Review: DFS and BFS

• Breadth-first search
  • Frontier is a queue: expand the shallowest node
  • **Complete**: always finds a solution, if one exists
  • **Optimal** (finds the best solution) *if all actions have the same cost.*
  • **Time complexity**: $O\{b^d\}$
  • **Space complexity**: $O\{b^d\}$.

• Depth-first search – utility depends on relationship between $m$ and $d$
  • Frontier is a stack: expand the deepest node
  • **Not complete** (might never find a solution, if $m$ is infinite)
  • **Not optimal** (returned solution is rarely the best one)
  • **Time complexity**: $O\{b^m\}$
  • **Space complexity**: $O\{bm\}$. 
Outline of today’s lecture

1. Uniform Cost Search (UCS): like BFS, but for actions that have different costs
   • **Complete**: always finds a solution, if one exists
   • **Optimal**: finds the best solution
   • **Time complexity** = # nodes that have cost < goal
   • **Space complexity** = # nodes that have cost < goal

2. Heuristics, e.g., Manhattan distance

3. Greedy Best-first search

4. A*: Like UCS but adds an estimate of the remaining path length
   • **Complete**: always finds a solution, if one exists
   • **Optimal**: finds the best solution
   • **Time complexity** = # nodes that have cost+heuristic < goal
   • **Space complexity** = # nodes that have cost+heuristic < goal
An example for which BFS is not optimal: Romania

BFS returns this path, because it requires only 3 actions. Cost = 450 km
An example for which BFS is not optimal: Romania

It would have been better to find this path!
Cost = 418 km
The solution: Uniform Cost Search

• Breadth-first search (BFS): Next node expanded is the one with the fewest required actions
  • Frontier is a queue
  • First node into the queue is the first one expanded (FIFO)

• Uniform cost search (UCS): Next node expanded is the one with the lowest accumulated path cost
  • Frontier is a *priority queue*
  • *Lowest-cost node* is the first one expanded
Example of UCS: Romania
Example of UCS: Romania

Zerind: 75, Timisoara: 118, Sibiu: 140
Example of UCS: Romania

Timisoara: 118, Sibiu: 140, Oradea: 146
Example of UCS: Romania

Sibiu:140, Oradea:146, Lugoj:239
Example of UCS: Romania

Oradea: 146, **Rimnicu Vilcea: 220**, Lugoj: 239, **Fagaras: 239**
Example of UCS: Romania

Rimnicu Vilcea: 220, Lugoj: 239, Fagaras: 239
Example of UCS: Romania

Lugoj: 239, Fagaras: 239, **Pitesti: 317, Craiova: 366**
Example of UCS: Romania

Fagaras: 239, **Mehadia: 309**, Pitesti: 317, Craiova: 366
Example of UCS: Romania

Mehadia: 309, Pitesti: 317, Craiova: 366, **Bucharest: 450**
Example of UCS: Romania

- Pitesti: 317
- Craiova: 366
- Dobreta: 384
- Bucharest: 450
Example of UCS: Romania

Craiova: 366, Dobreta: 384, **Bucharest: 418**
Example of UCS: Romania

Dobreta: 384, Bucharest: 418
Example of UCS: Romania

Bucharest: 418
GOAL!!!! GOL!!!!!

Outline of today’s lecture

1. Uniform Cost Search (UCS): like BFS, but for actions that have different costs
   • **Complete**: always finds a solution, if one exists
   • **Optimal**: finds the best solution
   • **Time complexity** = # nodes that have cost < goal
   • **Space complexity** = # nodes that have cost < goal

2. Heuristics, e.g., Manhattan distance
3. Greedy Best-first search
4. A*: Like UCS but adds an estimate of the remaining path length
   • **Complete**: always finds a solution, if one exists
   • **Optimal**: finds the best solution
   • **Time complexity** = # nodes that have cost+heuristic < goal
   • **Space complexity** = # nodes that have cost+heuristic < goal
Heuristics main idea

Instead of choosing the node with the smallest total cost so far (UCS), why not choose the node that’s CLOSEST TO GOAL, and expand that one first?
Why not choose the node CLOSEST TO GOAL?

• Answer: because we don’t know which node that is!!

• Example: which of these two is closest to goal?
We don’t know which state is closest to goal

• Finding the shortest path is the whole point of the search
• If we already knew which state was closest to goal, there would be no reason to do the search
• Figuring out which one is closest, in general, is a complexity $O(b^d)$ problem.
Search heuristics: estimates of distance-to-goal

- Often, even if we don’t know the distance to the goal, we can estimate it.
- This estimate is called a heuristic.
- A heuristic is useful if:
  1. **Accurate**: \( h(n) \approx d(n) \), where \( h(n) \) is the heuristic estimate, and \( d(n) \) is the true distance to the goal
  2. **Cheap**: It can be computed in complexity less than \( O\{b^d\} \)
Example heuristic: Manhattan distance

If there were no walls in the maze, then the number of steps from position \((x_n, y_n)\) to the goal position \((x_G, y_G)\) would be

\[ h(n) = |x_n - x_G| + |y_n - y_G| \]
Outline of today’s lecture

1. Uniform Cost Search (UCS): like BFS, but for actions that have different costs
   - **Complete**: always finds a solution, if one exists
   - **Optimal**: finds the best solution
   - **Time complexity** = # nodes that have cost < goal
   - **Space complexity** = # nodes that have cost < goal

2. Heuristics, e.g., Manhattan distance

3. Greedy Best-first search

4. A*: Like UCS but adds an estimate of the remaining path length
   - **Complete**: always finds a solution, if one exists
   - **Optimal**: finds the best solution
   - **Time complexity** = # nodes that have cost+heuristic < goal
   - **Space complexity** = # nodes that have cost+heuristic < goal
Greedy Best-First Search

Instead of choosing the node with the smallest total cost so far (UCS), why not choose the node whose HEURISTIC ESTIMATE indicates that it might be CLOSEST TO GOAL?
Greedy Search Example

According to the Manhattan distance heuristic, these two nodes are equally far from the goal, so we have to choose one at random.
Greedy Search Example

If our random choice goes badly, we might end up very far from the goal.

