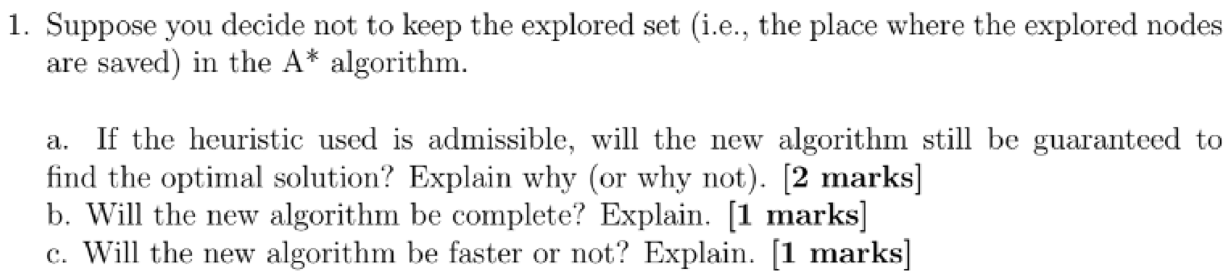
Artificial Intelligence

Assignment-2

Arnav Singh
2021019

Theory Assignment



1. Suppose you decide not to keep the explored set (i.e., the place where the explored nodes are saved) in the A* algorithm.

   a. If the heuristic used is admissible, will the new algorithm still be guaranteed to find the optimal solution? Explain why (or why not). [**2 marks**]
   b. Will the new algorithm be complete? Explain. [**1 marks**]
   c. Will the new algorithm be faster or not? Explain. [**1 marks**]

$f(n)=g(n)+h(n)$

a. Yes, the algorithm will still be guaranteed to find the optimal solution. Since the heuristic used is still admissible, i.e., $h(n)<=h^*(n)$, (where $h(n)$ is the heuristic function, and $h^*(n)$ is the actual cost to reach the goal state), the $f(n)$ at the last stage will always be greater than, or equal to the final stage $f(n)$ predicted by the heuristics at the penultimate stage

   Proof:

   $h(n)<=h^*(n)$

   $g(n)+h(n)<=g(n)+h^*(n)$

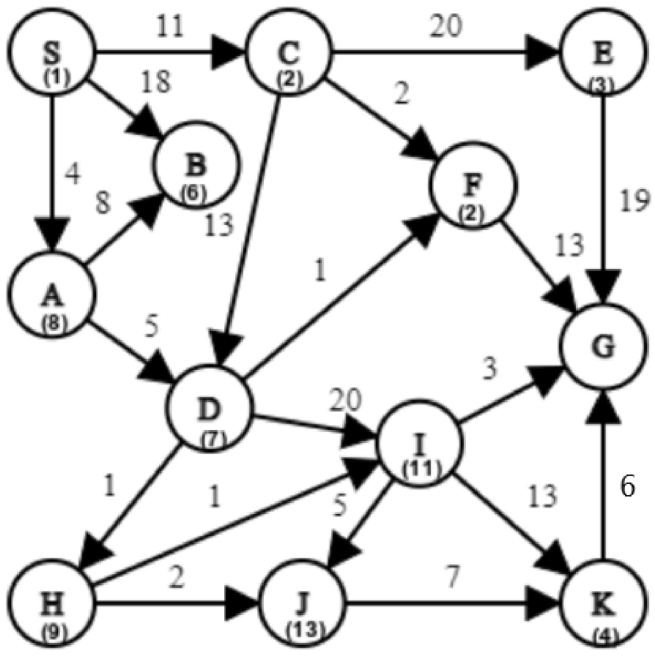
   $f_{heuristically\_estimated}(n)<=f_{actual}(n)$

   Should the final stage f(n)be greater than the expected value, we can try finding newer better paths from scratch, but if we have covered all possible paths, the answer is the one with the lowest f(n) for the final state. In the case that the final f(n) matches the f(n) predicted by heuristics in the penultimate stage, h(n)==h'(n). In this case, we don't need to explore any further, since the heuristic is accurate. [For the result to match the heuristics' prediction, h(n)==h'(n), since otherwise, there can be a possibility of a better path, should the better path have been assigned poor heuristic values.]

(b) Yes,this algorithm is complete since it will return the optimal solution for all search problems it's applied to in finite spaces, as could be inferred from the first part. For infinite spaces, there needs to exist a solution for the algorithm to terminate, as per Norvig.

