Title: Game Playing
AIMA: Chapter 5 (Sections 5.1, 5.2 and 5.4)

Introduction to Artificial Intelligence
CSCE 476-876, Spring 2003
URL: `www.cse.unl.edu/~choueiry/S03-476-876`

Berthe Y. Choueiry (Shu-we-ri)
`choueiry@cse.unl.edu, (402)472-5444`

Summary: CSP

- Definition, representation, types of domains/constraints
- Solving by consistency checking (arc-consistency, k-consistency,..)
- Solving with search (back-checking, intelligent backtracking, ..)
- Interleaving search and consistency filtering (look-ahead, forward checking, ..)
- Importance of modeling, ordering heuristics

Outline

- Introduction
- Minimax algorithm
- Alpha-beta pruning

Board game playing: an appealing target of AI research

Board game: Chess (since early AI), 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.
- More recently: Soccer (Robocup `www.robocup.org/02.html`)

Characteristics

- Environment: accessible
- '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
- 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

Discussion

- What are the theoretically best moves?
- Techniques for choosing a good move when time is tight
  - Pruning: ignore irrelevant portions of the search space
  - 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
- Set of operators: defining legal moves a player can take
- 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

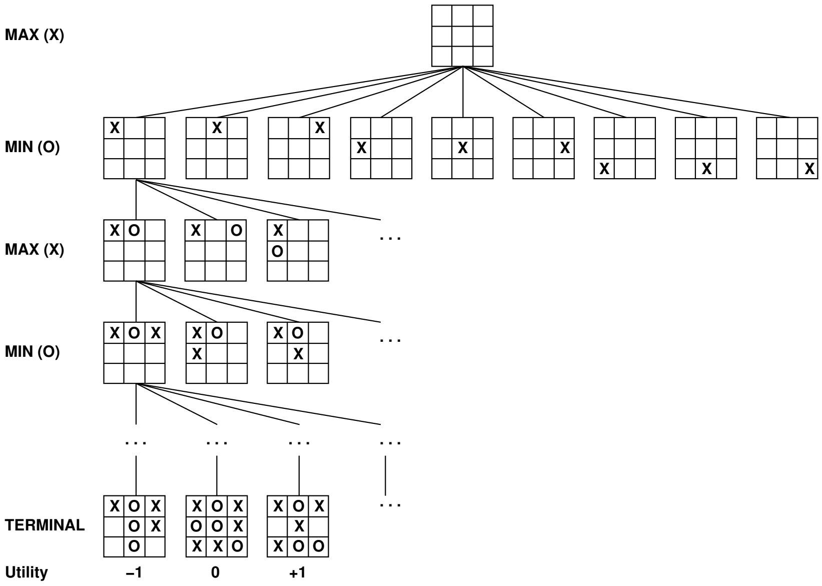
- Min actions are significant
- Max must find a strategy to win regardless of what Min does:
  $\longrightarrow$ correct action for Max for each action of Min

  Strategy may exist, but not the time for computing optimal/rational strategy

Example: Tic-Tac-Toe

Max has 9 alternative moves

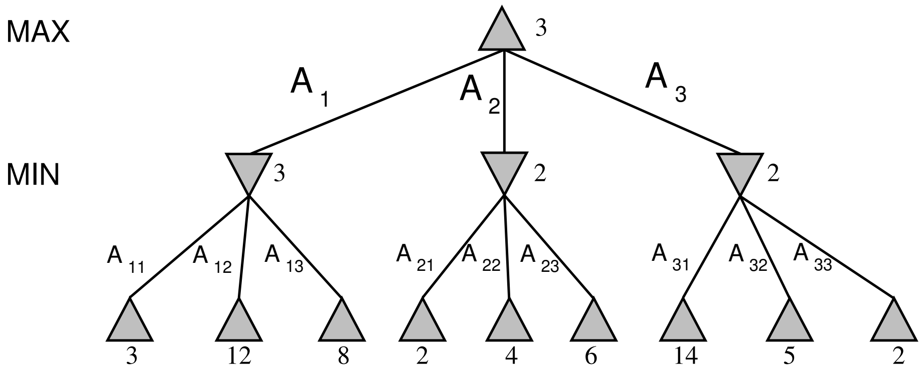
Terminal states' utility: Max wins=1, Max looses = -1, Draw = 0



