Lecture 8

AI in Game Playing

Adversarial and Stochastic Search

Textbook: Artificial Intelligence, A Modern Approach, IV edition
Authors: Russell and Norvig
Reading material: Chapter 5, sections 5.1 to 5.3; 5.5, pp. 146-161; 164-168

1

Outline

- AI in game playing
- Adversarial search
- Min-max tree
- Alpha-beta pruning
- Stochastic games and chance nodes

2

Games

- Multiagent environment
- Cooperative (collusion) vs. competitive
- Cooperative environment
  - multiple agents working on part of a problem and communicating
- Competitive environment: agents' goals are in conflict
  - adversarial search: two or more adversaries
  - maximize individual profit either by colluding or by competing
- Game Theory
  - a **branch of economics** borrowed in Artificial Intelligence
  - works on the principle **of individual profit maximization**
  - Cooperates (colludes) or competes with others for profit maximization

3

AI in Board-Games

- Small well-defined set of rules
- Well defined knowledge set
- Easy to evaluate performance
- Less control of continuous actions
  - opponent lowers the gain of an agent
  - game is based on stochastic moves based upon chance
  - complete information about the environment is not available
- Search spaces too large for exhaustive search
  - requires effective pruning

4

Types of Games

	Deterministic	Chance (probability based)
Perfect Information	Chess, checkers, go, othello	Backgammon, monopoly
Imperfect Information		Bridge, poker, scrabble, nuclear war

5

Status of AI in Games

- Chess – Deep Blue (World Champion 1997)
  - searches 200 million positions per second
- Checkers – Chinook (World Champion 1994)
  - used a precomputed database of 444 billion positions
- Othello – Logistello
  - humans are no match for computers at Othello
- Backgammon – TD-Gammon (top three)
- Go – AlphaGo(World champion 2016)
- Bridge (Bridge Barron 1997, GIB 2000)
  - imperfect information
  - multiplayer with two teams; each team has two players

6

Deterministic Games

- Fully observable
- Uses an evaluation function to evaluate the state of the game
- Two agents in games such as chess, Kalah
  - actions generally alternate (not in Kalah)
  - zero sum games: one player wins (+1), one player loses (-1)
  - utility values at the end of the game are equal and opposite
- Multiple agents' games such as checkers
  - compete or cooperate (collude) to maximize profits

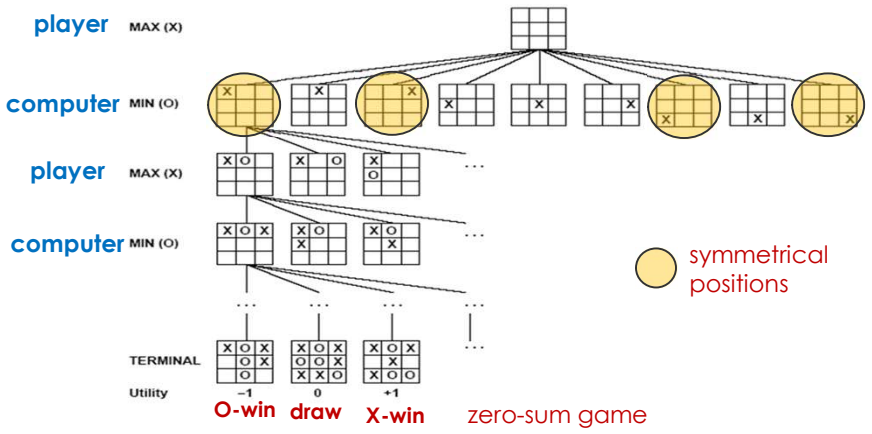
7

Games as State-space Search

- Games are modeled as a state-space search
  - each potential board or game position is a state
  - each possible move transits to another state
- The state space is huge
  - large branching factor (about 35 for chess during middle-game)
  - terminal state could be deep (about 50 levels for chess; 79 level for the game of Go)
- Modeled as adversarial search
  - a level corresponds to the same player's move
  - for two agents, search level generally alternates between player and the computer

8

Adversarial Search Illustration



9

Issues and Strategy

- Issues
  - opponents' moves are unpredictable
  - in stochastic (chance) games, such as dice-based games players have less control over moves they make
- Strategy
  - specify a counter-move for opponents' move to maximize profits
  - search is time-bound
  - **depth-limited search** due to imposed time-limits and computational power constraints
  - identify conditions to avoid exploring the part of search tree not containing goal state
  - identify features to evaluate the next state for a good move
  - use database lookup for previously visited equivalent nodes for improving computational efficiency

