# Lecture 5: Uninformed Search with Step Costs & Informed Search

*9/13/2016*

Last time, we mostly focused on uninformed search strategies that don't consider step costs.

Here, again, is our general purpose search recipe:
**Function** graph-search (problem): returns a *solution*, or *fail*

Init frontier list with root node (has init state of problem)

Init explored list to empty

**loop do**:

**1. If** frontier is empty :

**then return** *fail*

2. Choose & remove a node from frontier

**3. if** the node contains goal state :

**then return** *solution* (*i.e., path from current node back to root*)

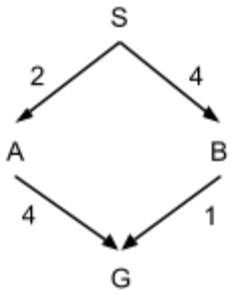
4. Add the state in that node to explored

5. Expand that node, add those children nodes points to successor states not already in explored (or already pointed to by another node in frontier) to frontier

Question: think about how you want to implement the check for whether the state is already in a node on the frontier

## Problems that have step costs

For problems like the 8-puzzle, the water jug problem, or the missionary and cannibal puzzle, every action has the same cost. Even though we care about the solution length (# of steps to solve the puzzle) we don't prefer certain moves over other moves.



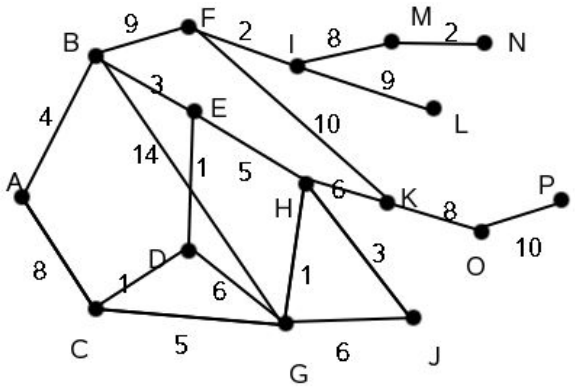

In many goal-oriented search problems, however, different actions may incur different costs. The optimal solution should tell us what actions to take to get from the init state to the goal state with the lowest total path cost, not necessarily the fewest actions.

To make this idea more formal, we will denote the function that assigns scores to nodes as f(n), typically referred to as an evaluation function. In the tree/graph-search algorithm, this is

calculated whenever we create a node. Different search strategies can be seen as different options of computing f(n).

- Uniform-Cost (aka Dijkstra's Algorithm)
  - Expand the node with the lowest *path* cost so far (i.e., starting from the root to the node containing the current state.
    - Notation: Let g(n) be a function that takes a node as its input and returns a value representing the path cost from the root to this node.
    - Here, we are using g(n) as f(n).
  - Question: What data structure should we use to implement frontier list in this case?
  - Question: What happens if all steps have the same cost?
  - Question: What happens if some state has an action that loops back to itself with 0 cost?
  - Question: In general, should we allow actions to have zero or even negative cost?
  - Question: If there are two possible paths to the same goal state, are we guaranteed to arrive at the goal state via the optimal path? Consider the four-node graph above. Does the algorithm find the right solution path?
  - Optimal? Yes
  - Complete? Yes
  - Time and Space Complexity:
    - We can't use depth level of the goal state here because the algorithm may (rightly) search down another branch to a deeper depth than the depth level where we find our goal.
    - Instead we'll denote that the optimal solution has a cost of C*
    - We also denote that the smallest step cost to be a positive value, epsilon.
    - We expect that the deepest level we need to explore will be no more than C*/epsilon.
    - So the time and space complexity is $O(b^{\text{rounc}(C^*/\text{epsilon})})$

  - **In class exercise:** Use the following graph as the state space. Suppose the initial state is A and the goal state is P. Trace the first few node expansions.



# Informed search

The search strategies that we have looked at so far pick what to explore next in a systematic but uninformed way. (Arguably, uniform cost is a little smarter because it takes the path cost (thus far) into considerations.) But is is still "uninformed" in the sense that aside from knowing how to get from our current state to these next states, we have no real knowledge of whether any of these next states are any good (more likely to lead us to the final goal).

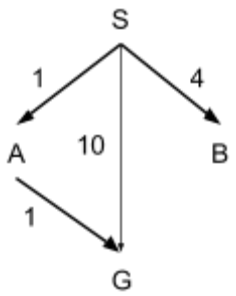
Informed search tries to explore actions that will lead to states that seem more promising, based on some problem-specific ideas.

Ideally, we know exactly how to predict whether a new node is on the optimal solution path or not. We can define a function $h_{oracle}(n)$ that returns 0 if node n is on the optimal solution path, 100 if it's on a suboptimal solution path, and infinity if it's not on the solution path at all. We can then simply set $f(n)=h_{oracle}(n)$, and the "search" is trivially about following the trail of states with 0 scores. We would never waste any effort.

> Question: What about a a function $h_{perfect}(n)$ that tells us the *exact cost* of going from node n to the goal state?

In practice, we don't have an oracle to help us score the nodes, so what we develop are heuristics for estimating how likely a node might be on the optimal path. Let's call this realistic heuristic function h(n), which estimates how much it'll cost to go from node n to the goal. For example, if we're lost in the mountains and want to find the basecamp, h(n) could be the elevation of the current location. However, since we don't actually know how far n is from the goal state, the output of this function is unreliable.

If we set the evaluation function f(n) to be h(n) and run graph-search, we'd get what is known as greedy search. It's called greedy because we always explore the candidate that seems to be the closest to the goal, even though it may actually not turn out to be the best choice. For example, in a path-finding problem, suppose we pick the next state to explore based on how far it is to the goal state in a straight line. Suppose we are trying to reach a particular spot in a building. So, we might be lead to search down a path going all around the building's walls before eventually finding the door, whereas if we considered the seemingly less optimal state of entering the parking lot, which is slightly further away from the goal state, we'd find a direct path into the building from underground. In some problems (e.g., one with an infinite state space) greedy is not even guaranteed to be complete. What if we have the magical $h_{perfect}(n)$? Would greedy be optimal and complete (consider the graph to the right)?



What if we set the evaluation function to be the sum of the path cost and the heuristic estimate?

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

That is, we've combined both uniform cost and greedy.

This is known as A* search. Intuitively, it is a better estimate of the overall solution path because it not only tries to guess how much further we have to go, it also keeps track of how far we've already traveled.

Is A* optimal and complete? Yes, under some conditions. More about this next time.

## Corresponding Readings

- Chapter 3.5 of Russell, S. and Norvig, P. (2010). Artificial Intelligence: A Modern Approach. 3rd ed. Prentice Hall.