Searching to solve problems

- Basic concepts of approach
  - *State* encodes all relevant information about a problem.
  - *Actions* take us to a different state; have some cost.
  - Consider possible sequences of actions until goal is reached.
  - Want the lowest cost sequence that reaches goal.
- Basic concepts of searching
  - *State space* is a graph where vertices are states and edges are actions
  - *Search tree* represents choice of action at each step, i.e. possible paths
  - *Nodes* in the search tree contain the state and other information (such as parent, cost, etc.)
- Properties of searches:
  - *Optimal* — will find the lowest cost path to G
  - *Complete* — will find a path to G if one exists and will return failure otherwise (i.e. terminate)
  - *Time complexity* — run time
  - *Space complexity* — memory requirements

Types of searches

- *Blind searches* consider alternative actions without any bias
  - Breadth-first
  - Depth-first
  - Iterative deepening
  - Uniform cost
- *Heuristic searches* use a heuristic to inform their choice of action
  - Greedy
  - A*

Depth-first search (DFS)

- Queue formulation of algorithm:
  - Put start node on a queue Q
  - Repeat:
    - If Q is empty, return failure
    - Remove first node N from Q
    - If N is goal, return success
    - Add children of N to *front* of Q
- Properties:

Optimal?	No
Complete?	No
Time complexity	$O(b^m)$
Space complexity	$O(mb)$

Breadth-first search (BFS)

- Queue formulation of algorithm:
  - Put start node on a queue Q
  - Repeat:
    - If Q is empty, return failure
    - Remove first node N from Q
    - If N is goal, return success
    - Add children of N to *back* of Q
- Properties:

Optimal?	Yes
Complete?	Yes
Time complexity	$O(b^d)$
Space complexity	$O(b^d)$

Greedy search

- A best-first search
- "Best" = closest to goal
- Queue formulation:
  - Put start node on a queue Q
  - Repeat:
    - If Q is empty, return failure
    - Remove from Q the node N with lowest $\hat{h}$ value
    - If N is goal, return success
    - Add children of N to of Q
- Properties:

Optimal?	No
Complete?	No

- Greedy search is like depth-first search:
  - tends to follow one path as far as possible
  - expanding a path which is close to the goal makes it closer to the goal!

The A* algorithm

- Queue formulation:
  - Put the root node on a queue Q
  - Repeat:
    - If Q is empty, return failure
    - Remove the node N from Q with the lowest value of $\hat{f}(\cdot) = \hat{g}(\cdot) + \hat{h}(\cdot)$
    - if N is the goal, return success
    - Add children of N to Q
- OPEN/CLOSED list formulation:
  - Put the start node on a list OPEN
  - Create an empty list CLOSED
  - Repeat:
    - If OPEN is empty, return failure
    - Remove node N from OPEN with lowest value of $\hat{f}(\cdot) = \hat{g}(\cdot) + \hat{h}(\cdot)$ and add to CLOSED
    - If N is a goal, return success
    - For each child C of N:
      - if C is not on OPEN or CLOSED, add to OPEN
      - if C is on OPEN, update $\hat{f}(C)$ if necessary
      - if C is on CLOSED and must be updated, remove C from CLOSED and add to OPEN

Notes on A* properties

- A* is a best-first search
- "Best" = total estimated cost from S to G, $\hat{f} = \hat{g} + \hat{h}$
- $\hat{g}(N)$ gives cost along current path from S to N
- Heuristic $\hat{h}(N)$ gives estimated cost from N to G
- An *admissible* heuristic is one that does not overestimate the distance to the goal.
- A* is like breadth-first search:
  - it tends to expand several paths at the same time
  - expanding one path makes it seem a little worse than other paths under consideration — actual cost is more than estimated cost!
- If the heuristic is *admissible*, A* is optimal
- Obviously, the better the heuristic, the more efficient the search…
- But *why* is A* optimal? What does admissibility have to do with it?

Optimality of A* (intuitively)

- The heuristic *guides* the search.
- An admissible heuristic is an optimistic heuristic.
- Why do we need an optimistic heuristic for optimality?
  - We stop at the first goal found.
  - A pessimistic heuristic could cause us to miss a route to the goal.
  - An optimistic heuristic may "mislead" the search, but we won't miss the optimal path.

Notation

- Path cost from S to N
  - $\hat{g}(N)$ is the cost of the actual path
  - $g(N)$ is the cost of the shortest path
  - Note: $\hat{g}(N) \geq g(N)$
- Distance from N to G
  - $\hat{h}(N)$ is an estimate of the cost
  - $h(N)$ is the cost of the shortest path
  - Admissibility means $\hat{h}(N) \leq h(N)$
- Total estimated cost
  - $\hat{f}(N) = \hat{g}(N) + \hat{h}(N)$
  - $f(N) = g(N) + h(N)$
  - Relationship between $\hat{f}(N)$ and $f(N)$ in general is unknown
- On an optimal path to a node N_K
  - Optimal path is $\xi \equiv N_0 = S, N_1, N_2, \ldots N_K$
  - Along ξ for any node N_L,

$$\hat{g}(N_L) = g(N_L)$$

  - If $N_K = G$, for any nodes N_I and N_J on ξ

$$f(N_I) = f(N_J)$$

Optimality of A* (formally)

Lemma 1: A* cannot terminate with a suboptimal goal N_{G2}

Proof:

1. To expand a suboptimal goal N_{G2}, it must have the lowest $\hat{f}(\cdot)$ value on OPEN
2. The estimated cost $\hat{f}(N_{G2})$ is greater than the cost to the optimal goal $f(N_G)$
   - Because $\hat{h}$ is admissible and N_{G2} is a goal state,

$$\begin{aligned} \hat{f}(N_{G2}) &= \hat{g}(N_{G2}) + \hat{h}(N_{G2}) = \hat{g}(N_{G2}) \\ f(N_{G2}) &= g(N_{G2}) + h(N_{G2}) = g(N_{G2}) \end{aligned}$$

   - Since $\hat{g}(\cdot) \geq g(\cdot)$

$$\hat{f}(N_{G2}) \geq f(N_{G2})$$

   - Since N_{G2} is suboptimal, $f(N_{G2}) > f(N_G)$, so

$$\hat{f}(N_{G2}) > f(N_G)$$

Proof of Lemma 1

3. There is a node on OPEN on the optimal path to N_G with estimated cost less than $f(N_G)$
   - Consider the optimal path to the optimal goal

$$\xi \equiv N_0 = S, N_1, N_2, \ldots N_K = G$$

   - There must be some node N^* from ξ on OPEN
   - (Otherwise, the entire path would have been expanded and we would have already found the optimal goal.)
   - A* has found an optimal path to N^*, so

$$\hat{g}(N^*) = g(N^*)$$

   - Because of admissibility,

$$\begin{aligned} \hat{g}(N^*) + \hat{h}(N^*) &\leq g(N^*) + h(N^*) \\ \hat{f}(N^*) &\leq f(N_G) \end{aligned}$$

4. Contradiction: N_{G2} does not have the lowest $\hat{f}(\cdot)$ value on open.

Optimality of A* (formally)

Lemma 2: At the beginning of each iteration of A*, there is always a node N^* on the OPEN list with the following properties:

- N^* is on an optimal path to a goal
- A* has found the optimal path to N^*
- $\hat{f}(N^*) \leq f(n_0)$

where n_0 is the start node.

Proof of Lemma 2 by induction

- Base case:
  - At the beginning, N^* is the start node n_0
- Inductive step:
  - Assume there is a node N^* on OPEN
  - In the next iteration:
    - Suppose N^* is not expanded — it will still have the same properties
    - Suppose N^* is expanded — one of its successors will be on the optimal path and added to the OPEN list. This successor will be the new N^*.
- N^* has estimated cost less than the optimal path
  - Because we have an optimal path to $N^*$and because $\hat{h}(\cdot)$ is admissible

$$\begin{aligned}\hat{f}(N^*) &= \hat{g}(N^*) + \hat{h}(N^*) \\ &= g(N^*) + \hat{h}(N^*) \\ &\leq g(N^*) + h(N^*) = f(N^*)\end{aligned}$$

  - Because N^* is on the optimal path to a goal

$$\hat{f}(N^*) \leq f(N^*) = f(n_0)$$

Optimality of A* (formally)

Theorem: Under the following assumptions:

1. The heuristic $\hat{h}$ is admissible (i.e. $\hat{h}(n) \leq h(n)$).
2. Each node has a finite number of successors.
3. All arcs in the state space graph have costs $> \epsilon > 0$ for some ϵ.

A* is guaranteed to terminate with a minimal-cost path to a goal.

Proof outline:

- Lemma 1: A* cannot terminate on a suboptimal goal
- A* cannot terminate with a suboptimal path to the optimal goal
- A* must terminate if there is a path to a goal

Theorem proof

- A* cannot terminate with a suboptimal path to the optimal goal
  - If we are about to expand the optimal goal reached by a suboptimal path, $\hat{f}(N_G)$ must be the lowest estimated cost on OPEN.
  - Since this path is suboptimal (and because the heuristic is admissible),

$$\hat{f}(N_G) = \hat{g}(N_G) > g(N_G) = f(N_G)$$

  - But by Lemma 2, there is a node N^* on the optimal path to N_G on OPEN and

$$\hat{f}(N^*) \leq f(N_G)$$

  - Contradiction: N_G does not have the lowest estimated cost on OPEN.
- A* terminates if there is a path to a goal
  - A* continues to expand nodes deeper into the search tree
  - Because every cost is at least ϵ, the $\hat{g}$ value of all nodes on OPEN would eventually exceed $f(n_0)$
  - *Unless* there are an infinite number of nodes with cost $< f(n_0)$. This cannot happen with conditions 2 and 3.

Uniform cost search

- Uniform cost search is essentially BFS for state spaces where edge costs are not the same.
- In the "standard" queue formulation, use the step:
  - Remove from Q the node N with lowest path cost $\hat{g}(\cdot)$
- Uniform cost search is equivalent to A* with $\hat{h}(\cdot) = 0$.
- This search maintains a "frontier" of nodes with (approximately) the same cost, hence the name.
- Like BFS:
  - it is optimal and complete,
  - has $O(b^d)$ time and space complexity

Iterative deepening

- Iterative deepening is a combination of BFS and DFS:
  - like BFS, it is optimal and complete
  - like DFS, its space complexity is not exponential
- Depth-limited search is a DFS with a depth cutoff
- Iterative deepening is repeated depth limited searches with an increasing depth cutoff, i.e. first to depth 0, then depth 1, then depth 2, and so on.
- Iterative deepening essentially throws away the results of every depth-limited search before going on to the next.
- But space complexity is $O(bd)$, not exponential.
- Time complexity is still $O(b^d)$ — prior depth-limited searches take a small amount of time compared to the final depth-limited search.
- For example, suppose $b = 10$ and $d = 6$

d	nodes visited
0	1
1	10
2	100
3	1,000
4	10,000
5	100,000
6	1,000,000
total	1,111,111