Pacman Game

Step 6, 7 and 8 - Corners Heuristic, Food Heuristic and Any Food Search Problem

Intelligent Systems

Master in Software Engineering

Manuel Cura (76546) | Carolina Albuquerque (80038)

universidade de aveiro
deti departamento de eletrónica, telecomunicações e informática

Context

The aim of this challenge of the Intelligent Systems course is to successfully complete steps 6, 7 and 8 of the Pacman agent, being the last phase of the project. This is only possible after the correct completion of the previous steps.

The goal was to implement two heuristics, the corners heuristic and the food heuristic, both must be admissible and consistent, also it was required the implementation of a function that finds the path to the closest dot.

The admissibility and consent criteria are:

- **admissibility:** the heuristic values must be lower bounds on the actual shortest path cost to the nearest goal and non-negative
- **consistency:** must additionally hold that if an action has cost *c*, then taking that action can only cause a drop in heuristic of at most c.

Implementation

Corners Heuristic

```python
def cornersHeuristic(state, problem):
    """
    A heuristic for the CornersProblem that you defined.

      state:   The current search state
               (a data structure you chose in your search problem)

      problem: The CornersProblem instance for this layout.

    This function should always return a number that is a lower bound on the
    shortest path from the state to a goal of the problem; i.e.  it should be
    admissible (as well as consistent).
    """
    corners = problem.corners # These are the corner coordinates
    walls = problem.walls # These are the walls of the maze, as a Grid (game.py)
    heuristic = 0
    unvisited = list(set(state[1]))
    currentNode = state[0]

    while(len(unvisited)!=0):
        distances = list()
        for corner in unvisited:
            distances.append((util.manhattanDistance(currentNode, corner), corner))
        cost, nextCorner = min(distances) # consistent condition
        heuristic += cost
        currentNode = nextCorner
        unvisited.remove(nextCorner)

    return heuristic
```

On the corners heuristic the goal is retrive the best path through all the corners, so the agent can follow the best sequence of corners. This path is generated based on the best heuristic to each corner from the pacman's current position, and after one is selected, the pacman's agent recalculates again from that position to the remaining corners, until the job is done. For each already unvisited corner, all distances are stored in a list of tuples containing (cost, corner) so that the minimum value of this list, i.e., the nearest corner, is returned. This approach can make the heuristic consistent and this subject will be addressed again with the results obtained.

The Manhattan distance is used to calculate the cost from the current position to the unvisited corners and from each unvisited corner to other univisited corners, since this heuristic presented the best results on previous testing.

Food Heuristic

```
def foodHeuristic(state, problem):
    """
    Your heuristic for the FoodSearchProblem goes here.

    This heuristic must be consistent to ensure correctness.  First, try to come
    up with an admissible heuristic; almost all admissible heuristics will be
    consistent as well.

    If using A* ever finds a solution that is worse uniform cost search finds,
    your heuristic is *not* consistent, and probably not admissible!  On the
    other hand, inadmissible or inconsistent heuristics may find optimal
    solutions, so be careful.

    The state is a tuple ( pacmanPosition, foodGrid ) where foodGrid is a Grid
    (see game.py) of either True or False. You can call foodGrid.asList() to get
    a list of food coordinates instead.

    If you want access to info like walls, capsules, etc., you can query the
    problem.  For example, problem.walls gives you a Grid of where the walls
    are.
    """
    position, foodGrid = state
    uneaten = foodGrid.asList()
    heuristic = 0

    while(len(uneaten)!=0):
        heuristicsList = list()
        for food in uneaten:
            distance = util.manhattanDistance(position, food)
            heuristicsList.append((distance, food))
        cost, nextFood = min(heuristicsList)
        heuristic += cost
        position = nextFood
        uneaten.remove(nextFood)

    return heuristic
```

The food heuristic follows the same logic as the corners heuristic, but instead of reaching the corners it considers all the uneaten food.

Both implemented heuristics are consistent and admissible because they always return a value that is the minimum cost found, and try to stay close to the real cost, always following the shortest and minimum cost path.

