Games and Search

Outline

- **Computer programs which play 2-player games**
  - game-playing as search
  - with the complication of an opponent

- **General principles of game-playing and search**
  - evaluation functions,
  - minimax principle
  - alpha-beta-pruning

- **Status of Game-Playing Systems**
  - in chess, checkers , etc, computers routinely defeat leading world players

Chess Rating Scale

Garry Kasparov (World Champion)

Deep Blue

Deep Thought

Ratings

3000
2800
2600
2400
2200
2000
1800
1600
1400
1200

1966 1971 1976 1981 1986 1991 1997

Game-Playing and AI

- **Game-playing is a good problem for AI research:**
  - all the information is available
    - i.e., human and computer have equal information
  - game-playing is non-trivial
    - need to display "human-like" intelligence
    - some games (such as chess) are very complex
    - requires decision-making within a time-limit
      - more realistic than other search problems
  - games are played in a controlled environment
    - can do experiments, repeat games, etc: good for evaluating research systems
  - can compare humans and computers directly
    - can evaluate percentage of wins/losses to quantify performance

Search and Game Playing

- **Consider a board game**
  - e.g., chess, checkers, tic-tac-toe
  - **configuration** of the board = unique arrangement of “pieces”
  - each possible configuration = state in search space

- **Statement of Game as a Search Problem**
  - **States** = board configurations
  - **Operators** = legal moves
  - **Initial State** = current configuration
  - **Terminal State (Goal**) = winning configuration

Game Tree Representation

S

Computer Moves

Opponent Moves

Computer Moves

G

Possible Goal State
lower in Tree (winning situation for computer)

Complexity of Game Playing

- **Imagine we could predict the opponent's moves given each computer move**

- **How complex would search be in this case?**
  - worst case, it will be $O(b^d)$
  - Chess:
    - b ~ 35 (average branching factor)
    - d ~ 100 (depth of game tree for typical game)
    - $b^d \sim 35^{100} \sim 10^{154}$ nodes!! ("only" about 10^{40} legal states)
  - Tic-Tac-Toe
    - ~5 legal moves, total of 9 moves
    - 5^9 = 1,953,125
    - 9! = 362,880 (Computer goes first)
    - 8! = 40,320 (Computer goes second)

- **well-known games can produce enormous search trees**

Utility Functions

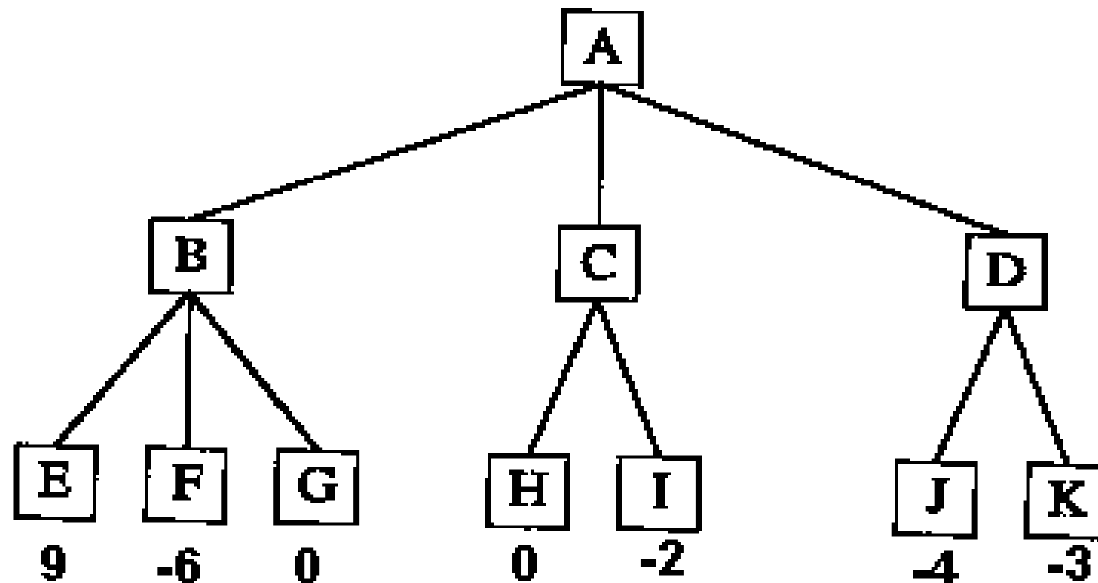
- **Utility Function:**
  - defined for each terminal state in a game
  - assigns a numeric value for each terminal state
  - these numbers represent how "valuable" the state is for the computer
    - positive for winning
    - negative for losing
    - zero for a draw
  - Typical values from -infinity (lost) to +infinity (won) or [-1, +1].