(c) No, Although it might conserve the memory otherwise utilized to store the explored set, it will have a greater time complexity since it will have to explore many useless paths while trying to find the optimal solution.

2. Given the graph, find the cost-effective path from S to G. S is the source node, and G is the goal node using A* [**2 marks**], BFS [**2 marks**], and Dijkstra algorithm. [**2 marks**] Please note: cost written on edges represents the distance between nodes. The cost written on nodes represents the heuristic value. Heuristic for Goal Node G is 0. Distance from K to G is 6.



(i) A*

f(n)=g(n)+h(n)

f(S)=0+1=1

f(A)=4+8=12

f(B)=18+6=24

f(C)=11+2=13

A is chosen to be the next node, since it's value is the minimum among A,B and C.

f(D)=9+7=16

There are no alternatives other than D, D is chosen to be the next node.

f(H)=19

f(I)=40

f(H)<f(I), So H is the next node.

f(I)=22

f(J)=25

f(I)<f(J), So I is the next node

f(G)=14. Since we had overestimated our heuristic function [h(n)>h'(n)], The algorithm terminates, and the most cost-effective path using the A* algorithm turns out to be S->A->D->H->I->G

(ii) BFS

<u>Best First Search</u>

path={}	
path={S}	
path={S,C}	C has the least heuristic cost among A,B,C
path={S,C,F}	F has the least heuristic cost among D,F,E
path={S,C,F,G}	G is the goal state

The most cost-effective path going by Best-First Search is S->C->F->G (i.e. Best First Search doesn't result in the most optimal path between the two nodes)

Breadth First Search

visited=[]

to_be_visited=[]

—---

Algorithm Begins

visited=[]	to_be_visited=[S]	
visited=[S]	to_be_visited=[C,B,A]	S=S
visited=[S,A]	to_be_visited=[D,C,B]	S->A=A
visited=[S,A,B]	to_be_visited=[D,C]	S->B=B
visited=[S,A,B,C]	to_be_visited=[F,E,D]	S->C=C
visited=[S,A,B,C,D]	to_be_visited=[I,H,F,E]	S->A->D=D
visited=[S,A,B,C,D,E]	to_be_visited=[G,I,H,F]	S->C->E=E
visited=[S,A,B,C,D,E,F]	to_be_visited=[G,I,H]	S->C->F=F
visited=[S,A,B,C,D,E,F,H]	to_be_visited=[J,G,I]	S->A->D->H=H
visited=[S,A,B,C,D,E,F,H,I]	to_be_visited=[K,J,G]	S->A->D->I=I
visited=[S,A,B,C,D,E,F,H,I,G]	to_be_visited=[K,J]	S->C->E->G
visited=[S,A,B,C,D,E,F,H,I,G,J]	to_be_visited=[K]	S->A->D->H->J=J
visited=[S,A,B,C,D,E,F,H,I,G,J,K]	to_be_visited=[]	S->A->D->I->K=K

