Chapter 5 Adversarial Search
5.1 – 5.4 Deterministic games

CS4811 - Artificial Intelligence

Nilufer Onder
Department of Computer Science
Michigan Technological University

Outline

Two-person games

Perfect play

- Minimax decisions
- $\alpha - \beta$ pruning

Resource limits and approximate evaluation

(Games of chance)

(Games of imperfect information)

Two-person games

- Games have always been an important application area for heuristic algorithms.
- The games that we will look at in this course will be two-person board games such as Tic-tac-toe, Chess, or Go.
- We assume that the opponent is “unpredictable” but will try to maximize the chances of winning.
- In most cases, the search tree cannot be fully explored. There must be a way to approximate a subtree that was not generated.

Two-person games (cont'd)

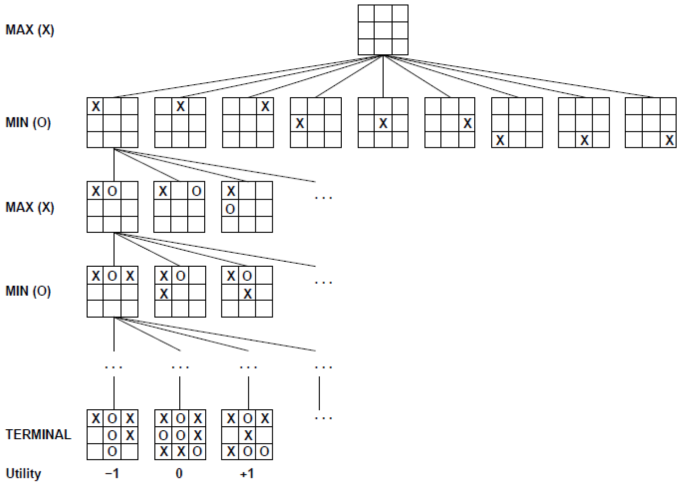
Several programs that compete with the best human players:

- Checkers: beat the human world champion
- Chess: beat the human world champion
- Backgammon: at the level of the top handful of humans
- Othello: good programs
- Hex: good programs
- Go: no competitive programs until 2008

Types of games

	Deterministic	Chance
Perfect information	Chess, checkers, go, othello ,	Backgammon monopoly
Imperfect information	Battleships, Minesweeper	Bridge, poker, scrabble “video games”

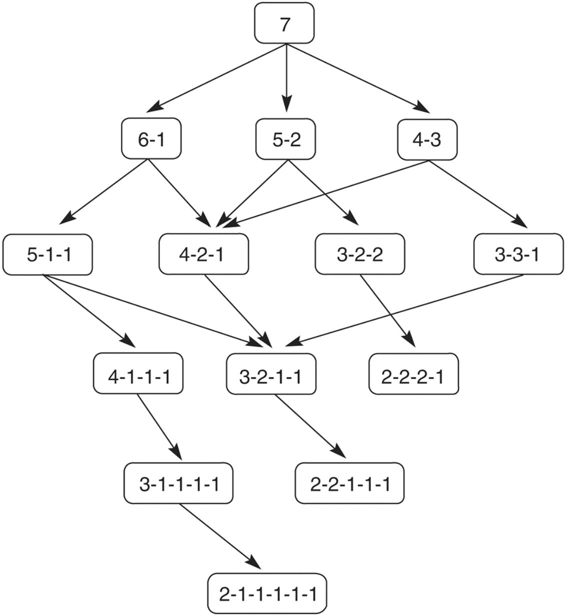
Game tree for tic-tac-toe (2-player, deterministic, turns)