Example: 2-ply game tree

Max's actions: A_1, A_2, A_3

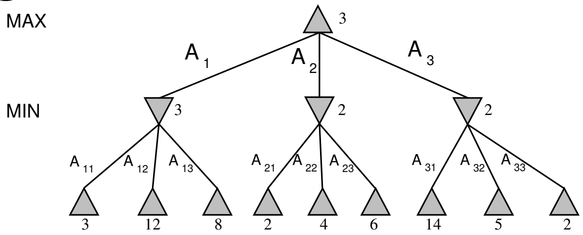
Min's actions: A_{11}, A_{12}, A_{13}



Minimax algorithm determines the optimal strategy for Max

$\rightarrow$ 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

MAX's decision: minimax decision maximizes utility under the assumption that the opponent will play perfectly to his/her own advantage

Minimax algorithm: Properties

m maximum depth
b legal moves

Using Depth-first search, space requirement is $O(bm)$

Time complexity $O(b^m)$

Real games: time cost totally unacceptable

Alpha-beta pruning (I)

Minimax requires computing all terminal nodes: unacceptable

Do we really need to do compute utility of all terminal nodes?

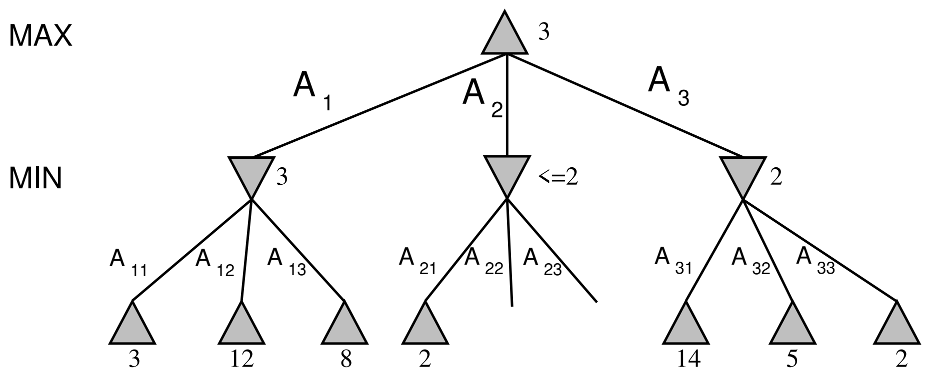
... No, says John McCarthy in 1956:

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)

Alpha-beta pruning (II) Example

BFSearch: A_1, A_{11}, A_{12}, A_{13},
A_2, A_{21}, ... stop!

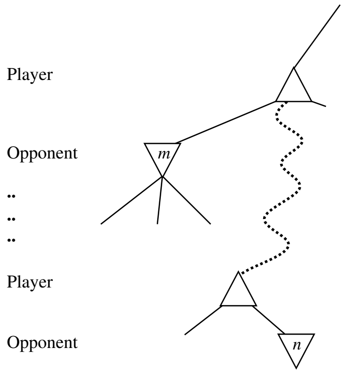


A_3, A_{31}, A_{32}, A_{33} stop! too late :—(

Alpha-beta pruning (III) General principal

If Player has a better choice m at $\begin{cases} \text{— a parent node of } n \\ \text{— any choice point further up} \end{cases}$

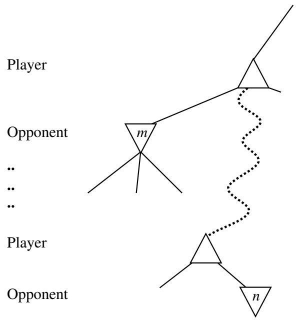
n will never be reached in <u>actual play</u>



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)

Alpha-beta pruning (IV) Mechanism

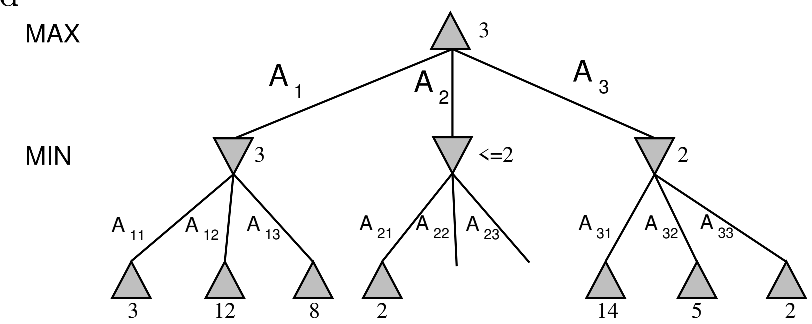
α: value of best choice so far for MAX, (maximum)
β: value of best choice so far for MIN, (minimum)



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

Alpha-beta pruning (V) Effectiveness of pruning

Effectiveness of pruning depends on the order of new nodes examined



Example: A_3, A_{33}, A_{31}, A_{32} instead of A_3, A_{31}, A_{32}, A_{33}

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