Greedy Search with Utilities

- **A greedy search strategy using utility functions**
  - Expand the search tree as far as the terminal states on each branch
  - Generate utility functions for each board configuration
  - Make the initial move that results in the board configuration with the maximum value
  - but this ignores what the opponent might do!
    - i.e. opponent's moves are interleaved with the computer's

Minimax Search

- Explore the tree as far as the terminal states
- Calculate utilities for the resulting board configurations
- The computer will make the move such that when the opponent makes his best move, the board configuration will be in the best position for the computer



Propagating Minimax Values up the Game Tree

- **Starting from the leaves**
  - Assign a value to the parent node as follows
    - Children are Opponent's moves: Minimum of all immediate children
    - Children are Computer's moves: Maximum of all immediate children

Computer Makes move that results in State C

A -2

B -6
C -2
D -4

Computers Possible Moves

E F G H I J K

9 -6 0 0 -2 -4 -3

Opponents Possible Moves

Board Evaluation from computers "point of view"

Deeper Game Trees

Minimax can be generalized to deeper game trees (than just 2 levels): "propagate" utility values upwards in the tree

A

3 B
4 C
Computer's Moves

D 3
E 5
5 F
8 G
H 4
Opponents

0 I
J 5
M 7
N 8
Computer's

K 0
L 7
Opponents

General Minimax Algorithm on a Game Tree

For each move by the computer

1. perform depth-first search as far as the terminal state
2. assign utilities at each terminal state
3. propagate upwards the minimax choices
    - if the parent is a minimizer (opponent)
        - propagate up the minimum value of the children
    - if the parent is a maximizer (computer)
        - propagate up the maximum value of the children
4. choose the move (the child of the current node) corresponding
    to the maximum of the minimax values if the children

Complexity of Minimax Algorithm

- Assume that all terminal states are at depth d
- **Space complexity**
  - depth-first search, so O(bd)
- **Time complexity**
  - branching factor b, thus, $O(b^d)$

Minimax Principle

- **"Assume the worst"**
  - say we are two plays (in the tree) away from the terminal states
  - high numbers favor the computer
    - so we want to choose moves which maximize utility
  - low numbers favor the opponent
    - so they will choose moves which minimize utility

- **Minimax Principle**
  - you (the computer) *assume that the opponent will choose the minimizing move next* (after your move)
  - so you now *choose the best move under this assumption*
    - i.e., the maximum (highest-value) option considering **both** your move and the opponent's optimal move.

  - we can generalize this argument more than 2 moves ahead: we can search ahead as far as we can afford.

Minimax with Evaluation Functions

For each move by the computer

1. perform depth-first search with depth-limit m
2. assign evaluation functions at each state at depth m
3. propagate upwards the minimax choices
   - if the parent is a minimizer (opponent)
     - propagate up the minimum value of the children
   - if the parent is a maximizer (computer)
     - propagate up the maximum value of the children
4. choose the move (the child of the current node) corresponding to the maximum of the minimax values if the children

- **The same as general minimax, except**
  - only goes to depth m
  - uses evaluation functions (estimates of utility)
- **How would this algorithm perform at chess?**
  - could look ahead about 4 “ply” (pairs of moves)
  - would be consistently beaten even by average chess players!

