National University of Singapore
School of Computing
CS2109S: Introduction to AI and Machine Learning
Semester I, 2023/2024

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 tree search vs graph search

Problems

1. 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* tree search finds an optimal solution with an admissible heuristic. Thus, devise a non-trivial admissible heuristic for this problem.

   A simple admissible heuristic $h(n)$ would be the number of pellets left at state n. Since Pacman can only eat at most one pellet per move, a state with n pellets will at least be n pellets away from the goal state. This means that $h(n) \leq h^*(n)$ where h^* is the true cost. A way to formulate admissible heuristics is to look at the cost of the optimal solution to a relaxed version of the problem - in this case, the relaxed problem would be that Pacman can move to any square without restrictions. More formally, we can view it as adding edges (less restrictions means extra moves) to the original graph. By definition, a solution to the original problem is also a solution to the relaxed problem as the original edges remain in the relaxed problem, but may not be optimal as the new edges may provide better paths to the goal node. This also means that the cost of optimal solution to the relaxed problem is always lower than the cost of

the optimal solution to the original problem, making it an admissible heuristic for the original problem.

Extra: since the heuristic is an exact cost for the relaxed problem, it satisfies the triangle inequality which also makes it a consistent heuristic as well.

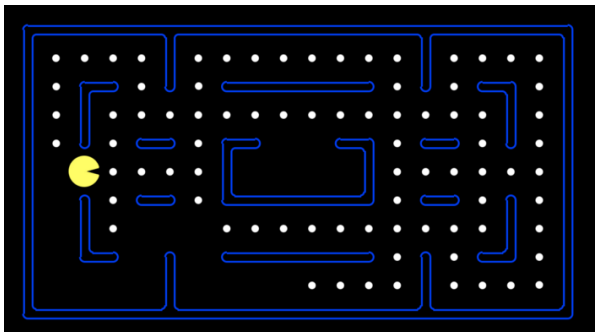


Figure 1: A state of Pacman.

2. 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:

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

Consider the following two heuristic functions

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

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

(a) Is $h_1(n)$ an admissible heuristic? If yes, prove it; otherwise, provide a counter-example.

Let $h^*(n)$ be the actual path distance from v_n to v_g. Since h_{SLD} is admissible, we have

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

Note that

$$\sqrt{(x_n - x_g)^2 + (y_n - y_g)^2} \geq \sqrt{(x_n - x_g)^2} = |x_n - x_g|$$

and similarly, $h_{SLD} \geq |y_n - y_g|$. Thus, $h_{SLD} \geq max\{|x_n - x_g|, |y_n - y_g|\} = h_1(n)$. This means that $h^*(n) \geq h_{SLD}(n) \geq h_1(n)$, so $h_1(n)$ is admissible.

(b) Is $h_2(n)$ an admissible heuristic? If yes, prove it; otherwise, provide a counter-example.

$h_2(n)$ is not an admissible heuristic. Consider a graph where v_s has coordinates $(0, 0)$ and v_g has coordinates $(1, 1)$. $h^*(v_s) = \sqrt{2}$ while $h_2(v_s) = 2$.

(c) Out of $h_1(n)$, $h_2(n)$ and $h_{SLD}(n)$, which heuristic function would you choose for A* search? Justify your answer.

$h_{SLD}(n)$. $h_2(n)$ is not admissible, so we may not get an optimal solution if we use it. Meanwhile, from part a, we have proven that $h_{SLD}(n) \geq h_1(n)$, i.e. $h_{SLD}(n)$ dominates $h_1(n)$, so we will incur less search cost (on average) if we use $h_{SLD}(n)$.

3. (a) 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.

Consistency is defined as $h(v) \leq c(v, a, u) + h(u)$.

Let v_n *be an arbitrary node and let* $v_n, v_{n-1}, \ldots, v_1, G$ *be the path with minimal cost to a goal state.*

By using the fact that h *is consistent multiple times, we can get:*

$$h(v_n) \leq c(v_n, a, v_{n-1}) + h(v_{n-1})$$

$$h(v_{n-1}) \leq c(v_{n-1}, a, v_{n-2}) + h(v_{n-2})$$

$$\ldots$$

$$h(v_1) \leq \ c(v_1, a, G) + h(G)$$

Note that $h(G) = h^*(G) = 0$, *so we can drop* $h(G)$.

