CSCI3230 (ESTR3108) Fundamentals of Artificial Intelligence

Lecture 10. Informed Search

Prof. Qi Dou

Email: qidou@cuhk.edu.hk
Office: Room 1014, 10/F, SHB

Dept. of Computer Science & Engineering
The Chinese University of Hong Kong

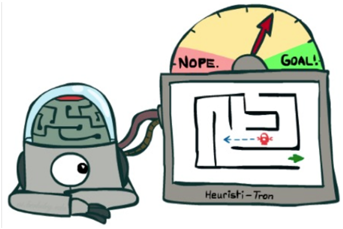
Outline

Part 1. Introduction

Informed search v.s. uninformed search

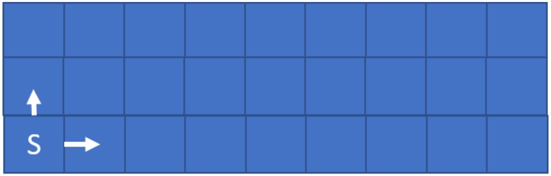
- By uninformed search strategies, we can find solutions to problems by systematically expanding new state and testing it against the goal.
- Unfortunately, these strategies would be inefficient in most cases.

- Informed search strategy uses problem-specific knowledge beyond the definition of the problem itself.
- Therefore, informed search can find solutions more efficiently than an uninformed strategy.



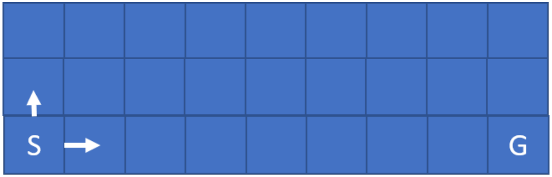
Informed search v.s. uninformed search - an example

- See the following example (uninformed search)
  - Assume that the initial state is S. Where should we go next?
  - Move Up or Move Right (the costs are the same)?
  - By uninformed search methods, we do not know the answer.



Informed search v.s. uninformed search - an example

- See the following example (informed search)
  - Assume the initial state is S. Where should we go next?
  - Assume we also know where is the goal state G.
  - Then, we know that it is probably better to move right, because the distance between current state and the goal state will be smaller.

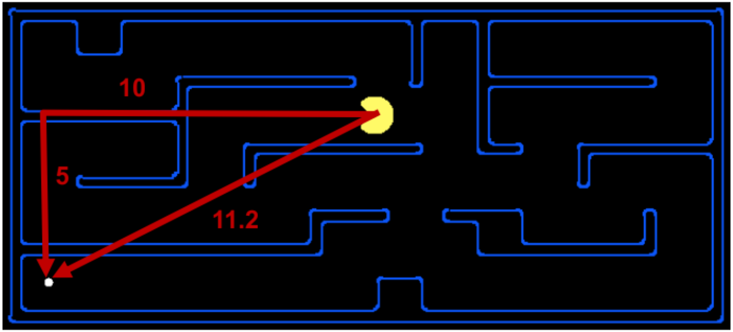


- Therefore, we may need a function that can estimate the cost from the current state to the goal state.

Search heuristics

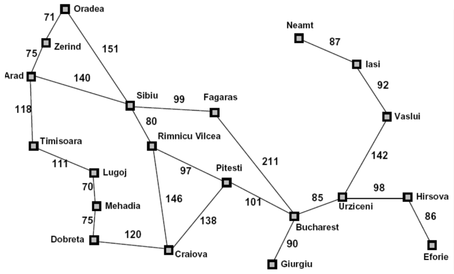
A heuristics is :

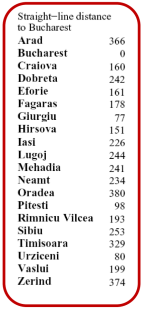
- A function that estimates how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance or Euclidean distance for path



Search heuristics

- Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm.
- $h(n) =$ estimated cost of the cheapest path from node n to goal
- If n is a goal node, then $h(n) = 0$
- For example, in the Romania map, one might estimate the cost of the cheapest path from Arad to Bucharest via the straight-line distance.





