UCS (Uniform Cost Search)

Algorithm 4 Uniform Cost Search(UCS): FindPathToGoal(u)

```
1: $F$(Frontier)$\leftarrow$ PriorityQueue($u$)                ▷ lowest cost node out first
2: $E$(Explored)$\leftarrow \{u\}$
3: $\hat{g}[u] \leftarrow 0$
4: while $F$ is not empty do
5:     $u \leftarrow F$.pop()
6:     if GoalTest(u) then
7:         return path(u)
8:     $E$.add($u$)
9:     for all children $v$ of $u$ do
10:        if $v$ not in $E$ then
11:            if $v$ in $F$ then
12:                $\hat{g}[v] = min(\hat{g}[v], \hat{g}[u] + c(u, v))$
13:            else
14:                $F$.push($v$)
15:                $\hat{g}[v] = \hat{g}[u] + c(u, v)$
16: return *Failure*
```

A few definitions before we analyze the algorithm (from the lecture 2 slides) :

Completeness

Defintion : An algorithm is **complete** if *it will find a solution when one exists* and *correctly report failure when it does not.*

Optimality

Definition : An algorithm is **optimal** if it finds a solution with the lowest path cost among all solutions (ie path cost optimal)

Performance of an algorithm

Intuition : The computational resources used by the algorithm.

UCS (Uniform Cost Search) details

Completeness: UCS algorithm is not complete. Let us consider the case of infinite number of zero weight edges deviating away from goal, UCS algorithm will not be able to find the path to goal. On the contrary, if every edge cost $\geq \epsilon$ (small small constant), then even if there are infinite such edges deviating away from goal, there summation will eventually be greater than edge cost between start and goal, and algorithm will find a path to goal.

Optimality: UCS algorithm is optimal. The main reason behind this, whenever we are performing *goaltest* on a particular node, we are always sure that we have found the optimal path to that node.

■

Time and Space Complexity : $O(b^{1+d})$

UCS is guided by path cost, assume every action costs at least some small constant ϵ and the optimal cost to reach goal is C, The worst case time and space complexity is $O(b^{1+\frac{C^*}{\epsilon}})$. 1 is added because UCS examines all the nodes at the goal's depth to see if one has a lower cost instead of stopping as soon as generating a goal. When all steps are equal, the time and space complexity is just $O(b^{1+d})$.

Note before the proof for optimality : Please pay special attention to the notation used. It is a bit different to what Prof. Daren has taught. The essential concepts are the same.

UCS Proof for optimality :

Claim 2 *When we pop u from F, we have found its optimal path.*

Proof: Let us use the following notations:

- $g(u)$ = minimum path cost from start to u.
- $\hat{g}(u)$ = estimated minimum path cost from start to u by UCS so far.
- $\hat{g}_{pop}(u)$ = estimated minimum path cost from start to u when u is popped from F.

Let the optimal path from start to goal u be:

$$S_0, S_1, S_2, \ldots, S_k, u$$

We will proof the claim using induction.

Base case: $\hat{g}_{pop}(S_0) = g(S_0) = 0$ is trivial.
Induction Hypothesis: $\forall_{i\in\{0,\ldots,k\}}\ \hat{g}_{pop}(S_i) = g(S_i)$.
Induction Step: As we know by definition:

$$g(S_0) \le g(S_1) \le \ldots g(S_k) \le g(u) \tag{1}$$

$$\hat{g}_{pop}(u) \ge g(u) \tag{2}$$

Therefore, by equation 1 and 2, $\hat{g}_{pop}(u) \ge g(u) \ge g(S_k)$
Since we have $g(u) \ge g(S_k)$, it is guaranteed that S_k is popped before u.
When we popped $S_k, \hat{g}(u)$ is computed as follows:

$$\begin{aligned}
\hat{g}(u) &= min(\hat{g}(u), \hat{g}(S_k) + C(S_k, u)) \\
\hat{g}(u) &\le \hat{g}(S_k) + C(S_k, u) \\
&\le g(S_k) + C(S_k, u) && \text{From induction hypothesis} \\
&\le g(u)
\end{aligned}$$

The estimate of u is always smaller than or equal to the value of actual minimum path cost to u along the path $\{S_0, S_1, \cdots, S_k, u\}$.