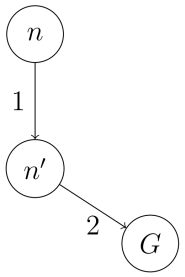
Combined,

$$\begin{aligned} & h(v_n) \\ & \leq c(v_n, a, v_{n-1}) + h(v_{n-1}) \\ & \leq c(v_n, a, v_{n-1}) + c(v_{n-1}, a, v_{n-2}) + h(v_{n-2}) \\ & \ldots \\ & \leq c(v_n, a, v_{n-1}) + c(v_{n-1}, a, v_{n-2}) + ... + c(v_1, a, G)) \end{aligned}$$

Because we chose the minimal cost path, $c(v_n, a, v_{n-1}) + c(v_{n-1}, a, v_{n-2}) + ... + c(v_1, a, G))$ *equals to the true cost* $h^*(n)$. *Hence we have shown* $h(v_n) \leq h^*(v_n)$ *for any node* v_n, *and proven that* h *is admissible.*

(b) Give an example of an admissible heurstic function that is not consistent.

Consider the graph



An example of an admissible heuristic function that is not consistent is as follows:

$$h(n) = 3$$

$$h(n') = 1$$

$$h(G) = 0$$

h *is admissible since*

$$h(n) \leq h^*(n) = 1 + 2 = 3$$

$$h(n') \leq h^*(n') = 2$$

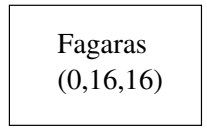
However, h *is not consistent as* $h(n) = 3 > c(n, a, n') + h(n') = 1 + 1 = 2$

4. Refer to the Figure 2 below. Apply the A* search algorithm to find a path from Fagaras to Craiova where $h(n) = |h_{SLD}(\text{Craiov}) - h_{SLD}(n)|$ and $h_{SLD}(n)$ is the straight-line distance from any city n to Bucharest given in Figure 4.1.

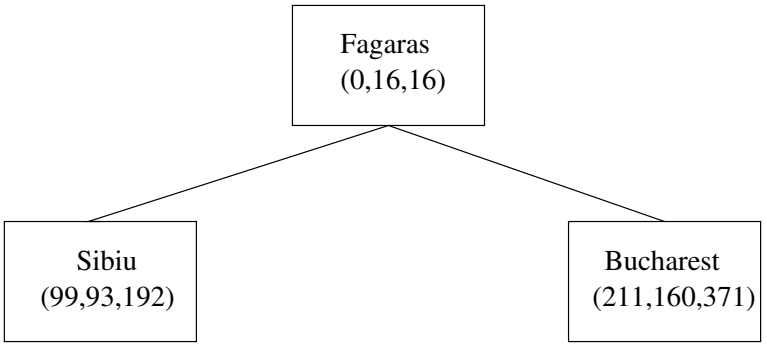
   (a) Trace the A* search algorithm by showing the series of search trees as each node is expanded, based on the TREE-SEARCH algorithm below.

   The series of search trees where the 3-tuple in each node denotes (g, h, f)*:*

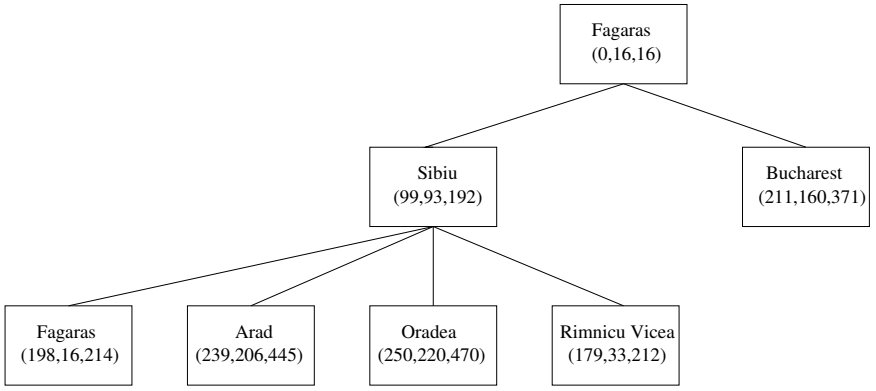
(a)



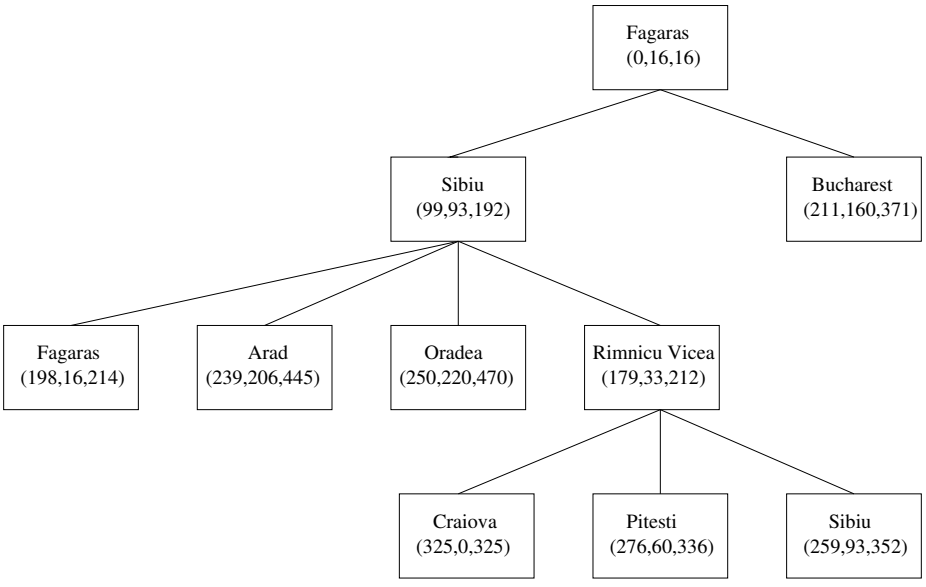
(b)