Closest Dots Problem and Any Food Search

The closests dots problem is solved using the ClosestDotSearchAgent. This agent always greedily eats the closest dot and depends on *findPathToClosestDot()* function to perform this approach. This implementation searches through an instance of the AnyFoodSearchProblem problem.

```
def findPathToClosestDot(self, gameState):
     """
     Returns a path (a list of actions) to the closest dot, starting from
     gameState.
     """
     # Here are some useful elements of the startState
     startPosition = gameState.getPacmanPosition()
     food = gameState.getFood()
     walls = gameState.getWalls()
     problem = AnyFoodSearchProblem(gameState)
     return search.aStarSearch(problem)
```

It was chosen A* algorithm for searching the problem due to its positive results during the last implementations of pacman's game. However, AnyFoodSearchProblem does not have the isGoalState function implemented, for that purpose the function returns if the state is in fact a food.

```
def isGoalState(self, state):
     """
     The state is Pacman's position. Fill this in with a goal test that will
     complete the problem definition.
       """
     x,y = state
     return (x,y) in self.food.asList()
```

The entire code is available on GitHub repository.

Results Analysis

Corners Heuristic

The results obtained with the corners heuristic implemented are shown below as well as the order of visiting the corners.

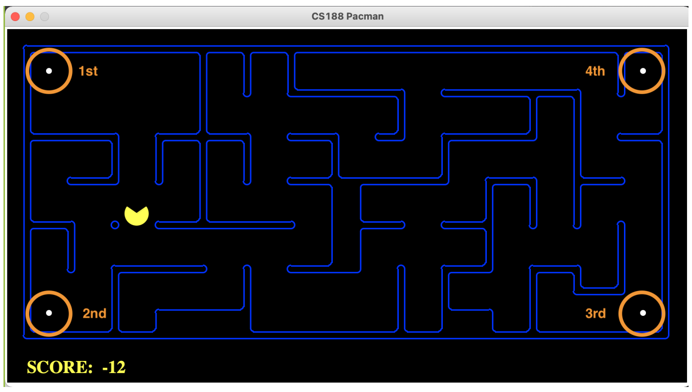
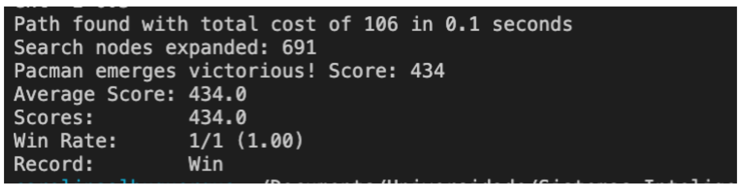


Figure 1 **-** Order of the corners visited by the agent using the Corners Heuristic implemented in mediumCorners map



```
Path found with total cost of 106 in 0.1 seconds
Search nodes expanded: 691
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figure 2 **-** Results obtained in mediumCorners map using the Corners Heuristic

As seen in Figure 1, Pacman agent visits corners closest to the last eaten food. This strategy is the same for the other maps, and below is the example of bigCorners maze.

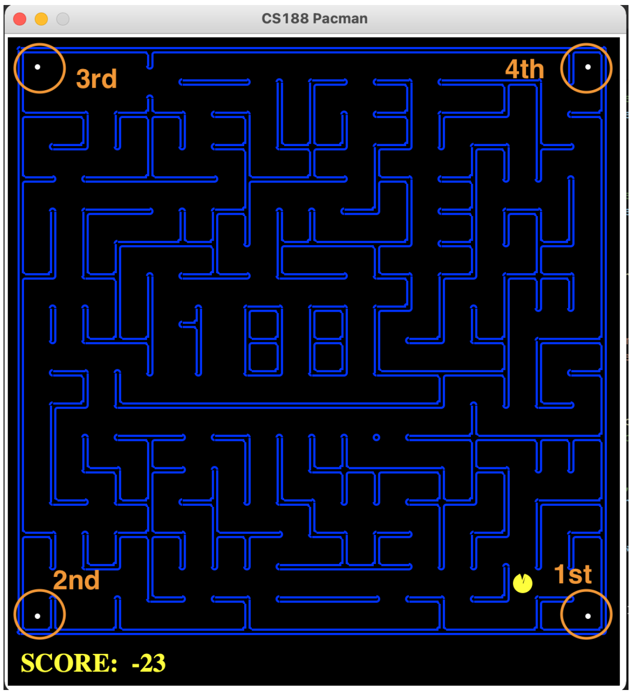
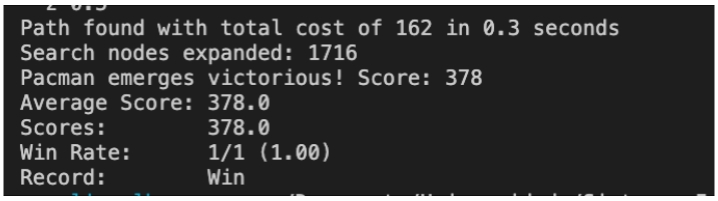


***Figure 3* -** Order of the corners visited by the agent using the Corners Heuristic implemented in bigCorners map



