Adversarial Search for Game Playing

CZ3005: Artificial Intelligence

Shen Zhiqi

NANYANG TECHNOLOGICAL UNIVERSITY

Outline

- Games as search problems
- Minimax search strategy
- Evaluation functions

Games as Search Problems

Abstraction

- Ideal representation of real world problems
  - e.g. board games, chess, go, etc. as an abstraction of war games
  - Perfect information, i.e. fully observable
- Accurate formulation: state space representation

Uncertainty

- Account for the existence of hostile agents (players)
  - Other agents acting so as to diminish the agent's well-being
  - Uncertainty (about other agents' actions):
    - not due to the effect of non-deterministic actions
    - not due to randomness

→ Contingency problem

Games as Search Problems...

Complexity

- Games are abstract but not simple
  - e.g. chess: average branching factor = 35, game length > 50

  → complexity = 35^{50}(only 10^{40} for legal moves)
- Games are usually time limited
  - Complete search (for the optimal solution) not possible

  → uncertainty on actions desirability
  - Search efficiency is crucial

Games as real-world problems

- Need to handle uncertainty
  - Contingency, time constraints
- Need to deal with competition
  - Two-player, three-player, multi-player games, etc

Types of Games

	Deterministic	Chance
Perfect information	Chess, Checkers, Go, Othello	Backgammon, Monopoly
Imperfect information		Bridge, Poker, Scrabble, Nuclear war

Perfect information

- each player has complete information about his opponent's position and about the choices available to him

Two-player Games

Mainly two-player games are considered

Two sources of uncertainty

- Opponent
  - does not know in advance what action the opponent will make
- Time limits
  - searching for optimal decisions in large state spaces not practical
  - unlikely to find goal
  - must approximate

