COMP 34111 — Artificial Intelligence and Games

Jonathan Shapiro

Department of Computer Science

Announcements

Reading for today: Chapter 3

Reading for this week: Chapter 4.

Next two weeks: Learning in Games

Announcements

- If there is a group of people you want to work with on the project, sign up together for a project group. (This is optional.)
- Sign up by Friday, Oct 27, or I will assign you to a group.
- Groups must be of size $3 - -4$. No exceptions.

Last time

1. For zero-sum games with perfect information, we can find the minimax solution on trees which are not too big.
2. We don't need to evaluate all nodes. We can prune those that cannot give any information. Alpha-beta pruning is a good way.
3. First, we make sure we understand alpha-beta pruning?
4. Next, we consider what to do if the tree is too large to search to the terminal nodes?

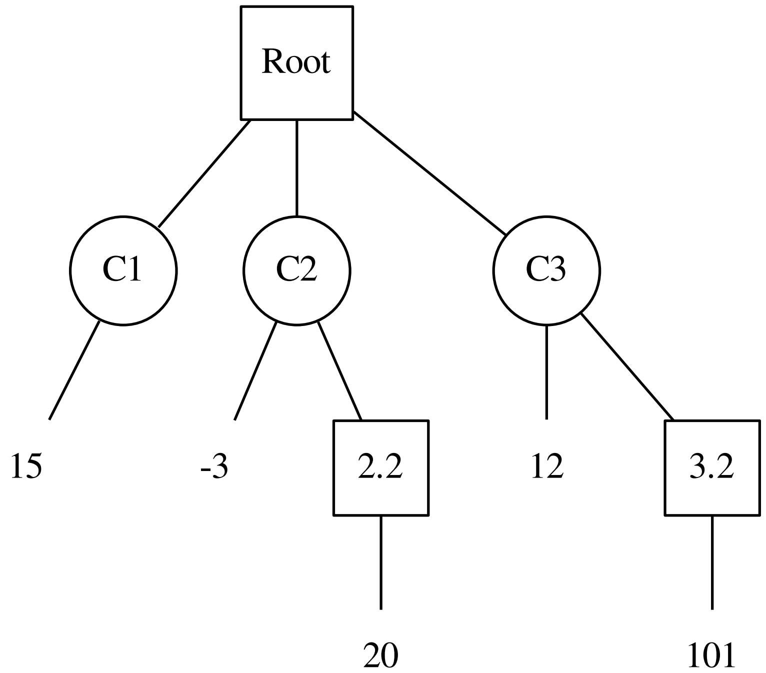
Reminder: alpha-beta pruning

- Pass the current range [alpha,beta] down the tree.
- When passed down the tree, the range is unchanged. But a child can change the range of its parent.
- Only `MAX` nodes can change alpha; only `MIN` nodes change beta.
- Each node contains a range, [alpha,beta] where
  - Alpha is the maximum lower bound of the value of the node
  - Beta is the minimum upper bound of the value of the node
  - when beta $\leq$ alpha prune the subtree containing that node.
- Start with the range $[-\infty, \infty]$.

function: evaluate(node J, α, β)