10

Adversarial Search

- Search tree alternates between computer and the player
- Uses **minimax** strategy
  - computer maximizes its state evaluation and minimizes opponent's
- Objective utility function to evaluate a state
- Initial state is the starting board position
- Goal state is a leaf-node when winning conditions are met
- Successor function
  - returns a list of (move, successor neighboring state) pairs
- **Terminal test** matches the current state with one of the goal states

11

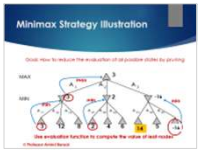
Adversarial Tree Structure

- The root of the tree is the initial state
  - next level is all of MAX's moves
  - next level is all of MIN's moves
  - …
- Example: Tic-Tac-Toe
  - root has 9 blank squares (MAX)
  - level 1 has 8 blank squares (MIN); Level 2 has 7 blank squares (MAX)
  - …
- Utility function: win for X is +1; win for O is -1

12

Minimax Strategy

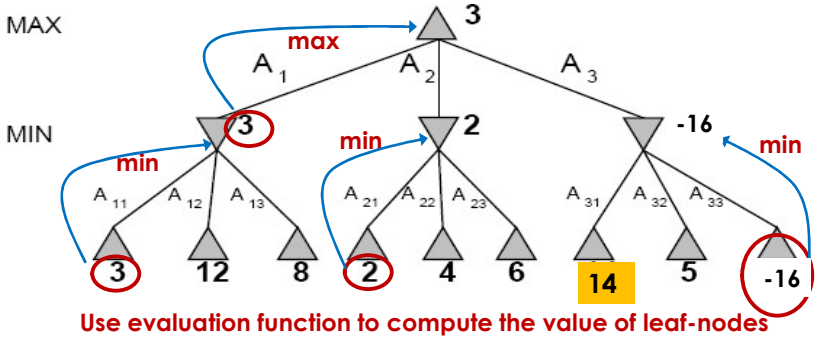
- Approach
  - choose the move with the highest possible max-value
  - remove equivalent states such as symmetric positions
  - max wins if value > 0
  - Min wins if value < 0 min (it is in the opposite direction)
- Max's goal: get highest value > 0
- Min's goal: get the lowest value < 0
- Utility value of a node (backed-up value):
  - start from the terminal nodes
  - if N is terminal, use the utility value
  - if N is a max move, take the max of utility value of successors
  - if N is a min move, take the min of utility values of successors



13

Minimax Strategy Illustration

Goal: How to reduce the evaluation of all possible states by pruning



Use evaluation function to compute the value of leaf-nodes

14

Minimax Algorithm

```
function MINIMAX-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state)
  return move

function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
    if v2 > v then
      v, move ← v2, a
  return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move ← v2, a
  return v, move
```

Algorithm taken from the book "Artificial Intelligence – A modern Approach by Russell and Norvig

15

Minimax Properties

- Evaluation function
  - **eval(s) = $w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s)$** where each function f_i is associated with some user-defined meaningful feature
  - an example of a feature in chess: *number of player's queen – number of opponent's queens; location and threat to pieces*
- Complexity properties
  - complete for finite trees
  - optimal
  - time complexity: $O(b^m)$ for depth-first exploration
  - space complexity: O(bm) for depth-first exploration

16

Multiplayer Game Playing

- Temporary alliance
- Replace single value at each node by a tuple of values
  - **example:** three player game: vector has three fields, one for each player
- terminal states: the utility value is calculated for each player
- nonterminal states: profit maximization

Example does not assume collusion

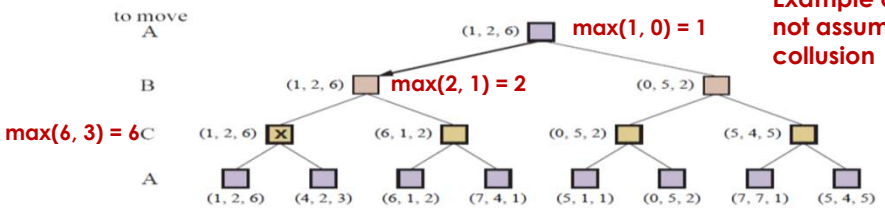


Figure 5.4 The first three ply of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

Figure taken from the book "Artificial Intelligence – A modern Approach by Russell and Norvig

17

Issues in Minmax Strategy

- Issues
  - number of examined game-states expand exponentially
  - search is depth-limited due to limited computational power and large branching factor
  - example: b (in chess) = 35; processing speed: 10^5 nodes/second; move-time 100 seconds gives depth-limit m as

$$\log_{35}(10^5 \times 100) \approx 5$$

- Solution: improve look-ahead depth
  - prune the branches and nodes which may not yield any solution

