Title: Adverserial Search

AIMA: Chapter 5 (Sections 5.1, 5.2 and 5.3)

Introduction to Artificial Intelligence

CSCE 476-876, Fall 2020

URL: www.cse.unl.edu/~choueiry/F20-476-876

Berthe Y. Choueiry (Shu-we-ri)

(402)472-5444

Outline

- Introduction
- Minimax algorithm
- Alpha-beta pruning

Context

- In an MAS, agents affect each other's welfare
- Environment can be cooperative or competitive
- Competitive environments yield adverserial search problems (games)
- Approaches: mathematical game theory and AI games

Game theory vs. AI

- AI games: fully observable, deterministic environments, players alternate, utility values are equal (draw) or opposite (winner/loser)
  In vocabulary of game theory: deterministic, turn-taking, two-player, zero-sum games of perfect information
- Games are attractive to AI: states simple to represent, agents restricted to a small number of actions, outcome defined by simple rules
  Not croquet or ice hockey, but typically board games
  Exception: Soccer (Robocup `www.robocup.org/`)

Board game playing: an appealing target of AI research

Board game: Chess (since early AI), Othello, Go, Backgammon, etc.

- Easy to represent
- Fairly small numbers of well-defined actions
- Environment fairly accessible
- Good abstraction of an enemy, w/o real-life (or war) risks :—)

But also: Bridge, ping-pong, etc.

Characteristics

- 'Unpredictable' opponent: contingency problem
  (interleaves search and execution)
- Not the usual type of 'uncertainty':
  no randomness/no missing information (such as in traffic)
  but, the moves of the opponent expectedly non benign
- Challenges:
  - huge branching factor
  - large solution space
  - Computing optimal solution is infeasible
  - Yet, decisions must be made. Forget A*...

Discussion

- What are the theoretically best moves?
- Techniques for choosing a good move when time is tight
  $\surd$ Pruning: ignore irrelevant portions of the search space
  $\times$ Evaluation function: approximate the true utility of a state without doing search

Two-person Games

- 2 player: Min and Max
- Max moves first
- Players alternate until end of game
- Gain awarded to player/penalty give to loser

Game as a search problem:

- Initial state: board position & indication whose turn it is
- Successor function: defining legal moves a player can take
  Returns $\{(\text{move, state})^*\}$
- Terminal test: determining when game is over
  states satisfy the test: terminal states
- Utility function (a.k.a. payoff function): numerical value for outcome e.g., Chess: win=1, loss=-1, draw=0

Usual search

Max finds a sequence of operators yielding a terminal goal scoring winner according to the utility function