Zero-sum game

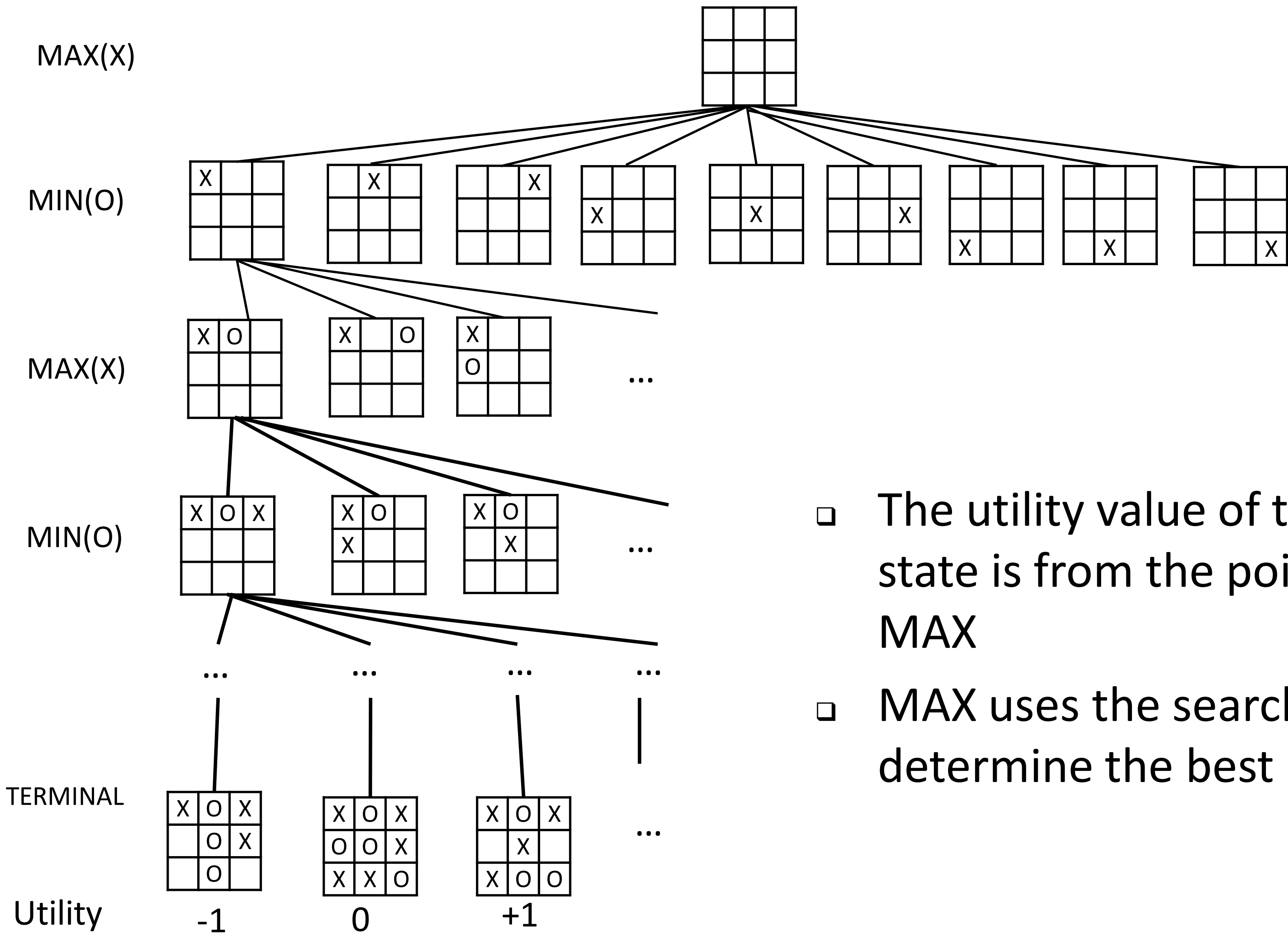
Game as a Search Problem

- Initial state: initial board configuration and indication of who makes the first move
- Operators: legal moves
- Terminal test: determines when the game is over
  - states where the game has ended: terminal states
- Utility function (payoff function): returns a numeric score to quantify the outcome of a game

Example: Chess

Win (+1), loss(-1) or draw (0)

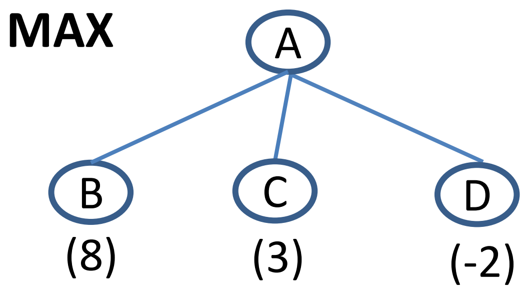
Game Tree for Tic-Tac-Toe



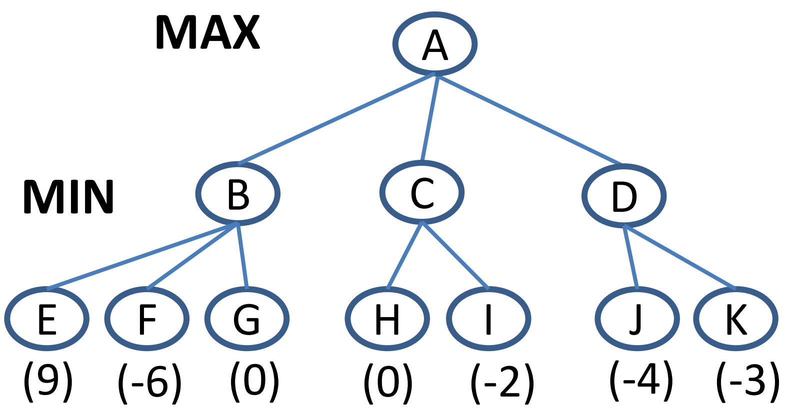
- The utility value of the terminal state is from the point of view of MAX
- MAX uses the search tree to determine the best move

What Search Strategy?

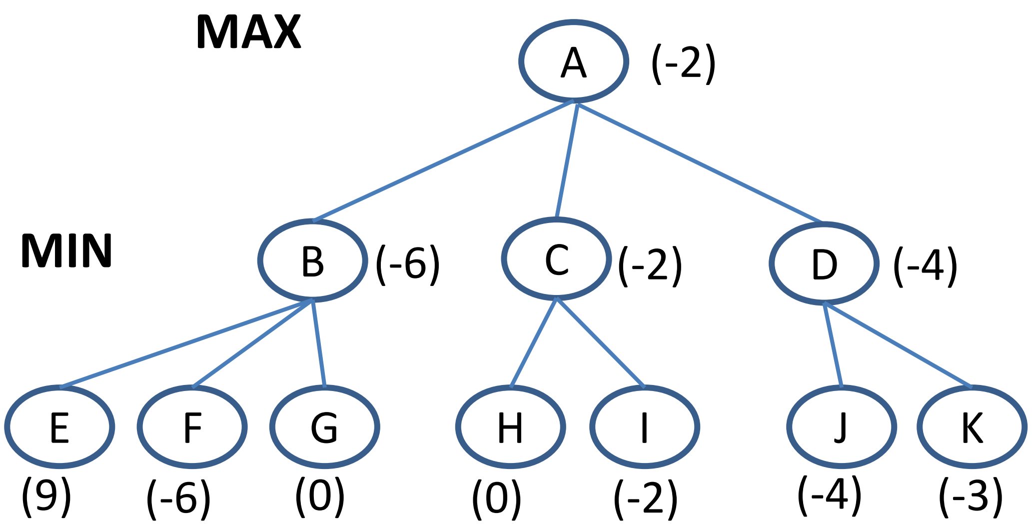
- One-play



- Two-play



What Search Strategy?...