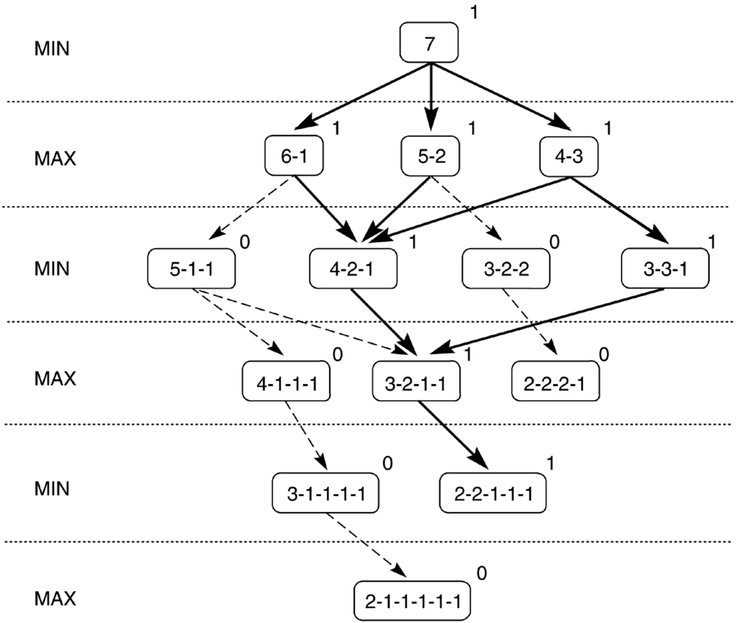
A variant of the game Nim

- A number of tokens are placed on a table between the two opponents.
- A move consists of dividing a pile of tokens into two nonempty piles of different sizes.
- For example, 6 tokens can be divided into piles of 5 and 1 or 4 and 2, but not 3 and 3.
- The first player who can no longer make a move loses the game.

The state space for Nim



Exhaustive Minimax for Nim



Search techniques for 2-person games

- The search tree is slightly different: It is a *two-ply tree* where levels alternate between players
- Canonically, the first level is “us” or the player whom we want to win.
- Each final position is assigned a payoff:
  - win (say, 1)
  - lose (say, -1)
  - draw (say, 0)
- We would like to maximize the payoff for the first player, hence the names MAX and MIN.

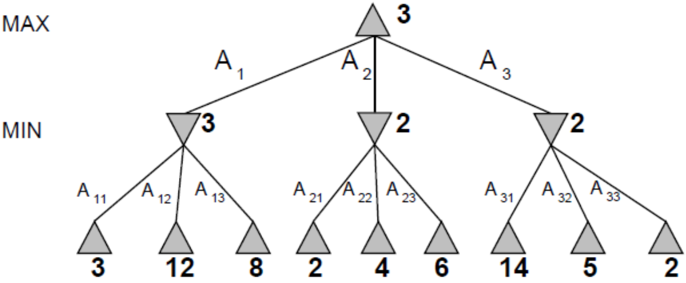
The search algorithm

- The algorithm called the *Minimax algorithm* was invented by Von Neumann and Morgenstern in 1944, as part of game theory.
- The root of the tree is the current board position, it is MAXs turn to play.
- MAX generates the tree as much as it can, and picks the best move assuming that MIN will also choose the moves for herself.

The Minimax algorithm

- Perfect play for deterministic, perfect information games.
- Idea: choose to move to the position with the highest mimimax value.
  Best achievable payoff against best play.

Minimax example



Minimax algorithm pseudocode

function MINIMAX-DECISION (*state*)
returns an action
    return $\text{argmax}_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

function MAX-VALUE (*state*)
returns a utility value
    if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
    $v \leftarrow -\infty$
    for each *a* **in** ACTIONS(*state*) **do**
        $v \leftarrow$ MAX(v,MIN-VALUE(RESULT(*state*, *a*)))
    return v

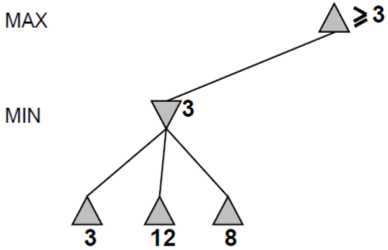
function MIN-VALUE (*state*)
returns a utility value
    if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
    $v \leftarrow \infty$
    for each *a* **in** ACTIONS(*state*) **do**
        $v \leftarrow$ MIN(v,MAX-VALUE(RESULT(*state*, *a*)))
    return v

Properties of minimax

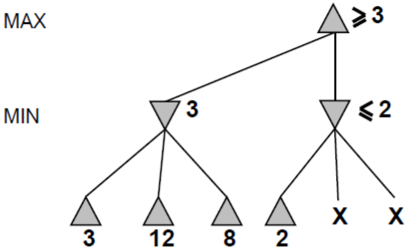
- *Complete:* Yes (if the tree is finite)
  chess has specific rules for this
- *Time:* $O(b^m)$
- *Space:* $O(bm)$ with depth-first exploration
- *Optimal:* Yes, against an optimal opponent. Otherwise ??