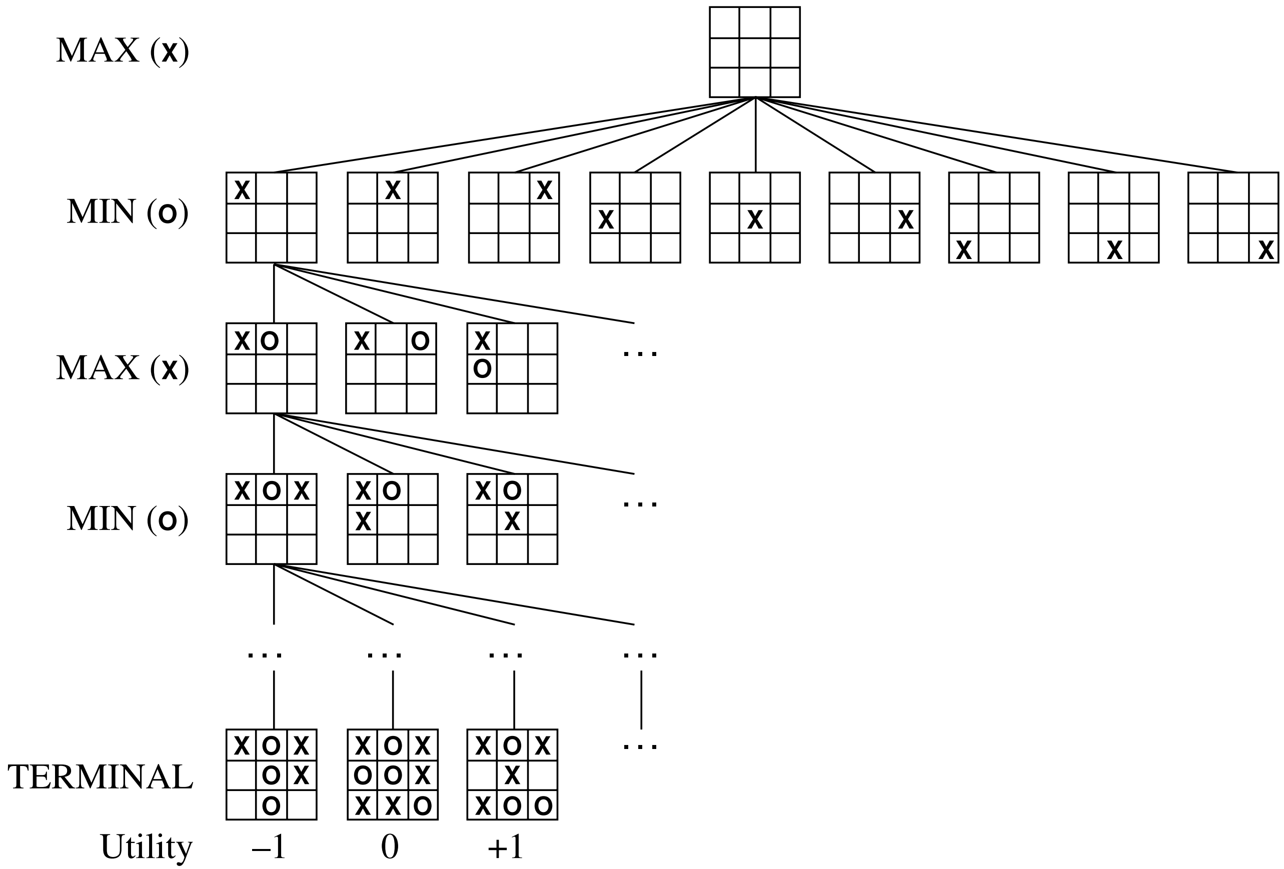
Game search

- Min actions are significant
  Max must find a strategy to win regardless of what Min does:
  $\longrightarrow$ correct action for Max for each action of Min
- Need to approximate (no time to envisage all possibilities difficulty): a huge state space, an even more huge search space
  e.g., chess: $\begin{cases} 10^{40} \text{ different legal positions} \\ \text{Average branching factor=35, 50 moves/player} = 35^{100} \end{cases}$
- Performance in terms of time is very important