Minimax Search Strategy

Search strategy

- Find a sequence of moves that leads to a terminal state (goal)

Minimax search strategy

- Maximize one's own utility and minimize the opponent's
  - Assumption is that the opponent does the same

- 3-step process:
  1. Generate the entire game tree down to terminal states
  2. Calculate utility
     1. Assess the utility of each terminal state
     2. Determine the best utility of the parents of the terminal state
     3. Repeat the process for their parents until the root is reached
  3. Select the best move (i.e. the move with the highest utility value)

Perfect Decisions by Minimax Algorithm

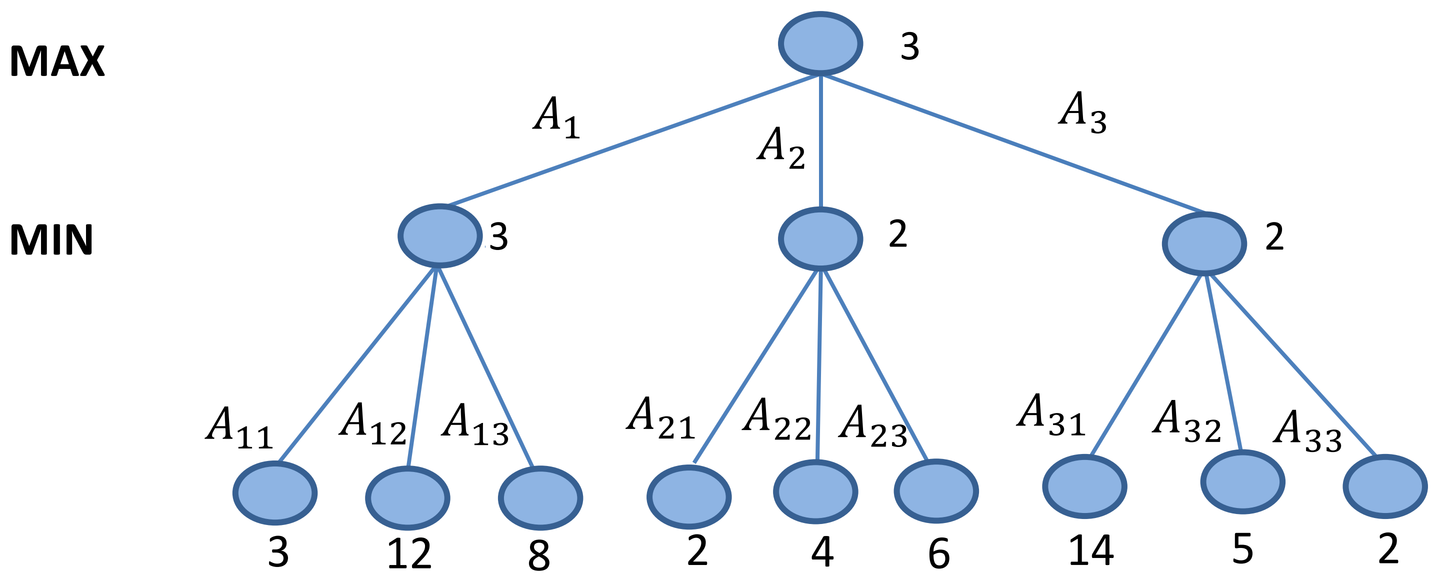
Perfect decisions: no time limit is imposed

- generate the complete search tree

Two players: MAX and MIN

- Choose move with best achievable payoff against best play
- MAX tries to max the utility, assuming that MIN will try to min it

Example



Minimax Decision

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

  for each *op* **in** OPERATIORS[*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*)

- Depth-first
- Value passed up the path one level at a time (back up)

Othello 4

	X	O	
	O	X	

- A player can place a new piece in a position if there exists at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another piece of the same kind, with one or more contiguous pieces from the opponent player between them
- After placing the new piece, the pieces from the opponent player will be captured and become the pieces from the same Player
- The player with the most pieces on the board wins

