CS3243 **AY20/21 Sem1**

Lecture 3: August 26

Lecturer: MEEL, Kuldeep S. *Scribes: Song Qifeng*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

3.1 Uninformed search

Definition 3.1 *Uninformed search*

Uninformed search strategies have no additional information other than that provided in the problem definition. Uninformed strategies can do generate successors and testing whether a state is goal.

3.1.1 Uniform cost search

Uniform cost search is a modification based on BFS. It stores states in priority queue.

3.1.1.1 Motivation for developing UCS

While BFS meets completeness, it does not guarantee optimality because it ends as soon as the goal state is popped from frontier, however, there may be shorter paths not explored yet. To fix this issue, we develop BFS to ensure that whenever goal state is popped, we have found the optimal path.

3.1.1.2 Psuedocode of UCS

```
FindPathToGoal(u):
    F ← PriorityQueue(u)
    E ← {u}
    ĝ[u] = 0 // minimum cost discovered so far to reach u

    while F is not empty
        u ← F.pop()
        if GoalTest(u)
            return path(u)
        E.add(u)

        for all children v of u:
            if v not in E
                if v in F
                    ĝ[v] = min(ĝ[v], ĝ[u] + c(u, v))
```

else
$\hat{g}[v] = \hat{g}[u] + c(u, v)$
$F.push(v)$

return $Failure$

How is this different from Dijkstra?

Dijkstra will initiate every $\hat{g}[v_i] = 0$, this will cause trouble when the graph is infinite.

3.1.1.3 Analysis

4 properties to be analyzed:

- Completeness
- Optimality
- Time complexity
- Space complexity

Completeness
UCS is not complete there is an infinite sequence of 0-cost edges, in this case, UCS will dive into this sequence and never find the goal.
However, provided that every edge $e_i > \epsilon$ and $\epsilon > 0$ in the graph, UCS will be complete.

Optimality
UCS is optimal.

Proof:

claim: $\hat{g}_{pop}[u] = g[u]$

Let the optimal path to goal be $S_0, S_1, \ldots, S_n, goal$.

It is obvious that

- $g[S_i] + c(S_i, S_{i+1}) = g[S_{i+1}]$
- derived from the above statement, $g[S_0] \leq g[S_1] \leq \ldots \leq g[S_n] \leq g[goal]$
- $\hat{g}_{pop}[S_i] \geq g[S_i]$, $0 \leq i \leq n$

base: $\hat{g}_{pop}[S_0] = g[S_0] = 0$

mathematical induction:

$\forall k < n, \forall\, 0 \leq i < k$, $\hat{g}_{pop}[S_i] = g[S_i]$.

From the psuedocode, $\hat{g}_{pop}[S_{k+1}] \leq \hat{g}_{pop}[S_k] + c(S_k, S_{k+1}) = g[S_k] + c(S_k, S_{k+1})$.

Because this sequence is the optimal path, therefore $g[S_{k+1}] = g[S_k] + c(S_k, S_{k+1})$.

Therefore we know: $\hat{g}_{pop}[S_{k+1} \leq g[S_{k+1}]]$.

Because $\hat{g}_{pop}[S_{k+1}] \geq g[S_{k+1}]$, therefore $\hat{g}_{pop}[S_{k+1}] = g[S_{k+1}]$.

■

Time complexity

$\forall$ edge e, $e \geq \epsilon > 0$, let branch factor be b, let the optimal path length be L