Example: Tic-Tac-Toe

Max has 9 alternative moves

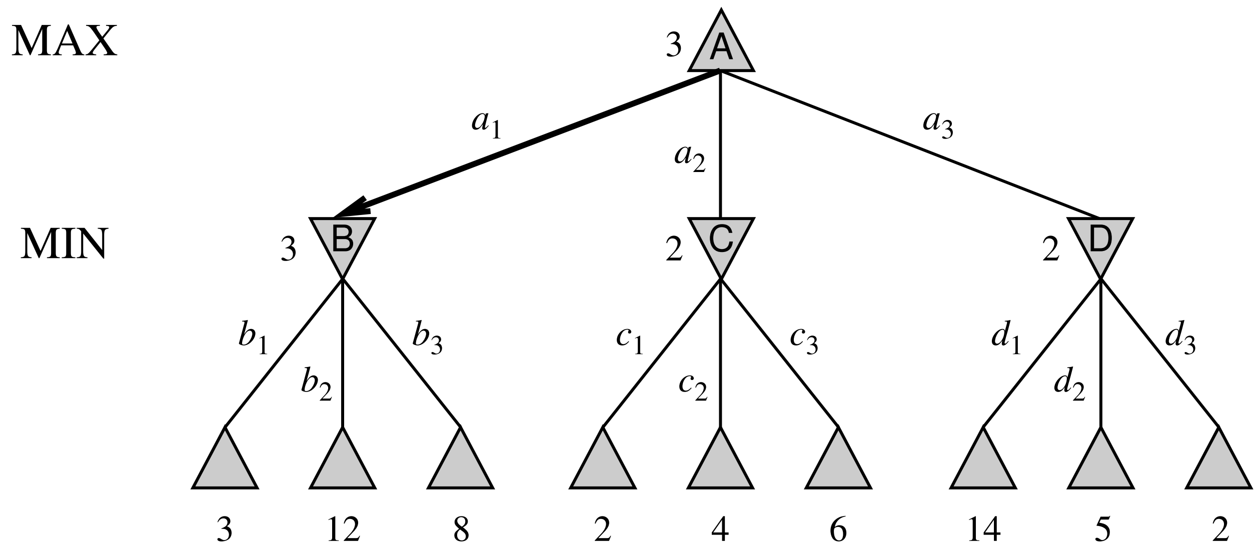
Terminal states' utility: Max wins=1, Max loses = -1, Draw = 0



Example: 2-ply game tree

Max's actions: a_1, a_2, a_3

Min's actions: b_1, b_2, b_3



Minimax algorithm determines the optimal strategy for Max
→ decides which is the best move

Minimax algorithm

- Generate the whole tree, down to the leaves
- Compute utility of each terminal state
- Iteratively, from the leaves up to the root, use utility of nodes at depth d to compute utility of nodes at depth $(d-1)$:

MIN 'row': minimum of children

MAX 'row': maximum of children

MINIMAX-VALUE (n)

$$\begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal node} \\ max_{s \in Succ(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a Max node} \\ min_{s \in Succ(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a Min node} \end{cases}$$

Minimax decision

- MAX's decision: minimax decision maximizes utility under the assumption that the opponent will play perfectly to his/her own advantage
- Minimax decision maximes the worst-case outcome for Max (which otherwise is guaranteed to do better)
- If opponent is sub-optimal, other strategies may reach better outcome better than the minimax decision

