Artificial Intelligence

Game Playing — Adversarial Search

Outline

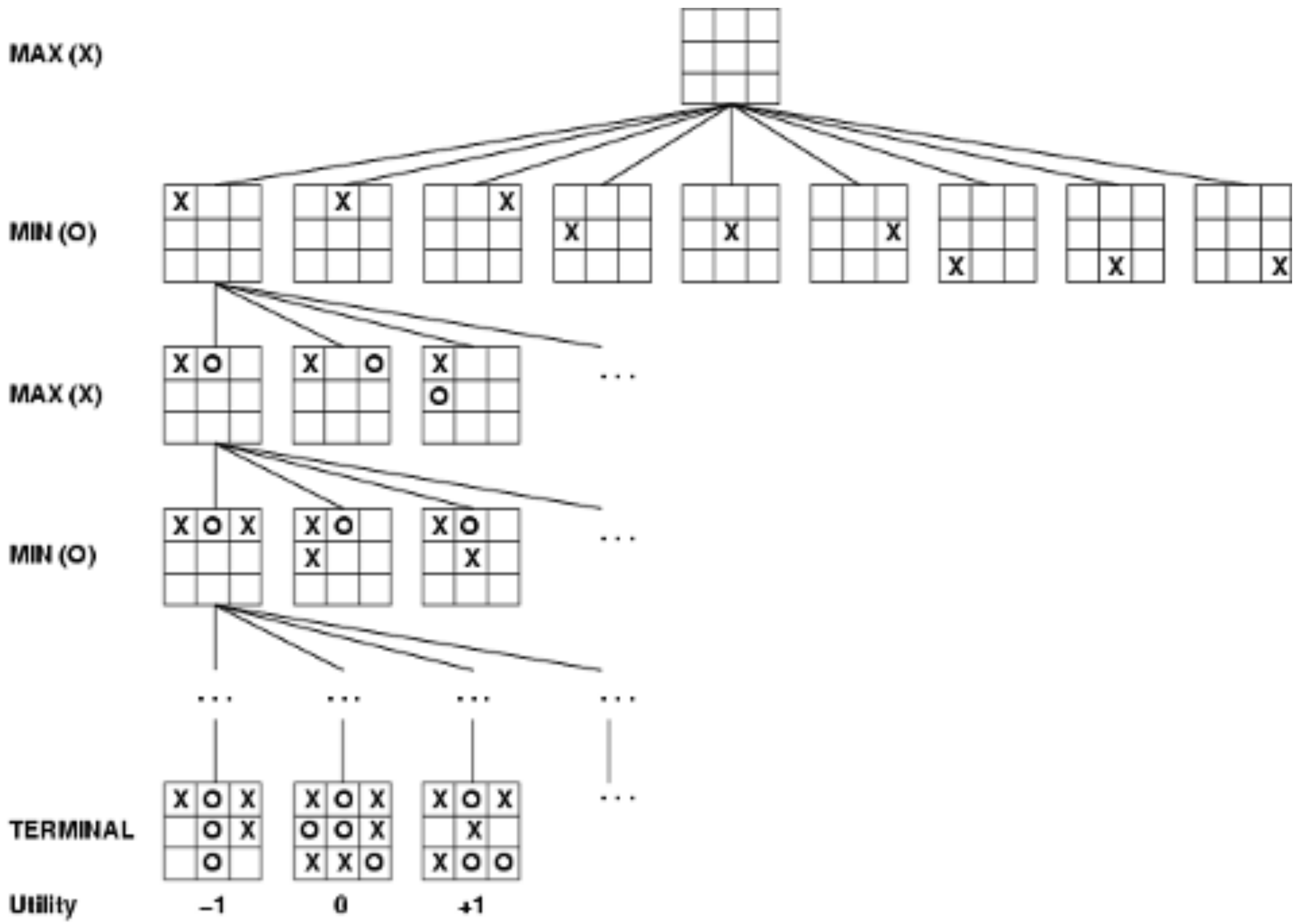
- Optimal decisions
- α-β pruning
- Imperfect, real-time decisions

Games vs. search problems

- "Unpredictable" opponent
  —> specifying a move for every possible opponent reply