$\rightarrow$ depth d= L/ϵ

$\rightarrow$ time complexity= $O(b^{d+1}) = O(b^{(L/\epsilon)+1})$

Space complexity

space complexity= $O(b^{(L/\epsilon)+1})$

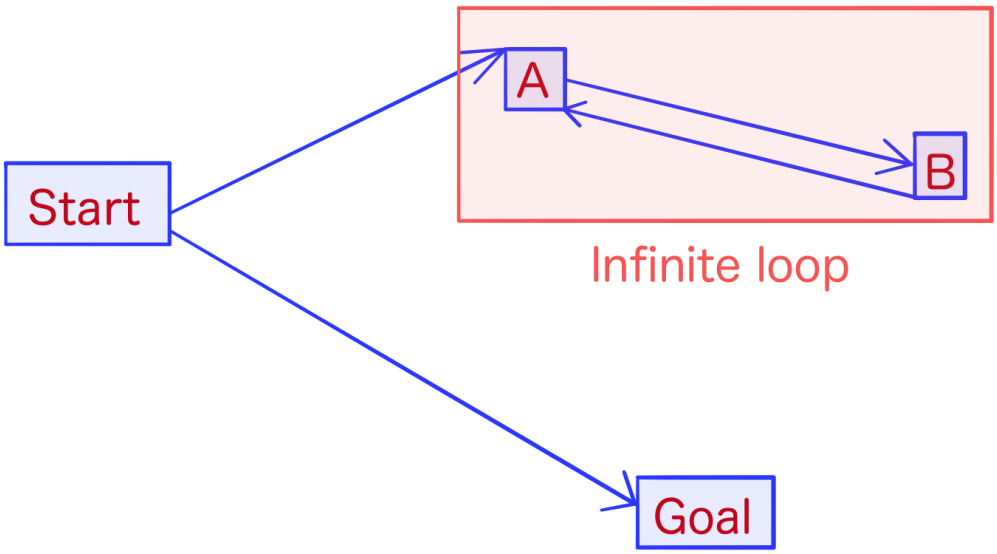
3.1.2 Depth first search

DFS always pops the deepest node in the frontier. DFS is very similar to BFS, the only difference is that DFS stores frontier nodes in a stack instead of a queue. There are two versions of DFS- tree version and graph version.

3.1.2.1 Analysis

Completeness

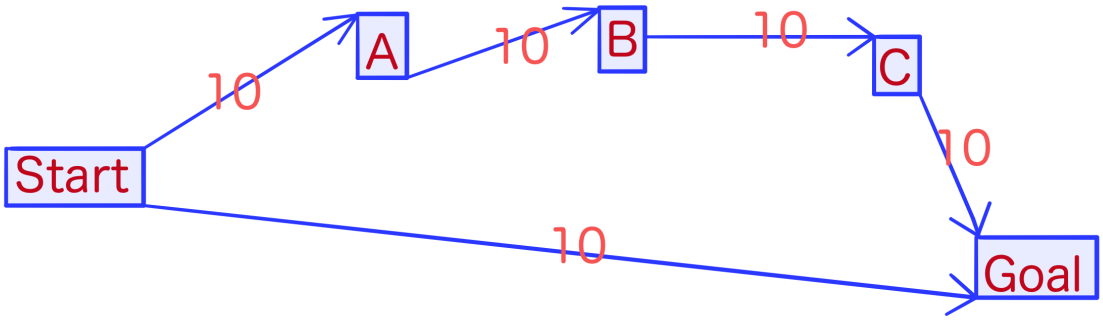
For the tree search version, DFS can end up in infinite loop. For example, in the graph shown in the figure below, DFS will be stuck in the infinite loop A $\rightarrow B \rightarrow A \rightarrow \ldots$



For the graph version, however, provided that the graph does not have infinite depth, will be complete.

Optimality

Both versions of DFS are nonoptimal. The figure below is an example of DFS not returning the optimal path, DFS returns the nonoptimal path $Start \rightarrow A \rightarrow B \rightarrow C \rightarrow Goal$.



Time complexity

Let m be the maximum depth, b be the branch factor.