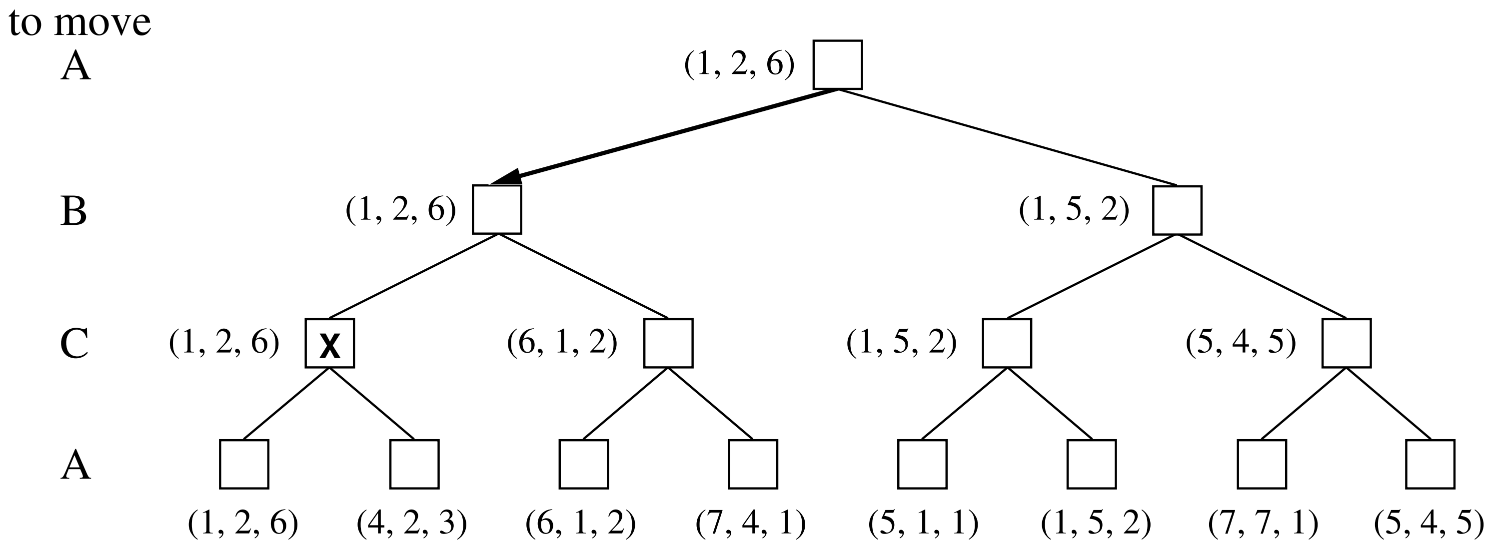
Minimax algorithm: Properties

- m maximum depth
  b legal moves
- Using Depth-first search, space requirement is:
  $O(bm)$: if generating all successors at once
  $O(m)$: if considering successors one at a time
- Time complexity $O(b^m)$
  Real games: time cost totally unacceptable

Multiple players games

Utility(n) becomes a vector of the size of the number of players

For each node, the vector gives the utility of the state for each player



Alliance formation in multiple players games

How about alliances?

- A and B in weak positions, but C in strong position
  A and B make an alliance to attack C (rather than each other
  $\rightarrow$ Collaboration emerges from purely selfish behavior!
- Alliances can be done and undone (careful for social stigma!)
- When a two-player game is not zero-sum, players may end up automatically making alliances (for example when the terminal state maximizes utility of both players)