$$\hat{g}_{pop}(u) \le \hat{g}(u) \le g(u) \tag{3}$$

Thus, by equation 2 and 3

$$\hat{g}_{pop}(u) = g(u) \tag{4}$$

■

A* Search

3.1 A* Algorithm

The key idea of A* algorithm comes from Uniform Cost Search. A* is same as UCS except that it has extra knowledge that can guide the search, that is called as heuristic $h(n)$. $h(n)$ is an estimate of the cost of the shortest path from node n to a goal node, and we have to compute $h(n)$ also. For a time being, let us assume that we know $h(n)$ for all the nodes, later we will discuss in detail about how to compute such heuristics. A^* evaluates nodes using an estimated cost of the optimal solution, therefore

$$\hat{f}(n) = \hat{g}(n) + h(n)$$

Algorithm 6 A* Algorithm: FindPathToGoal(u)

```
1: $F$(Frontier)← PriorityQueue($u$)                ▷ it should be implement with $\hat{f}$ minimum
2: $E$(Explored)← $\{u\}$
3: $\hat{g}[u] \leftarrow 0$
4: while $F$ is not empty do
5:     $u \leftarrow F$.pop()
6:     if GoalTest(u) then
7:         return path(u)
8:     $E$.add($u$)
9:     for all children $v$ of $u$ do
10:        if $v$ not in $E$ then
11:            if $v$ in $F$ then
12:                $\hat{g}[v] = min(\hat{g}[v], \hat{g}[u] + c(u,v))$
13:                $\hat{f}[v] = h[v] + \hat{g}[v]$
14:            else
15:                $F$.push($v$)
16:                $\hat{g}[v] = \hat{g}[u] + c(u,v)$
17:                $\hat{f}[v] = h[v] + \hat{g}[v]$
18: return *Failure*
```

A* Search Details

While in UCS, the frontier priority queue is implemented with $\hat{g}$; in A^* Search, frontier priority queue should be implemented with $\hat{f}$, and $\hat{g}$. It is still used to keep track of the minimum path cost of reaching n discovered so far. A^* is a mix of lowest-cost-first and best-first search, Algorithm 6 describes the A* algorithm.

Let us discuss A^* algorithm in detail.

Completeness A^* is complete, and it will always find a solution if there is one, because the frontier always contains the initial part of a path to a goal, before that goal is selected.

As we have already discussed that UCS is optimal. The main reason for UCS being optimal was along the every possible optimal path $S_0, S_1, \dots, S_k, S_u$, we know that before S_i is popped, $S_1, \dots, S_{i-1}$ are already explored. To ensure this, UCS maintains $\hat{g}_{pop}(S_0) \leq \hat{g}_{pop}(S_1), \leq, \dots, \hat{g}_{pop}(S_k) \leq \hat{g}_{pop}(S_u)$, by ensuring $\hat{g}_{pop}(S_i) = g(S_i)$. We want to use the same proof to show that A^* is optimal.

Therefore in order to make A^* optimal, we would like to ensure following two properties:

Please turn to the next page for the properties and how we approach the proof for optimality of A* search

Property 1

$$\hat{f}_{pop}(S_0) \leq \hat{f}_{pop}(S_1), \leq, \ldots, \hat{f}_{pop}(S_k) \leq \hat{f}_{pop}(S_u)$$

.

Property 2

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

Notice that If $f(S_0) \leq f(S_1) \leq \ldots \leq f(S_k) \leq f(u)$ is True, and property 2 holds then property 1 would hold.

We know $\forall_{i \in \{1,\ldots,k\}}$ $f(S_i) = g(S_i) + h(S_i)$, and let us assume $f(S_0) \leq f(S_1) \leq \ldots \leq f(S_k) \leq f(u)$.

$\forall_{i \in \{1,\ldots,k\}}$, we have:

$$\begin{aligned} f(S_i) &\leq f(S_{i+1}) \\ g(S_i) + h(S_i) &\leq g(S_{i+1}) + h(S_{i+1}) \\ h(S_i) &\leq c(S_i, S_{i+1}) + h(S_{i+1}) \end{aligned}$$

The property $h(S_i) \leq c(S_i, S_{i+1}) + h(S_{i+1})$ is known as **consistency**.

$$\begin{aligned} &h(S_i) \leq c(S_i, S_{i+1}) + h(S_{i+1}) \\ &h(S_i) \leq c(S_i, S_{i+1}) + c(S_{i+1}, S_{i+2}) + h(S_{i+2}) \\ &\quad .. \\ &\quad .. \\ &h(S_i) \leq c(S_i, S_{i+1}) + c(S_{i+1}, S_{i+2}) + \ldots + c(S_k, S_u) + h(S_u) \end{aligned}$$

And, as we know, $h(S_u) = 0$,

$$\begin{aligned} &h(S_i) \leq c(S_{i+1}, S_i) + c(S_{i+1}, S_{i+2}) + \ldots + c(S_k, S_u) \\ &h(S_i) \leq OPT(S_i) \end{aligned}$$

where $OPT(S_i)$ is optimal path cost from S_i to goal u.