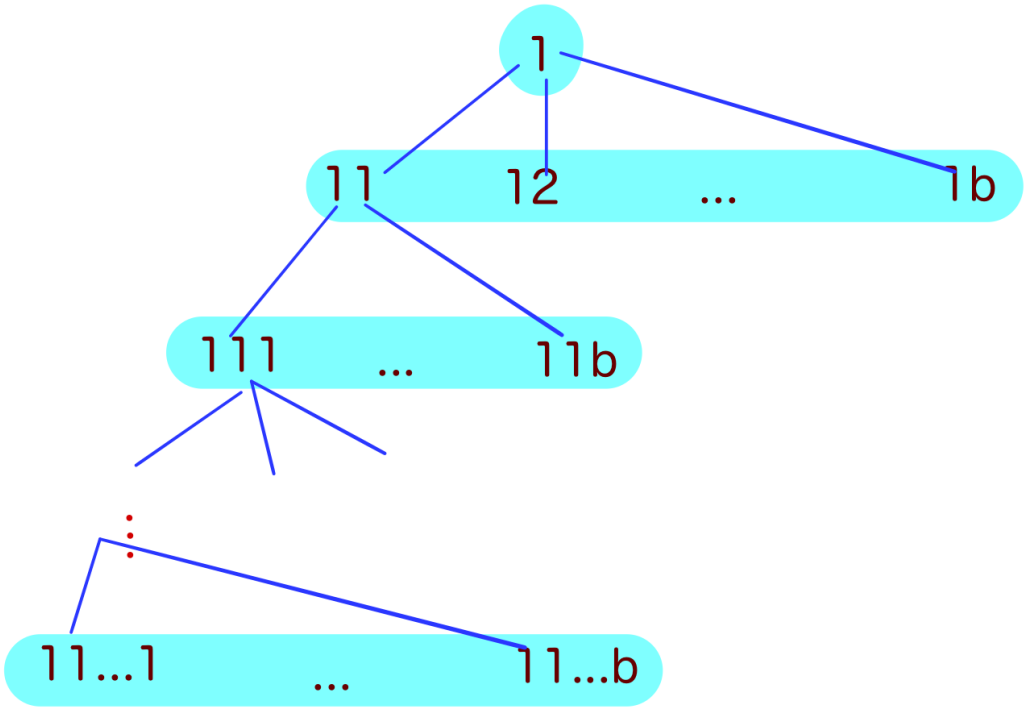
Time complexity= $O(b^m)$

Space complexity

Let m be the maximum depth, b be the branch factor.

Space complexity= $O(b \times m)$

Here's a figure to show why DFS has such small space complexity.



3.2 Informed search

Definition 3.2 *Informed search*

Informed search strategies use problem-specific knowledge in addition to problem definition to find solutions more efficiently.

3.2.1 A* search

3.2.1.1 Motivation for developing A* search

Among all uninformed searching strategies we have met so far, UCS is the only strategy to meet both completeness and optimality, however, when an agent applying UCS, it will inevitably dive into wrong branches. Can we avoid choosing the wrong branch to search, or at least reduce the occurrence of such situations?

Suppose we have a black box function $h(u)$ that estimates the cost from u to goal, then we can have another function $\hat{f}(u) = \hat{g}[u] + h(u)$ that estimates the cost of the path $start \to \ldots \to u \to \ldots \to goal$. Then we can sort the priority queue by $\hat{f}[u]$ instead of $\hat{g}[u]$, which will hopefully reduce the chance of choosing incorrect branches.

3.2.1.2 Psuedocode of A* search

```
FindPathToGoal(u):
    F ← PriorityQueue(u)
    E ← {u}
    ĝ[u] = 0 // minimum cost discovered so far to reach u

    while F is not empty
        u ← F.pop()
        if GoalTest(u)
            return path(u)
        E.add(u)

        for all children v of u:
            if v not in E
                if v in F
                    ĝ[v] = min(ĝ[v], ĝ[u] + c(u, v))
                    f̂[v] = ĝ[v] + h[v]
                else
                    ĝ[v] = ĝ[u] + c(u, v)
                    f̂[v] = ĝ[v] + h[v]
                    F.push(v)

    return Failure
```

3.2.1.3 Conditions of h() to ensure A* search's optimality

Recall that UCS's optimality was guaranteed by the following 2 properties:

- $\hat{g}_{pop}[S_0] \leq \hat{g}_{pop}[S_1] \leq \ldots \leq \hat{g}_{pop}[S_n] \to$ **property 1**
- $\forall S_i \in graph,\ \hat{g}_{pop}[S_i] = g[S_i] \to$ **property 2**

Also note that A* search simply replaces $\hat{g}[u]$ in UCS with $\hat{f}[u]$. If $\hat{f}[u]$ also holds the following two properties, then A* search's optimality is guaranteed.

- $\hat{f}_{pop}[S_0] \leq \hat{f}_{pop}[S_1] \leq \ldots \leq \hat{f}_{pop}[S_n] \to$ **property 1**
- $\forall S_i \in graph,\ \hat{f}_{pop}[S_i] = f[S_i] \to$ **property 2**

Moreover, there is a third property such that if **property 2** and **property 3** both hold, **property 1** will also hold.