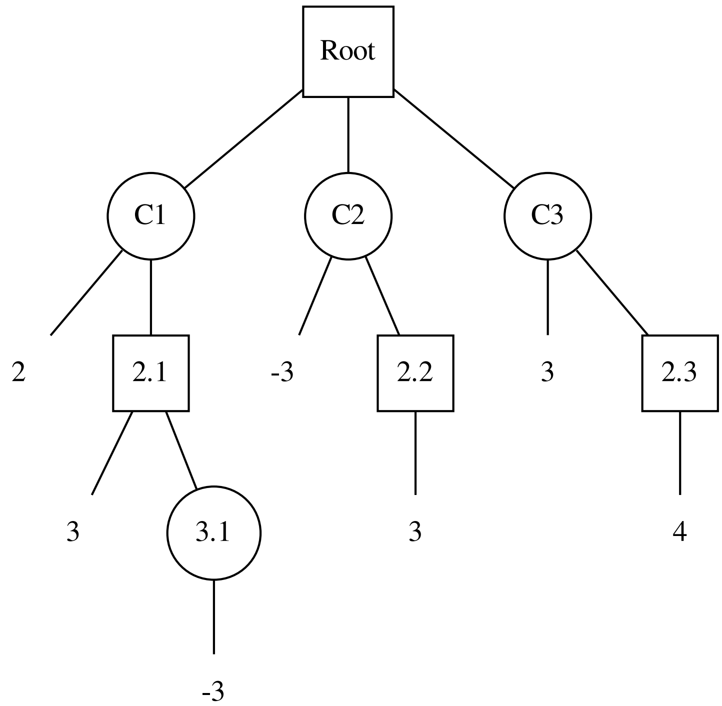
1. If J is a terminal node, return $U(J)$ and a pointer to `null`. Else, move to next step.
2. If J is a `MAX` node,
   2.1 Get next unvisited child of $J \rightarrow n_i$.
   2.2 If $i = 1$, $\alpha = J(n_i)$, else $\alpha = \max[\alpha, \textit{evaluate}(n_i, \alpha, \beta)]$.
   2.3 If $\beta \leq \alpha$, break
   2.4 Until all child nodes are of J are evaluated. Return α and pointer to a child with value α.
3. If J is a `MIN` node,
   3.1 Get next unvisited child of $J \rightarrow n_i$.
   3.2 If $i = 1$, $\beta = J(n_i)$, else $\beta = \min[\beta, \textit{evaluate}(n_i, \alpha, \beta)]$.
   3.3 If $\beta \leq \alpha$, break
   3.4 Until all child nodes are of J are evaluated. Return β and pointer to a child with value β.

$\alpha - \beta$ pruning worked examples



(Then we will work through it.)

$\alpha - \beta$ pruning worked examples



(Then we will work through it.)

What to do in large games

- Often the game tree is so large, it is not possible to perform minimax search, because you cannot reach terminal nodes except towards the end of the game.
- Use an “evaluation function” (“heuristic”) to *approximate* the value of deepest nodes we can reach, if they are not terminal nodes.

Evaluation functions

(aka board evaluation functions, heuristic evaluation function, “heuristic”)

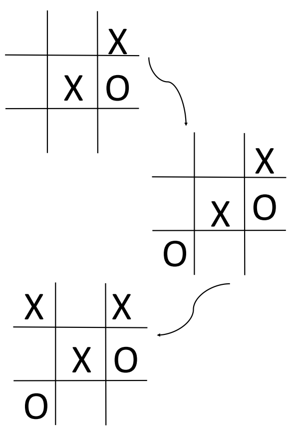
- Introduce an evaluation function, which is an approximation to $V(J)$. Use it just like the value.
- Search the tree to a given depth or until terminal nodes are found.
- Use U or the evaluation function to evaluation the node.
- Propagate that value up the mini-max tree.

Evaluation functions

- Unlike in A*, notions of admissible or monotonic do not apply.
- Approximate pay-off reachable from this node.
  - More positive evaluation function — Player 1 more likely to win.
  - More negative evaluation function — Player 2 more likely to win.

Examples of Heuristics — Tic-Tac-Toe

What would be a good heuristic?



