1115200600110_project1_pacman

Αριστείδης Μουτσάτσος sdi0600110

Questions 1 , 2 , 3 (DFS , BFS , A* search)

In all these search methods i used a dictionary called parent which uses the states as keys and the values are a tuple of (parent , action) . Parent is the parent state and action is a string which represents the action taken to reach the current node from the parent node. I also used a reconstruct path function with arguments the goal state and a dictionary. This function returns the list of actions taken to reach the goal state from the start state. I am using the algorithms BFS - DFS as presented in class with the difference that i change the early exit so it would pass the autograder. For A* search i also used an extra dictionary called cost with states as keys and the actual path costs (g) as values. I could have included this in the parent dictionary but i thought the code was cleaner this way. I also used a priority queue as my frontier and states are pushed with priority g + h . A state is pushed in the frontier if it hasn't been discovered yet (not in cost dict) or if its cost is less than the one already in the cost dict(i decided to use the actual cost g here other than g+h after doing several tests and reading articles on A* performance and found that this way i save a lot of calculations while my results are still correct thus increasing performance).

Question 4

State representation : a tuple of two tuples. first tuple is position (x,y) and second tuple is a tuple of tuples with the corners explored so far.

getStartState : a tuple of pacman position and an empty tuple since no corners have been visited yet.As the problem is described the starting position is not allowed to be a corner so i did not check for that

isGoalState: if state[1] (the list of corners explored so far) has a length of 4

expand: calls getNextState to create the nextstate and append it to children with the other parameters needed

getNextState: produces the next pacman position and checks if its a corner. if it is and hasn't been discovered yet its added to state[1]. Returns a tuple ((next pacman position) , (corners visited so far)). Corners visited so far is a tuple of tuples (x,y)

Question 5

Changes in __init__
I added self.start which is the starting game state of the problem and self.info which is a dictionary to store info.

Since this is a very small problem and we are looking for minimum expansions i decided to brute force it to get a good estimate.First i precompute all food real distances. Then the code produces all permutations of the unexplored corners and calculates all possible paths from pacman to goal.Finally the minimum of these paths is returned.This is also trivially consistent and admissible since it is exact.

It expands 246 nodes and there is a commented section which when uncommented it will keep pacman always in the right track expanding only the nodes of the optimal path (106 nodes).

Question 6

For foodHeuristic i used the weight of the minimum spanning tree in a complete graph where nodes are the food positions + maze distance (real distance) to the closest food from pacman.The function pre-calculates all pairs of food real distances and keeps them in the heuristicInfo dict (i used it as a nested dict for these cases). It also keeps in the dict every food list MST being calculated with the MST weight as value since this remains unchanged until pacman eats another food It expands 255 nodes. Even if this is computationally heavy and requires a lot of memorization i decided to use it since all graphs of q6 are small or sparse

Extra:
Since i wanted to also solve bigSearch there is a commented section and when uncommented it will turn the heuristic into an approximation function for a while. Stoping the approximation at 60 food is a sweet spot(in my opinion) to run in a logical time while getting a great path in bigSearch. It basically discards not promising nodes until the graph is sparse enought to be solved optimally When run like this it will solve mediumSearch in about 100 seconds finding a path of 152 and bigSearch in about 3 mins finding a path of 278 I m not sure if these are 100% optimal but even if not the are very close

Proof of admissibility

To prove admissibility lets first consider how pacman will eat the dots in an optimal way. At first he will have to travel the distance to the first dot lets call that $distance_a$ and then eat all the rest by following an optimal path of distance $distance_{opt}$. So he will travel a total of $distance_a + distance_{opt}$ and i will call that summation TrueSP . For my heurstic i find first the minimum distance from pacman to a dot, lets call that $distance_{min}$ and since this is the closest

dot we know that $distance_{min} \leq distance_a$ (1).By definition we know that "A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight" therefore we get that the MST weight lets call that $mst_w \leq distance_{opt}$ (2) . Adding (1) and (2) by parts we get

$distance_{min} + mst_w \leq distance_a + distance_opt \Leftrightarrow$

My heuristic estimate $\leq$ TrueSP

and therefore my heuristic is always an underestimate thus admissible.

Proof of consistency

I will call a parent state (a) and its heuristic estimate h(a) and a successor state (b) and and its heuristic estimate h(b). I will also call the weight of the MST mst_w To prove Consistency we need to consider 2 cases

Case 1. Pacman doesn't eat a dot in the successor state (b)

In this case the weight of the MST remains the same since pacman doesn't eat a dot and doesn't affect the MST. We have :

h(a) = distance to the closest dot (x) from node (a) (lets call that $dist_{a-x}$) + mst_w

h(b) = distance to the closest dot (y) from node (b) (lets call that $dist_{b-y}$ + mst_w

what we need for consistency is that $h(a) \leq c(a,b) + h(b)$ where $c(a,b)$ is the true cost from a to b. By replacing we have

$dist_{a-x} + mst_w \leq c(a,b) + dist_{b-y} + mst_w \Leftrightarrow dist_{a-x} \leq c(a,b) + dist_{b-y}$ which is always true since if it wasn't , when pacman is at node (a) he will pick (y) as his closest dot since he can first travel to (b) and then go from there to (y) with a shortest distance. Therefore in this case the heuristic is consistent.