`X' plays first

X considers the game now

	X	O	
	O	X	

O considers the game now

	X	O	
	X	X	
	X		

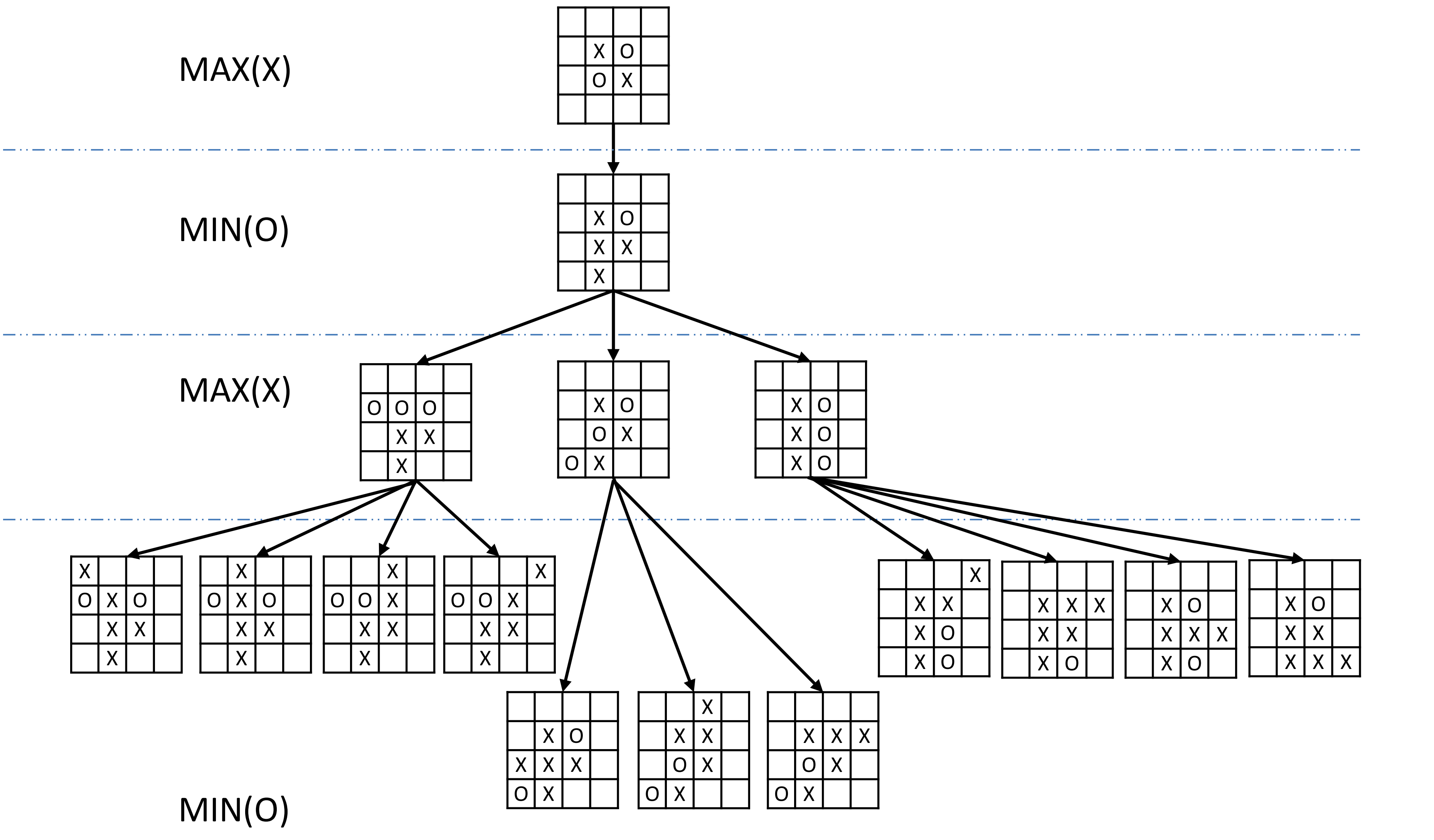
X considers the game now

O	O	O	
	X	X	
	X		

	X	O	
	O	X	
O	X		

	X	O	
	X	O	
	X	O	

Game Tree Othello 4



Imperfect Decisions

For chess, branching factor ≈ 35, each player typically makes 50 moves ⟶ for the complete game tree, need to examine 35^{100} positions

