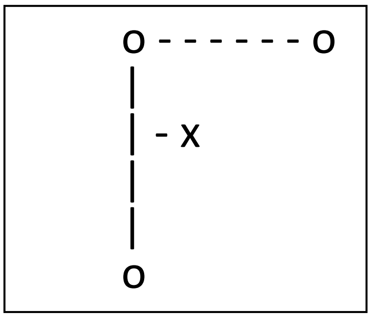
Niels Van den Broeck
S0203844

Report

<u>CornerHeuristic:</u>

For the corners problem, I quickly found an heuristic that calculates the Manhattan distance to the nearest dot, sets the players position to that dot and repeats this for all dots. The heuristic is the addition of all the distances. I thought this was admissible because there is no possibility that the actual cost could be less than this heuristic, although after some experimenting, I did find an example which proofs that the heuristic is not admissible:



o represents food, x represents the player. My heuristic calculates the path from x tot he left top corner, because that is the closest one to the player. In this case the cost is 4. Then it goes from the top left to the bottom left, which cost is 5. Finally, it goes to the last corner, which cost is 12. The heuristic returns a value of 21. There is a better path which cost is less than 21, when it goes to the bottom left corner first, to the top left and then the top right. This gives a cost of 17, which makes my heuristic inadmissible.

The solution to this is pretty straight forward. Instead of calculating the closest corner to the player, I chose the minimum of all possible permutations of paths to the corners. So that it can find the best path through all corners. This is admissible because we are still working with the Manhattan distance, which can only be equal or smaller than the actual distance (depending on walls) to the corners. This is also consistent, because every step the player takes, the heuristic can only drop by 1, stay equal, or add 1. This is because we calculate the best path and 1 step cannot lead to another path that is 2 steps ahead, otherwise that path + 1 step would have been the initial path.

<u>Foodheuristic:</u>

Taking the permutation of all dots works for a maze with few dots like 4 in the corners problem, however we cannot do that for mazes with more dots. In tricky search, there are 14 dots that the player needs to eat. Taking a permutation of 14 items gives us 87.178.291.200 combination (each time we want to calculate the heuristic). So, we got to find another solution for the foodheuristic.

I started to just add up all the distances from the dots to the player, so that the player gets punished for moving away from dots. This is not admissible because if 2 dots and the player are in the same line, the heuristic gives a higher value than the actual cost.

Then I thought about taking the average of all distances, which is admissible because the average is always lower or equal than the actual distance the player must take. This is also consistent because each step taken, the different distances can add 1, stay equal, or drop by 1. Dividing this by the number of dots will always be lower than the heuristic of the initial state + 1. This is a decent solution, it solves the tricky maze in around 11254 nodes

Niels Van den Broeck
S0203844

expanded. Although, taking the average makes the heuristic less precise. So, I started to think about an heuristic that does not work with an average.
(I commented the implementation out of the heuristic by average)

Taking only the distance of the furthest dot as the heuristic is a great solution to this. It expands only 9551 nodes. It is admissible because it can never have a value less than the actual cost since it must go to the furthest dot anyway. It is also consistent because taking one step can lead the furthest dot to be (again) 1 place further, stay the same distance or get 1 step closer. If the furthest dot switches, it can also never be more than 1 step difference, because than that dot had to be the furthest away in the original position.

In all these heuristics, I used the Manhattan distance from the player to the dots. We can improve those heuristics by using the function MazeDistance, which calculates the distance from the player to a dot while considering the walls in the maze. This gives an even more accurate heuristic which makes the tricky maze expand only 4137 nodes.

<u>MediumSearch:</u>

My heuristic is not able to solve the mediumsearch maze in a short time. There are too many foods to calculate the best path through the maze. This is similar to the traveling salesman problem and thus it is hard to solve.

<u>Suboptimal Search:</u>

In question 8, we were asked to create an agent that always eats the closest dot. This is exactly like my first implementation of cornersproblem, where it calculates the path by going towards the closest dots. Although this is not optimal.

```
o - - - o
x
|
o
```

In the following example, the player would first go up, then down, to finally reach the upper right corner, with 12 steps taken. If the player went down first (and not eat the closest dot first) it takes only 9 steps to eat all dots.

```
o o o o o
o wwww
o wwww
x o www
```

Another scenario could be crucial when walls are present, and the agent has 2 or more dots that are equally close to the player. If the player chooses to go up first, which is fine for the ClosestDotSearchAgent, it must go all the way back for the last dot in lowest column. The cost would be 15, while taking the lowest dot first, would conclude in a cost of only 9.