Case 2. Pacman eats a dot in the successor state (b)

I will call the weight of the old MST in state (a) mst_a and the weight of the new MST when pacman eats a dot in state (b) mst_b. Also note that in order for pacman to eat a dot in state (b) he would have to be right next to it in state (a) so we can tell that $c(a,b) = 1$ for these cases. We need to again consider 2 cases

Case 1. Node (b) was a leaf node of the MST

Since (b) is a leaf node is degree of 1 and we know that in order to reconstruct the new MST all we have to do is remove the edge from (b) to a node (y) that connects (b) with the MST. Lets call that edge e_{b-y}.

We need to prove that $h(a) \leq c(a,b) + h(b)$. Since (b) contains a dot and the minimum possible distance is 1 we can say without loss of generality that the closest dot from (a) has a distance of 1 and therefore $h(a) = 1 + mst_a$. Now we know that mst_a also contains the edge e_{b-y}. Since by definition of the MST this is the shortest edge to connect b with the other dots we know that (y) is the closest dot from (b). We also know that when we remove e_{b-y} from mst_a we will be left with mst_b. So we get that :

$h(a) = 1 + e_{b-y} + mst_b \leq 1 + h(b)$ (1) but we also know that

$h(b) = e_{b-y} + mst_b$ as shown above and by replacing in (1) we get $0 \leq 0$ which is true and my heuristic is consistent in this case also

Case 2. Node (b) was a cut vertex of the MST

In this case the MST will be affected severely since if (b) is of degree n removing (b) will break the MST into n components. As shown previously we know that the MST in a will contain the edge e_{b-y} which is the closest dot from b. Lets now define mst_c which is mst_a without the edge e_{b-y}. So $h(a) = 1 + e_{b-y} + mst_c$ and $h(b) = e_{b-y} + mst_b$ where mst_b is the new reconstructed MST.And since as shown before $c(a,b) = 1$ we see that what we need to prove is

$1 + e_{b-y} + mst_c \leq 1 + e_{b-y} + mst_b \Leftrightarrow mst_c \leq mst_b$

Now we know that removing (b) is going to break the MST in n components and we also know from MST theory that we can reconstruct the MST by greedily connecting these components with n-1 edges.

mst_c has n-1 edges different than mst_b and since mst_c was constructed in a complete graph the edges chosen where the shortest possible to connect the tree otherwise other edges would have been selected for the original mst_a (minimal cut property can show that) . So if we call an edge that exists in mst_c and not in mst_b ,$edge_c$ and an edge that exists in mst_b and not in mst_c ,$edge_b$ we conclude that

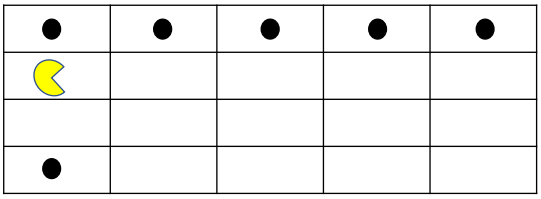
$$\sum_{i=1}^{n-1} Wedge_{c_i} \leq \sum_{i=1}^{n-1} Wedge_{b_i}$$

and since all the other edges remain the same we get that $mst_c \leq mst_b$, so consistency holds in this case also and we can conclude that this heuristic is consistent under any case.

NOTES: In my algorithm i reconstruct the MST when it changes but this doesn't affect the theory since the result will be a MST of equal weight.

Inadmissible - Inconsistent heuristics that pass the autograder

```
position, foodGrid = state
food = foodGrid.asList()
if problem.isGoalState(state):
    return 0
if len(food) == 1:
    return util.manhattanDistance(position, food[0])
sum_distances = min(util.manhattanDistance(position , x) for x in food)
first_food = food.pop(0)
while food:
    next_distance , next_food = min((util.manhattanDistance(first_food , f) , f ) for f in food)
    first_food = next_food
    sum_distances += next_distance
    food.remove(next_food)
return sum_distances
```



Consider this case. The heuristic will return a value of 12 while the true cost has of value 9. I found this while trying to design a greedy-like heuristic like in the corners problem. This works for their cases though since greedy can sometimes be consistent. The code is also commented in searchAgents.py if you like to run it. This is also graded 5/4 since it expands 6260 nodes.

2) My heuristic when used only as approximation (removing if foods ¿ 60 and aligning the other lines properly) will also pass the autograder resulting in an expansion of just 200 nodes.

And some more

Question 7

I defined the goal state in AnyFoodSearchProblem as if there is food in the graph in that position. Then the findPathToClosestDot function only needs to run a BFS since BFS will find the closest dot. A* could also do the job instead of BFS.