National University of Singapore
School of Computing
CS2109S: Introduction to AI and Machine Learning
Semester 2, 2024/2025

Tutorial 2
Informed Search

These questions will be discussed during the tutorial session. Please be prepared to answer them.

Summary of Key Concepts

In this tutorial, we will discuss and explore the following key learning points/lessons from Lecture:

1. Heuristic functions
   (a) Formulation of heuristic functions
   (b) Admissible and consistent heuristics
   (c) Dominance
2. A* Search
   (a) Optimality of `SEARCH` vs `SEARCH WITH VISITED MEMORY`

Problems

A Pacman

Pacman is a maze action video game. For this question, we will take a look at a simplified version of Pacman. In this version, Players control Pacman, with the objective of eating all the pellets in the maze while executing the least amount of moves. More formally, the initial state is a game state where Pacman is at a starting location and pellets are scattered around the maze (refer to Figure 1). Available actions are 4-directional movement unless blocked by walls, and if Pacman moves to a square with a pellet, he will automatically eat it with no additional cost (each movement has a cost of 1). The goal state is when Pacmac has eaten all the pellets. Recall that A* search finds an optimal solution with an admissible heuristic.

1. Thus, devise a non-trivial admissible heuristic for this problem.

B Route Finding

We learnt that for route-finding problems, a simple admissible heuristic is the straight line distance. More formally, given a graph $G = (V, E)$ where each node v_n having coordinates (x_n, y_n), each edge (v_i, v_j) having weight equals to the distance between v_i and v_j, and a unique goal node v_g with coordinates (x_g, y_g), the straight line distance heuristic is given by:

$$h_{SLD}(n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2}$$

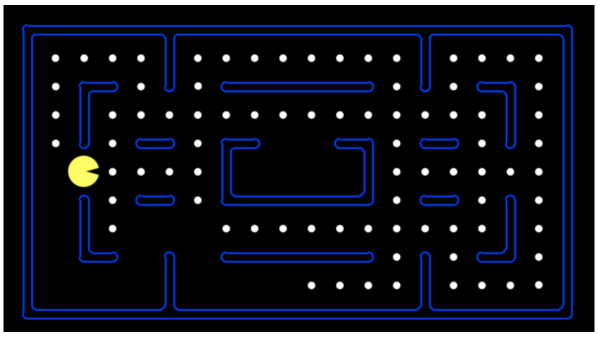


Figure 1: A state of Pacman.

Consider the following two heuristic functions

$$h_1(n) = \max\{|x_n - x_g|, |y_n - y_g|\}$$

$$h_2(n) = |x_n - x_g| + |y_n - y_g|$$

1. Is $h_1(n)$ an admissible and consistent heuristic? If yes, prove it; otherwise, provide a counterexample.
2. Is $h_2(n)$ an admissible heuristic? If yes, prove it; otherwise, provide a counterexample.
3. Out of $h_1(n)$, $h_2(n)$ and $h_{SLD}(n)$, which heuristic function would you choose for A* search? Justify your answer.