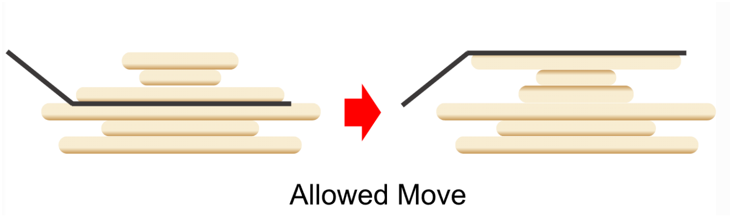
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Search heuristics: pancake sorting problem

Pancake sorting problem

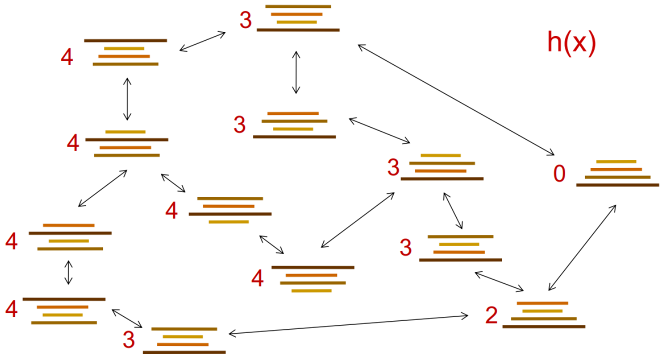
- It is the mathematical problem of sorting a disordered stack of pancakes in order of size.
- The smallest is on top, the largest is at bottom.
- A spatula can be inserted at any point in the stack and used to flip all pancakes above it.



Search heuristics: pancake sorting problem

Pancake sorting problem

- Assume that there are four pancakes: pancake-1, pancake-2, pancake-3, pancake-4 with size 10, 15, 25 and 35, respectively. (The larger the index is, the larger the pancake size is).
- Heuristic: the index of the largest pancake that is still out of place



Heuristic search

- Informed search algorithms (a.k.a. heuristic search) have information on the goal state which helps for more efficient searching.
- They have a heuristic function $h(n)$ that calculates cost estimates from current state n to the goal.
- They have an evaluation function $f(n)$ that returns a number purporting to describe the desirability of expanding the node
    - Greedy best-first search: minimize the estimated cost to reach the goal.
    - A* search: minimizing the total path cost.

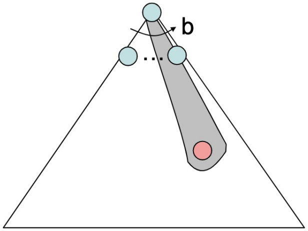
Part 2. Greedy best-first search

Greedy best-first search

- Greedy best-first search (a.k.a. greedy search) tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.
- It evaluates nodes by using just the heuristic function $f(n) = h(n)$.

Greedy best-first search

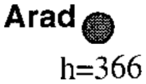
- Greedy best-first search (a.k.a. greedy search) tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.
- It evaluates nodes by using just the heuristic function $f(n) = h(n)$.
- Strategy: expand the node that you think is closest to a goal state.
- Heuristic: estimate of distance to nearest goal for each state.
- A common case: takes you straight to the goal.



Greedy best-first search - an example

Assume the initial state is city Arad, the goal is city Bucharest.

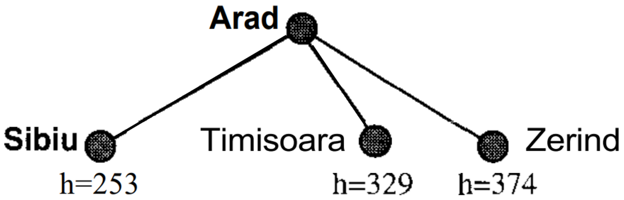
- The initial state is Arad with $h(\text{Arad}) = 366$



Greedy best-first search - an example

Assume the initial state is city Arad, the goal is city Bucharest.