Min Max : The Alpha Beta Principle

1

1 0 -8

3 4 1 2 0 -8

X X X

Computer's

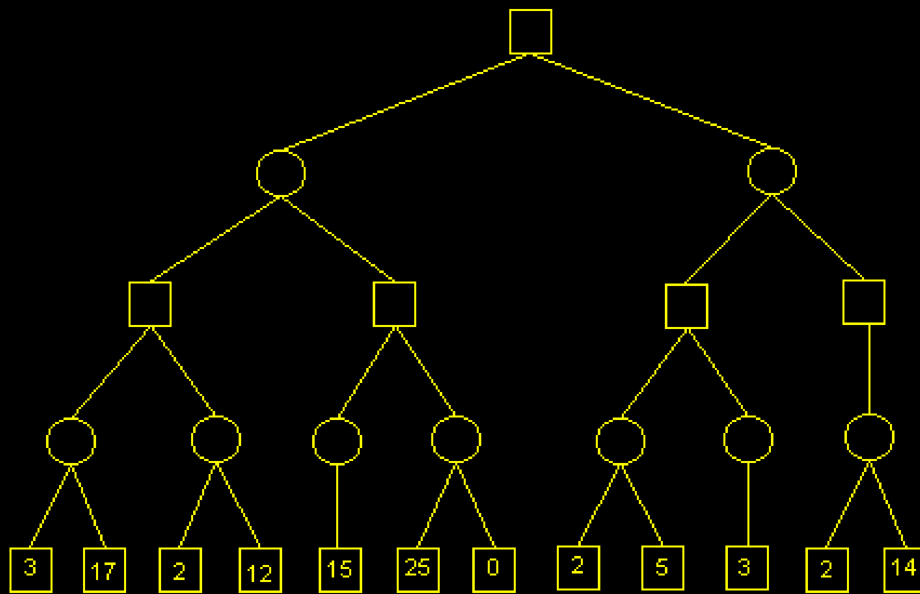
Opponent’s Moves
(min of evaluations)

(X indicates ‘not evaluated’)

Alpha Beta Procedure

- **Depth first search of game tree, keeping track of:**
  - Alpha: Highest value seen so far on maximizing level
  - Beta: Lowest value seen so far on minimizing level

- **Pruning**
  - When Maximizing,
    - do not expand any more sibling nodes once a node has been seen whose evaluation is smaller than Alpha
  - When Minimizing,
    - do not expand any sibling nodes once a node has been seen whose evaluation is greater than Beta



$\beta = \infty$
$\alpha = 3$

$\beta = 3$
$\alpha = -\infty$
3

$\beta = 3$
$\alpha = 3$

$\beta = \infty$
$\alpha = 3$
3

$\beta = 3$
$\alpha = 15$
15

$\beta = \infty$
$\alpha = 3$
3

$\beta = 3$
$\alpha = -\infty$

$\beta = 3$
$\alpha = -\infty$
3

$\beta = 2$
$\alpha = 3$
2

15

$\beta = 2$
$\alpha = 3$
2

$\beta = 3$
$\alpha = 3$
3

3
17
2
15
2
3

Effectiveness of Alpha-Beta Search

- **Worst-Case**
  - branches are ordered so that no pruning takes place
  - alpha-beta gives no improvement over exhaustive search

- **Best-Case**
  - each player's best move is the left-most alternative (i.e., evaluated first)
  - in practice, performance is closer to best rather than worst-case

- **In practice often get $O(b^{(d/2)})$ rather than $O(b^d)$**
  - this is the same as having a branching factor of sqrt(b),
    - since $(\text{sqrt}(b))^d = b^{(d/2)}$
    - i.e., we have effectively gone from b to square root of b
  - e.g., in chess go from b ~ 35 to b ~ 6
    - this permits much deeper search in the same amount of time
    - makes computer chess competitive with humans!

Computers can play GrandMaster Chess

- **“Deep Blue” (IBM)**
  - parallel processor, 32 nodes
  - each node has 8 dedicated VLSI “chess chips”
  - each chip can search 200 million configurations/second
  - uses minimax, alpha-beta, heuristics: can search to depth 14
  - memorizes starts, end-games
  - power based on speed and memory: no common sense

- **Kasparov v. Deep Blue, May 1997**
  - 6 game full-regulation chess match (sponsored by ACM)
  - Kasparov lost the match (2.5 to 3.5)
  - a historic achievement for computer chess: the first time a computer is the best chess-player on the planet

- Note that Deep Blue plays by “brute-force”: there is relatively little which is similar to human intuition and cleverness

Status of Computers in Other Games

- **Checkers/Draughts**
  - current world champion is Chinook, can beat any human
  - uses alpha-beta search
- **Othello**
  - computers can easily beat the world experts
- **Backgammon**
  - system which learns is ranked in the top 3 in the world
  - uses neural networks to learn from playing many many games against itself
- **Go**
  - branching factor b ~ 360: very large!
  - $2 million prize for any system which can beat a world expert

Summary

- Game playing is best modeled as a search problem
- Game trees represent alternate computer/opponent moves
- Evaluation functions estimate the quality of a given board configuration for each player
- Minimax is a procedure which chooses moves by assuming that the opponent will always choose the move which is best for them
- Alpha-Beta is a procedure which can prune large parts of the search tree and allow search to go deeper
- For many well-known games, computer algorithms based on heuristic search match or out-perform human world experts.