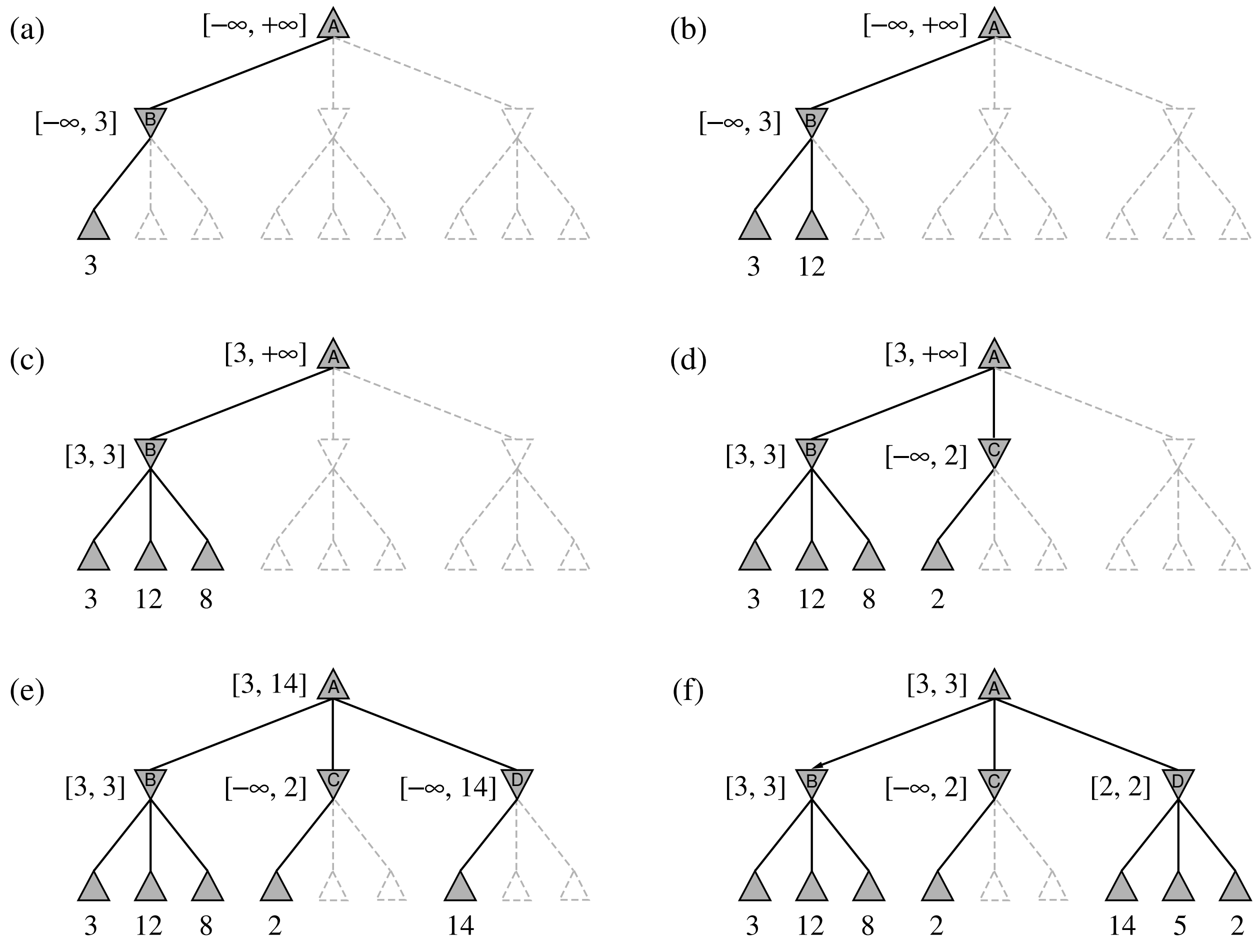
Alpha-beta pruning

- Minimax requires computing all terminal nodes: unacceptable
- Do we really need to do compute utility of all terminal nodes?
  ... No, says John McCarthy in 1956:

  It is possible to compute the correct minimax decision without looking at every node in the tree, and yet get the correct decision
- Use pruning (eliminating useless branches in a tree)

B.Y. Choueiry

Example of alpha-beta pruning

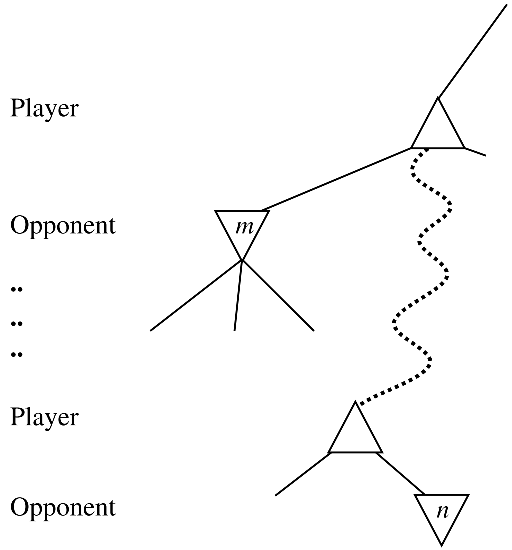


Try 14, 5, 2, 6 below D

General principal of Alpha-beta pruning

If Player has a better choice m at
- — a parent node of n
- — any choice point further up

n will never be reached in <u>actual play</u>

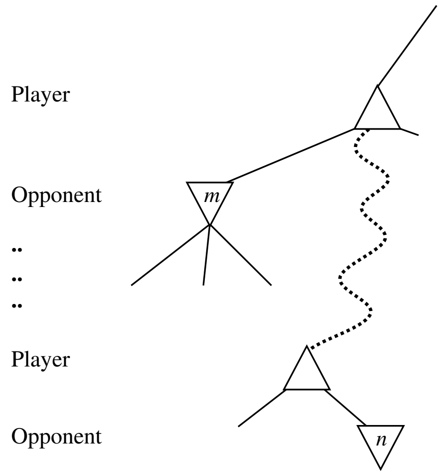


Once we have found enough about n (*e.g.*, through one of it descendants), we can prune it (*i.e.*, discard all its remaining descendants)