18

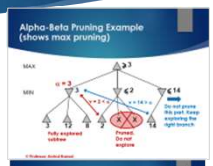
Optimizing Minmax Search

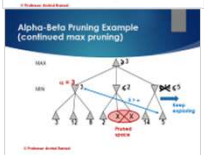
- Use depth-limited search
- Use database lookup for prior solutions from the current state
- Avoid generating equivalent states (e.g. 4 different first corner moves in tic-tac-toe)
- Use alpha-beta cutoffs:
  - improves depth-limit around a factor of two

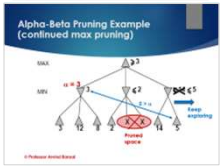
19

Alpha-Beta Pruning

- Advantages
  - prunes the branches that will not yield a solution
  - reduces effective branching factor
  - improves the depth-limit
- Approach
  - start evaluating from left-most branches
  - α = current highest value along the left path for max
  - β = current lowest value along the left path for min
  - v = value of the left-most child of right subtrees
- If v < α, *max* will prune the subtree rooted at v
  - *max*(3, *min*(v = 2, x, y) …) is 3 irrespective of values of x and y because 2 < 3
- if v > β, *min* will prune the subtree rooted at v
  - *min*(2, *max*(v = 3, x, y) …) is 2 irrespective of value of x and y

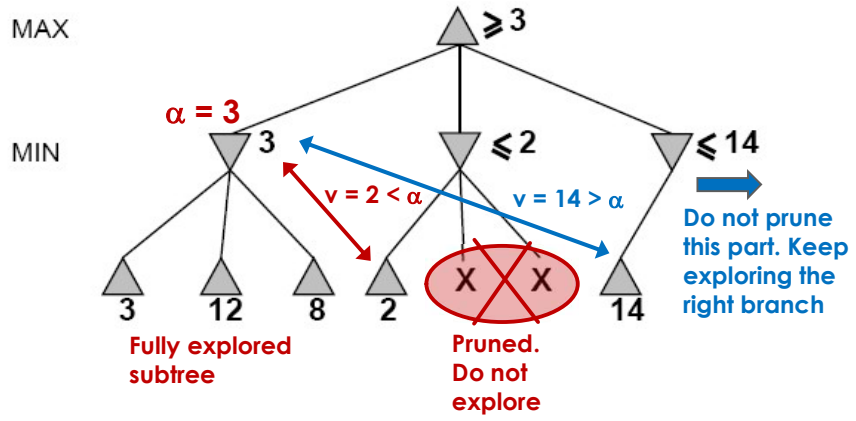






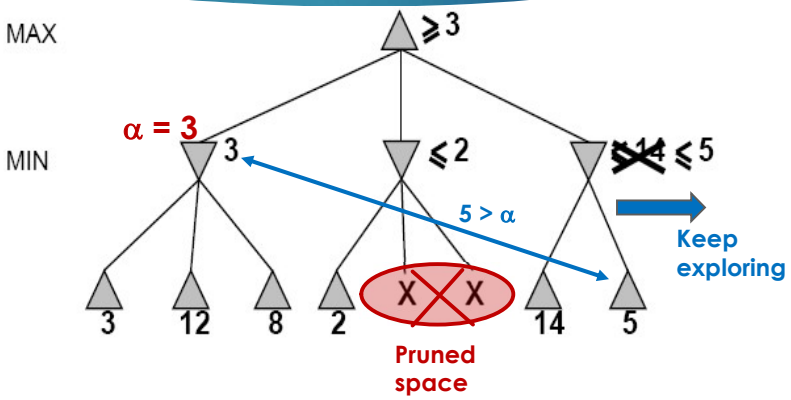
20

Alpha-Beta Pruning Example (shows max pruning)



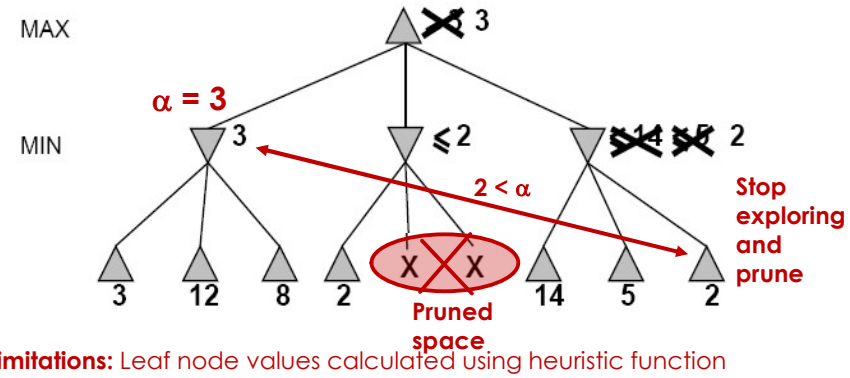
21

Alpha-Beta Pruning Example (continued max pruning)



