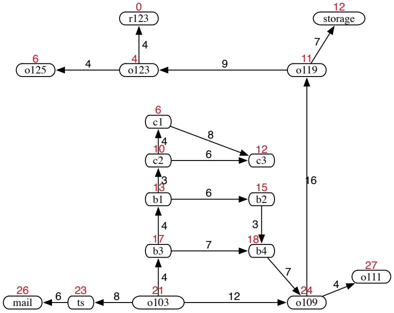
Heuristic Search

- Idea: don't ignore the goal when selecting paths.
- Often there is extra knowledge that can be used to guide the search: heuristics.
- $h(n)$ is an estimate of the cost of the shortest path from node n to a goal node.
- $h(n)$ uses only readily obtainable information (that is easy to compute) about a node.
- h can be extended to paths: $h(\langle n_0, \ldots, n_k \rangle) = h(n_k)$.
- $h(n)$ is an underestimate if there is no path from n to a goal that has path length less than $h(n)$.

Example Heuristic Functions

- If the nodes are points on a Euclidean plane and the cost is the distance, we can use the straight-line distance from n to the closest goal as the value of $h(n)$.
- If the nodes are locations and cost is time, we can use the distance to a goal divided by the maximum speed.
- If the goal is to collect all of the coins and not run out of fuel, the cost is an estimate of how many steps it will take to collect the rest of the coins, refuel when necessary, and return to goal position.

Straight-line distance heuristics for the Delivery Robot



- This heuristics is an underestimate because the its value is less than or equal to the exact cost of a lowest-cost path from the node to a goal.
- It is the exact cost for node o123.
- It is very much an underestimate for node b1, which seems to be close, but there is only a long route to the goal.
- It is very misleading for c1, which also seems close to the goal, but no path exists from that node to the goal.

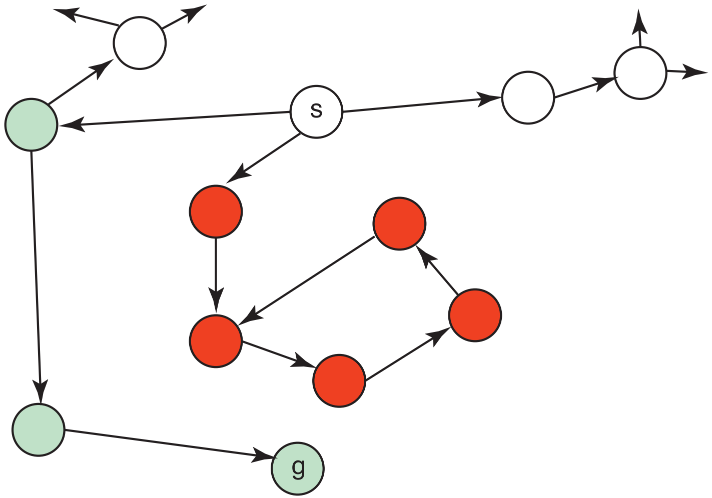
Heuristic Depth-first Search

- A simple use of a heuristic function is to order the neighbors that are added to the stack representing the frontier in depth-first search.
- The neighbors can be added to the frontier so that the best neighbor is selected first.
- This search chooses the locally best path, but it explores all paths from the selected path before it selects another path.
- Although it is often used, it suffers from the problems of depth-fist search.

Best-first Search

- Idea: select the path whose end is closest to a goal according to the heuristic function.
- Best-first search selects a path on the frontier with minimal *h*-value.
- It treats the frontier as a priority queue ordered by *h*.

Illustrative Graph — Best-first Search



Complexity of Best-first Search

- It uses space exponential in path length.
- It isn't guaranteed to find a solution, even if one exists.
- It doesn't always find the shortest path.

A^* Search

- A^* search uses both path cost and heuristic values
- $cost(p)$ is the cost of path p.
- $h(p)$ estimates the cost from the end of p to a goal.
- Let $f(p) = cost(p) + h(p)$. $f(p)$ estimates the total path cost of going from a start node to a goal via p.

$$\underbrace{\underbrace{start \xrightarrow{\text{path } p} n}_{cost(p)} \underbrace{\xrightarrow{\text{estimate}} goal}_{h(p)}}_{f(p)}$$

A^* Search Algorithm

- A^* is a mix of lowest-cost-first and best-first search.
- It treats the frontier as a priority queue ordered by $f(p)$.
- It always selects the node on the frontier with the lowest estimated distance from the start to a goal node constrained to go via that node.

Admissibility of A^*

- Admissibility of A*
  - A* always finds an optimal path, if one exists, and
  - the first path found to a goal is optimal.
- If a solution exists, even when the search space is infinite, a solution will be found and the first one found will be a lowest-cost solution.

Proposition *(A* admissibility):* If there is a solution, A^* always finds an optimal solution – the first path to a goal selected – if the following holds:

- the branching factor is finite,
- arc costs are bounded above zero (there is some $\epsilon > 0$ such that all of the arc costs are greater than ϵ), and
- $h(n)$ is an underestimate of the cost of the cheapest path from n to a goal node.

Admissibility of A^*

Proposition *(A* admissibility):* If there is a solution, A^* always finds an optimal solution – the first path to a goal selected – if the following holds:

- the branching factor is finite,
- arc costs are bounded above zero (there is some $\epsilon > 0$ such that all of the arc costs are greater than ϵ), and
- $h(n)$ is an underestimate of the cost of the cheapest path from n to a goal node.

Proof:

Part A: a solution will be found.
If the arc costs are all greater than some $\epsilon > 0$, eventually, for all paths p in the frontier, $cost(p)$ will exceed any finite number and, thus, will exceed a solution cost if one exists (at depth in the search tree no greater than m/ϵ, where m is the solution cost). Because the branching factor is finite, only a finite number of nodes must be expanded before the search tree could get to this size, but the A* search would have found a solution by then.

Part B: the first path to a goal selected is an optimal path.
The f-value for any node on an optimal solution path is less than or equal to the f-value of an optimal solution. This is because h is an underestimate of the actual cost from a node to a goal. Thus, the f-value of a node on an optimal solution path is less than the f-value for any non-optimal solution. Thus, a non-optimal solution can never be chosen while a node exists on the frontier that leads to an optimal solution (because an element with minimum f-value is chosen at each step). So, before it can select a non-optimal solution, it will have to pick all of the nodes on an optimal path, including each of the optimal solutions.

The role of the heuristics

- The admissibility of A^* does not ensure that every intermediate node selected from the frontier is on an optimal path from the start node to a goal node.
- How the heuristic function improves the efficiency of A^*:
-
    - c is the cost of a shortest path from a start node to a goal node.
    - With an admissible heuristic, every path is expanded from a start node in the set:

      $$\{p : cost(p) + h(p) \leq c\},$$

      and some of the paths in the set:

      $$\{p : cost(p) + h(p) = c\}.$$

    - Improving h affects the efficiency of A^* if it reduces the size of the first of these sets.

Dominance

- An admissable heuristic *dominates* another admissable heuristic if, for all states (i.e., nodes), the former has a higher value than the latter.
- Observation: Given two admissable heuristics that don't dominate each other, we can always make a better heuristic by *combining* them, i.e., by simply taking the max of the two heuristic values.