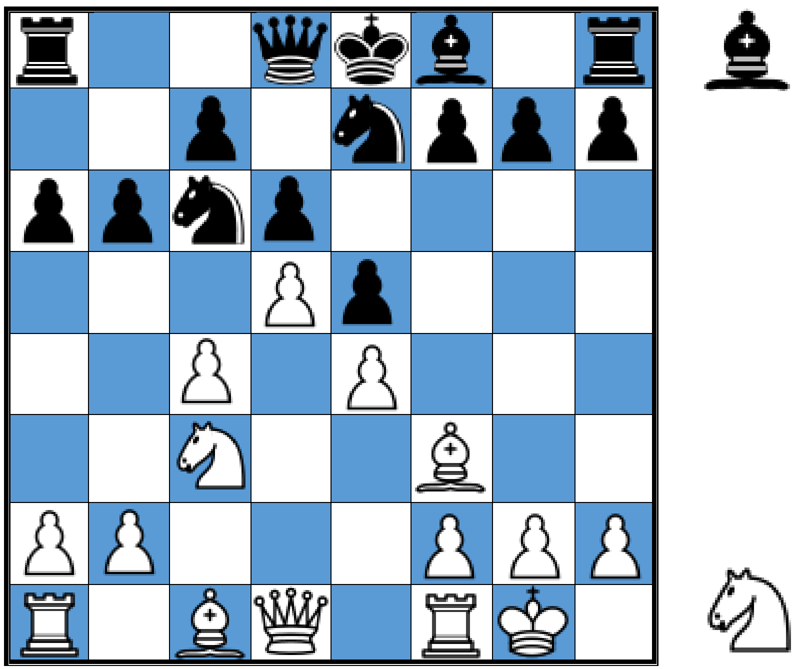
Time/space requirements ⟶ complete game tree search is intractable ⟶ impractical to make perfect decisions

Modifications to minimax algorithm

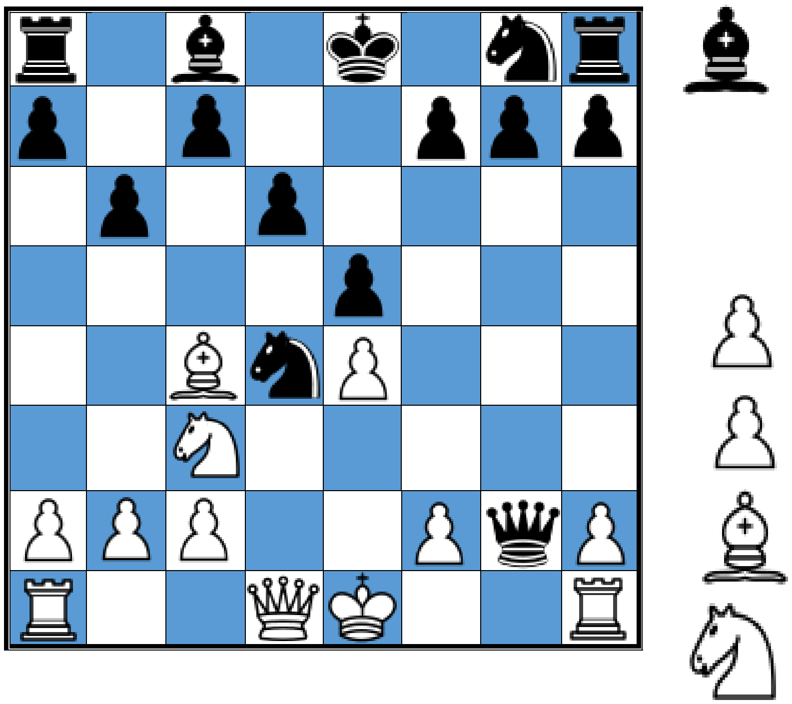
1. replace utility function by an estimated desirability of the position
   - Evaluation function
2. partial tree search
   - E.g., depth limit
   - Replace terminal test by a cut-off test

Evaluation Functions

Returns an estimate of the expected utility of the game from a given position



Black: to move
White: slightly better



White: to move
Black: winning

Evaluation Functions...

Requirements

- Computation is efficient
- Agrees with utility function on terminal states
- Accurately reflects the chances of winning

Trade off between accuracy and efficiency

Example (Chess)

Define features

- e.g. , (number of white queens) – (number of black queens)

Use a weighted sum of these features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots w_n f_n(s)$$

- Need to learn the weight

Evaluation Functions for Othello 4

- A corner of the board is one of the most important positions. A piece at the corner can help capture other pieces from the opponent player
- A square at the border is also more important than any position in the middle of the board

Heuristics for `X' is proposed as follows:

- For any non-terminal game state, the evaluation function is computed as

$$3(X_C - O_C) + 2(X_b - O_b) + (X_m - O_m)$$

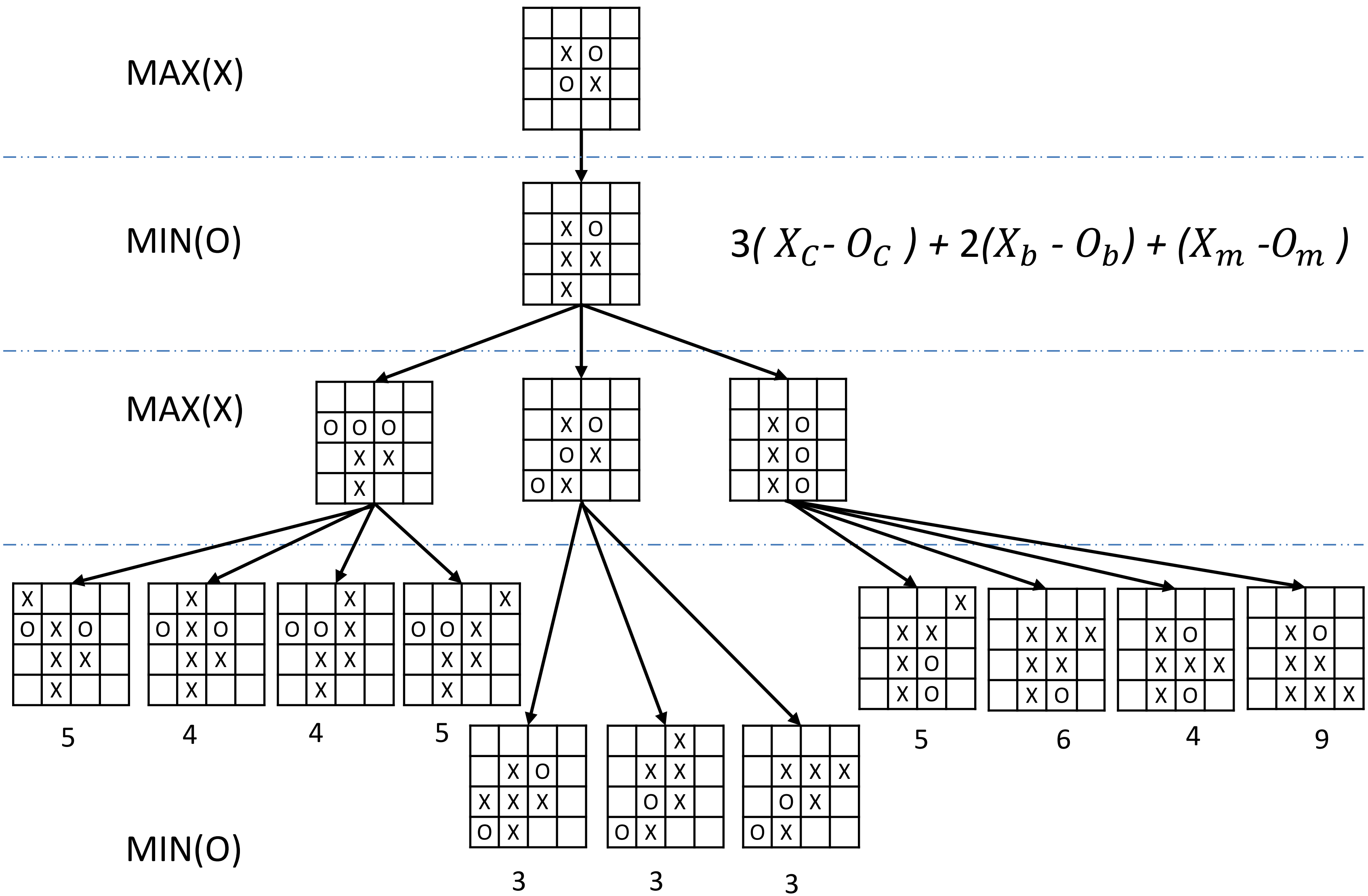
where X_C is the number of *X*'s at corners,

