Artificial Intelligence

[week #5]

Game Playing

Hilmy. A. T

hilmi.tawakal@gmail.com

Adversarial Search

In which we examine the problems that arise when we try to plan ahead in a world where other agents are planning against us.

- Competitive environments, in which the agents' goals are in conflict, giving rise to adversarial search problems—often known as games.
- For AI researchers, the abstract nature of games makes them an appealing subject for study.
- The state of a game is easy to represent
- Agents are usually restricted to a small number of actions whose outcomes are defined by precise rules.

Why games?

- Games are well-defined problems that are generally interpreted as requiring intelligence to play well.
- Games are interesting because → too hard to solve.
- Introduces uncertainty → opponents moves can not be determined in advance.
- Search spaces can be very large. For chess:
  - Branching factor: 35
  - Depth: 50 moves each player
  - Search tree: 35^{100} nodes (~10^{40} legal positions)
- Despite this, human players do quite well without doing much explicit search. They seem to rely on remembering many patterns.
- Good test domain for search methods and development of pruning methods that ignore portions of the search tree that do not affect the outcome.

Game Types

	deterministic	stochastic
perfect information	chess, checkers, go, othello	monopoly, backgammon
imperfect information		bridge, poker, nu-clear war

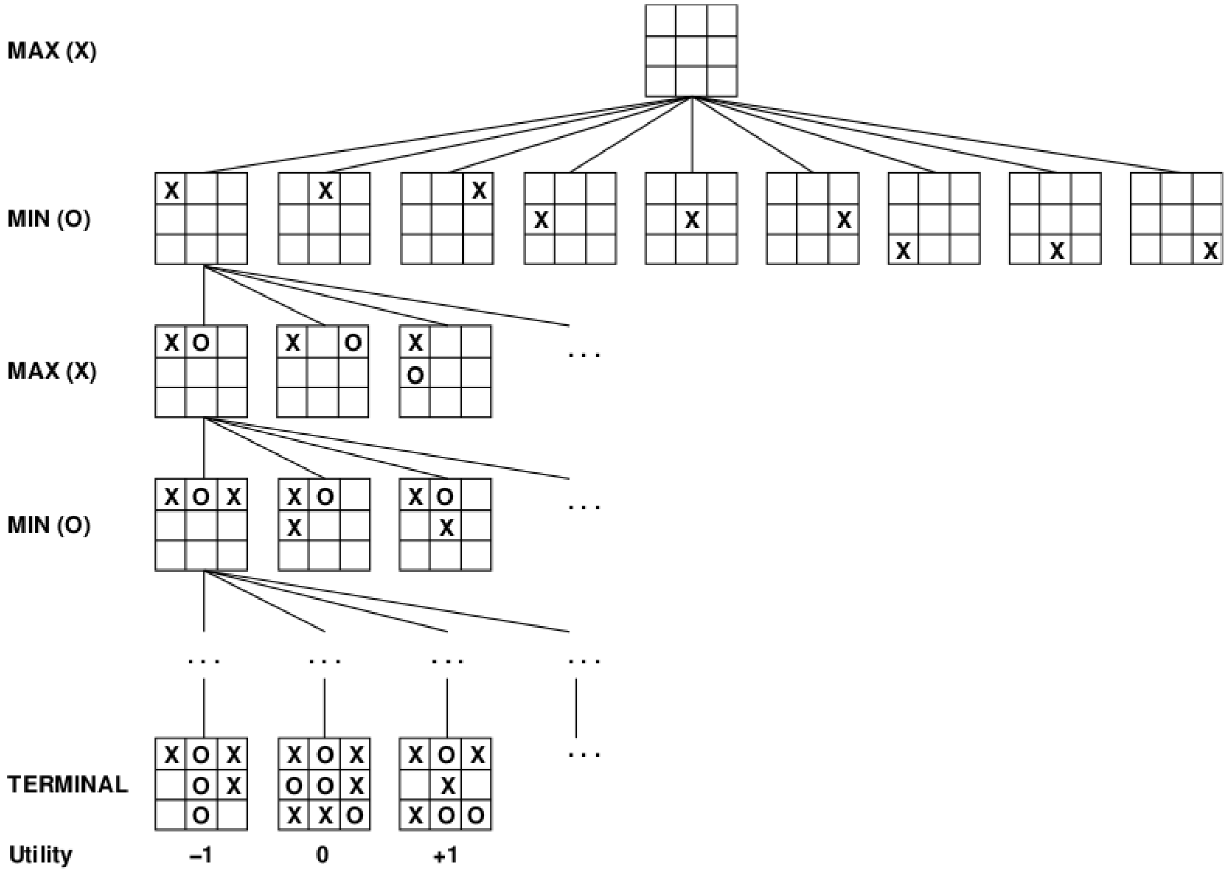
Games for AI research usually:

- Deterministic
- Perfect information
- 2 player
- Zero-sum game

Game Playing Problem

- Instance of the general search problem
- States where the game has ended are called terminal states
- A utility (payoff) function determines the value of terminal states, e.g. win=+1, draw=0, lose=-1.
- In two-player games, assume one is called MAX (tries to maximize utility) and one is called MIN (tries to minimize utility).
- In the search tree, first layer is move by MAX, next layer by MIN, and alternate to terminal states.
- Each layer in the search is called a **ply**

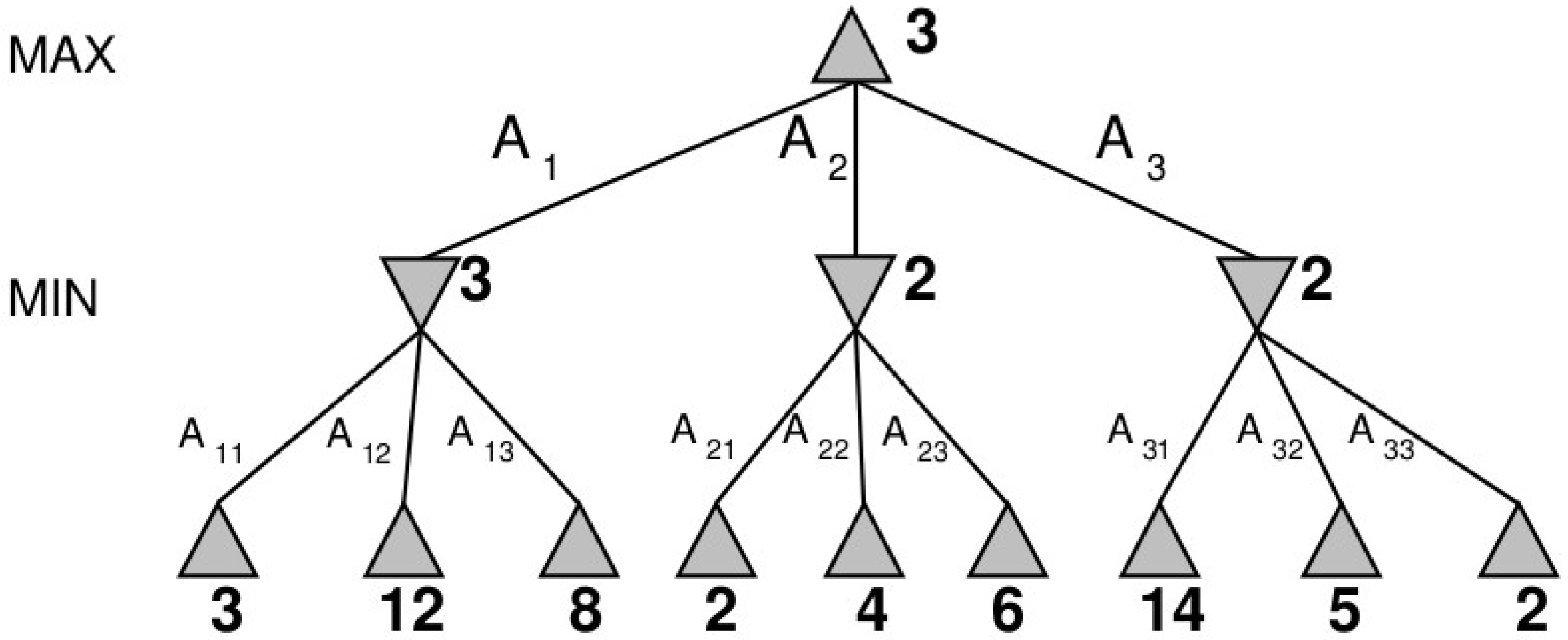
Example(tic tac toe)



