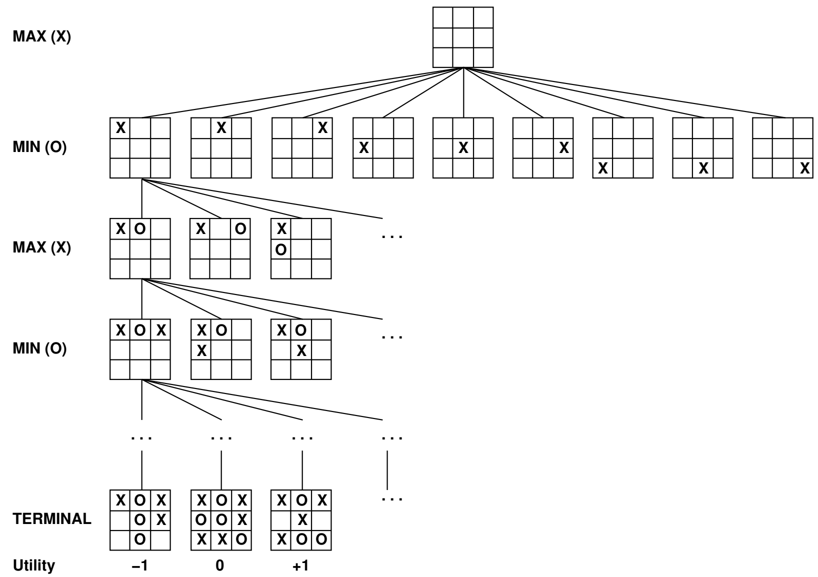
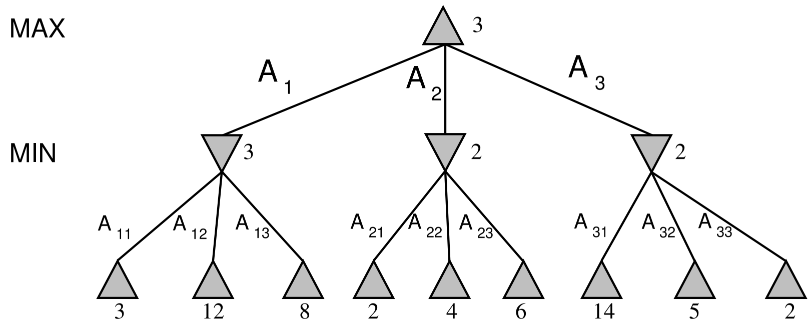
Games as Search Problems

- Mainly two-player games are considered.
- Two-player game-playing problems require an agent to plan ahead in an environment in which another agent acts as its opponent.
- Two sources of uncertainty:
  - Opponent:
    The agent does not know in advance what action its opponent will take.
  - Time-bounded search:
    Time limit has to be imposed because searching for optimal decisions in large state spaces may not be practical. The "best" decisions found may be suboptimal.
- Two-player game-playing problems are contingency problems.
- Search problem:
  - Initial state: initial board configuration and indication of who makes the first move.
  - Operators: legal moves.
  - Terminal test: determines when a terminal state is reached.
  - Utility function (payoff function): returns a numeric score to quantify the outcome of a game.

Game Tree for Tic-Tac-Toe



Perfect Decisions by Minimax Algorithm



- Two players: MAX and MIN
- Minimax algorithm:
  - No time limit is imposed.
  - The complete search tree is generated.
  - MAX tries to *maximize* the utility, assuming that MIN will act to *minimize* it.

Minimax Algorithm

function MINIMAX-DECISION(*game*) **returns** *an operator*

  for each *op* **in** OPERATORS[*game*] **do**
    VALUE[*op*] ← MINIMAX-VALUE(APPLY(*op*, *game*), *game*)
  end
  return the *op* with the highest VALUE[*op*]

---

function MINIMAX-VALUE(*state, game*) **returns** *a utility value*

  if TERMINAL-TEST[*game*](*state*) **then**
    return UTILITY[*game*](*state*)
  else if MAX is to move in *state* **then**
    return the highest MINIMAX-VALUE of SUCCESSORS(*state*)
  else
    return the lowest MINIMAX-VALUE of SUCCESSORS(*state*)

Imperfect Decisions

- It is often impractical to make perfect decisions, as the time and/or space requirements for complete game tree search (to terminal states) are intractable.
- Two modifications to minimax algorithm:
  - Partial tree search:
    Terminal test is replaced by a *cutoff test*.
  - Evaluation function:
    Utility function is replaced by a heuristic *evaluation function*.

Evaluation Functions

- An evaluation function returns an *estimate* of the *expected utility* of the game from a given position.
- Requirements:
  - Computation of evaluation function values is efficient.
  - Evaluation function agrees with utility function on terminal states.
  - Evaluation function accurately reflects the chances of winning.
- Tradeoff between accuracy and efficiency is needed.
- *Learning* good evaluation functions *automatically* from past experience is a promising new direction.

TIC-TAC-TOE

f(p) = + ∞ if p is a win for computer

= - ∞ if p is a win for opponent

= (# of complete r,c or d that are still open for max)-(# of complete r,c or d that are still open for MIN

Ex.