```
Path found with total cost of 162 in 0.3 seconds
Search nodes expanded: 1716
Pacman emerges victorious! Score: 378
Average Score: 378.0
Scores:        378.0
Win Rate:      1/1 (1.00)
Record:        Win
```

***Figure 4* -** Results obtained in bigCorners map using the Corners Heuristic

In order to test the performance of the corners heuristic implemented, firstly the problem was tested returning a null heuristic to serve as a basis for comparison. Returning a null heuristic, corners heuristic considers a uniform-cost search where *h(n)* = 0, i.e. uniform-cost search is *g(n)*. Comparing this null heuristic with the corners heuristic implemented it is clear that the last one is significantly faster and finds the least cost path expanding fewer nodes. For this reason, the heuristic implemented is considered admissible due to h(n) =< g(n) as seen in the next Table and it was tested in several mazes which contain food in its corners.

Map	Null Heuristic			Corners Heuristic			Reduction of Nodes
	Cost	Expanded Nodes	Time	Cost	Expanded Nodes	Time	
tinyCorners	28	243	0.0s	28	153	0.0s	37%
mediumCorners	106	1921	0.3s	106	691	0.1s	64%
bigCorners	162	7862	1.6s	162	1716	0.3s	79%
openMaze	116	7200	2.5s	116	1319	0.3s	82%

Table 1 - Comparison between null heuristic performance and Corners Heuristic implemented

The corners heuristic implemented is consistent because it returns optimal cost solutions, i.e, the cost returned by the corners heuristic is equal to the cost returned by the null heuristic. In fact, it returns the least cost solution with an incredible performance especially in larger maps. For example, for bigCorners maze, the heuristic reduces expanded nodes by 79% to achieve the least cost solution and the time spent during the search process is drastically lower. There is also a marked reduction in the time spent by the agent during the search.

Food Heuristic

The results of the food heuristic implementation are described below.

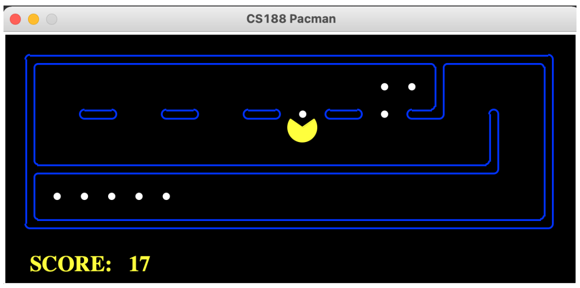


Figure 4 - *Path taken by agent in trickySearch maze*

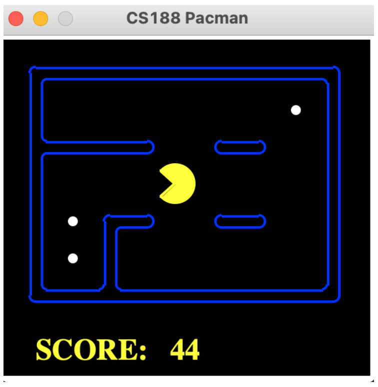
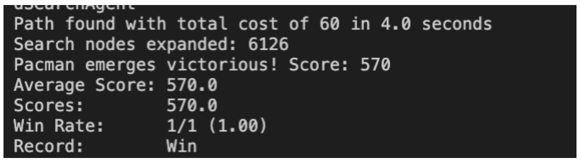
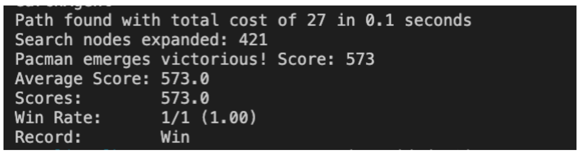


Figure 5 - *Path taken by agent in tinySearch maze*