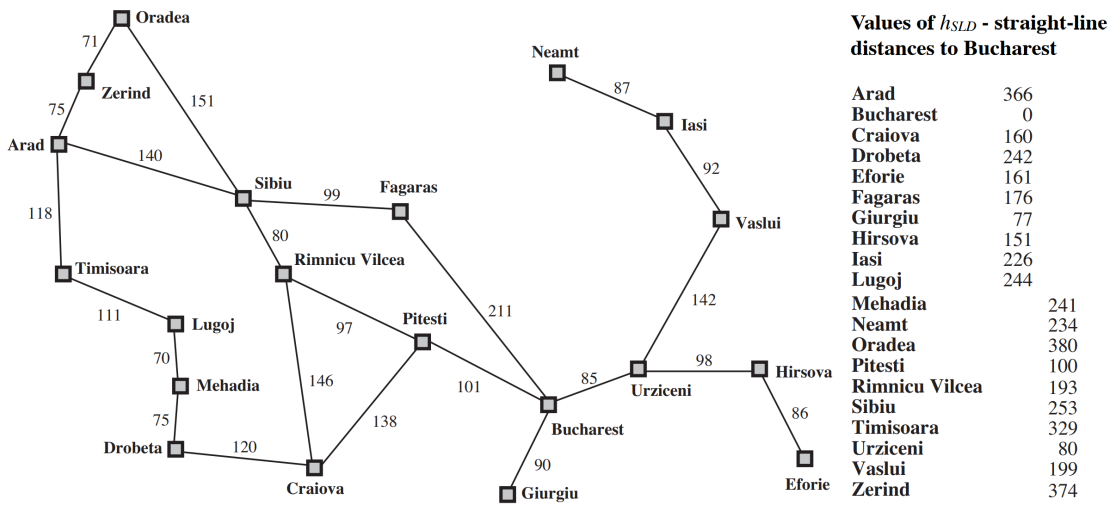
C Admissibility and Consistency

1. Given that a heuristic h is such that $h(G) = 0$, where G is any goal state, prove that if h is consistent, then it must be admissible. (Hint: Think of the path the algorithm takes)
2. Give an example of an admissible heuristic function that is not consistent.

D Romania

Refer to the Figure 2 below. Apply the A* search algorithm to find a path from Fagaras to Craiova where $h(n) = |h_{SLD}(\text{Craiova}) - h_{SLD}(n)|$ and $h_{SLD}(n)$ is the straight-line distance from any city n to Bucharest given in Figure 2.

1. Trace the A* search algorithm by showing the nodes and their priorities in the fringe (frontier) as each node is expanded, based on the SEARCH algorithm below in its first *three* iterations. Also, draw the search trees for the first three iterations.
2. (Optional) Prove that $h(n)$ is an admissible heuristic.

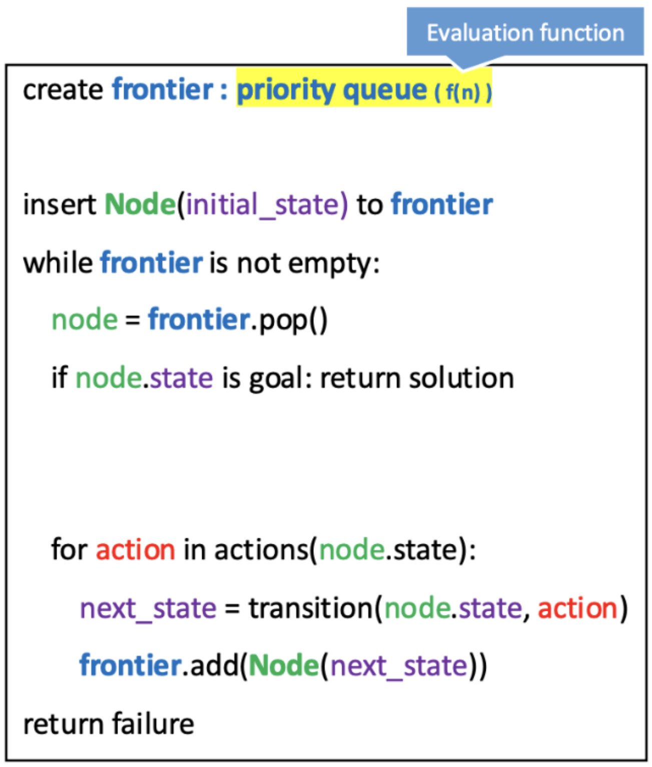


Values of h_{SLD} - straight-line distances to Bucharest

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Figure 2: Graph of Romania.