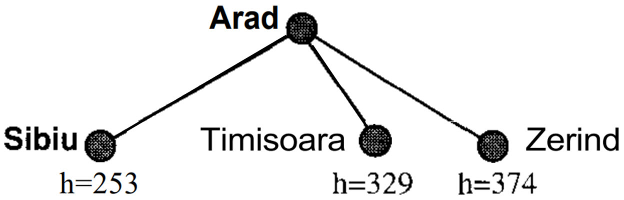
- The initial state is Arad with $h(\text{Arad}) = 366$
- Arad connects to three nodes, i.e., Sibiu, Timisoara and Zerind.



Greedy best-first search - an example

Assume the initial state is city Arad, the goal is city Bucharest.

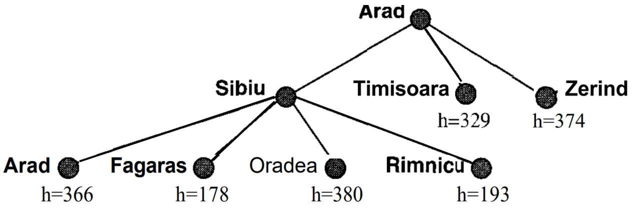
- The initial state is Arad with $h(\text{Arad}) = 366$
- Arad connects to three nodes, i.e., Sibiu, Timisoara and Zerind.
- The first node to be expanded from Arad is Sibiu, because $h(\text{Sibiu})$ is the smallest one.



Greedy best-first search - an example

Assume the initial state is city Arad, the goal is city Bucharest.

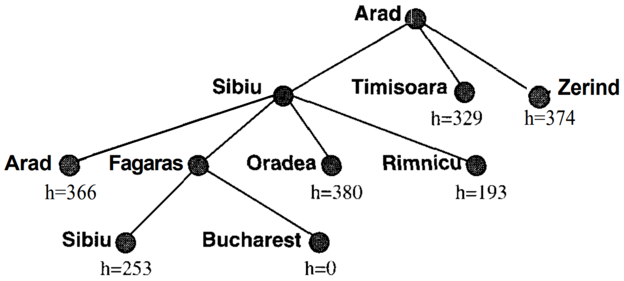
- The initial state is Arad with $h(\text{Arad}) = 366$
- Arad connects to three nodes, i.e., Sibiu, Timisoara and Zerind.
- The first node to be expanded from Arad is Sibiu, because $h(\text{Sibiu})$ is the smallest one.
- The next node to be expanded is Fagaras, because it is closer to the goal compared to Arad, Oradea, Rimnicu, Timisoara and Zerind.



Greedy best-first search - an example

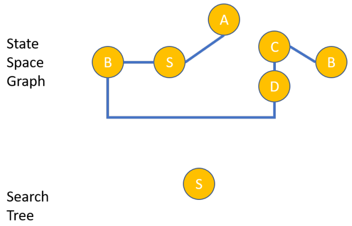
Assume the initial state is city Arad, the goal is city Bucharest.

- The initial state is Arad with $h(\text{Arad}) = 366$
- Arad connects to three nodes, i.e., Sibiu, Timisoara and Zerind.
- The first node to be expanded from Arad is Sibiu, because $h(\text{Sibiu})$ is the smallest one.
- The next node to be expanded is Fagaras, because it is closer to the goal compared to Arad, Oradea, Rimnicu, Timisoara and Zerind.
- Fagaras finally generates Bucharest, which is the goal.



Greedy best-first search analysis: completeness

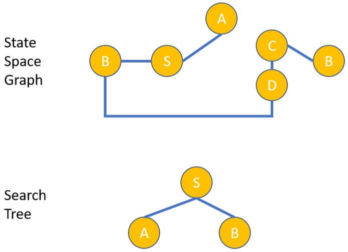
- Greedy search is not complete. It is susceptible to false starts.
- See the example, get from the initial node S to the goal node G.