★ = states in the explored set

○ = states on the frontier
The problem with Greedy Search

Having gone down a bad path, it’s very hard to recover, because now, the frontier node closest to goal (according to the Manhattan distance heuristic) is this one:
The problem with Greedy Search

That’s not a useful path...
The problem with Greedy Search

Neither is that one...
What went wrong?
Outline of today’s lecture

1. Uniform Cost Search (UCS): like BFS, but for actions that have different costs
   - **Complete**: always finds a solution, if one exists
   - **Optimal**: finds the best solution
   - **Time complexity** = # nodes that have cost < goal
   - **Space complexity** = # nodes that have cost < goal

2. Heuristics, e.g., Manhattan distance

3. Greedy Best-first search

4. A*: Like UCS but adds an estimate of the remaining path length
   - **Complete**: always finds a solution, if one exists
   - **Optimal**: finds the best solution
   - **Time complexity** = # nodes that have cost+heuristic < goal
   - **Space complexity** = # nodes that have cost+heuristic < goal
The problem with Greedy Search

Among nodes on the frontier, this one seems closest to goal (smallest $h(n)$, where $h(n) \leq d(n)$).

But it’s also farthest from the start. Let’s say $g(n) =$ total path cost so far.

So the total distance from start to goal, going through node $n$, is

$$c(n) = g(n) + d(n) \geq g(n) + h(n)$$
The problem with Greedy Search

Of these three nodes, this one has the smallest $g(n) + h(n)$

$$(g(n) + h(n) = 4 + 28 = 32)$$

So if we want to find the lowest-cost path, then it would be better to try that node, instead of this one, which has

$$g(n) + h(n) = 21 + 14 = 35$$
A* notation

- \( c(n) = \text{cost} \) of the total path (START,...,n,...,GOAL).
- \( d(n) = \text{distance} \) of the remaining partial path (n,...,GOAL).
- \( g(n) = \text{gone-already} \) on the path so far, (START,...,n).
- \( h(n) = \text{heuristic} \), \( h(n) \leq d(n) \).

\[
c(n) = g(n) + d(n) \geq g(n) + h(n)
\]
A* Search

• Idea: avoid expanding paths that are already expensive
• The evaluation function \( f(n) \) is the estimated total cost of the path through node \( n \) to the goal:

\[
f(n) = g(n) + h(n)
\]

- \( g(n) \): cost so far to reach \( n \) (path cost)
- \( h(n) \): estimated cost from \( n \) to goal (heuristic)

• This is called A* search if and only if the heuristic, \( h(n) \), is admissible. The word “admissible” just means that \( h(n) \leq d(n) \), and therefore, \( f(n) \leq c(n) \).
Admissible heuristic

• Suppose we’ve found one path to $G$; the path goes through node $m$. Since we’ve calculated the whole path, we know its total path cost to be $c(m)$.

• For every other node, $n$, we don’t know $c(n)$, but we know $f(n) = g(n) + h(n)$, and we know that

$$c(n) \geq f(n)$$

• Therefore we know that

$$\text{IF } f(n) \geq c(m) \text{ THEN } c(n) \geq c(m)$$

• So if $f(n) \geq c(m)$ for every node $n$ that’s still in the frontier, then we know that $m$ is the best path.
**A* Search**

**Definition: A* SEARCH**

- If $h(n)$ is **admissible** ($d(n) \geq h(n)$), and
- if the frontier is a priority queue sorted according to $g(n) + h(n)$, then
- the FIRST path to goal uncovered by the tree search, path $m$, is guaranteed to be the **SHORTEST** path to goal

\[
(h(n) + g(n) \geq c(m) \text{ for every node } n \text{ that is not on path } m)
\]
Example of A*: Romania

Suppose we don’t know the distance from Sibiu to Bucharest on highways, but we DO know the distance “as the crow flies.”

\[ h(n) = \text{Euclidean distance (as the crow flies)} \]

- Sibiu: \( h(n) = 260\text{km} \)
- Timisoara: \( h(n) = 410\text{km} \)
- Zerind: \( h(n) = 422\text{km} \)
Romania using UCS

Zerind: 75, Timisoara: 118, Sibiu: 140

Pick this one first?
Romania using A*

Sibiu: 140 + 260 = 400, Zerind: 75 + 422 = 495, Timisoara: 118 + 410 = 528

No, pick this one first!!!
BFS vs. A* Search Example
The heuristic $h(n) =$ Euclidean distance favors nodes on the main diagonal. Those nodes all have the same $g(n) + h(n)$, so A* evaluates them first.

Outline of today’s lecture

1. Uniform Cost Search (UCS): like BFS, but for actions that have different costs
   - **Complete**: always finds a solution, if one exists
   - **Optimal**: finds the best solution
   - **Time complexity** = # nodes that have cost < goal
   - **Space complexity** = # nodes that have cost < goal

2. Heuristics, e.g., Manhattan distance

3. Greedy Best-first search

4. A*: Like UCS but adds a lower bound of the remaining path length
   - **Complete**: always finds a solution, if one exists
   - **Optimal**: finds the best solution
   - **Time complexity** = # nodes that have cost+heuristic < goal
   - **Space complexity** = # nodes that have cost+heuristic < goal