Trees and Adversarial Search
—
Winston, Chapter 6

Michael Eisenberg and Gerhard Fischer
TA: Ann Eisenberg

AI Course, Fall 1997

Why Study Games in AI?

- problems are formalized

- real world knowledge (common sense knowledge) is not too important

- rules are fixed

- adversary modeling is of general importance (e.g., in economic situations, in military operations, ...)
  - opponent introduces uncertainty
  - programs must deal with the contingency problem

- complexity of games??
  - number of nodes in a search tree (e.g., 10^{40} legal positions in chess)
  - specification is simple (no missing information, well-defined problem

Game Playing - Overview

- games as search problems
- perfect decisions in two person games
- imperfect decisions
- alpha-beta pruning
- games with a chance
- state-of-the-art game programs

Examples

- Nim
- Tic-Tac-Toe
- Checkers (Arthur Samuel — classic papers on tree-pruning heuristics, adaptive parameter improvement)
- Othello/Reversi (see program in Norvig: “AI Programming” and Boecker, Eden, Fischer: “Interactive Problem Solving Using LOGO”)
- Chess (programs play at Grandmaster level; Deep Blue beat Kasparov)
- Go
- Backgammon (program has beaten the world champion, but was lucky)
- Blackjack (strategies for playing in casinos — Thorp “How To Beat the Dealer”)

Example: Nim

rules:

- 2 person game
- players alternate
- one move: take 1, 2 or 3 objects
- player who takes the last object will loose

27

26 25 24

25 24 23 24 23 22 23
22 21

Example: Modified Version of Nim

o

o o

o o o

o o o o

o o o o o

Formal Definition of Games as Search Problems

- initial state: board position + whose move
- a set of operators defining the legal moves of a player
- terminal test determining when the game is over
- utility function giving a numeric value for the outcome of a game (chess: +1, 0, and -1; backgammon: +192 to -192

Game Tree

- A game tree is a representation that is a semantic tree
  In which nodes denote board configurations

  Branches denote moves

- With writers that
  Establish that a node is for the maximizer or for the minimizer

  Connect a board configuration with a board-configuration description

- With readers that
  Determine whether the node is for the maximizer or minimizer

  Produce a board configuration's description

Search Procedures

- **MINI-MAX** ---> static evaluation; conclusions about what to do at the deeper nodes of the search tree percolate up to determine what should happen at higher nodes

- **ALPHA-BETA**
  - there is no need to explore disastrous moves any further
  - can be augmented by a number of heuristic pruning procedures (danger: optimal moves may not be selected)

- **general trade-off:**
  - look-ahead operations
  - pattern-directed play

Minimax Algorithm

To perform a minimax search using MINIMAX,

- If the limit of search has been reached, compute the static value of the current position relative to the appropriate player. Report the result.

- Otherwise, if the level is a minimizing level, use MINIMAX on the children of the current position. Report the minimum of the results.

- Otherwise, the level is a maximizing level. Use MINIMAX on the children of the current position. Report the maximum of the results.

Heuristic Evaluation Functions

- allow us to approximate the true utility of a state without doing a complete search

- **changes:**
  - utility function is replaced by an heuristic evaluation function
  - terminal test is replaced by a cutoff test

- example for **Tic-Tac-Toe** (and Number Scrabble): static value associated with each field:
  - center: 4
  - corners: 3
  - middle field of a row: 2

- **chess:**
  - material value: pawn=1 — knight or bishop=3 — rook=5 — queen=9
  - other features: good pawn structure, king safety, mobility, ...

Alpha-Beta Pruning

- **basic idea:** it is possible to compute the correct minimax decision without looking at every node in the search tree ---> **pruning** (allows us to ignore portions of the search tree that make no difference to the final choice)

- **general principle:**
  - consider a node n somewhere in the tree, such that a player has a chance to move to this node
  - if player has a better chance m either at the parent node of n (or at any choice point further up) then n will never be reached in actual play

- **effectiveness:**
  - depends on the ordering in which the successors are examined
  - try to examine first the successors that are likely to be best

ALPHA-BETA Procedure

To perform minimax search with the ALPHA-BETA procedure:

- If the level is the top level, let alpha be -∞ and let beta be ∞.
- If the limit of search has been reached, compute the static value of the current position relative to the appropriate player. Report the result.
- If the level is a minimizing level,
  - Until all children are examined with ALPHA-BETA or until alpha is equal to or greater than beta,
    - Use the ALPHA-BETA procedure, with the current alpha and beta values, on a child; note the value reported.
    - Compare the value reported with the beta value; if the reported value is smaller, reset beta to the new value.
  - Report beta.
- Otherwise, the level is a maximizing level:
  - Until all children are examined with ALPHA-BETA or alpha is equal to or greater than beta,
    - Use the ALPHA-BETA procedure, with the current alpha and beta value, on a child; note the value reported.
    - Compare the value reported with the alpha value; if the reported value is larger, reset alpha to the new value.
  - Report alpha.

Game Playing: Case Study Othello — Questions to Think about

- how would you write a game playing program for Othello?

- what kind of evaluation function would you use or would you not use?

- what is the most difficult aspect of playing the game well?

- if you are an experienced Othello player, articulate some of your Othello knowledge

Rules

- each player takes 32 discs and chooses one color (64 discs are available to play)
- move: "outflanking" your opponent ---> then flipping the outflanked discs to your color
- definition of "outflank": ..... *establishing a domain vocabulary*
- black moves first ---> then take turns if legal moves are available
- if a move is available ---> one **must** take it
- outflanking occurs in all directions: horizontally, vertically, diagonally
- **all** discs outflanked in any one move must be flipped (even if it is to the player's disadvantage)
- end of game: when it is no longer possible for either player to move (either because all squares are filled or no legal move is available)

Incremental Development of Game Playing Programs

- let humans play against each other
  - the program serves as a representational media
  - the program checks for legal moves

- humans against program
  - legal moves by the program
  - good moves by the program

- humans against program — the program being in the role of a coach

- program against program

- learning component (program improve its play by playing games)

Humans against Program — Incremental Additions to the "Smartness" of the Program

- play randomly (but legal; may involve a non-trivial amount of knowledge / computation)

- have a static value associated with each square on the board

- have a dynamically value associated with each square on the board

- have an evaluation function taking other factors into account (e.g., number of pieces)

- search / look-ahead / exploring alternatives (using the evaluation function):
    - look one move ahead
    - look several moves ahead using minimax, alpha-beta,

Strategy

- goal is clear -- but how can we achieve the goal?
- corners are special: they can **never** be outflanked ---> question: how do we get one of our pieces into the corner (backward reasoning)
- squares next to corners are not good
- border squares are desirable (they can only be outflanked in only two directions)
- squares next to border squares are not desirable
- get **control** of the game: have many possible moves to choose from
- try to have as pieces of your color at any time in the game as possible

Rules themselves may be changed

- original set-up can vary:

b w
w b
or
b b
w w

- turn
  - **one** direction
  - **all** directions

- let the player decide

- an extended version of the program could handle all strategies

- in chess: many variations

Other Issues

- Othello as a computer game — claim: brute-force search based on a good evaluation function can yield excellent play
  - number of legal moves is small (in most situations)
  - humans have difficulties to "visualize" the long range consequences of a move
- knowledge elicitation / acquisition techniques: two humans play the game against each other and think-aloud
- thin spread of domain knowledge: claim: any amount of programming knowledge (e.g., in Lisp, C, ...) will not allow you to write a program which plays Othello well