E Admissibility and Inconsistency



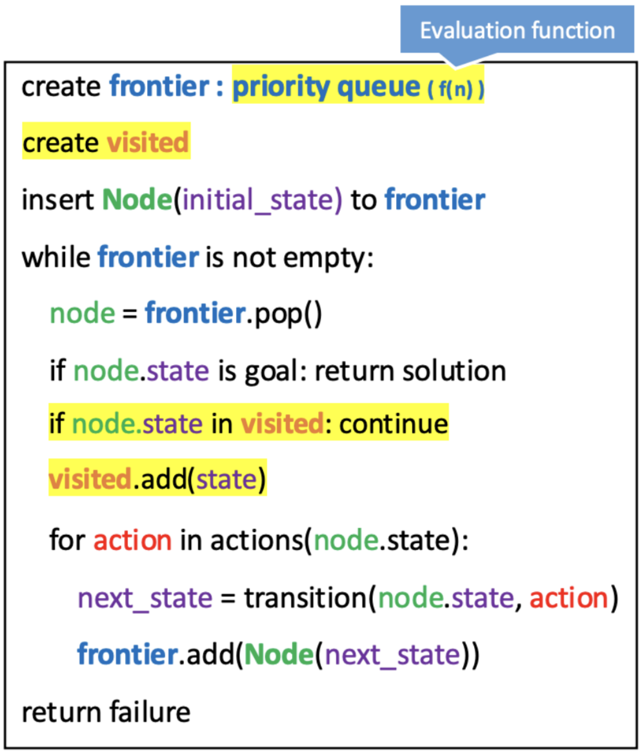
```
create frontier : priority queue ( f(n) )


insert Node(initial_state) to frontier
while frontier is not empty:
    node = frontier.pop()
    if node.state is goal: return solution



    for action in actions(node.state):
        next_state = transition(node.state, action)
        frontier.add(Node(next_state))
return failure
```

Figure 3: A* search (`SEARCH` implementation).


Evaluation function

```
create frontier : priority queue ( f(n) )
create visited
insert Node(initial_state) to frontier
while frontier is not empty:
    node = frontier.pop()
    if node.state is goal: return solution
    if node.state in visited: continue
    visited.add(state)
    for action in actions(node.state):
        next_state = transition(node.state, action)
        frontier.add(Node(next_state))
return failure
```

Figure 4: A* search (SEARCH WITH VISITED MEMORY implementation).

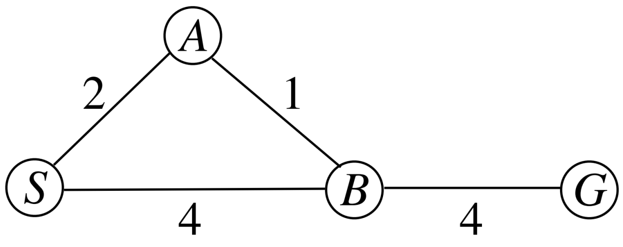


Figure 5: Graph

You have learned before that A* using search with visited memory is optimal if $h(n)$ is consistent. Does this optimality still hold if $h(n)$ is admissible but inconsistent? Use the heuristic function $h(n)$ defined as follows:

$$h(n) = \begin{cases} 7 & \text{if } n = S \\ 0 & \text{if } n = B \\ 3 & \text{if } n = A \\ 0 & \text{if } n = G \end{cases}$$

You may refer to Figure 3 and Figure 4 for A* Search vs. A* Search with visited memory implementation.

1. Using the graph in Figure 5, show that A* using SEARCH WITH VISITED MEMORY returns a non-optimal solution path from start node S to goal node G when using an admissible but inconsistent $h(n)$.
2. Show that search will return the optimal solution with the same heuristic.

F (Extra & Optional) Negativity

1. Would A* work with negative edge weights? Assume no negative cycles. If yes, prove it; otherwise, provide a counterexample. What if there are negative cycles?
2. We have seen how A* performs with negative *edge weights*. A* with negative *heuristics* is different: negative heuristics are heuristic functions that can return negative values for some nodes. Can A* work with negative heuristics? If yes, prove it; otherwise, provide a counterexample.