Node n	h(n)
S	2
A	1
B	3
C	2
D	2
E	2

Greedy best-first search analysis: completeness

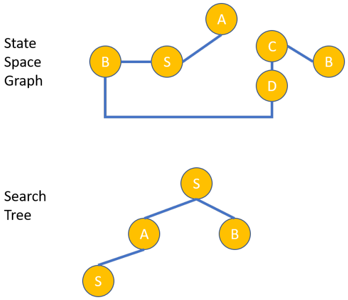
- Greedy search is not complete. It is susceptible to false starts.
- See the example, get from the initial node S to the goal node G.
- We first expand the node A, because $h(\text{A})$ is smaller than $h(\text{B})$



Node n	h(n)
S	2
A	1
B	3
C	2
D	2
E	2

Greedy best-first search analysis: completeness

- Greedy search is not complete. It is susceptible to false starts.
- See the example, get from the initial node S to the goal node G.
- We first expand the node A, because $h(\text{A})$ is smaller than $h(\text{B})$
- Then, we go back to S because $h(\text{S})$ is smaller than $h(\text{B})$.

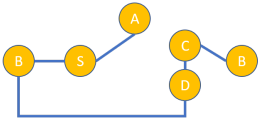


Node n	h(n)
S	2
A	1
B	3
C	2
D	2
E	2

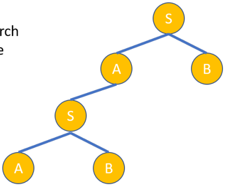
Greedy best-first search analysis: completeness

- Greedy search is not complete. It is susceptible to false starts.
- See the example, get from the initial node S to the goal node G.
- We first expand the node A, because $h(\text{A})$ is smaller than $h(\text{B})$
- Then, we go back to S because $h(\text{S})$ is smaller than $h(\text{B})$.
- So, it goes to an infinite path and never try other possibilities.

State
Space
Graph



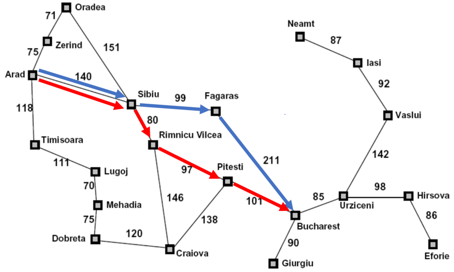
Search
Tree



Node n	h(n)
S	2
A	1
B	3
C	2
D	2
E	2

Greedy best-first search analysis: optimality

- Greedy search is not guaranteed to be optimal.
- For the given greedy search example, the solution is Arad → Sibiu → Fagaras → Bucharest
- However, it is not optimal, the path is 32 miles longer than the path Arad → Sibiu → Rimnicu Vilcea → Pitesti → Bucharest



- This shows why the algorithm is called "greedy" – at each step it tries to get as close to the goal as it can.

Greedy best-first search analysis: time & space complexity

- Greedy best-first search resembles depth-first search in the way it prefers to follow a single path all the way to the goal.
- Let $\boldsymbol{m}$ be the maximum depth of the search space, in the worst case,
- Time complexity is $O(\boldsymbol{b}^{\boldsymbol{m}})$
  - For example, $h(n) = 0$ for all nodes. We may visit all nodes.
- Space complexity is $O(\boldsymbol{b}^{\boldsymbol{m}})$
  - Because greedy search retains all nodes in memory, its space complexity is the same as its time complexity
  - For example, $h(n) = 0$ for all nodes. In this situation, the time complexity is $O(\boldsymbol{b}^{\boldsymbol{m}})$. Hence, the space complexity is also $O(\boldsymbol{b}^{\boldsymbol{m}})$.

Greedy best-first search analysis: time & space complexity

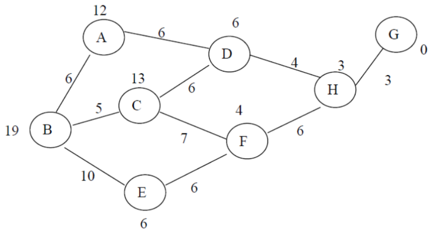
- Greedy best-first search resembles depth-first search in the way it prefers to follow a single path all the way to the goal.
- Let $\boldsymbol{m}$ be the maximum depth of the search space, in the worst case,
- Time complexity is $O(\boldsymbol{b}^{\boldsymbol{m}})$
  - For example, $h(n) = 0$ for all nodes. We may visit all nodes.
- Space complexity is $O(\boldsymbol{b}^{\boldsymbol{m}})$
  - Because greedy search retains all nodes in memory, its space complexity is the same as its time complexity
  - For example, $h(n) = 0$ for all nodes. In this situation, the time complexity is $O(\boldsymbol{b}^{\boldsymbol{m}})$. Hence, the space complexity is also $O(\boldsymbol{b}^{\boldsymbol{m}})$.

