Search

1 Nets and search

1.1 introduction

When finding a path, you can create a **search tree** that denotes all possible paths.
A search tree is a semantic tree which consists of:

1. Nodes that denote paths
2. Branches which connect paths to one step-path extensions
3. Writers can connect a path in the tree to a description of the path
4. Readers can produce a path's description based on the tree

Determining the children of a node is called **expanding** the node. Nodes are **closed** if they're not expanded and **open** if they are.

1.2 algorithms

Hill Climbing: a modification of deapth first search where we order choices according to a heuristic, such as straight line distance, measure of how far we are from the goal. We order the new elements we are inserting into our queue by this heuristic
Problems with hill climbing:

1. Could get stuck in local optimum
2. The search space could be flat with a dramatic peak in the middle
3. The gradient goes against the directions you're stepping, and all steps seem to take you down.

beam search / best first search: at each step in breadth-first search, only expand the w best nodes. Best first search, the best node no matter where it's from is selected.

1.3 optimal search

Branch-and-bound search: keep track of all partial paths contending for future consideration. Extend the shortest one, consider the new paths and repeat with the shortest one.
We can add in a guess of the remaining distance to help us. Over estimates can cause us to miss the optimal path, but underestimates never can. Straight line distance will always be an underestiamte

To modify our search, we just instead order the priority queue we keep track of maths with by the sum of the path length and estimate of remaining cost. We can further modify it by adding the condition: if two or more paths reach a common node, delete all paths except the one that reaches the common node with the minimum cost.
A* is just branch-and-bound with a distance heuristic and path compression.

1.4 adversarial search

We can represent games, such as chess, as a decision tree of moves where opponents get to take turns moving.
static evaluation: the process of determining the quality of a static board. For our purposes, assume one player is trying to maximize their score and another player is trying to minimize their score. Negative scores are good for the minimum player and maximum scores are good for the maximum player.
In the **minimax** procedure:

1. If the limit of the search has been reached, compute the static value relative to current player and return the result

2. otherwise, if we are at a minimizing level, use minimax on the children of the current level and report the minimum.

3. else if we are at a maximizing level, use minimax on the children of the current level and report the maximum.

If we are testing a minimizing level, there is no point in investigating children where we know the maximizor can get a higher score than if we picked another path.
This procedure is called **alpha-beta** once we know we can force a particular score to occur, there's no point in investigating other worse scores.
On a minimizing level, keep track of the minimum score among the children, if a child is found to have a direct child with a higher score, we don't need to investigate tha node anymore.
Alpha-beta proceudre:

- If the level is the top layer, let alpha be ∞ and beta be $-\infty$
- If the search limit has been reached, compute the static vlaue of the

current position relative to the appropriate player and return the result

- If the level is a minimizing level,
  - until all children are examined with alha=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 and record the value reported
    - compare the value reported with the beta value, and if the reported value is smaller than the betra value update beta
  - report beta
- If the level is a maximizing level,
  - until all children are examined with alha=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 and record the value reported
    - compare the value reported with the alpha value, and if the reported value is larger than the alpha value update alpha
  - report alpha

To improve the algorithm, we can use progressive deppening and continue doing our alpha,beta to the n-1 level and expanding.
For chess, we add in the **singular-extension heuristic** that states that as long as one move stands out from the rest for the opponent we need to keep going, as to avoid a capture.