Informed (Heuristic) Search Algorithms

The generic informed/heuristic search algorithm is called best-first search. Best-first search works just like uniform-cost search (UCS) except we store our frontier as a priority queue sorted by an *evaluation function* known as f(n). Just like UCS, best-first search chooses to examine nodes with the lowest values of f(n) first. Where this algorithm differs from UCS is that f(n) is an estimate of the lowest-cost of path from the initial state, to the state at node n, to any goal state. Recall that UCS's frontier is sorted by g(n), which is the lowest cost of the path from the initial state to the state at node n, so UCS does not try to estimate the cost of the path *after* n towards the goal; it only takes into account the cost of the path *before* n.

Best-first search typically uses a heuristic function, denoted h(n), as an estimate of the cost from node n to a goal state. By changing the definition of f(n) to various functions involving g(n) and/or h(n), we obtain three different algorithms:

If we define f(n) = g(n), best-first search degrades to uniform cost search.
If we define f(n) = h(n), best-first search becomes an algorithm called greedy best-first search.
If we define f(n) = g(n) + h(n), best-first search becomes an algorithm called A* search (or just A*, pronounced "A-star").

The pseudocode for best-first search below is identical to UCS, except the frontier's priority queue is kept sorted by f(n), rather than g(n).

```
BEST-FIRST-SEARCH(problem)    // aka A*, greedy best-first search, or uniform cost search/Dijkstra, for various f/g/h
    node ← a new node corresponding to the initial state, with f/g/h set to appropriate values
    frontier ← a priority queue of nodes sorted by f(n), initialized to contain only node
    explored ← an empty set of states
    loop forever:
        if frontier is empty, return failure
        node ← pop(frontier)  // remove node with smallest f(n) value from frontier
        if IS-GOAL(node.state), then return the corresponding solution
        add node.state to the explored set
        for each action in ACTIONS(node.state):
            child_node ← new node with   child_node.state = RESULT(node.state, action)
                                         child_node.g = node.g + COST(node.state, action, child_node.state) [if using g]
                                         child_node.h = h(child_node)                                       [if using h]
                                         child_node.f = [whatever function f(n) is defined as]
                                         child_node.action = action
                                         child_node.parent = node
            if child_node.state is not in explored or frontier:
                add child_node to frontier
            else if child_node.state is already in frontier, but child_node.f is better than the frontier's:
                replace that frontier node with child_node
```

Admissibility and consistency of the heuristic function

An admissible heuristic is one that **never overestimates the cost to reach the goal**. Because h(n) estimates the cost from node n to any goal state, h(n) must never be greater than the true cost from n along the cheapest path to a goal state.

A consistent heuristic is, informally, one that always decreases in a "consistent" manner as one moves along a path from the initial state to the goal state(s). Formally, a heuristic h(n) is consistent if, for every node n and every successor n' of n, the estimated cost of reaching the goal from n is no greater than the step cost from n to n' plus the estimated cost of reaching the goal from n': $h(n) <= COST(n, a, n') + h(n')$ or equivalently, $h(n) - h(n') <= COST(n, a, n')$.

Another way to interpret this is a consistent heuristic **never overestimates the cost of a single step** from n to n'.