Cut off search earlier (replace TERMINAL-TEST by CUTOFF-TEST)
  - E.g., introduce a fixed depth limit (selected so that the amount of time will not exceed what the rules of the game allow), or iterative deepening
- Apply heuristic ***evaluation function*** EVAL (replacing utility function of alpha-beta)
- Change**:**
  - **if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

  into
  - **if** CUTOFF-TEST(*state*,*depth*) **then return** EVAL(*state*)

34

Heuristic evaluation function

- Idea: produce an estimate of the expected utility of the game from a given position.
- Performance depends strongly on quality of evaluation function
- Requirements:
  - EVAL should order terminal-nodes in the same way as UTILITY.
  - Computation may not take too long.
  - For non-terminal states the EVAL should be strongly correlated with the actual chance of winning.
- Exact values don't matter, only the order matters

35

Evaluation functions

- Most evaluation functions work by calculating various features of the state
  - Compute numerical contributions from each feature and combine them to find the total value
- For chess, *material value* for each piece: pawn=1, knight=bishop=3, rook=5, queen=9
- 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)

36

Real-time Decisions

- Minimax with alpha-beta pruning
- An evaluation function
- More sophisticated cutoff test: e.g., wild swings in value in near future
- A large *transposition table* (closed list): repeated states problem

37

Real-time Decisions

- Extensively tuned evaluation function
- Pruning heuristics
- A large database of optimal opening and endgame moves: table lookup instead of search
  - Chess endgames with up to 7 pieces solved

38

Go

- Minimax not successful in Go

39

Monte Carlo tree search

- Find the most promising moves by playing out many random moves to the end, using techniques in Markov Decision Processes
- Doesn't necessarily require game-specific knowledge -> no need for evaluation function; simply implementing the game's mechanics is sufficient
- Does well in games with a high branching factor
- Used in Go, real-time video games
- AlphaGo: uses a Monte Carlo tree search guided by deep neural networks extensively trained from games played by human experts and games against itself
- AlphaGo Zero (2017) learns to play using Reinforcement Learning simply by playing games against itself, starting from completely random play

40

State-of-the-Art Deterministic games

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. In July 2007, Chinook's developers announced that the program has been improved to the point where it cannot lose a game. 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.
- Othello/Reversi: humans are no match for computers
- Go: AlphaGo beat a professional player without handicaps on a full-sized 19×19 board in 2015, beat a top Go player in 2016.

41

Types of Games

- We covered games of deterministic, perfect information: chess, checkers
- Stochastic Games: backgammon (Chapter 5.5)
- Partially observable games: bridge, poker (Chapter 5.6)

42