For chess, $b \approx 35, m \approx 100$ for "reasonable games. The same problem with other search trees: the tree grows very quickly, exhaustive search is usually impossible.
But do we need to explore every path?
Solution: Use $\alpha - \beta$ pruning

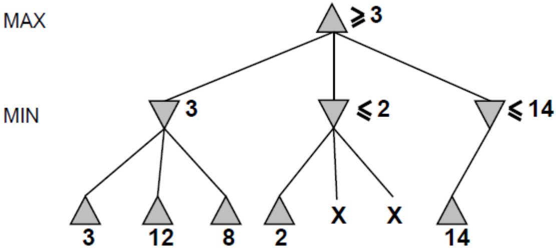
$\alpha - \beta$ pruning example



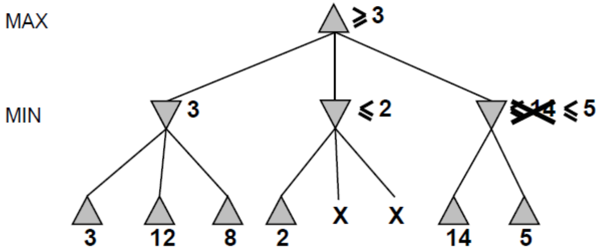
$\alpha - \beta$ pruning example



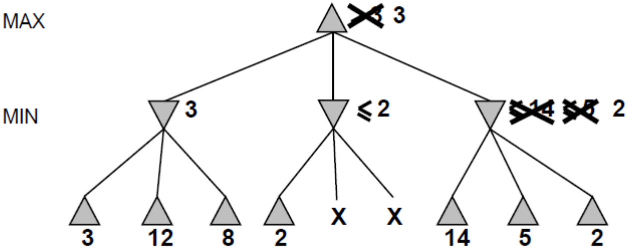
$\alpha - \beta$ pruning example



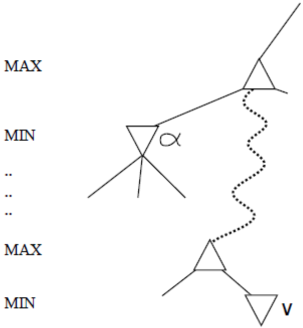
$\alpha - \beta$ pruning example



$\alpha - \beta$ pruning example



Why is it called $\alpha - \beta$?



α is the best value to MAX found so far off the current path.
If V is worse than α then MAX will avoid by by pruning that branch.
Define β similarly for MIN.

The $\alpha - \beta$ algorithm

function ALPHA-BETA SEARCH (*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE (*state*, $-\infty, \infty$)
  return the *action* **in** ACTIONS(*state*) with value v

function MAX-VALUE (*state*, α, β) **returns** a utility value
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v,MIN-VALUE (RESULT(*state*, a),α, β)
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

function MIN-VALUE (*state*) **returns** a utility value
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v,MAX-VALUE (RESULT(*state*, a),α, β)
    if $v \leq \alpha$ **then return** v
    $\alpha \leftarrow$ MIN(α, v)
  return v

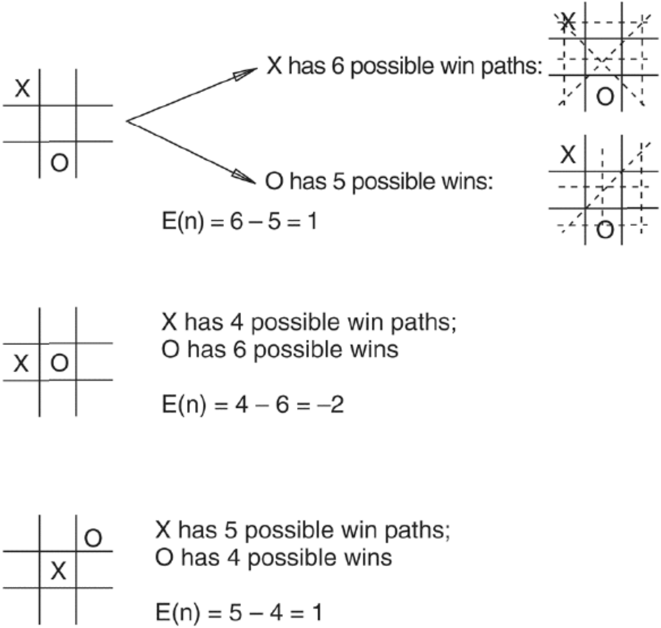
Properties of $\alpha - \beta$

- A simple example of the value of reasoning about which computations are relevant (a form of *metareasoning*)
- Pruning does not affect the final result
- Good move ordering improves the effectiveness of pruning
- With "perfect ordering," time complexity $= O(b^{m/2})$ doubles solvable depth
- Unfortunately, 35^{50} is still impossible!