X_b is the number of *X*'s at the border (excluding corners) ,

X_m is the number of *X*'s in the middle of the grid,

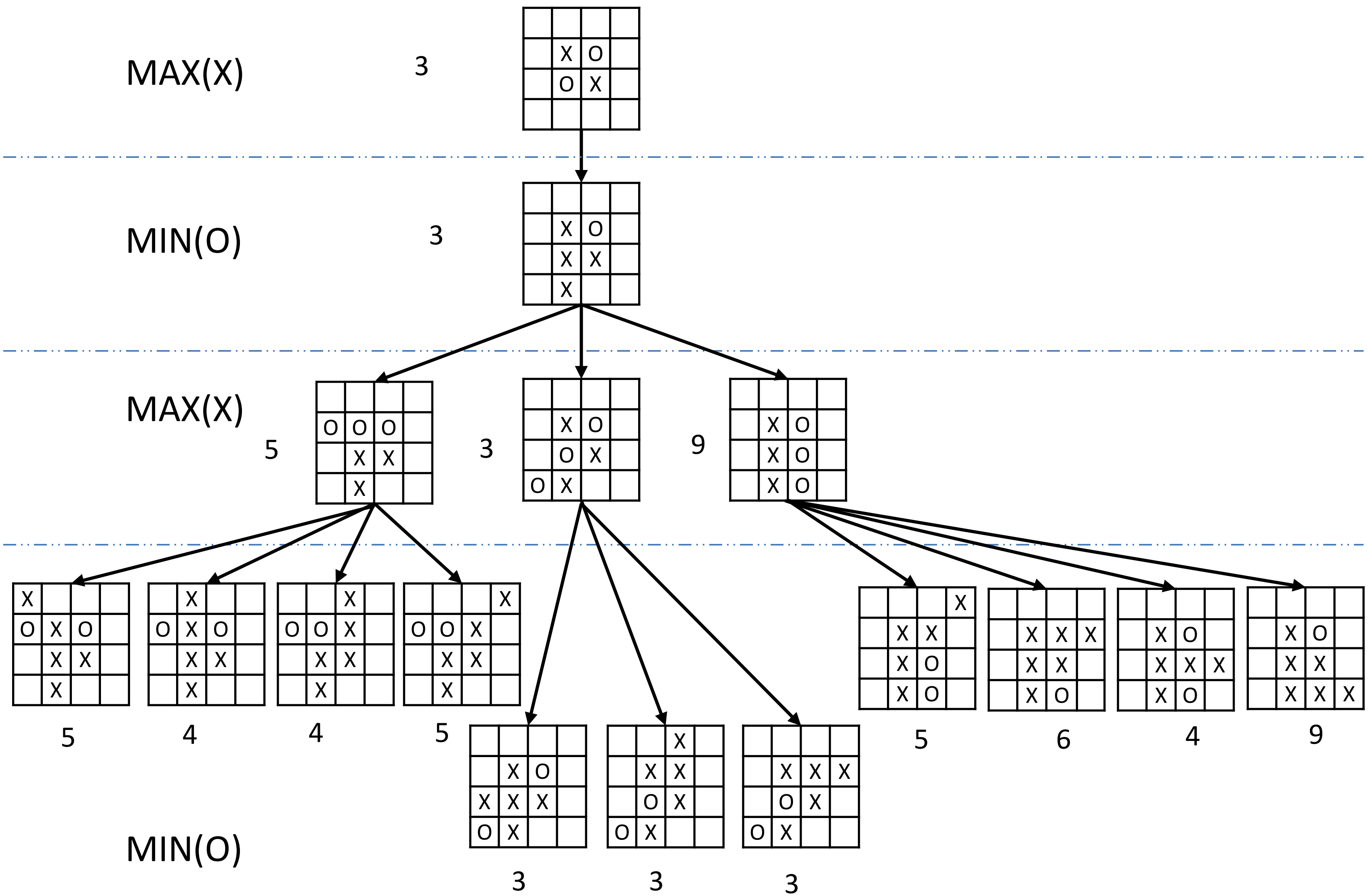
O_C , O_b and O_m are the number of *O*'s at the corners, the border and the middle of the board

Evaluation Functions for Othello 4 ...



