CS3243: Introduction to Artificial Intelligence
Lecture Notes 3: Informed Search

1. Uniform-Cost Search (UCS) Algorithm

The Uniform-Cost Search algorithm improves on the Breadth-First Search (BFS) algorithm such that it is **optimal**.

- Instead of a Queue, a Priority Queue is used for the Frontier.
- $\hat{g}(u)$ keeps track of the minimum cost path of reaching node u discovered so far.

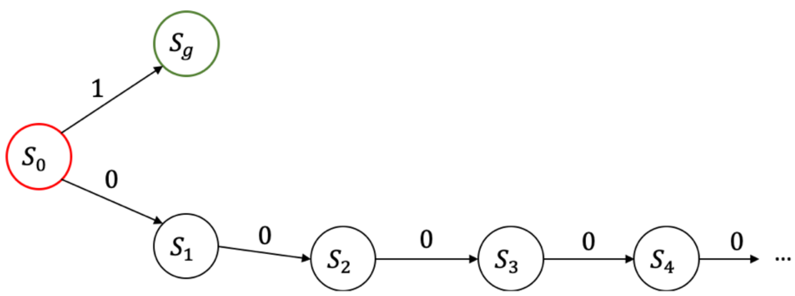
```
FindPathToGoal(u):
    Frontier ← PriorityQueue(u)
    Explored ← {u}
    ĝ[u] = 0

    while Frontier is not empty:
        u ← Frontier.pop()
        if GoalTest(u):
            return path(u)
        Explored.add(u)
        for all children v of u:
            if (v not in Explored):
                if (v in Frontier):
                    ĝ[v] = min(ĝ[v], ĝ[u] + c(u,v))
                else:
                    Frontier.push(v)
                    ĝ[v] = ĝ[u] + c(u,v)
    return Failure
```

Note that UCS is different from Djikstra's algorithm. In Djikstra's algorithm, we initialize $\hat{g}(u)$ for all the nodes u.

- **Completeness:** If we assume that every edge of the graph has cost $\geq \varepsilon$, then the algorithm will eventually reach the goal state.

  However, this algorithm is not complete for graphs with <u>edges of cost 0</u>. The number of steps required to reach the goal state might potentially be infinite, and the algorithm may not be able to reach the goal. For example:



- **Optimality:** The algorithm is optimal. When we perform the Goal Test on a node u, we have already found the optimal path to u.

Proof. We will prove that when we pop a node u from the frontier, we have found the optimal path to u.

We make the following definitions:
- $g(u)$ is the actual minimum path cost from the start node to node u.
- $\hat{g}(u)$ is the minimum path cost from the start node to node u discovered so far.
- $\hat{g}_{pop}(u)$ is the value of $\hat{g}(u)$ when we pop u from the frontier.

Let us consider the optimal path $\pi: S_0, S_1, S_2, \cdots, S_k, u$.
We will perform induction on the path.

<u>Base Case:</u>
When we pop node S_0 from the frontier,

$\hat{g}_{pop}(S_0) = g(S_0) = 0$ as S_0 is the start node.
$\hat{g}_{pop}(S_1) = g(S_1)$ as ensured by the Priority Queue.
$\hat{g}(S_1) = min(\hat{g}(S_1), \hat{g}(S_0) + c(S_0, S_1))$ as ensured by the algorithm.

Observe that $g(S_0) \leq g(S_1) \leq g(S_2) \leq \cdots \leq g(u)$.
Hence, at all times,

$$\hat{g}(u) \geq g(u) \geq g(S_k) \text{ --- (1)}$$

<u>Inductive Hypothesis:</u>
Let us assume that for all nodes $\{S_0, \cdots, S_k\}$, we find that $\hat{g}_{pop}(S_i) = g(S_i)$.
Then, when we pop S_k, $\hat{g}(u) \leq \hat{g}_{pop}(S_k) + c(S_k, u) = g(S_k) + c(S_k, u) = g(u)$

Hence,

$$\hat{g}(u) \leq g(u) \text{ --- (2)}$$

From (1) and (2), we can conclude that

$$\hat{g}_{pop}(u) = g(u)$$

- **Time:** $O(b^{1+d})$. Assuming that the optimal cost to reach the goal node is $C*$, and each edge has cost $\geq \varepsilon$. Then $\frac{C*}{\varepsilon} = d$.