The property $h(S_i) \leq OPT(S_i)$ is known as **admissibility**. Therefore, consistency implies admissibility.

Claim 3 *If h is consistent, then A^* with graph search is optimal.*

Proof: Let Frontier priority queue is be implemented with $\hat{f}(n)$, and $\hat{g}(n)$, it is used to keep track of the minimum path cost of reaching n discovered so far. $f(n), g(n)$ be the optimal cost, and $f_{pop}(n), g_{pop}(n)$ represents the value of f and g for node n, when n is popped.

Given h is consistent, therefore:

$$h(S_i) \leq c(S_i, S_{i+1}) + h(S_{i+1}) \tag{5}$$

and, as discussed above, consistency implies admissibility, hence:

$$h(S_i) \leq OPT \tag{6}$$

Where OPT is optimal value of cost of S_i to goal. Also as per algorithm, for any path to goal u, we have:

$$\forall_{i \in \{1,\ldots,u\}}\ f(S_i) = g(S_i) + h(S_i) \tag{7}$$

To prove A^* is optimal, i.e $\hat{f}_{pop}(S_u) = f(S_u)$, where $\hat{f}_{pop}(S_u)$ is estimated cost when S_u is popped, and $f(S_u)$ is minimum cost to reach goal u.
By definition, we know:

$$\hat{f}_{pop}(S_{k+1}) \geq f(S_{k+1}) \tag{8}$$

We will proof $\hat{f}_{pop}(S_u) = f(S_u)$ by induction. Let us assume we have a path Π to reach goal u.

$$\Pi = S_1, S_2, \ldots, S_k, S_{k+1}, \ldots, S_u$$

Base Case: $\hat{f}_{pop}(S_0) = f(S_0) = h(S_0)$.

Induction Hypothesis: $\forall_{i \in 1,\ldots,k}\ \hat{f}_{pop}(S_i) = f(S_i)$.

Induction Step: Estimated optimal cost for S_{k+1} is:

$$\begin{aligned}
&\hat{f}_{pop}(S_{k+1}) = \hat{g}_{pop}(S_{k+1}) + h(S_{k+1}) \\
&\hat{f}_{pop}(S_{k+1}) = g(S_k) + c(S_k, S_{k+1}) + h(S_{k+1}) \\
&\qquad\qquad \text{from: } \hat{g}_{pop}(S_{k+1}) = g(S_{k+1}) \\
&\hat{f}_{pop}(S_{k+1}) \geq g(S_k) + h(S_k) && \text{from equation 5} \\
&\hat{f}_{pop}(S_{k+1}) \geq \hat{f}_{S_k} && \text{from equation 7}
\end{aligned}$$

Hence, S_{k+1} explored after S_k:

$$\begin{aligned}
&\hat{f}_{pop}(S_{k+1}) = min(\hat{f}(S_{k+1}), \hat{g}_{pop}(S_k) + c(S_k, S_{k+1}) + h(S_{k+1})) \\
&\hat{f}_{pop}(S_{k+1}) \leq \hat{g}_{pop}(S_k) + c(S_k, S_{k+1}) + h(S_{k+1}) \\
&\hat{f}_{pop}(S_{k+1}) \leq g(S_k) + c(S_k, S_{k+1}) + h(S_{k+1}) \\
&\qquad\qquad \text{From UCS proof } \forall_{i \in 1,\ldots,u}\ (g_{pop}(S_i) = g(S_i)) \\
&\hat{f}_{pop}(S_{k+1}) \leq g(S_{k+1}) + h(S_{k+1}) \\
&\hat{f}_{pop}(S_{k+1}) \leq f(S_{k+1}) && \text{From Equation 5}
\end{aligned}$$

Hence, from above equation and equation 8

$$\hat{f}_{pop}(S_{k+1}) = f(S_{k+1})$$

■

A* for Tree Search Observe that admissibility is a weaker property than consistency, and with tree search, it is possible to design an optimal algorithm that can only uses admissibility property. In the case of graph search, a node is popped from fronter only once, so when it is explored, the optimal path must have been found, but with tree search that is not the case. A node can be popped multiple times, and it allows us to use relaxed conditions.