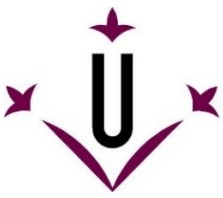
Artificial Intelligence

Practice 1 – Search algorithms with *Pac-Man*

Pere Muñoz Figuerol



Universitat de Lleida

Table of contents

Table of figures

Table of tables

1 Search algorithms

For the implementation of the search algorithms, I've followed the pseudocodes of the slides seen at class.

But if I must remark some important change, I will say the final part of the *UCS* and *A** algorithms.

In the [Fig 1] we can see the original checks for every successor node we generate:

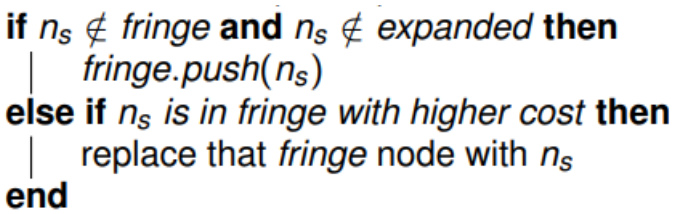


Fig 1. Original successor node checks

But if we take advantage of one function implemented for the *Priority Queue*, called *update()*, we can optimize the above code to the below one [Fig 2]:

```
if state not in expanded:
    fringe.update(successorNode,
    priorityOfSuccessorNode)
```

Fig 2. Optimized successor node checks

As we can see, we only must worry about not having the successor node in expanded, because the *update* function checks all the other requirements internally and acts consequently.

I think doing it this way the code looks more readable, compact, and comprehensive.

2 Heuristics' design

In this section, I will describe the implemented heuristics, for the Corners Problem and Food Problem.

2.1 Corners Heuristic

The Corners Heuristic is designed for the Corners Problem.

The Corners Problem consists of having four foods in the map, one at every corner of it. The objective is to eat all the four foods.

For the design of the heuristic, first I calculate the Manhattan distance between the actual position and the closest unvisited corner of the map. Once calculated this distance, we add to it all the distances from each corner to its closest corner.
For a better understanding of the process, I visually represent it with a diagram [Fig 3]:

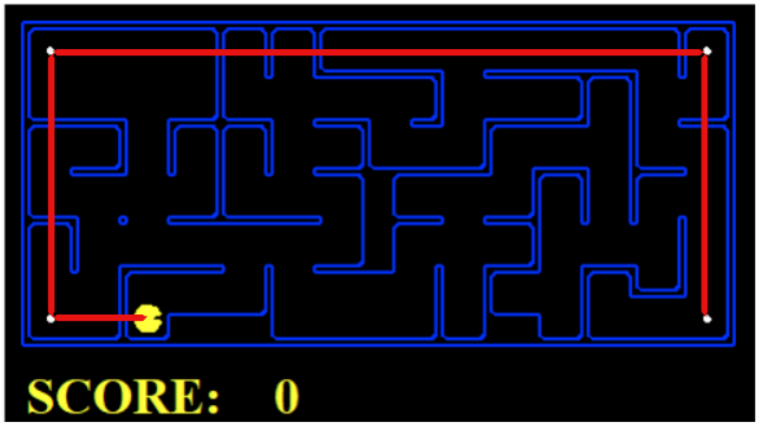


Fig 3. Visual representation of Corners Heuristic at the initial state

The red lines are every distance that added all together they create the state heuristic. This calculation is done for every state. Every time a corner is visited (means that *Pac-Man* ate the corner's food already) it's removed from the calculation of the next states for not overestimating the cost (shown in [Fig 4]).

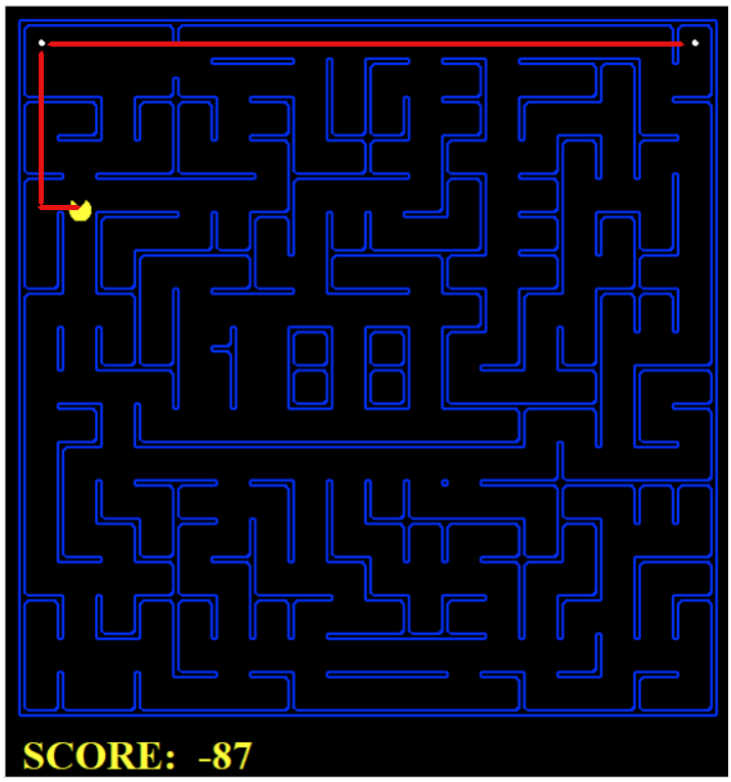


Fig 4. Visual representation of Corners Heuristic at a random middle state

2.2 Food Heuristic

The Food Heuristic is designed to solve the Food Problem.