	O	
	X	

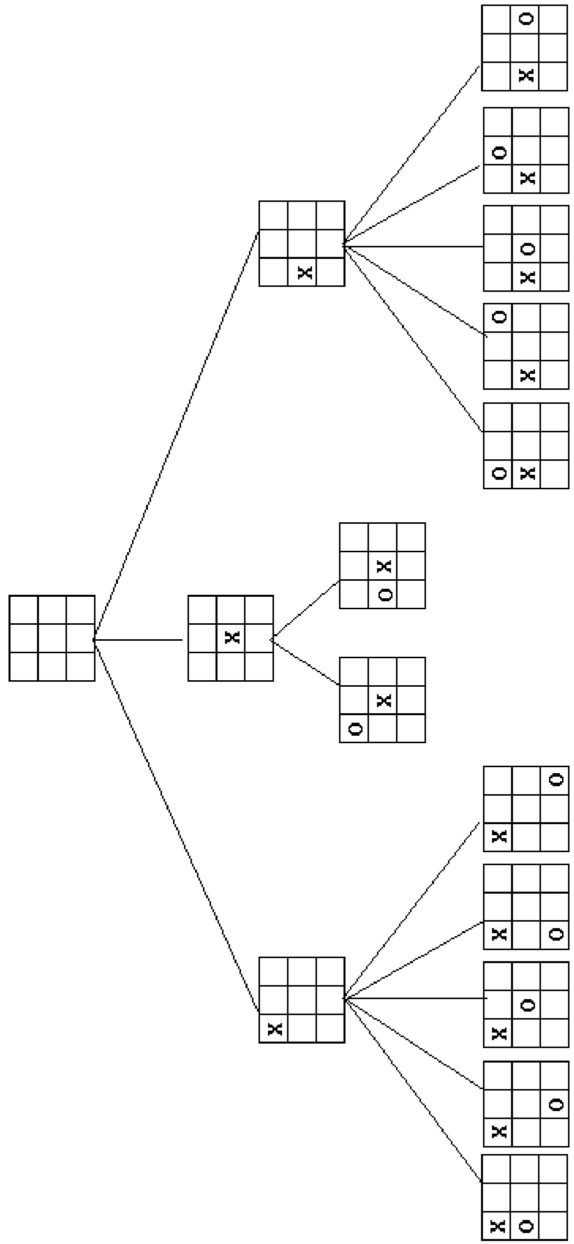
X - MAX

O - MIN

X has 2 rows, 2 cols and 2 diagonals

O has 2 rows, 2 cols

f(p) = 6 - 4 = 2

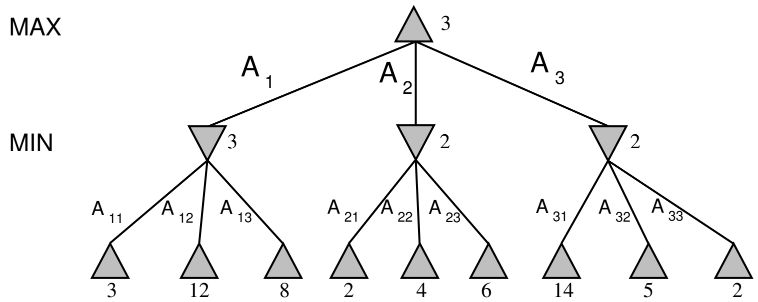


Partial Tree Search

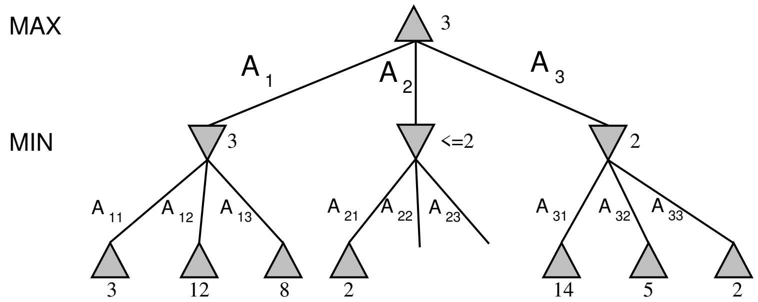
- Different approaches:
  - Depth-limited search
  - Iterative deepening search
  - Quiescent search
- Quiescent positions are positions that are not likely to have large variations in evaluation in the near future.
- Quiescent search:
  - Expansion of nonquiescent positions until quiescent positions are reached.