- $f[S_0] \leq f[S_1] \leq \ldots \leq f[S_n] \rightarrow$ **property 3**

property 3 can also be written as: $\forall i \in \mathbb{N}$, $f[S_i] \leq f[S_{i+1}]$

$\rightarrow g[S_i] + h[S_i] \leq g[S_{i+1}] + h[S_{i+1}]$

$\rightarrow g[S_i] + h[S_i] \leq g[S_i] + c(S_i, S_{i+1}) + h[S_{i+1}]$

$\rightarrow h[S_i] \leq c(S_i, S_{i+1}) + h[S_{i+1}]$

This is called the *triangle inequality*, also known as *consistency*. From *consistency*, we further derive:

$\rightarrow h[S_i] \leq c(S_i, S_{i+1}) + h[S_{i+1}]$

$\rightarrow h[S_i] \leq c(S_i, S_{i+1}) + c(S_{i+1}, S_{i+2}) + h[S_{i+2}]$

$\ldots$

$\rightarrow h[S_i] \leq c(S_i, S_{i+1}) + c(S_{i+1}, S_{i+2}) + \ldots + c(S_n, goal) + h[goal]$, $h[goal] = 0$

$\rightarrow h[S_i] \leq c(S_i, S_{i+1}) + c(S_{i+1}, S_{i+2}) + \ldots + c(S_n, goal) =$ real coat of the path

Therefore we notice that $h[S_i]$ will always be optimistic and we call it *admissibility*. $h[S_i]$ is the lower bound for the path cost of $start \rightarrow \ldots \rightarrow S_i \rightarrow \ldots \rightarrow goal$.

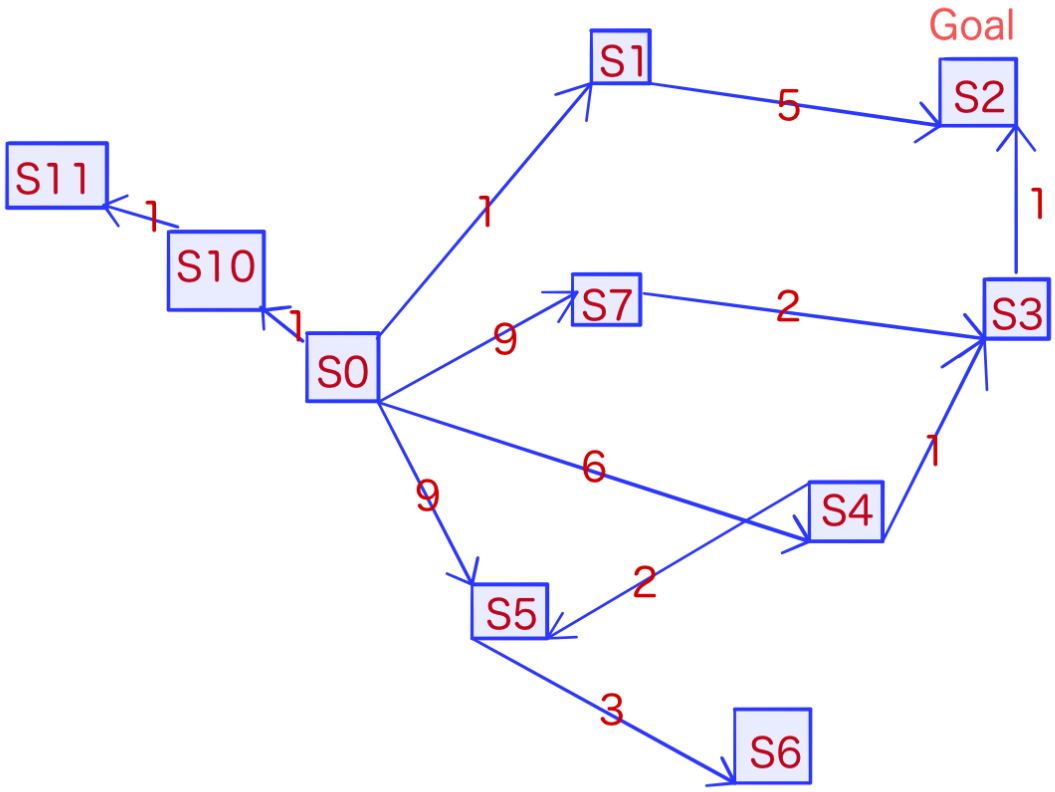
Now we have obtained the two property of $h[]$:

- Consistency

- Admissibility

3.2.1.4 A common mistake: is $f[S_i]$ non-decreasing on any path?

No. The graph below is a counter example.



For the path $S_0 \to S_1 \to S_2$, their values of f are 8, 10, and 8.

The textbook went wrong in this step:

$g[n'] + h[n'] = g[n] + c(n, n') + h[n'$, where $g[n']$ may not necessarily equal to $g[n] + c(n, n')$ because there may be another shortcut to n' without going through n.

$g[n'] + h[n'] = g[n] + c(n, n') + h[n'$ is guaranteed to hold true only on the optimal path. However, the textbook has generalized it to 'any path'.