1 Uninformed Search

1.1 *Formulate* - Search - Execute.

A goal is the set of world states in which the goal is satisfied.

Problem formulation is deciding what actions and states to consider and abstracting them to a level of detail that is processable. (Uber Micro or Uber Macro)

Most uninformed search algorithms assume the environment is static, discreet and non-deterministic with no surprises. (Open-Loop)

A well defined problem has:

- An initial state.
- A successor function. (A way to generate new states from the current state).
- A state space and connected states.
- A goal test.
- Step costs that can be used to calculate a path cost.

Solution quality is measured with regards to proximity to the optimal solution.

1.2 Formulate - *Search* - Execute.

Searching...

- ...has a search tree of connected nodes.
- ...applies the successor function to generate new states. (Unexpanded but generated nodes are called the fringe.)

Performance is measured in:

- Completeness. (Is the search guaranteed to find a solution?)
- Time Complexity. (How long does it take?)
- Space Complexity. (How much memory does it pwn?)

Complexity = branching factory (the max successors a node can have) + depth of the shallowest goal node + max length of any path.

Search cost = time (possibly factoring in memory costs too). Total cost = time + path cost.

1.2.1 Uninformed (Blind) Search

Breadth-First Search

Root node is expanded first then all successors are expanded next, then their successors.

- Huge memory requirements. $O(b^{d+1})$
- Huge time requirements (although memory is slightly more of a problem).

Uniform-Cost Search

Instead of expanding shallowest node, expands node with *lowest path cost*. If all path costs are equal, this is the same as breadth-first.

Will get stuck in an infinite loop if it expands a path with zero cost that leads to itself.

Depth-First Search

Always expands the *deepest* node in the current fringe of the search tree.

- Uses far less memory than breadth-first. $O(bd)$
- Time complexity of $O(b^d)$
- Not complete or optimal - it can go down a subtree of unbounded depth and never terminate.

Depth-Limited Search

Depth-First but to a certain depth in the tree.

Put the depth too low: Incomplete. May never go down far enough to find the solution. Put the depth too high: Non-Optimal. Wastes time searching down unnecessarily.

Iterative Deepening

Many Depth-Limited searches, increasing in depth by one on each try.

Seems wasteful but is actually not that costly. In a search tree with the same branching factor each level, most nodes are at the bottom. It does not matter if the upper levels are generated many times. Even with this overhead, IDS is still much faster than BFS. It is also complete.

1.3 Heuristic Search

Uses a heuristic evaluation function to try and always pick the best node first. Maintains an ordered queue of nodes.

A^* Search

Evaluates nodes by combining the cost to reach the node ($g(n)$) with the cost to get from the node to the goal ($h(n)$).

$f(n) = g(n) + h(n)$

- It is optimal if $(h(n))$ is an admissible heuristic - one that never overestimates the cost to reach the coal.
- A heuristic is consistent if, for every node n and every successor n' of n generated by and action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n':

  $h(n) \leq c(n, a, n') + h(n')$