Alpha-Beta Pruning

- Pruning:
  - The process of eliminating a branch of the search tree from consideration without actually examining it.
- Minimax search without pruning:

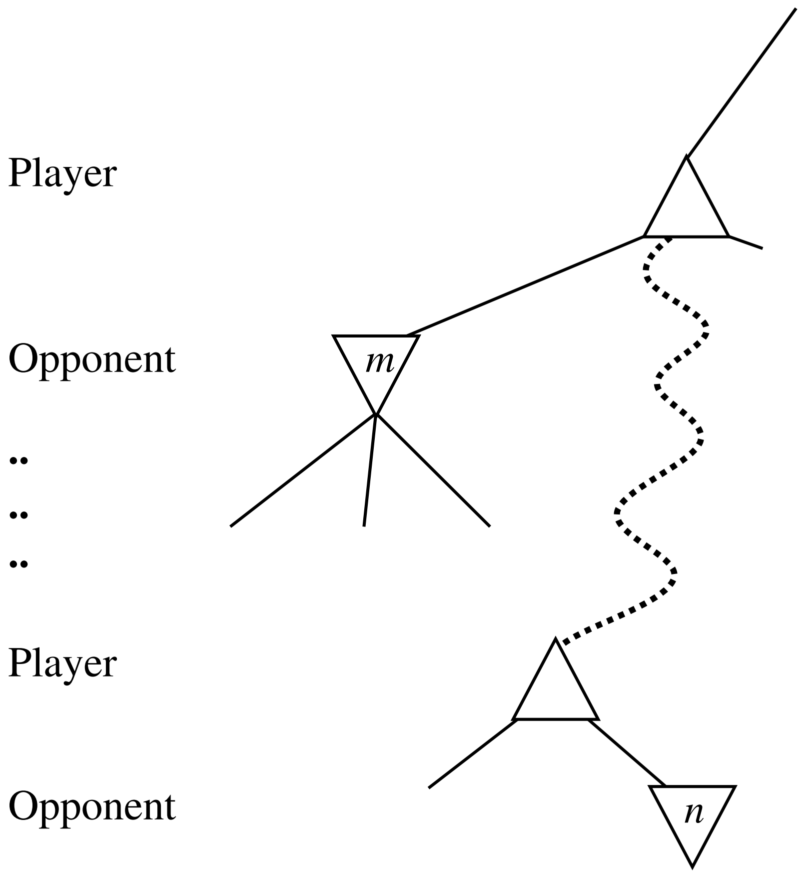


- Minimax search with alpha-beta pruning:



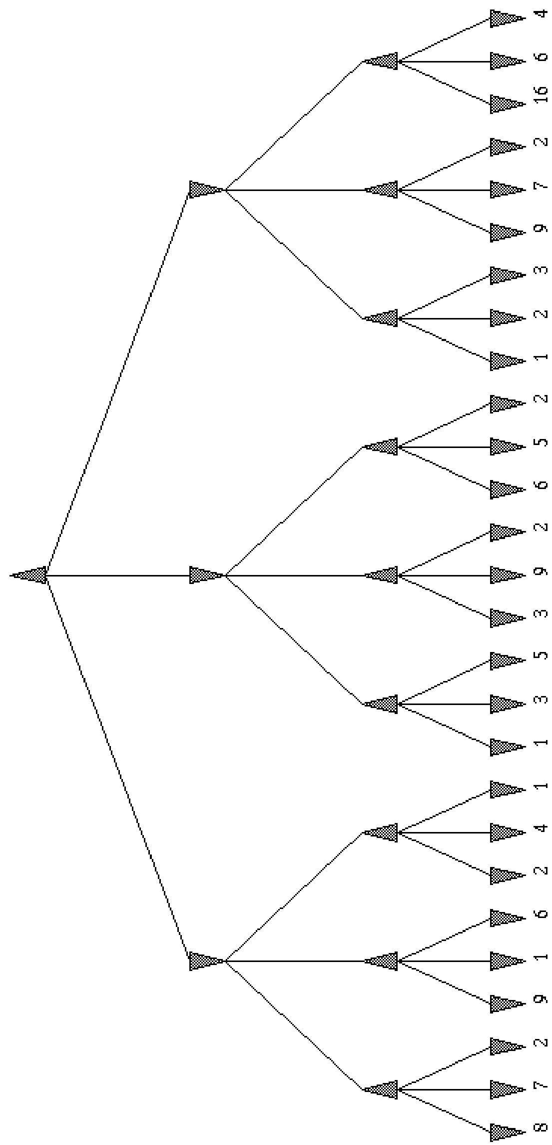
- Effectiveness of alpha-beta pruning depends on order in which successor nodes are examined.

Alpha-Beta Pruning (cont'd)



- General idea:
  - If m is better than n for Player, then n will never be reached in actual play and hence can be pruned away.

Example: Alpha-Beta Pruning



ALPHA-BETA SEARCH ALGORITHM

function MAX-VALUE(*state, game*, α, β) **returns** the minimax value of *state*
  inputs: *state*, current state in game
    game, game description
    α, the best score for MAX along the path to *state*
    β, the best score for MIN along the path to *state*

  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each *s* **in** SUCCESSORS(*state*) **do**
    $\alpha \leftarrow$ MAX(α, MIN-VALUE(*s, game*, α, β))
    if $\alpha \geq \beta$ **then return** β
  end
  return α

---

function MIN-VALUE(*state, game*, α, β) **returns** the minimax value of *state*

  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each *s* **in** SUCCESSORS(*state*) **do**
    $\beta \leftarrow$ MIN(β, MAX-VALUE(*s, game*, α, β))
    if $\beta \leq \alpha$ **then return** α
  end
  return β