- Note that, with a good heuristic function, however, the complexity can be reduced substantially. The amount of the reduction depends on the particular problem and on the quality of the heuristic.

Quiz

The value at each node represents the heuristic cost from that node to the goal node (G). The value at each arcs represents the path cost between two nodes. If the initial node is node B, what is the solution if use greedy best-first search?

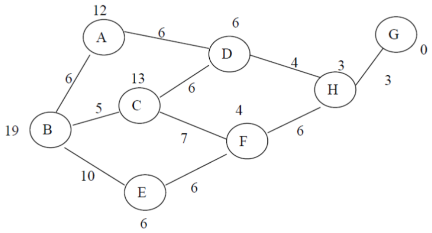
A. BEFHG
B. BCFHG
C. BCDHG
D. BADHG



Quiz

The value at each node represents the heuristic cost from that node to the goal node (G). The value at each arcs represents the path cost between two nodes. If the initial node is node B, what is the solution if use greedy best-first search?

A. BEFHG
B. BCFHG
C. BCDHG
D. BADHG



Answer: A

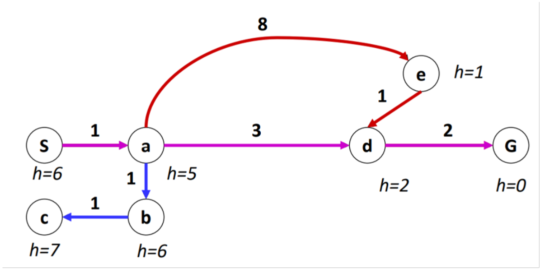
Part 3. A^* search

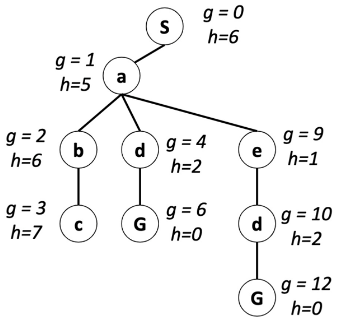
A^* search

- A^* search evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the cost to get from the node to the goal.
- In other words, $g(n)$ gives the path cost from start node to node n, and $h(n)$ is the estimated cost of the cheapest path from n to goal, so we have $f(n) = g(n) + h(n)$.
- $f(n) =$ estimated cost of the cheapest solution through n

A^* search

- A^* search evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the cost to get from the node to the goal.
- Uniform-cost search orders by path cost, or backward cost, i.e., $g(n)$
- Greedy search orders by goal proximity, or forward cost, i.e., $h(n)$

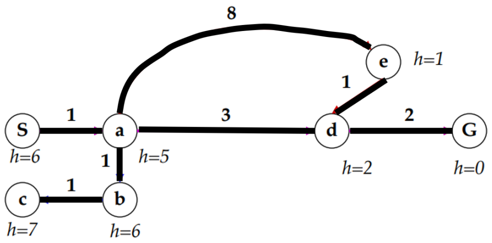


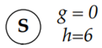


A^* search - an example

The initial state is node S. The goal node is node G.

- Step1: Start from the initial node S, and expand it to the search tree.

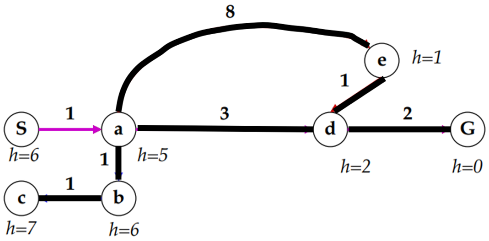


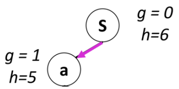


A^* search example

The initial state is node S. The goal node is node G.

