

Q1. Searching with Heuristics

Consider the A* searching process on the connected undirected graph, with starting node S and the goal node G. Suppose the cost for each connection edge is **always positive**. We define $h^*(X)$ as the shortest (optimal) distance to G from a node X.

Answer Questions (a). **Question (b) is optional.**

(a) Suppose h is an **admissible** heuristic, and we conduct A* **tree search** using heuristic h' and finally find a solution. Let C be the cost of the found path (directed by h' , defined in part (a)) from S to G

(i) Choose **one best** answer for each condition below.

- | | | | |
|---|---|------------------------------------|--|
| 1. If $h'(X) = \frac{1}{2}h(X)$ for all Node X , then | <input checked="" type="radio"/> $C = h^*(S)$ | <input type="radio"/> $C > h^*(S)$ | <input type="radio"/> $C \geq h^*(S)$ |
| 2. If $h'(X) = \frac{h(X)+h^*(X)}{2}$ for all Node X , then | <input checked="" type="radio"/> $C = h^*(S)$ | <input type="radio"/> $C > h^*(S)$ | <input type="radio"/> $C \geq h^*(S)$ |
| 3. If $h'(X) = h(X) + h^*(X)$ for all Node X , then | <input type="radio"/> $C = h^*(S)$ | <input type="radio"/> $C > h^*(S)$ | <input checked="" type="radio"/> $C \geq h^*(S)$ |

(ii) In which of the conditions above, h' is still **admissible** and for sure to dominate h ? Check all that apply. Remember we say h_1 dominates h_2 when $h_1(X) \geq h_2(X)$ holds for all X .

1 2 3 4 5 6

(b) [Optional] Suppose h is an **admissible** heuristic, and we conduct A* **tree search** using heuristic h' and finally find a solution.

If $\epsilon > 0$, and X_0 is a node in the graph, and h' is a heuristic such that

$$h'(X) = \begin{cases} h(X) & \text{if } X = X_0 \\ h(X) + \epsilon & \text{otherwise} \end{cases}$$

- Alice claims h' can be inadmissible, and hence $C = h^*(S)$ does not always hold.
- Bob instead thinks the node expansion order directed by h' is the same as the heuristic h'' , where

$$h''(X) = \begin{cases} h(X) - \epsilon & \text{if } X = X_0 \\ h(X) & \text{if otherwise} \end{cases}$$

Since h'' is admissible and will lead to $C = h^*(S)$, and so does h' . Hence, $C = h^*(S)$ always holds.

The two conclusions (underlined) apparently contradict with each other, and **only exactly one of them are correct and the other is wrong**. Choose the **best** explanation from below - which student's conclusion is wrong, and why are they wrong?

- Alice's conclusion is wrong, because the heuristic h' is always admissible.
- Alice's conclusion is wrong, because an inadmissible heuristics does not necessarily always lead to the failure of the optimality when conducting A* tree search.
- Alice's conclusion is wrong, because of another reason that is not listed above.
- Bob's conclusion is wrong, because the node visiting expansion ordering of h'' during searching might not be the same as h' .
- Bob's conclusion is wrong, because the heuristic h'' might lead to an incomplete search, regardless of its optimally property.
- Bob's conclusion is wrong, because of another reason that is not listed above.

Choice 4 is incorrect, because the difference between h' and h'' is a constant. During searching, the choice of the expansion of the fringe will not be affected if all the nodes add the same constant to the heuristics.

Choice 5 is incorrect because there will never be an infinite loop if there are no cycle has negative COST sum (rather than HEURISTICS). If there is a cycle, such that its COST sum is positive, and all the nodes in the cycle have negative heuristics, when we do $g+h$, g is getting larger and larger, while h remains a not-that-large negative value. Soon, the search algorithm will be favoring other paths even if the h in there are not negative.

The true reason: h'' violate a property of admissible heuristic. Since h is admissible, we have $h(G) = 0$. If $X_0 = G$, we have a negative heuristic value at $h''(G)$, and it is no longer admissible. If $X_0 \neq G$, then it is indeed that the optimality holds - the only change is that more nodes will be likely to be expanded for h' and h'' compared to h .