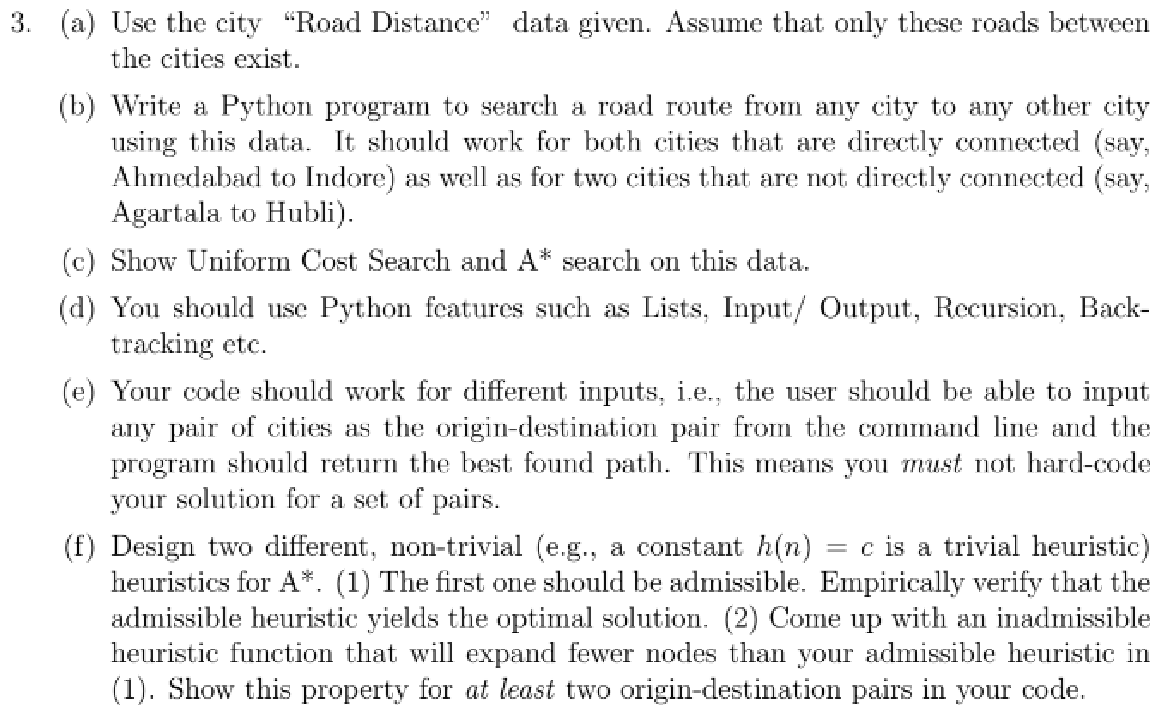
Using BFS, It can be observed that the shortest path from S to G is S->C->E->G. It can be empirically proven that this in fact, is not the most cost effective path from S to G (BFS can only work effectively on weighted graphs, should the weights be evenly distributed among the edges writ large). It's worth noting that it is the shallowest path from S to G.

(iii) Dijkstra

Current Node	S	A	B	C	E	H	D	F	J	I	K	G
S	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
A	0	4	18	11	∞	∞	∞	∞	∞	∞	∞	∞
C	0	4	12	11	∞	∞	9	∞	∞	∞	∞	∞
D	0	4	12	11	∞	10	9	10	∞	29	∞	∞
H	0	4	12	11	∞	10	9	10	12	11	∞	∞
F	0	4	12	11	∞	10	9	10	12	11	∞	23
C	0	4	12	11	31	10	9	10	12	11	∞	23
I	0	4	12	11	31	10	9	10	12	11	24	14
B	0	4	12	11	31	10	9	10	12	11	19	14
J	0	4	12	11	31	10	9	10	12	11	19	14
G	0	4	12	11	31	10	9	10	12	11	19	14
K	0	4	12	11	31	10	9	10	12	11	19	14
E	0	4	12	11	31	10	9	10	12	11	19	14

From Dijkstra’s Algo, It can be seen that the most cost-effective path from S to G is S->A->D->H->I->G

```
# def ucs(graph, start, goal):
#     pqueue = [(0, start, [])]
#     while pqueue:
#         cost, current_node, path = heapq.heappop(pqueue)
#
#         if current_node == goal:
#             return path + [current_node]
#
#         for neighbor,edge_cost in graph[current_node]:
#             heapq.heappush(pqueue, (cost + int(edge_cost), neighbor, path + [current_node]))
#
#     return None
# separate function for ucs should the ta prefer the same
```



3. (a) Use the city "Road Distance" data given. Assume that only these roads between the cities exist.
   (b) Write a Python program to search a road route from any city to any other city using this data. It should work for both cities that are directly connected (say, Ahmedabad to Indore) as well as for two cities that are not directly connected (say, Agartala to Hubli).
   (c) Show Uniform Cost Search and A* search on this data.
   (d) You should use Python features such as Lists, Input/ Output, Recursion, Back-tracking etc.
   (e) Your code should work for different inputs, i.e., the user should be able to input any pair of cities as the origin-destination pair from the command line and the program should return the best found path. This means you *must* not hard-code your solution for a set of pairs.
   (f) Design two different, non-trivial (e.g., a constant $h(n) = c$ is a trivial heuristic) heuristics for A*. (1) The first one should be admissible. Empirically verify that the admissible heuristic yields the optimal solution. (2) Come up with an inadmissible heuristic function that will expand fewer nodes than your admissible heuristic in (1). Show this property for *at least* two origin-destination pairs in your code.

This Property(f) can be showcased for e.g., on the following two routes:-

Hubli-Chandigarh

Delhi-Imphal