```
Path found with total cost of 60 in 4.0 seconds
Search nodes expanded: 6126
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figure 6 - *Results of Food Heuristic in trickySearch maze*



```
Path found with total cost of 27 in 0.1 seconds
Search nodes expanded: 421
Pacman emerges victorious! Score: 573
Average Score: 573.0
Scores:        573.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Figure 7 - *Results of Food Heuristic in tinySearch maze*

Similar to corners heuristic, the food heuristic was tested too with a null heuristic in order to serve as a basis for comparison.

Map	**Null Heuristic**			**Food Heuristic**			**Reduction of Nodes**
	Cost	**Expanded Nodes**	**Time**	**Cost**	**Expanded Nodes**	**Time**	
testSearch	7	13	0.0s	7	12	0.0s	8%
tinySearch	27	4627	1.6s	27	421	0.1s	91%
trickySearch	60	15878	7.7s	60	6126	4.0s	61%
smallSearch	34	61511	541.2s	34	60	0.0s	99.9%
greedySearch	16	567	0.1s	16	37	0.0s	93%
tinyCorners	28	243	0.0s	28	153	0.0s	37%
mediumCorners	106	1921	1.8s	106	691	0.4s	64%
bigCorners	162	7862	14.6s	162	1716	2.7s	78%

Table 2 - Comparison between null heuristic performance and Food Heuristic implemented

Note: For mediumSearch and bigSearch mazes, the null heuristic spent too much long running time, so that it was interrupted the search before obtaining results

As expected, the implemented food heuristic returns the least cost solution (it is consistent) and increases the agent's performance both in expanded nodes and time compared to the null heuristic. In several mazes, the food heuristic reduces by at least 90% of the expanded nodes needed to find the optimal solution.

Closest Dots Problem

The closest dots problem was tested several times varying the search algorithm used in order to compare the performance between all already implemented algorithms. For example, using depth-first search algorithm, the results obtained are the following:

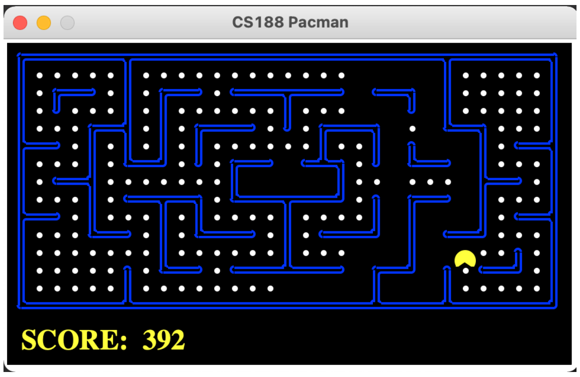


***Figure 8** - Path taken by CloseDotsSearchAgent in bigSearch maze*

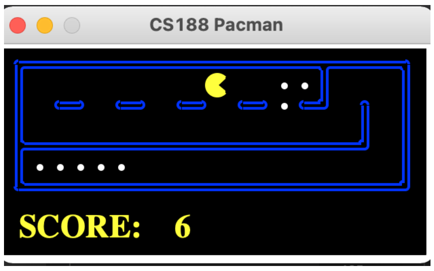
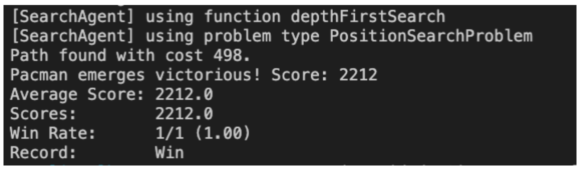
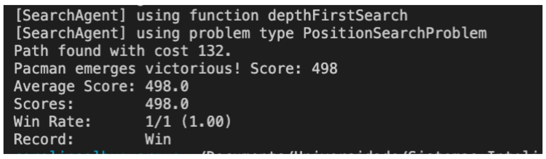


***Figure 9** - Path taken by CloseDotsSearchAgent in trickySearch*