- Step 1: Start from the initial node S, and expand it to the search tree.
- Step 2: There is one node connected to the initial node S, Therefore, expand node a to the search tree

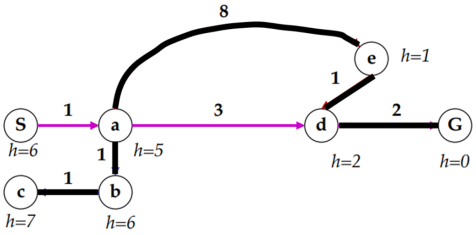


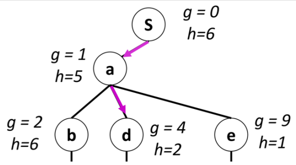


A^* search example

The initial state is node S. The goal node is node G.

- Step 1: Start from the initial node S, and expand it to the search tree.
- Step 2: There is one node connected to the initial node S, Therefore, expand node a to the search tree
- Step 3: Node a connects to three nodes, i.e., b, d, e.
  We expand node d because the $f(d) = h(d) + g(d) = 4 + 2 = 6$ is the smallest one in $f(b)$, $f(d)$, $f(e)$.

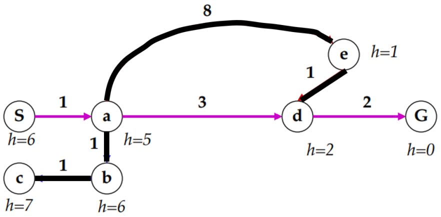


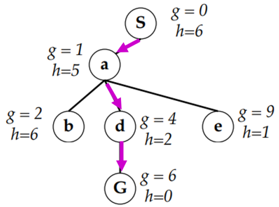


A^* search example

The initial state is node S. The goal node is node G.

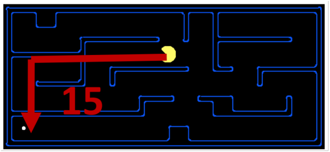
- Step 1: Start from the initial node S, and expand it to the search tree.
- Step 2: There is one node connected to the initial node S, Therefore, expand node a to the search tree
- Step 3: Node a connects to three nodes, i.e., b, d, e.
  We expand node d because the $f(d) = h(d) + g(d) = 4 + 2 = 6$ is the smallest one in $f(b)$, $f(d)$, $f(e)$.
- Step 4: Totally three nodes are available, i.e., b, e, G. We expand G because $f(G) = 6 + 0$ is the smallest one in $f(b)$, $f(e)$ and $f(G)$. Therefore, we reach the goal node, and finish the A^* search.

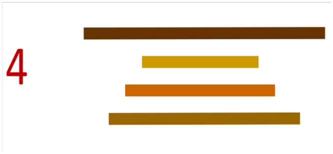




Admissible heuristic function

- $h(n)$ is estimated cost to the goal from the state n. It is an admissible heuristic if it never overestimates the cost to reach the goal
- In other words, a heuristic h is admissible if: $0 \leq h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost to a nearest goal.
- For examples:

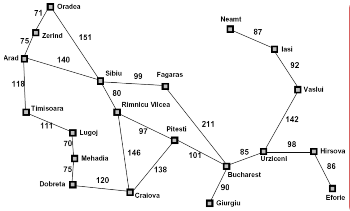


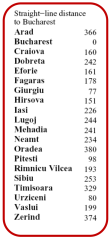


- Coming up with admissible heuristics is most of what's involved in using A^* in practice.

Admissible heuristic function

- Admissible heuristic functions are by nature optimistic, because they think the cost of solving the problem is less than it actually is.
- This optimism transfers to the function $f(n)$ as well.
- Since $g(n)$ is the exact cost to reach n, we have as immediate idea that $f(n)$ never overestimates the true cost of a solution through n.
- An obvious example of an admissible heuristic is the straight-line distance in city map.





Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Combing heuristic functions

Assume that we have two heuristic functions h_1 and h_2. We can combine the two heuristic functions to get a better one.

- Dominance: $h_1 \geq h_2$ if
  - $h_1(n) \geq h_2(n)$ for any state n.
- We know that larger is better as long as both are admissible
- What if have two heuristic functions, but neither dominates another?
  - $h_1(n) < h_2(n)$, existing such state n
  - $h_1(n) > h_2(n)$, existing such state n

Combing heuristic functions

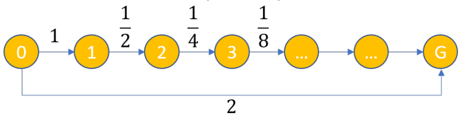
Assume that we have two heuristic functions h_1 and h_2. We can combine the two heuristic functions to get a better one.

