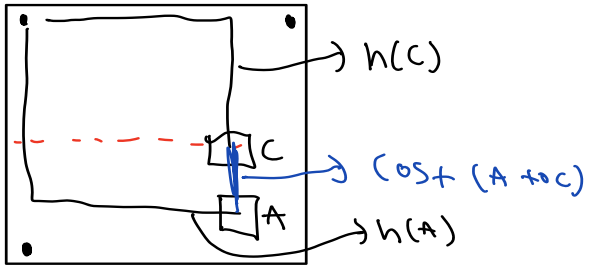
Ali Taylan Akyürek
64229

1) DFS uses a stack as frontier, however BFS uses a queue as frontier. These leads different behavior. If you use a stack, you should visit a node's child before you visit its siblings because if that node has added to stack at height n, it's child should be added at height n+1, that means you put a node's childs after you put it's sibling. In stacks, you pop the last added item first so you visit childs first, that leads you visit high depts first (like vertically if you imagine tree). But that's other way around in BFS because of First In First Out property of queue. You visit siblings first because it added first. Other differences are BFS uses more memory because it adds an entire level to its queue. But although DFS uses less memory, it's not optimal because you may visit a goal state that is in higher depth even same state exists in upper levels. That prevents optimality of solution.

2) They both uses costs to below nodes (minimum cumulative cost) when it comes to decide which node to visit next. But their main difference is that A* uses also heuristics when it comes to decide. It uses both. So, uniform cost search decides upon cost but A* decides upon sum of cumulative cost and heuristic. I use A* if I have a good heuristic, I would use UCS otherwise.

3) I used location and an empty list as state. That empty list will be filled when a corner visited. I mean when pacman visits a corner, that corner will be added to list. And I compare my state's list with actual corner list, that makes detecting goal state possible. If their length are same, that means all corners have visited and putted to list. That makes solving problem possible.

4) My algorithm basically calculates Manhattan distance of a position to all the corners and returns max. This allows me to solve the problem because heuristic means best possible approximation to goal state, and distance to most far corner makes sense because you definitely should visit that corner in order to solve problem so cost is at least that. Summing them is also a makes sense but that makes heuristic inadmissible because true cost may be lower than sum. My solution is admissible because cost of eating every dot is definitely costier than cost to most far dot. And the solution is consistent, below there is a maze that has no walls, and 4 dots, and there is Manhattan distances to most far dot.



See that cost to a to c is always equal to h(c) – h(a) in no wall situation. If there were walls, cost(A to C) will be even higher so cost(A to C) >= h(c) – h(a) for sure.

5) Algorithm looks to every dot in map and returns most far dot as heuristic value. It literally calculates real distances (it makes BFS to other dot) with mazeDistance algorithm that has implemented for us. This makes sense because it's a good approximation in order to solve problem of eating every dot. To eat every dot, pacman should at least eat most far dot. Closest dot will be inaccurate because being close to closest dot means nothing in terms of solving problem, other dots may be much far away than closest dot. But other dots cannot be farer from most far dot. It's admissible because of same reason above. And consistent because same reason above. Think that pacman is at location A. Even if pacman goes to location C first then go to goal, cost(A to C) will be equal to h(C) – h(A) because of BFS. Although, this algorithm may cause long execution times because of BFS'. That is a good example for question below.

6) Consistent heuristics always find optimal solutions, but that's not the case for inadmissible ones. But there may be inadmissible heuristics that can calculate predicted heuristic costs faster. Also, an inadmissible heuristic may result less nodes expanded if it makes more sense than consistent one. I would prefer consistent heuristic if I want an optimal solution, but I may use an inadmissible one if its faster and expand less nodes.