MINIMAX Algorithm

- General method for determining optimal move.
- Generate complete game tree down to terminal states.
- Compute utility of each node bottom up from leaves toward root.
- At each MAX node, pick the move with maximum utility.
- At each MIN node, pick the move with minimum utility (assumes opponent always acts correctly to minimize utility).
- When reach the root, optimal move is determined

MINIMAX Algorithm



MINIMAX Algorithm

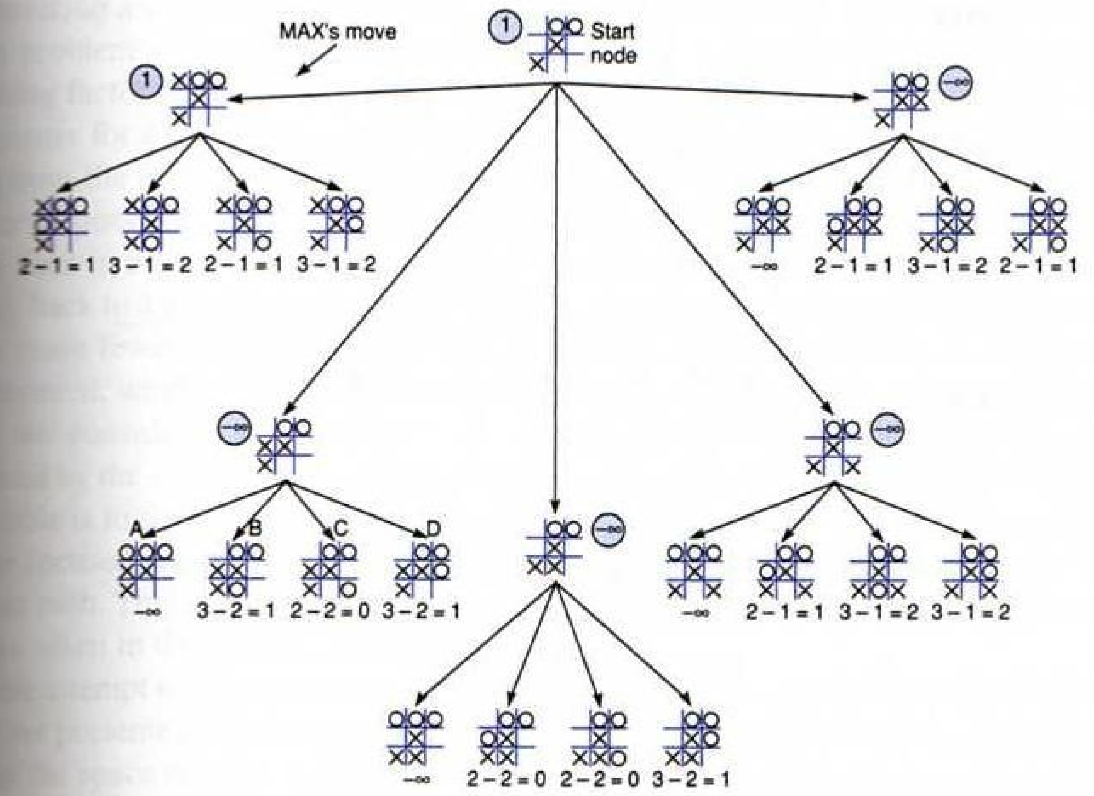


Figure 4.25 Two-ply minimax applied to X's move near the end of the game, from Nilsson (1971).

Imperfect Decisions

- Generating the complete game tree for all but the simplest games is intractable.
- Instead, cut off search at some nodes and estimate expected utility using a heuristic evaluation function.
- Ideally, a heuristic measures the probability that MAX will win given a position characterized by a given set of features.
- Sample chess evaluation function based on "material advantage:" pawn=1, knight/bishop=3, rook=5, queen=9
- An example of a weighted linear function:

$$\boxed{Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)}$$

State of the art program

- Chess: DeepBlue beat world champion
  - Customized parallel hardware
  - Highly-tuned evaluation function developed with expert
  - Comprehensive opening and end-game databases
- Checkers:
  - Chinook is world champion, previous champ held title for 40 years had to withdraw for health reasons.
- Othello: Programs best players (e.g. Iago).
- Backgammon: Neural-net learning program TDGammon one of world's top 3 players.
- Go: Branching factor of ~360 kills most search methods. Best programs still mediocre