Overview

- formal $\alpha - \beta$ pruning algorithm
- $\alpha - \beta$ pruning properties
- games with an element of chance
- state-of-the-art game playing with AI
- more complex games

- project #1: full description
  - core routines for 8-puzzle

$\alpha - \beta$ Pruning Algorithm: Max-Value

function Max-Value (state, game, $\alpha$, $\beta$) return minmax(state)
$\alpha$: best MAX on path to state ; $\beta$: best MIN on path to state
if Cutoff(state) then return Eval(state)
for each $s$ in Successor(state) do
  • $\alpha \leftarrow$ Max($\alpha$, Min-Value($s$,game,$\alpha$, $\beta$))
  • if $\alpha \geq \beta$ then return $\beta$ /* CUT!! */
end
return $\alpha$

$\alpha - \beta$ Pruning Algorithm: Min-Value

function Min-Value (state, game, $\alpha$, $\beta$) return minmax(state)
$\alpha$: best MAX on path to state ; $\beta$: best MIN on path to state
if Cutoff(state) then return Eval(state)
for each $s$ in Successor(state) do
  • $\beta \leftarrow$ Min($\beta$, Max-Value($s$,game,$\alpha$, $\beta$))
  • if $\beta \leq \alpha$ then return $\alpha$ /* CUT!! */
end
return $\beta$
Ordering is Important for Good Pruning

- For MIN, sorting successor's utility in an **increasing** order is better (shown above; left).
- For MAX, sorting in **decreasing** order is better.

Games With an Element of Chance

Rolling the dice, shuffling the deck of card and drawing, etc.

- **chance nodes** need to be included in the minimax tree
- try to make a move that maximizes the **expected value** → **expectimax**
- expected value of random variable $X$:
  \[
  E(X) = \sum_x xP(x)
  \]
- **expectimax**
  \[
  \text{expectimax}(C) = \sum_i P(d_i) \max_{s \in S(C,d_i)}(\text{utility}(s))
  \]

Design Considerations for Probabilistic Games

- the **value** of evaluation function, not just the **scale** matters now! (think of what expected value is)
- time complexity: $b^m n^m$, where $n$ is the number of distinct dice rolls
- pruning can be done if we are careful

Game Tree With Chance Element

- chance element forms a new ply (e.g. dice, shown above)
State of the Art in Gaming With AI

- Backgammon: Tesouro’s Neural Network → top three (1992)
- Othello: smaller search space → superhuman performance
- Checkers: Samuel’s Checker Program running on 10Kbyte (1952)

Genetic algorithms can perform very well on select domains.

Hard Games

The game of Go, popular in East Asia:

- \(19 \times 19 = 361\) grid: branching factor is huge!
- search methods inevitably fail: need more structured rules
- the bet is high: $2,000,000 prize

Project 1: 8-Puzzle with Search

- Input: a board configuration
  ‘(1 3 4 8 6 2 7 0 5)
- Output: sequence of moves
  ‘(UP RIGHT UP LEFT DOWN)
- Search methods to be used:
  Depth-First, Iterative Deepening Search, Breadth-First, Heuristic search, A*, IDA*.
- Use \(h_1\) (tiles out-of-place), and \(h_2\) (sum of manhattan distance) for the last three.
- This is an individual project.

This project is heavily inspired by:

Project 1: Required Material

Use the exact filename as shown below (in bold).

- Program code (eight.lsp): put it in a single text file.
  – Ample indentation and documentation is required.
- Documentation (README): user manual
- Inputs and outputs (include in README)
  – Easy: ‘(1 3 4 8 6 2 7 0 5)
  – Medium: ‘(2 8 1 0 4 3 7 6 5)
  – Hard: ‘(5 6 7 4 0 8 3 2 1)
  – Include 5 examples of your own
For each run, report the time taken, and the number of nodes expanded.

- Compare the performance of various search methods using the Easy, Medium, and Hard case examples.
- Compare and contrast different search methods based on the results.
- Some search methods may fail to produce an answer. Analyse why it failed and report your findings.

**Project 1 Tips (1)**

Timing execution: use \( \text{(time (your-function-to-run))} \) to get current time.

\( \text{>(time (car '(x x)))} \)

real time : 0.000 secs
run time : 0.000 secs
X

\( \text{>} \)

**Project 1 Tips (2)**

Checking for duplicate states

\( \text{(defun dupe (state node-list)} \)
\( \text{  (dolist (node node-list nil)} \)
\( \text{     (if (equal state (first node)} \)
\( \text{         (return-from dupe T)))}) \)

A general expand function:

\( \text{(defvar *expand-func*)} ; \text{name of expand function} \)
\( \text{(defun expand (node)} \)
\( \text{  (funcall *expand-func* node))} \)

**Project 1: State Representation**

A node in the search tree has the following structure:

\( \text{\'(1 3 4 8 6 2 7 0 5) ; blank is stored as 0} \)
\( \text{  ; heuristic function value} \)
\( \text{  depth} \)
\( \text{  ; depth from the root} \)
\( \text{  path}) \)
\( \text{  ; list of moves from} \)
\( \text{  ; the start} \)
### Project 1: Utility Routines

Source will be available on the course web page:

- `(apply-op <operator> <node>)`: return new node after applying operator on current node
- `(print-tile <state>)`: prints out the board
- `(print-answer <state> <path>)`: prints boards after each move in the path, starting from the state.

### Project 1: Submission

- Turnin using the `turnin` command on CS unix machines. The folder is 625–600. Send either separate files or a single `tar` file.
- Submission deadline is 10/14/02 before class (12:39am).
- No late submissions will be allowed.
- If more than half have problems meeting the deadline, I will consider an extension.
- Only send **plain ASCII** text files. **Do not send MS-Word documents or other formatted text.**

### Key Points

- formal $\alpha - \beta$ pruning algorithm: know how to apply pruning
- $\alpha - \beta$ pruning properties: evaluation
- games with an element of chance: what are the added elements? how does the minmax tree get augmented?

### Next Time: Logic

- Propositional Logic: Chapter 6, 6.3–6.6