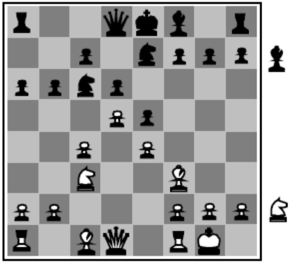
Resource limits

- The Minimax algorithm assumes that the full tree is not prohibitively big
- It also assumes that the final positions are easily identifiable.
- Use a two-tiered approach to address the first issue
  - Use CUTOFF-TEST instead of TERMINAL-TEST
    e.g., depth limit
  - Use EVAL instead of UTILITY
    i.e., evaluation function that estimates desirability of position

Evaluation function for tic-tac-toe

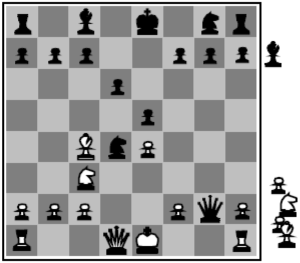


Evaluation function for chess



Black to move

White slightly better



White to move

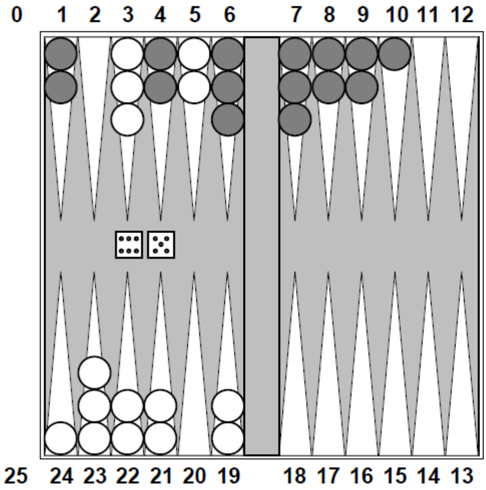
Black winning

For chess, typically linear weighted sum of features:
$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \sum_{i=1}^{n} w_n f_n(s)$
e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) - (number of black queens)

Deterministic games in practice

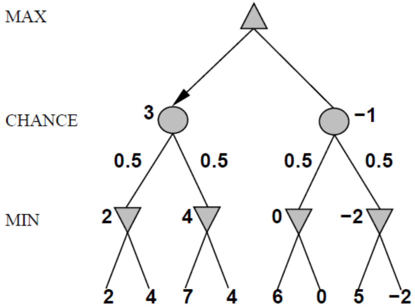
- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.
- Chess: Deep Blue defeated human world champion Gary Kasparov in a six- game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
- Othello: human champions refuse to compete against computers. Computers are too good.
- Go: human champions refuse to compete against computers. Computers are too bad.
  In Go, $b > 300$. Most programs used pattern knowledge bases to suggest plausible moves. Recent programs used Monte Carlo techniques.

Nondeterministic games: backgammon



Nondeterministic games in general

Chance is introduced by dice, card shuffling.



Algorithms for nondeterministic games

- EXPECTIMINIMAX gives perfect play.
- As depth increases, probability of reaching a given node shrinks, the value of lookahead is diminished.
- $\alpha - \beta$ is less effective.
- TDGAMMON uses depth 2 search and a very good evalution function. It is at the world-champion level.

Games of imperfect information

- E.g., card games where the opponent's cards are not known.
- Typically, we can calculate a probability for each possible deal.
- Idea: Compute the minimax value for each action in each deal, then choose the action with highest expected value over all deals.
- However, the intuition that the value of an action is the average of its values in all actual states is not correct.

Summary

- Games are fun to work on!
- They illustrate several important points about AI
  - perfection is unattainable, must approximate
  - good idea to think about what to think about
  - uncertainty constrains the assignment of values to states
  - optimal decisions depend on information state, not real state
- Games are to AI as grand prix racing is to automobile design

Sources for the slides

- AIMA textbook (3^{rd} edition)
- AIMA slides (http://aima.cs.berkeley.edu/)
- Luger's AI book (5^{th} edition)
- Tim Huang's slides for the game of Go
- Othello web sites
  www.mathewdoucette.com/artificialintelligence
  home.kkto.org:9673/courses/ai-xhtml
- Hex web sites
  hex.retes.hu/six
  home.earthlink.net$\tilde{v}$anshel
  cs.ualberta.ca/$\tilde{j}$avhar/hex
  www.playsite.com/t/games/board/hex/rules.html