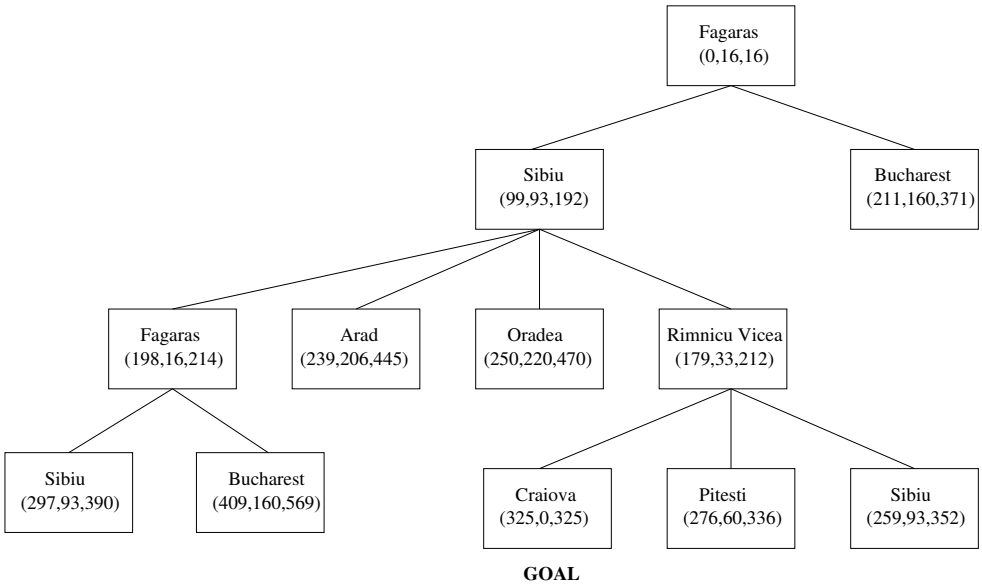


(c)



(d)



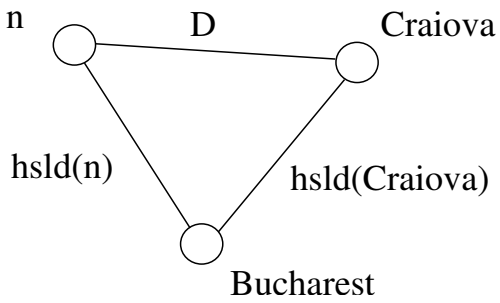


(e)

Note the we need to expand Faragas even though we had reached the destination the third step because getting to the destination is not sufficient. A finds the optimal solution.*

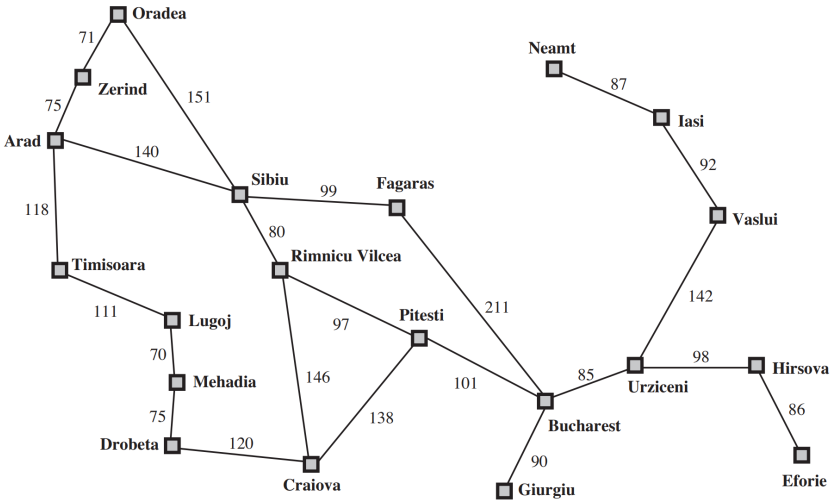
(b) Prove that $h(n)$ is an admissible heuristic.