The Food Problem consists of having any amount (>1) of foods spread out throughout the map. The objective, same as the Corners Problem, is to eat all the foods.

The design of this heuristic is so like the Corner's one, but instead of calculating the distance to each corner, now we calculate the Manhattan distance from the actual position to the closest food, and then we add to it the distance of the closest food to its closest food, and so on… We do it recursively, like in the previous heuristic.

For a better understanding of the calculation, it's shown graphically in the [Fig 5]:

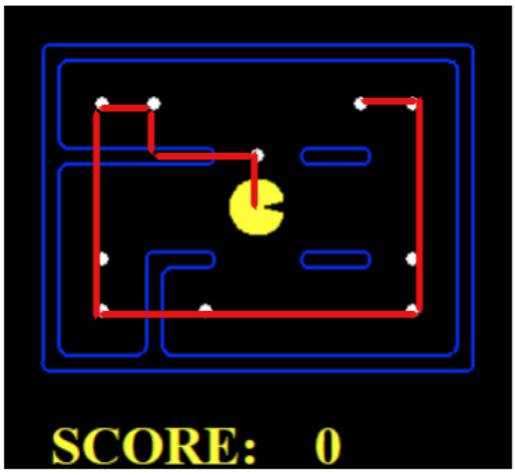


Fig 5. Visual representation of Food Heuristic at the initial state

Same as the previous one, the red lines are the distances that if we add all up together, we get the heuristic value of that specific state.
As the *Pac-Man* eats some food, the heuristic value is calculated according to the foods left on the map for not overestimating the cost (shown in the [Fig 6]).

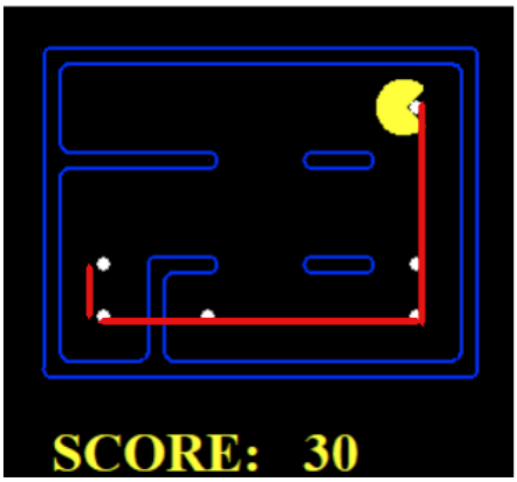


Fig 6. Visual representation of Food Heuristic at a random middle state

3 Heuristics' performance

After the design of the two heuristics, it comes the time to test them.
In this section we are going to analyze the performance of the two heuristics and compare them to the trivial heuristic or *UCS* results.

3.1 Corners Heuristic

After the execution using *OptiLog*, the results have been these [Table 1]:

MAP	Value measured	A* with Corners Heuristic	UCS or A* with trivial heuristic
tinyCorners	Cost	28	28
	Expanded Nodes	165	252
mediumCorners	Cost	106	106
	Expanded Nodes	1153	1966
bigCorners	Cost	162	162
	Expanded Nodes	2933	7949

Table 1. Comparison of the Corner Heuristic and the trivial heuristic

From the results, we can extract that the designed heuristic improves the computational cost of the UCS. The number of expanded nodes reduces significantly when executing the maps with the heuristic designed. The average reduction is almost of 2 times respect the *UCS* execution.

Being curious about these numbers, and how to improve them, I implemented by myself some other heuristics (that are not admissible) that improves a lot of these results.
But as you asked for a consistent heuristic, these one is the best I could design.

3.2 Food Heuristic

After the execution of the heuristic in different maps, the results have been these [Table 2]:

MAP	Value measured	A* with Food Heuristic	UCS or A* with trivial heuristic
greedySearch	Cost	16	16
	Expanded Nodes	178	692
smallSearch	Cost	34	34
	Expanded Nodes	8345	70726
testSearch	Cost	7	7
	Expanded Nodes	12	14

tinySearch	Cost	27	27
	Expanded Nodes	1851	5057
trickySearch	Cost	60	60
	Expanded Nodes	9791	16688
mediumDottedMaze	Cost	74	74
	Expanded Nodes	1275	3696
bigCorners	Cost	162	162
	Expanded Nodes	2933	7949
mediumCorners	Cost	106	106
	Expanded Nodes	1153	1966
tinyCorners	Cost	28	28
	Expanded Nodes	165	252

Table 2. Comparison of the Food Heuristic and the trivial heuristic

In the table above [Table 2] only appears the maps that its execution finished at least for one of the algorithms in 5 minutes.

As we can see, the results with the heuristic designed improved a lot. The execution with our heuristic reduces an average of 4 times the expanded nodes from the execution of the trivial heuristic or *UCS*. I think that's a huge improvement.