- Dominance: $h_1 \geq h_2$ if
  - $h_1(n) \geq h_2(n)$ for any state n.
- We know that larger is better as long as both are admissible
- What if have two heuristic functions, but neither dominates another?
  - $h_1(n) < h_2(n)$, existing such state n
  - $h_1(n) > h_2(n)$, existing such state n
- Form a new one that takes the max of the both.
  - $h(n) = max(h_1(n), h_2(n))$
- Max of admissible functions is admissible and dominates both.
  - $h(n)$ is admissible because $h(n)$ never overestimates the actual cost.
  - $h(n)$ dominates both, because $h(n) \geq h_1(n)$ and $h(n) \geq h_2(n)$ for any state n.

A^* search analysis: completeness

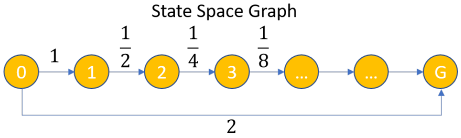
- As A* search expands nodes in order of increasing, it must eventually expand to reach a goal state.
- Unless there are infinitely nodes n with $f(n) \leq f(n^*)$, while the $f(n^*)$ is the optimal solution cost.
  - For example, there is a node with an infinite branching factor or there is a path with a finite path cost but an infinite number of nodes along it
  - Consider the state graph with vertices 0, 1, 2, 3, ... and G. Assume that the start node is node 0, and the goal node is G.
    - Let the weight of the edge between i and i+1 be $\frac{1}{2^i}$, let the weight of the edge between 0 and G be 2, let the heuristic value be 0
    - We know that f(G)=h(G)+g(G)=0+2=2.
    - And there are infinite nodes with cost smaller than 2 because $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots \frac{1}{N}$ is equal to 2 only when N is infinite. Then A* will not find (in finite time) the path from 0 to S.

State Space Graph



A^* search analysis: completeness

- As A* search expands nodes in order of increasing, it must eventually expand to reach a goal state.
- Unless there are infinitely nodes n with $f(n) \leq f(n^*)$, while the $f(n^*)$ is the optimal solution cost.
  - For example, there is a node with an infinite branching factor or there is a path with a finite path cost but an infinite number of nodes along it



- Thus, A* search is complete on locally finite graphs (graphs with a finite branching factor), provided there is some positive constant ϵ such that every operator costs at least ϵ. (*recall completeness of uniform-cost search.*)

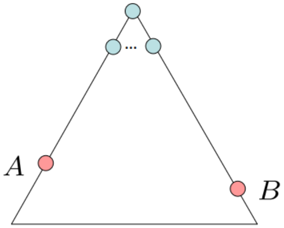
A^* search analysis: optimality

Assume:

- A is an optimal goal node
- B is a suboptimal goal node
- heuristic function h is admissible

Claim:

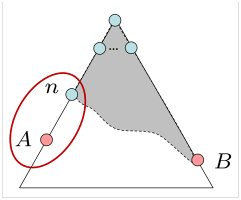
- A will exit the fringe before B



A^* search analysis: optimality

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will exit the fringe before B
  1. $f(n)$ is less or equal to $f(A)$

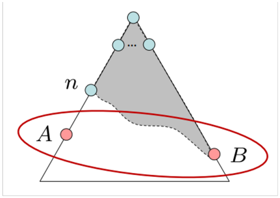


$f(n) = h(n) + g(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$f(A) = g(A)$	h=0 at a goal

A^* search analysis: optimality

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will exit the fringe before B
  1. $f(n)$ is less or equal to $f(A)$
  2. $f(A)$ is less than $f(B)$

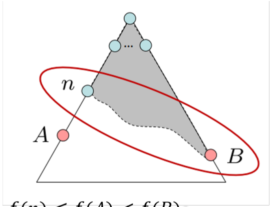


$g(A) < g(B)$ B is suboptimal
$f(A) < f(B)$ h=0 at a goal

A^* search analysis: optimality

Proof:

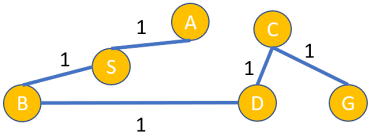
- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will exit the fringe before B
  1. $f(n)$ is less or equal to $f(A)$
  2. $f(A)$ is less than $f(B)$
  3. n expands before B
- All ancestors of A expand before B
- A expands before B
- A^* search is optimal



$f(n) \leq f(A) < f(B)$

A^* search analysis: time complexity

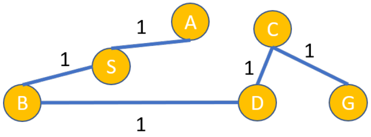
- Assume the initial node is node S, the goal node is node G.
- The time complexity of A* depends on the heuristic.
- In worst case, the time complexity is $O(\boldsymbol{b^m})$, with $\boldsymbol{b}$ as branching factor and $\boldsymbol{m}$ as the maximum depth of the search tree.
  - Assume that the heuristic function is uninformative and the edge costs could all be the same.
  - Then, the A^* search will become breadth-first search. Therefore, the time complexity is $O(\boldsymbol{b^m})$.



Node n	h(n)
S	0
A	0
B	0
C	0
D	0
G	0

A^* search analysis: space complexity

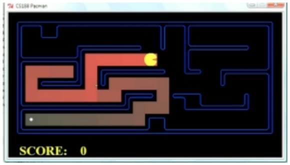
- Assume the initial node is node S, the goal node is node G.
- The space complexity of A* depends on the heuristic.
- In the worst case, the heuristic function is uninformative and the edge costs could all be the same.
- Then, the A^* search will become breadth-first search. Therefore, the space complexity is $O(\boldsymbol{b^m})$.

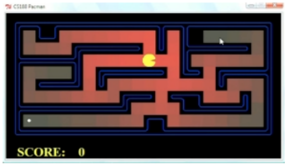


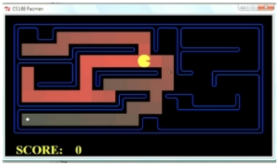
Node n	h(n)
S	0
A	0
B	0
C	0
D	0
G	0

Comparison

- Greedy expands the nodes whose estimated distance to the goal is the smallest
- Uniform-cost expands equally in all "directions"
- A^* expands mainly toward the goal, but does hedge its bets to ensure optimality







Greedy

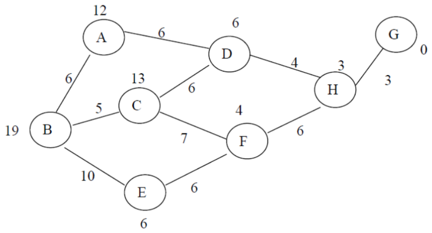
Uniform Cost

A*

Quiz

The value at each node represents the heuristic cost from that node to the goal node (G). The value at each arcs represents the path cost between two nodes. If the initial node is node B, what is the solution if use A^* search?

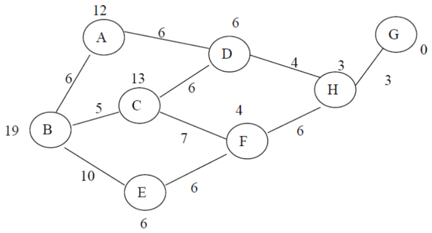
A. BEFHG
B. BCFHG
C. BCDHG
D. BADHG



Quiz

The value at each node represents the heuristic cost from that node to the goal node (G). The value at each arcs represents the path cost between two nodes. If the initial node is node B, what is the solution if use A^* search?

A. BEFHG
B. BCFHG
C. BCDHG
D. BADHG



Answer: C

Part 4. Summary

Summary

- Informed search and heuristics
- Greedy best-first search
- A* search

Reading materials

- Chapter 4.1, Artificial Intelligence A Modern Approach by Stuart J. Russell and Peter Norvig
- Slides, CS 188 Introduction to Artificial Intelligence Summer 2021