Lecture 6: Transposition table, hashing and GHI

Misc notes

- Milestone 3 details posted
- Server code posted
  - Doesn't check for end-of-game
- HW marks in Milestone folder
  - run “svn update”
- Preferred way to test on server?

Course Presentations

- Chinook (Checkers)
- TD-Gammon (Backgammon)
- Logistello (Othello) - Chris
- GiB (Bridge)
- Maven (Scrabble) - Will
- Fuego/others (Go) - Jason

- Will post detailed instructions soon

Negamax

- Instead of implementing two functions (max/min) implement just one!
  - Works because a game is zero-sum
- Instead of debugging two functions, only debug one
  - A bit harder to get right the first time
  - A bit harder to debug
- Another option for upcoming milestones
**Negamax**

Negamax\( (depth, scoreWeight) \)

if \( depth == 0 \) || Done()
    return scoreWeight·CutoffEval();
currVal ← ∞
for each successor s in 1… # successors
    ApplyMove(s)
    currVal = max(currVal, -Negamax\( depth-1, -scoreWeight \))
    UndoMove(s)
return currVal

**Using alpha-beta bounds**

- When performing alpha-beta pruning, don’t have to start with alpha = -∞, beta = ∞
- If you have evidence that the value is in a smaller range, can search with a tighter bound
  - Produces smaller tree
  - But, if the value returned is at the bounds of your range you have to re-search

**Graphs versus Trees**

- A graph is not a tree
  - Turning a graph into a tree results in exponential blow-up
- Transposition (hash) Table
  - Key: Minimax uses (almost) no memory
    - Use some memory to detect tree
  - When two paths transpose into the same state

**Hop Step example**

- When is the first transposition in Hop Step?
- Assume you are searching 8 ply (HW)
  - 3-ply into the game
  - Save 5-ply \( (8^5 \approx 33k \text{ states}) \)
Transpositions

• When should we look for transpositions?
  • Near the top of the tree
    • Large savings
    • Likely to find transpositions

• When shouldn’t we look for transpositions?
  • Near the bottom of the tree
    • Minimal savings
    • Unlikely to find transpositions

Zobrist hashing

• Quick incremental hashing
  • Add to apply/undo functions
  • Hash function is always computed

Transpositions

• What is needed to test for transpositions?
  • Naïve - list of states, and a linear search
  • Better - tree of states log(s) search
  • Best - hash table

• What hash function should we use?

Zobrist hashing

• For each state element, compute a (64-bit) random number (can be pre-computed)
• Calculate the initial heuristic as the XOR of all the initial state elements
• When a move is applied:
  • XOR “out” the state elements that changed
  • XOR “in” the new state elements
• XOR has the nice property that XOR’ing the same value twice does not change the original value
Hop Step Zobrist Hashing

- 121 positions on board
  - 2 states each (blocked/unblocked)
  - 2 player positions each
  - 2 sides each (1 or 2 steps)
  - 968 random numbers
- Can we do better?
  - Could hash next hop distance independently
  - Can we reach the same position with different hop counts?

Hash table size

- Do we need to store the full state?
  - Expensive
  - Hope that the hash is enough
- If we have m balls in n bins, when $m = \sqrt{n}$ there is 50% chance that collisions occur
  - 32-bit hashing – 65535 elements (too small)
  - 64-bit hashing – 4 billion elements (good)

Transposition tables

- What is stored in the table?
  - For minimax?
    - Just the value of the state
  - For alpha-beta pruning?
    - Need the bounds
    - May reach the same state with different bounds
  - For some games
    - Need/want the depth

Graph History Interaction

- Some games have repetition constraints
  - Chess, Checkers, Go
- State depends on history of moves, which isn’t encoded in the state for efficiency purposes
- Transposition tables can cause issues
  - Examples and figures from this paper
First-Player Loss GHI Example

- Search A→B→E→H→E
  - A loss is stored in the table entry for H, because the position repetition cannot be avoided.
- Search A→B→D
  - A loss is stored for AND node B
- Expand A → C → F → H
  - A table look-up for H retrieves a loss which is backed up to F and C
- A is now incorrectly labeled as a loss because losses are stored for both successors B and C. However, A is a win by the sequence A→C→F→H→E→G.

Why do we need solutions?

- If we are going to solve games:
  - Checkers, Chess, Go
- How do we fix the problem?
  - It’s a bit complicated…
  - Zobrist hashing of move sequences
  - Move sequences stored and proofs verified

Repeating-Player Loss

- Search A→B→E→H
  - H is stored as a win because the opponent does not have a legal move at H.
- Search A → C → F → H
  - The win stored for H is backed up and a win is stored for C as well.
- A is now incorrectly labeled as a win since C’s table entry shows a win. However, A is a losing position, since the sequences A → B → D, A → C → F → H → E → G and A → C → F → H → E → H all lose.
Transposition tables

• Debugging:
  • Verify every transposition table lookup
  • Perform search and verify values stored!

• VERY important
  • Much more likely to be correct
  • Can also do with alpha-beta pruning