- **Space:** $O(b^{1+d})$ which is still expensive.

2. Depth-First Search (DFS) Algorithm

The Depth-First Search (DFS) algorithm is just BFS but with the frontier changed to a last-in-first-out stack. This reduces the space complexity to $O(bd)$. However, the tradeoff is that the algorithm is not complete.

3. A* Algorithm

BFS, DFS and UCS are all *uninformed* search algorithms where there is no information on how much it might actually cost to reach a goal. In contrast, the A* algorithm is an *informed* search algorithm.

This algorithm is similar to UCS, except that it makes use of the function $\hat{f}(u)$ instead of $\hat{g}(u)$, where:

- $\hat{g}(u)$ is the minimum path cost from the start node to node u discovered so far.
- $\hat{h}(u)$, or the **heuristic**, is an estimate of the path cost from node u to the goal node.
- $\hat{f}(u) = \hat{g}(u) + \hat{h}(u)$

```
FindPathToGoal(u):
    // PriorityQueue to be implemented with f̂
    Frontier ← PriorityQueue(u)
    Explored ← {u}
    ĝ[u] = 0

    while Frontier is not empty:
        u ← Frontier.pop()
        if GoalTest(u):
            return path(u)
        Explored.add(u)
        for all children v of u:
            if (v not in Explored):
                if (v in Frontier):
                    ĝ[v] = min(ĝ[v], ĝ[u] + c(u,v))
                    f̂(v) = ĝ(v) + ĥ(v)
                else:
                    Frontier.push(v)
                    ĝ[v] = ĝ[u] + c(u,v)
                    f̂(v) = ĝ(v) + ĥ(v)
    return Failure
```

What is the property of $h(u)$ that makes sure that this algorithm is optimal?

Recall: UCS was optimal because on the optimal path π: $S_0, S_1, S_2, \cdots, S_k, u$, we have:

1. $\hat{g}_{pop}(S_0) \leq \hat{g}_{pop}(S_1) \leq \cdots \leq \hat{g}_{pop}(S_k) \leq \hat{g}_{pop}(u)$
2. $\hat{g}_{pop}(S_i) = g(S_i)$

Similarly, the A* algorithm is optimal if we have the following properties:

P1: $\hat{f}_{pop}(S_0) \leq \hat{f}_{pop}(S_1) \leq \cdots \leq \hat{f}_{pop}(S_k) \leq \hat{f}_{pop}(u)$

P2: $\hat{f}_{pop}(S_i) = f(S_i)$

Properties of the Heuristic, $\hat{h}(u)$

In order to ensure that **P1** and **P2** holds (such that the A* algorithm is optimal), the heuristic must fulfill the following properties:

- **Consistency:** $h(S_i) \leq c(S_i, S_{i+1}) + h(S_{i+1})$, also known as the triangle inequality.

Derivation. If $f(S_0) \leq f(S_1) \leq \cdots \leq f(S_k) \leq f(u)$ is true, and **P2** holds, then **P1** would hold. Hence, we want to make sure that $f(S_i) \leq f(S_{i+1})$.

Expanding and manipulating the above inequality,

$$g(S_i) + h(S_i) \leq g(S_{i+1}) + h(S_{i+1})$$
$$h(S_i) \leq g(S_{i+1}) - g(S_i) + h(S_{i+1})$$

Therefore, we get $h(S_i) \leq c(S_i, S_{i+1}) + h(S_{i+1})$.

- **Admissibility:** $h(S_i) \leq OPT(S_i)$, where $OPT(S_i)$ refers to the optimal path cost from node.

Derivation: Expanding from the property of consistency,

$$h(S_i) \leq c(S_i, S_{i+1}) + h(S_{i+1})$$
$$h(S_i) \leq c(S_i, S_{i+1}) + c(S_{i+1}, S_{i+2}) + h(S_{i+2})$$
$$\vdots$$
$$h(S_i) \leq c(S_i, S_{i+1}) + c(S_{i+1}, S_{i+2}) + \cdots + c(S_k, u) + h(u)$$

Since $h(u) = 0$ as u is the goal node,

$$h(S_i) \leq c(S_i, S_{i+1}) + c(S_{i+1}, S_{i+2}) + \cdots + c(S_k, u) = OPT(S_i)$$

Therefore, we get $h(S_i) \leq OPT(S_i)$.