22

Alpha-Beta Pruning Example (continued max pruning)



Limitations: Leaf node values calculated using heuristic function
Or evaluation function of the current state which may not be accurate

23

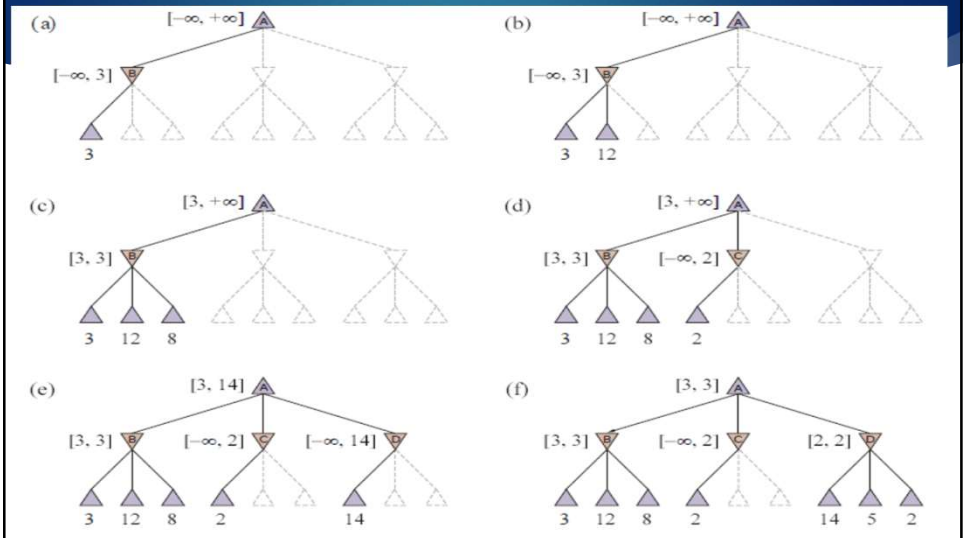
Alpha-Beta Algorithm

```
function ALPHA-BETA-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state, −∞, +∞)
  return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

Algorithm taken from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

24

Alpha-Beta Pruning Book Example (Fig. 5.5)

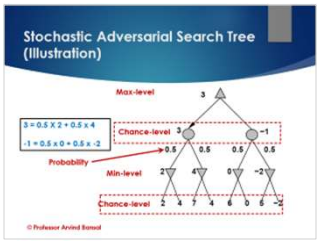


Algorithm taken from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

25

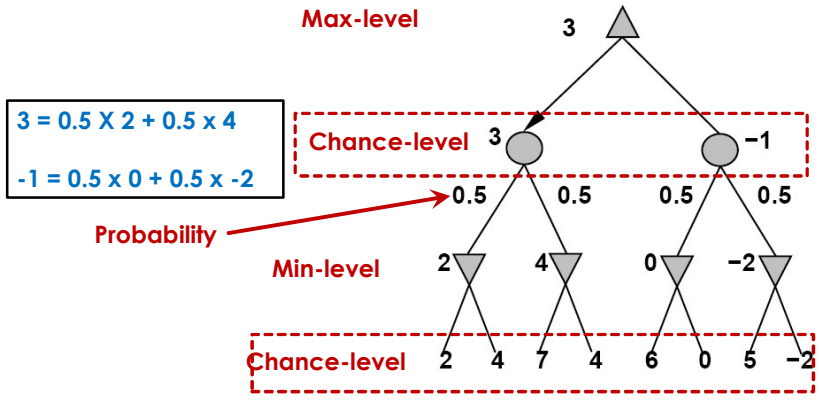
Stochastic Games

- In stochastic games, chance is introduced by probability such as dice, coin-flipping, or card-shuffling
- An extra level (**chance-level**) is introduced between **max and min**, or **min and max** to handle the probability of stochastic process
- Values are calculated as:
  - chance-value = $\sum_{i=1}^{i=n} \mathbf{probability_i \times nodeValue_i}$
  - *max-value* = maximum of corresponding children chance-nodes)
  - *min-value* = minimum of corresponding children chance-nodes)



26

Stochastic Adversarial Search Tree (Illustration)



27

Summary

- Various classes of games
  - deterministic games; stochastic games; games with partial observability; two player games; multiplayer games
- Two player games are adversarial, and strategy involves MinMax
- Multiplayer games involve profit maximization
  - profit maximization involves both collusion and competition
- Depth-limiting, alpha-beta pruning and database lookup improve the efficiency for adversarial search
- Stochastic games embed a chance layer between min-layers and max-layers
- Limitations of game search algorithms
  - evaluation function should be accurate
  - in stochastic games, look ahead does not work due to mix-up in probability

28