Q2. Iterative Deepening Search

Pacman is performing search in a maze again! The search graph has a branching factor of b , a solution of depth d , a maximum depth of m , and edge costs that may not be integers. Although he knows breadth first search returns the solution with the smallest depth, it takes up too much space, so he decides to try using iterative deepening. As a reminder, in standard depth-first iterative deepening we start by performing a depth first search terminated at a maximum depth of one. If no solution is found, we start over and perform a depth first search to depth two and so on. This way we obtain the shallowest solution, but use only $O(bd)$ space.

But Pacman decides to use a variant of iterative deepening called **iterative deepening A***, where instead of limiting the depth-first search by depth as in standard iterative deepening search, we can limit the depth-first search by the f value as defined in A* search. As a reminder $f[node] = g[node] + h[node]$ where $g[node]$ is the cost of the path from the start state and $h[node]$ is a heuristic value estimating the cost to the closest goal state.

In this question, all searches are tree searches and **not** graph searches.

- (a) Complete the pseudocode outlining how to perform iterative deepening A* by choosing the option from the next page that fills in each of these blanks. Iterative deepening A* should return the solution with the lowest cost when given a consistent heuristic. Note that cutoff is a boolean and new-limit is a number.

```
function ITERATIVE-DEEPENING-TREE-SEARCH(problem)
  start-node ← MAKE-NODE(INITIAL-STATE[problem])
  limit ←  $f[\textit{start-node}]$ 
  loop
    fringe ← MAKE-STACK(start-node)
    new-limit ← 
    cutoff ← 
    while fringe is not empty do
      node ← REMOVE-FRONT(fringe)
      if GOAL-TEST(problem, STATE[node]) then
        return node
      end if
      for child-node in EXPAND(STATE[node], problem) do
        if  $f[\textit{child-node}] \leq \textit{limit}$  then
          fringe ← INSERT(child-node, fringe)
          new-limit ← 
          cutoff ← 
        else
          new-limit ← 
          cutoff ← 
        end if
      end for
    end while
    if not cutoff then
      return failure
    end if
    limit ← 
  end loop
end function
```

A ₁	$-\infty$	A ₂	0	A ₃	∞	A ₄	<i>limit</i>
B ₁	True	B ₂	False	B ₃	<i>cutoff</i>	B ₄	not <i>cutoff</i>
C ₁	<i>new-limit</i>	C ₂	<i>new-limit</i> + 1	C ₃	<i>new-limit</i> + $f[node]$	C ₄	<i>new-limit</i> + $f[child-node]$
C ₅	MIN(<i>new-limit</i> , $f[node]$)	C ₆	MIN(<i>new-limit</i> , $f[child-node]$)	C ₇	MAX(<i>new-limit</i> , $f[node]$)	C ₈	MAX(<i>new-limit</i> , $f[child-node]$)

- (i) A₁ A₂ A₃ A₄
- (ii) B₁ B₂ B₃ B₄
- (iii) C₁ C₂ C₃ C₄
 C₅ C₆ C₇ C₈
- (iv) B₁ B₂ B₃ B₄
- (v) C₁ C₂ C₃ C₄
 C₅ C₆ C₇ C₈
- (vi) B₁ B₂ B₃ B₄
- (vii) C₁ C₂ C₃ C₄
 C₅ C₆ C₇ C₈

The cutoff variable keeps track of whether there are items that aren't being explored because of the limit. If cutoff is false and the algorithm has exited the while, no nodes were cutoff (not added to the fringe because of the limit). This scenario suggests that there is no solution.

In order to ensure that iterative deepening A* obtains the lowest cost solution efficiently, we want to increase the limit as much as we can while guaranteeing optimality. Setting new-limit to the smallest f cost of nodes that were cutoff achieves this. When nodes aren't cutoff (part iii), the new-limit should not change. Hence C1, C7, C8, or a combination of the three were accepted as answers.

(b) Assuming there are no ties in f value between nodes, which of the following statements about the number of nodes that iterative deepening A* expands is True? If the same node is expanded multiple times, count all of the times that it is expanded. If none of the options are correct, mark None of the above.

- The number of times that iterative deepening A* expands a node is greater than or equal to the number of times A* will expand a node.
- The number of times that iterative deepening A* expands a node is less than or equal to the number of times A* will expand a node.
- We don't know if the number of times iterative deepening A* expands a node is more or less than the number of times A* will expand a node.
- None of the above

Iterative deepening A* runs depth first search multiples at different limit values. This causes iterative deepening A* to expand certain nodes multiple times.