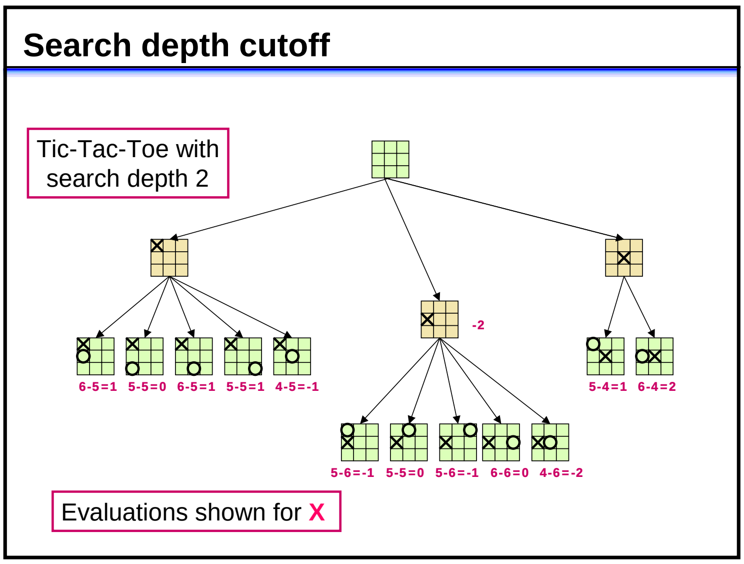
Heuristics for Tic-Tac-Toe

https://www.cs.duke.edu/courses/summer04/cps001/lectures/Lecture22.pdf Broken link

- There are eight lines where a player can get three in a row: 3 rows, 3 columns, and 2 diagonals.
- Heuristic: (number of lines where X can win) - (number of lines where O can win).

Player 1 wants to maximise this; Player 2 wants to minimise this.

Figure from https://www.cs.duke.edu/courses/summer04/cps001/lectures/Lecture22.pdf Broken link



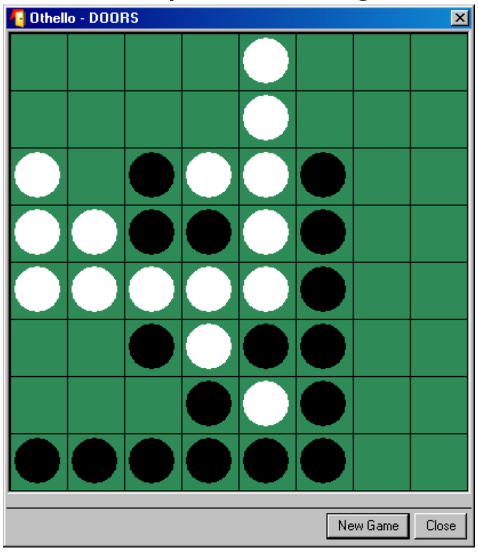
Example of Heuristics — Connect-4

Good heuristics for Connect-4? Thoughts?

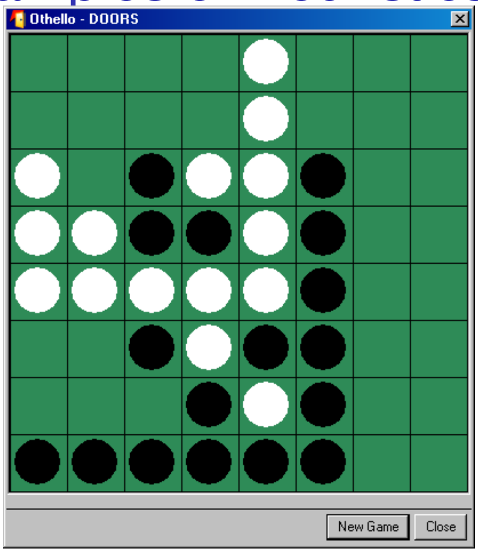
- Number of unblocked rows, columns, diagonals for Player 1 minus number of unblocked rows, columns, diagonals for Player 1.
- Distance to nearest win for Player 2 - distance to nearest win for Player 1.

Examples of Heuristics — Othello/Reversi

An example of the game. `https://eothello.com/`



Examples of Heuristics — Othello/Reversi



Some ideas,

1. Relative difference in number of coins
2. Relative difference in the number of legal moves (Mobility)
3. Difference between the number of corners captured.
4. Stability of coins (number that never be flanked – number that be flanked in the next move).

Combined in some way.