```
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with cost 498.
Pacman emerges victorious! Score: 2212
Average Score: 2212.0
Scores:        2212.0
Win Rate:      1/1 (1.00)
Record:        Win
```

***Figure 10** - Results of CloseDotsSearchAgent in bigSearch maze*



```
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with cost 132.
Pacman emerges victorious! Score: 498
Average Score: 498.0
Scores:        498.0
Win Rate:      1/1 (1.00)
Record:        Win
```

***Figure 11** - Results of CloseDotsSearchAgent in trickySearch*

As seen before, the agent is able to eat all nearby foods scattered across the maps. The closest dots problem can be addressed by varying the search algorithm used to get the actions for the nearest food. Therefore, the problem was tested by varying the search algorithm and the results obtained are available below.

Map	Depth-First		Breadth-First		Uniform-Cost		A*	
	Cost	Score	Cost	Score	Cost	Score	Cost	Score
testSearch	7	513	7	513	7	513	7	513
tinySearch	41	559	31	569	31	569	31	569
trickySearch	132	498	68	562	68	562	68	562
smallSearch	66	604	48	622	48	622	48	622
mediumSearch	295	1285	171	1409	171	1409	171	1409
bigSearch	498	2212	350	2360	350	2360	350	2360
tinyCorners	49	491	32	508	32	508	32	508
mediumCorners	209	331	106	434	106	434	106	434
bigCorners	302	238	162	378	162	378	162	378

***Table 3* -** Comparison between all search algorithms applied to Closest Dots problem

In a first analysis, as expected, the depth-first search algorithm does not present an optimal solution compared to the others. The remaining algorithms present exactly the same results to each other for the different maps. In addition, it is clear that the score obtained is influenced by the cost of the solution obtained, that is, optimal costs will obtain a better score against less optimal costs. However, the costs obtained stand out negatively compared to the costs obtained for food heuristic, for example. With this, it was decided to analyze map by map which is the least cost solution and which implementation returns the least associated cost. The result is described in the following Table.

Map	Least Cost	Food Heuristic		Closest Dots (Using A* search)	
		Cost Returned	Returns the Least Cost	Cost Returned	Returns the Least Cost
testSearch	7	7	✓	7	✓
tinySearch	27	27	✓	31	✗
trickySearch	60	60	✓	68	✗
smallSearch	34	34	✓	48	✗
tinyCorners	28	28	✓	32	✗
mediumCorners	106	106	✓	106	✓
bigCorners	162	162	✓	162	✓

Table 4 - Least cost solution analysis between Food Heuristic and Closest Dots using A* search

The mediumSarch and bigSearch mazes were not considered for the previous analysis due to the lack of results for food heuristic as seen before. It is clear that closest dots do not always return the least cost solution unlike food heuristic. Thus, food heuristic is considered an optimal option allowing pacman agents to eat all dots with a least cost solution.

Conclusions

The implementation of the two heuristics allowed a significant optimization regarding the expanded nodes and time spent by the agent in searches. Besides corners heuristic is considered good to reach food in all corners, when a map has more dots beyond those dots in the corners, pacman agent cannot eat those ones ending up dying. Thus, the food heuristic becomes more complete due to the agent having a greater perception of the behavior of the map. In addition, as seen in the results, the closest dots problem does not always return optimal solutions regardless of the search algorithm it uses. In conclusion, eating the closest dots is not the better solution as shown.