Mechanism of Alpha-beta pruning

α: value of best choice so far for MAX, (maximum)

β: value of best choice so far for MIN, (minimum)

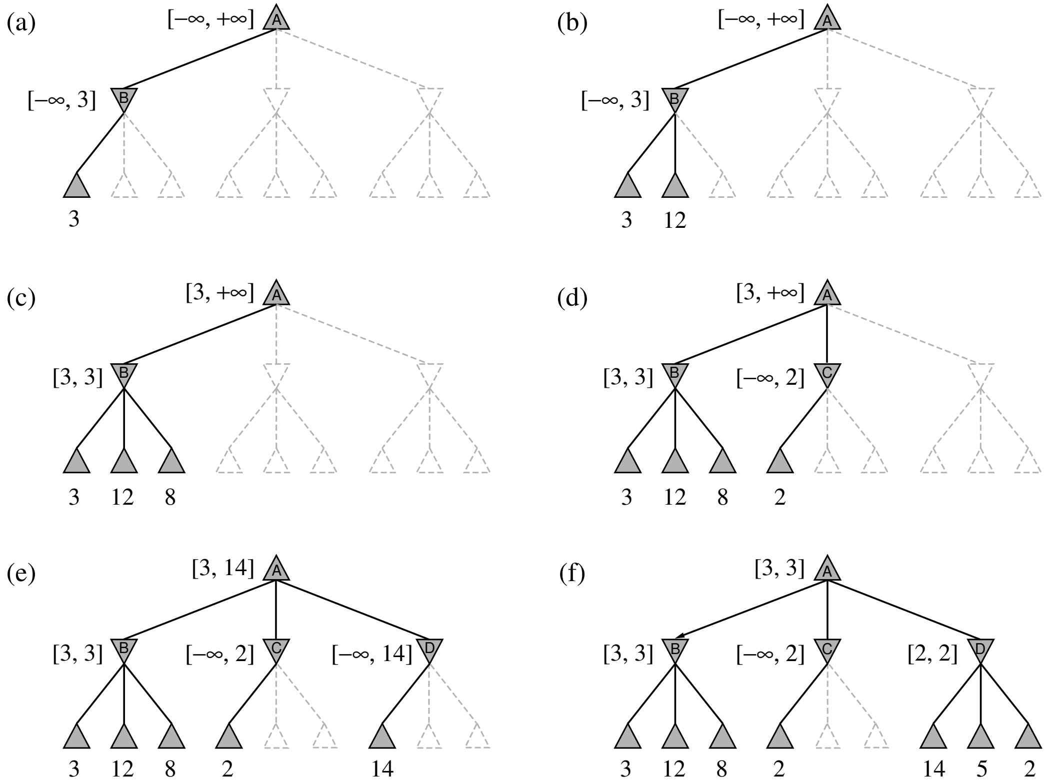


Alpha-beta search:

- updates the value of α, β as it goes along
- prunes a subtree as soon as its worse then current α or β

Effectiveness of pruning

Effectiveness of pruning depends on the order of new nodes examined



Savings in terms of cost

- Ideal case:
  Alpha-beta examines $O(b^{d/2})$ nodes (vs. Minimax: $O(b^d)$)
  $\rightarrow$ Effective branching factor $\sqrt{b}$ (vs. Minimax: b)

- Successors ordered randomly:
  $b > 1000$, asymptotic complexity is $O((b/\log b)^d)$
  b reasonable, asymptotic complexity is $O(b^{3d/4})$

- Practically: Fairly simple heuristics work (fairly) well