How to find good heuristics

- Often a hard problem. Requires knowledge of the game.
- Need to make sure the heuristic is correctly normalized against values returned at actual terminal nodes.
- Often trial and error is used: given two heuristics,
  - Play them against each other in multiple games (from both starting positions), and
  - play them both against other heuristics
  - The one that wins the most is likely better.

Finding a good combination of heuristics

- Sometimes it is necessary to combine heuristics.
- For example, backgammon

  Make aggressive progress: $e_1 =$ (the distance to remove all of your opponent's pieces from the board) $-$ (the distance to remove all of your pieces from the board).

  Protect your pieces: $e_2 = -$ (the number of your vulnerable pieces).
- You need a balance of both of these.
- Too much e_1 is too aggressive (too much risk); too much e_2 is too conservative.

Finding a good combination of heuristics

- Let $\{e_1(J), e_2(J), \ldots, e_k(J)\}$ be a set of k heuristics
- You combine them into a single heuristic via a linear combination:

  $$H(J) = w_1 e_1(J) + w_2 e_2(J) + \cdots + w_k e_k(J).$$

  How to find the best or good set of weights?

Method I — hand-tweaking

Often just done by guess work or by manual tweaking the weights.

Method II — stochastic hillclimber

Start with a set of weights (e.g. random, sensible guess)

Repeat:

1. Perturb the weights slightly (using a random number generator).
2. Compare the heuristic using the original weights to that using the perturbed weights by playing the two against each other in one or multiple games.
3. Choose the best of the two.

To evaluate which heuristic is best, play them against each other N times and average the payoffs.

Stochastic hillclimber

Input: A set of heuristics $\{e_1(J), e_2(J), \ldots, e_k(J)\}$, a small quality ϵ.
Output: A set of weights $(w_1, w_2, \ldots, w_k)$.
Initialise weights (random or using game-dependent knowledge)
Normalise the weights to sum to 1 {optional}
Set current heuristic to be $H = \sum_1^k w_i e_i$.
while time left **do**
  Generate k **zero-mean** random numbers, $(r_1, \ldots, r_k)$.
  Remove mean from r, $r_i \leftarrow r_i - \frac{1}{k}\sum_j^k r_j$ for all i.
  Set test heuristic to be $H^{\text{test}} = \sum_1^k (w_i + \epsilon r_i) e_i$.
  if H^{test} evaluates higher than H **then**
    $H \leftarrow H^{\text{test}}$.
  end if
end while

An example of 'Black-box optimization'

To evaluate a heuristic, you must play it in a game.

- The game is like a black box — cannot be expressed as a function.
- Also an example of *expensive optimization* — evaluation has a high computational cost.

Summary

The standard way to create agents which play two-player, perfect information games.

- Use a good mini-max tree search algorithm with effective pruning.
- Use effective heuristic evaluation function.

How to find good heuristic evaluation functions is a major issue.

The default approach

For decades, Mini-Max search was the default approach.

- Arthur Samuel's check playing program (1950s-60s).
- Deep Blue beating World Champion Garry Kasparov (1996)
- Many game AI in between.

All had MiniMax search, pruning, optimised evaluation heuristic as the heart.

With extensions

- Database of opening moves.
- Database of end games.
- Move reordering to increase pruning.

See Chapter 4 of Schalk for further discussion of these techniques.
See Chess Programming Wiki

The game Go

These approaches completely failed at this game.

- Game tree huge, 10^{360} on 19×19 board[1].
- No known effective heuristics.

AI was defeated by Go.

[1] Wikipedia artical on 'Game Complexity'

Next topic — Learning in games

Which solved Go and other games.

Conclusions

1. Minimax search + alpha-beta pruning + a good heuristic function $=$ a widely-used approach for game algorithms.
2. It is possible to learn effective heuristic functions, for example using stochastic hillclimbing.
3. Evaluation of heuristics requires playing the game.

Next time: Learning in games.