$$3(X_C - O_C) + 2(X_b - O_b) + (X_m - O_m)$$

Minimax Search for Othello 4



Cutting O Search

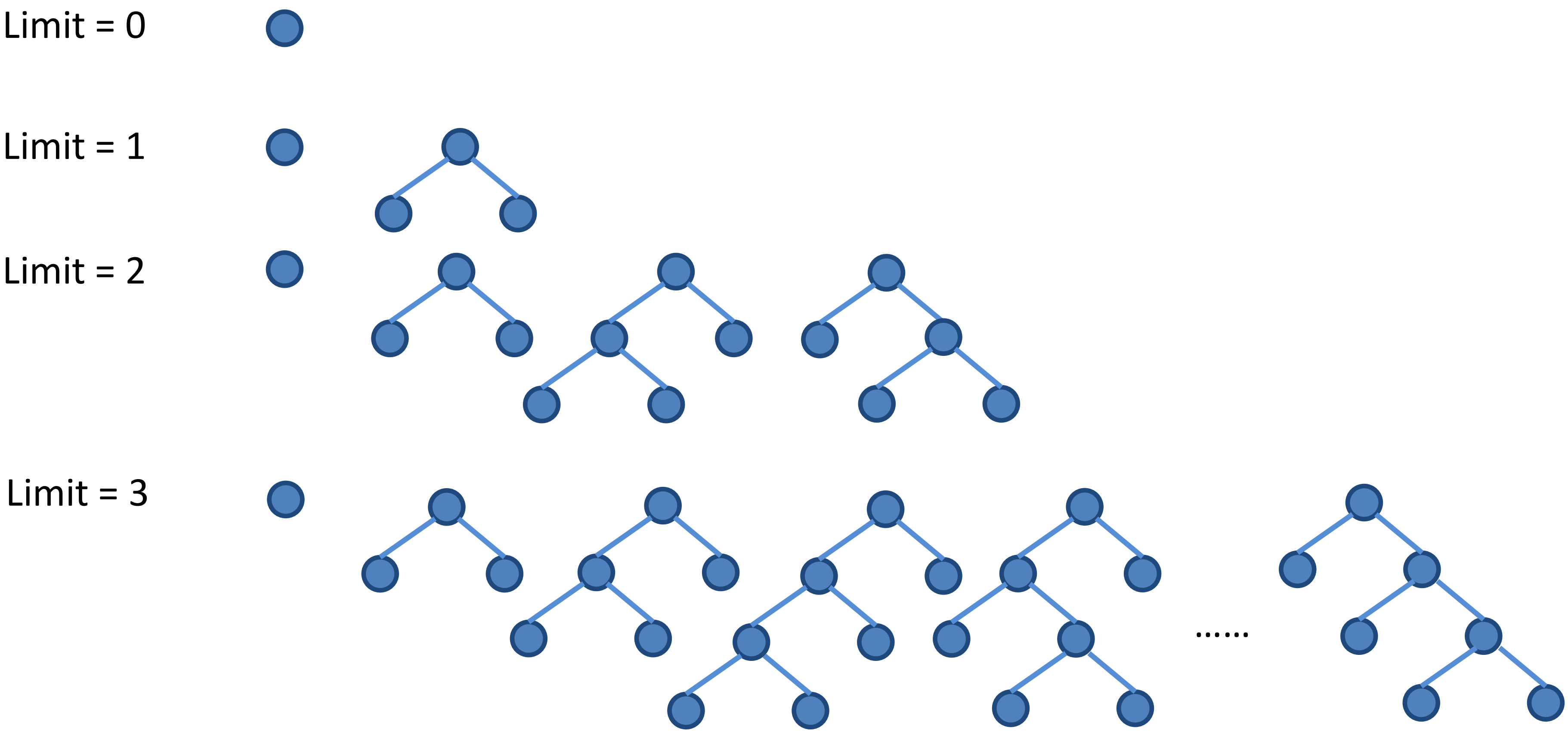
Depth-limited search

- depth is chosen such that the amount of time allowed will not be exceeded

Question? How to tell that in advance?

Answer: Iterative deepening search

Iterative deepening search



When time runs out, returns the move selected by the deepest completed search

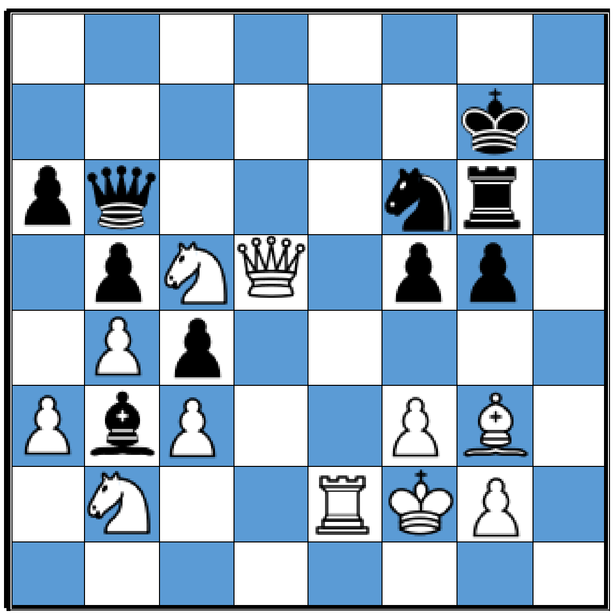
Quiescent Search

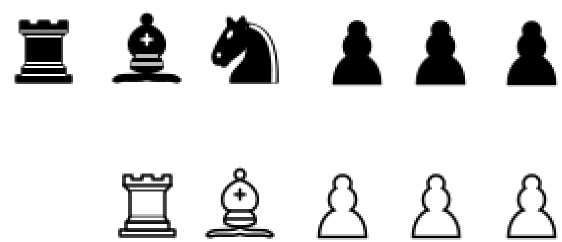
The evaluation function should only be applied to quiescent positions

- positions that are not likely to have large variations in evaluation in the near future

Example (Chess)

Positions in which favorable captures can be made





Black: to move
White: about to lose

Expansion of non-quiescent positions until quiescent positions are reached