Now consider the following triangle:



Let D be the straight-line distance between n and Craiova. From the above triangle, by the Triangle Inequality D must be at least as much as the difference of the 2 other sides. $D \geq |h_{SLD}(Craiova) - h_{SLD}(n)|$. Hence, $D \geq h(n)$. But we also know that $h^*(n) \geq D$, so $h^*(n) \geq h(n)$, and $h(n)$ is admissible.

Note: The definition of triangle inequality is, sum of the lengths of any two sides must be greater than or equal to the the length of the third side.



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.

function TREE-SEARCH(*problem, fringe*) **returns** a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]),*fringe*)
  loop do
    if *fringe* is empty **then return** failure
    node ← REMOVE-FRONT(*fringe*)
    if GOAL-TEST[*problem*] applied to STATE(*node*) succeeds **return** *node*
    fringe ← INSERTALL(EXPAND(*node*,*problem*),*fringe*)

Figure 3: Tree Search Algorithm.

5. You have learned before that A* using graph search is optimal if $h(n)$ is consistent. Does this optimality still hold if $h(n)$ is admissible but inconsistent? Using the graph in Figure 4, let us now show that A* using graph search returns a non-optimal solution path from start node S to goal node G when using an admissible but inconsistent $h(n)$. We assume that $h(G) = 0$. Then, show that tree search will return the optimal solution with the same heuristic.

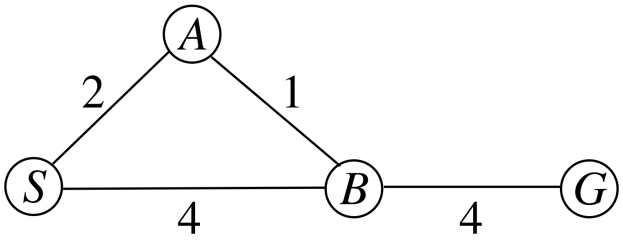


Figure 4: Graph.

First let us recall that A maintains a priority queue in increasing order of* $f(n) = g(n)+h(n)$ *(ties broken arbitrarily). The key difference between tree search and graph search is that under graph search, once a node gets explored, it will never be added to the priority queue again. The intuition here is that if we do not revisit nodes, we may miss out on some shorter paths that we mistakenly think are long after reaching the node as the heuristic function violates the triangle inequality. There are several possible solutions one might consider. First, let us consider an admissible* h *where*

$$h(S) = 7$$

$$h(B) = 0$$

$$h(A) = 3, 4 \text{ or } 5$$

$$h(G) = 0$$

Assume that $h(A) = 3$*. A* will first add* $(S, 7, (null))$ *to the priority queue and explore it as it is the only element in the queue, which adds* $(A, 2 + 3, (S))$ *and* $(B, 4 + 0, (S))$ *into the priority queue, resulting in*

$$|(B, 4, (S)), (A, 5, (S))|$$

It will then explore B*, finding* A *and* G*, and since both nodes have not been explored yet, it adds both* $(A, 8)$ *and*$(G, 8)$ *into the priority queue resulting in*

$$|(A, 5, (S)), (A, 8, (S, B)), (G, 8, (S, B))|$$

Now, it explores A *finding* S *and* B*, but does not add either of them into the priority queue since both nodes have already been explored. Next, it explores* A *again and adds nothing into the priority queue. With only* G *left in the queue, we explore* G*, ending up with a suboptimal solution of* (S, B, G)*.*

With tree search, we add nodes that we have already visited back into the priority queue. At the start, we add S *into the priority queue:*

$$|(S, 7, (null))$$

Exploring it, we find A *and* B*, adding them into the priority queue, resulting in*

$$|(B, 4, (S)), (A, 5, (S))|$$

It still explores B *first, finding* S*,* A *and* G*, resulting in*

$$|(A, 5, (S)), (A, 8, (S, B)), (G, 8, (S, B)), (S, 15, (S, B))|$$

Then, it explores A*, finding* S *and* B*, resulting in*

$$|(B, 3, (S, A)), (A, 8, (S, B)), (G, 8, (S, B)), (S, 11, (S, A)), (S, 15, (S, B))|$$

This time, it explores B *first (from the path from A), finding* S*,* A *and* G*, resulting in*

$$|(G, 7, (S, A, B)), (A, 7, (S, A, B)), (A, 8, (S, B)), (G, 8, (S, B)), ..., (S, 15, (S, B))|$$

Finally, G *is explored and we get the optimal path* (S, A, B, G)*.*

6. *Recall that in the proof of A*, negative weights do not affect the optimality of A* as long as there are no negative cycles. Therefore, the same proof applies.*