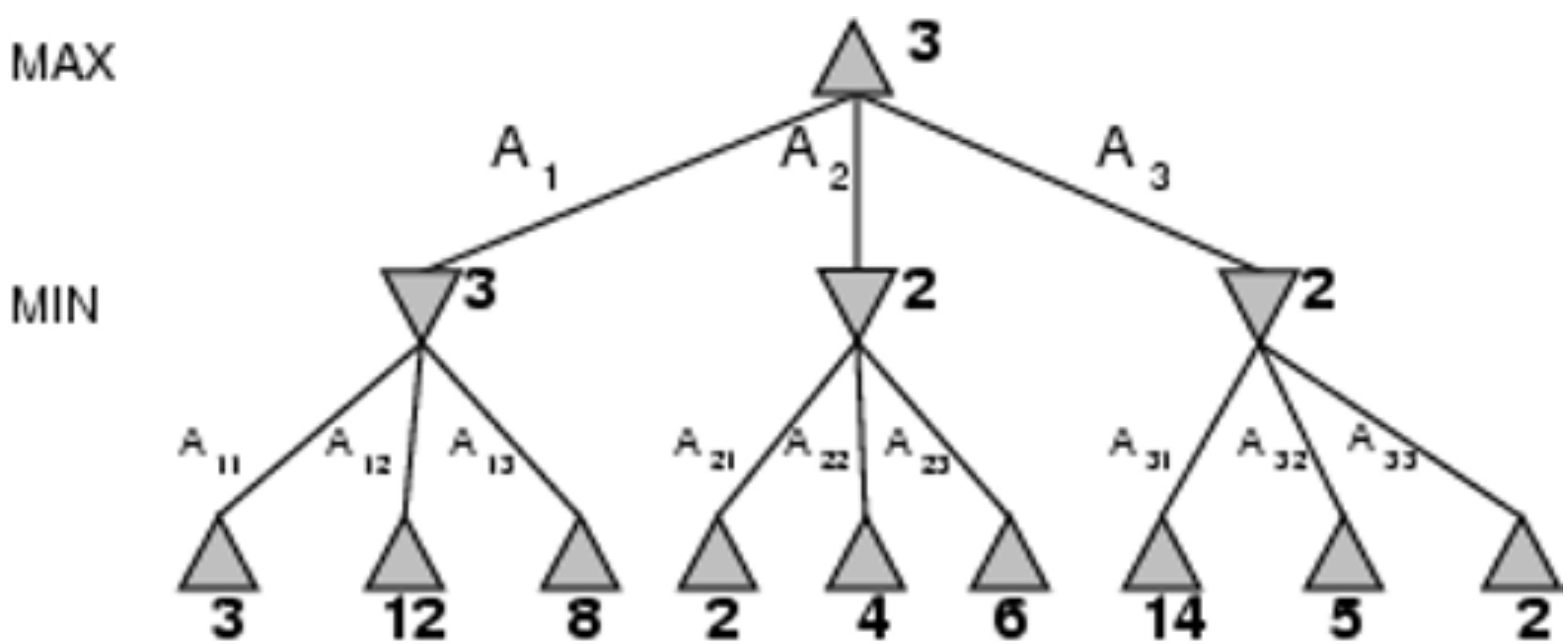
- Time limits
  —> unlikely to find goal, must approximate

Game tree (2-player, deterministic, turns)



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*

$v \leftarrow$ 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*)
$v \leftarrow -\infty$
for a, s in SUCCESSORS(*state*) **do**
  $v \leftarrow$ MAX(v, 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**
  $v \leftarrow$ 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

α-β pruning

```
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
```

α-β pruning (cont.)

function MIN-VALUE($state, \alpha, \beta$) **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($state$)
  $v \leftarrow +\infty$
  for a, s in SUCCESSORS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(s, α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

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

Resource limits

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

- Standard approach to wrap up search in time:

  - cutoff test:
    e.g., depth limit (perhaps add quiescence search)

  - evaluation function
    = estimated desirability of position

Cutting off search

- MinimaxCutoff is identical to MinimaxValue except
  - Terminal? is replaced by Cutoff?
  - Utility is replaced by Eval
- Does it work in practice?
  $b^m = 106$, $b=35$ —> $m=4$
- 4-ply look-ahead is a hopeless chess player!
  - 4-ply ≈ human novice
  - 8-ply ≈ typical PC, human master
  - 12-ply ≈ Deep Blue, Kasparov

Summary

- Games are fun to work on!
- They illustrate several important points about AI
- Perfection is unattainable —> must approximate
- Good idea to think about what to think about
- Alternative approach? —> Learning