AI third lecture

1 Search

1.1 general search

When drawing a path, we can grow a tree associated with a search. We grow it in lexographical order.
To grow the tree for each node, starting from a starting node S, we add each node we can visit without retracing our steps.
note: search is not about maps, it's about choicse
Breadth first search is stupid because it extends paths multiple times.
We will modify the algorithm such that it won't be added if the final node hasn't been seen.

Hill climbing search: on each decision, pick the one that's closer to the goal. This is an analog of DFS.
beam search: only keep the best beamWidth paths on each level, this is an analog of BFS.
best first search: always take the easiest path.

1.2 Fastest path

When we say which node is cloests, we mean a heuristic difference.
branch and bound: always extend the path with the slowest accumilated path.
improvement on branch and bound (dyksras) once we know a path can't get us to the final, don't continue it.
Airline improvement: add in the "airline (absolute shortest length ignoring connections)" to the guess.
Airline improvement adds in an admissible heuristic: lower bound of the actual distance.
A*: branch and bound combined with extended list and admissible heuristic, both don't repeat paths and add in an admissible heuristic, or "airline" distance"

Admissible heuristic can lead you into trouble, it must be a map.
When it's not a map, heuistic must be:

1. Admissible: $H(x, G) \leq D(x, G)$
2. Consistent: $H(x, g) - H(y_1, g) \leq D(x, y)$

AI third lecture

1.3 games, minimax, alpha-beta

Ways computers could play chess:

1. A computer could play chess by analyzing analysis, strategy and tactics to decide to make a move, but no one can do that.
2. if-then rules, checks to see what moves are availalbe ranks them and picks the strongest, not a good way to do it
3. Look ahead an evaulate, we must have a way to evaluate situations.
4. British muesum, test all possible, really not possible
5. **Look ahead as far as possible**

Evaluations functions:
$staticValue = g(f_1, f_2, f_3)$ where fs are features of the board.
We reduce this to a linear polynimal $c_1 * f_1 + c_2 * f_2 + c_3 * f_3$ and that is a linear scoring polynomial.

Vocabulary:
Branching factor, b: how many children each node in the tree has
Depth, d: how far down the tree goes, starting with the root at 0
Note that the number of leafs is b^d

Two players, maximum player and minimum player. One wants the minimum value and the other one wants the maximum value.
In the **minimax** strategy, you go down to the bottom and combute static values, then go up the game tree and assign a value to each node based on what the play whose move it is will pick.
Alpha beta: once you devolop a situation such that you know the best value you can get, stop computing. This is an adaptation of minimax. To do this, we propogate the tree upwards and assign a maximum value that player can get from that path. If we know one of our moves could allow the other player to get a better path, we will not use that move.
deep cutoff: if you know that a path can get the opponent to a winning move, you can cut it off no matter how deep.
Static evaluations do $S = 2b^{d/2}$ in the optimal case. Average case is close to this, can never make it worse. Aka we always want to do it.
Saving time, to save time we can do all the moves from the $d - 1$ level,

which takes $1/d$ as much time. We propogate upwards and do the minimax algorithm at each level before moving onto the next.

$$S = 1 + b + b^2 + \cdots + b^{d-1} \tag{1}$$

$$b * S = b + b^2 + \cdots + b^d \tag{2}$$

$$b * S - S = b^d - 1 \tag{3}$$

$$S = \frac{b^d - 1}{b - 1} \approx \frac{b^d}{b} = b^{d-1} \tag{4}$$

$$\tag{5}$$