Report for Project 1: Search in Pacman

Abstract
This report explains how Pac-man agent finds paths through his maze world to reach a particular location and to collect food efficiently. It will introduce the generic search algorithm(for DFS, BFS, UCS, and A*) first, and then step into the development of related heuristic functions with proved admissibility and consistency. Some pertinent design details are also included.

Contents

1 Generic Search Algorithm 1
Q1: Depth First Search 1
Q2: Breath First Search 1
Q3: Varying the Cost Function 2
Q4: A* Search 2
2 Heuristic Functions! 2
Q5: Corners Problem: Representation 2
Q6: Corners Problem: Heuristic 3
Method 1 • Method 2 • Method 3
Q7: Eating All The Dots 3
Method 2 • Method 3 • Inconsistent Methods for mediumSearch
Q8: Suboptimal Search 4

1. Generic Search Algorithm

For the first 4 questions, a full-fledged generic search functions is implemented to help Pac-man plan out a path. It roughly follows the pseudo-code given in the lecture slides, with some additional parameters control the function to manifest differences between search algorithms. Since each algorithm is quite similar except for the details of how the the fringe is managed, so the function "genericSearch" accepts parameters $(problem, thefringe, dfs, bfs)$ (dfs=$True$ and bfs=$False$ by default) to pass in different data structures for the fringe. All the structures are imported from $util.py$.

For these 4 questions the state space size is literally the number of possible positions for Pac-man(positions out of the walls). The algorithm steps for DFS, UCS and A* in graph search form are listed as follow, and some of them have different orders in BFS, which will be specified later.

1. Select and goal test a new note from the fringe
2. Expand it, adding unvisited child into the fringe
3. Repeat step 1&2 until the fringe is empty

Q1: Depth First Search

DFS is an **Uninformed** search algorithm. Its strategy is to expand a deepest node first, with the help of a **LIFO** data structure. Here **Stack** is used to store the fringe, and some test results are shown below.

Table 1. Some Test Results of DFS

Maze	Cost	Expanded	Score
tinyMaze	10	15	500
mediumMaze	130	146	380
bigMaze	210	390	300

Note that although the idea behind DFS is straightforward, this algorithm can't guarantee completeness and optimality if the maximum search depth is infinite. The implementation of DFS in graph search form for Q1 limits the possible depth of the algorithm by the size of the maze, while its search strategy(deepest node each time) still keeps itself from finding the least cost solution.

Q2: Breath First Search

BFS is an **Uninformed** search algorithm. Its strategy is to expand a shallowest node first, with the help of a **FIFO** data structure. Here **Queue** is used to store the fringe. and some test results are shown below. As an aside, this implementation also solves the eight-puzzle search problem.

Table 2. Some Test Results of BFS

Maze	Cost	Expanded	Score
tinyMaze	8	15	502
mediumMaze	68	267	442
bigMaze	210	617	300

Pertinent Implementation Details It should be pointed out that, for BFS, the parameters passed in $genericSearch$ are $(problem, thefringe, dfs = False, bfs = True)$. Generally, the space the fringe takes is roughly the last tier, but the time to do goal test will make a change. Suppose b is the branching factor and s is the solution at various depths. If goal test the node while expanding, the fringe will take $O(b^{s+1})$. However, if goal test the node while generating, that is before it's added into the fringe, the fringe will take $O(b^s)$. The algorithm steps we takes here for BFS in graph search form is listed as follow.

1. Select a new note from the fringe and expand it
2. Goal test unvisited children, add them into the fringe
3. Repeat step 1&2 until the fringe is empty

Comparison Note that the nodes expanded in BFS is more than DFS, however the final cost is less. Remember BFS is complete and optimal, therefore the path found has least cost.

Q3: Varying the Cost Function

Sometimes Pac-man is encouraged to find different paths in specific scenarios, like ghost-ridden areas or uneven food-rich areas, which gave birth to a new algorithm, UCS. Instead of expanding the node with the shallowest depth, UCS will expand the cheapest node first, thus ensuring that the optimal solution can be found regardless of the different costs at each step by changing the cost function. It is also an **Uninformed** search algorithm. Here **Priority Queue with Function**(function: cumulative cost) is used to store the fringe, and some test results are shown below.

Table 3. Some Test Results of UCS

Maze	Cost	Expanded	Score
medium	68	269	4422
mediumDotted	1	186	646
mediumScary	68719479864	108	418

Note that it got very low and very high path costs for the StayEastSearchAgent and StayWestSearchAgent, due to their exponential cost functions, $g((x,y)) = \frac{1}{2^x}$ and $g((x,y)) = 2^x$ respectively.

Q4: A* Search

Often there's some other knowledge about nodes, and this information can be used to boost performance. A* is an **Informed** search algorithm, since it takes into account both past and future costs, $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the path represented by node n, $h(n)$ is the heuristic estimate of the cost of achieving the goal from n. Its strategy is to expand a "best" node-nodes with least value of $f(n)$-first. Here **Priority Queue with Function**(function: $g(n) + h(n)$, $h(n)$ is the manhattan distance) is used to store the fringe, and the comparison between UCS and A* is shown below.

Table 4. Comparison between A* and UCS

Alg	Maze	Cost	Expanded	Time
UCS	bigMaze	210	620	0.1s
A*	bigMaze	210	**549**	**0.0**s

Note that A* finds the optimal solution slightly faster than uniform cost search. Remember instead of expanding in all "directions" as UCS, A* expands mainly toward the goal, however hedges its bets by admissible and consistent heuristic functions to ensure optimality.

Open Maze Below are the results of different algorithms on openMaze. It's obvious that A* has the best performance. Also notice that the nodes expanded by BFS and UCS are similar, which is reasonable since UCS is similar to BFS when the steps have the same cost.

Table 5. Comparison on Open Maze

Alg	Cost	Expanded	Score
DFS	298	576	216
BFS	54	679	456
UCS	54	682	456
A*	54	**535**	456

2. Heuristic Functions!

The real power of A* will only be apparent with a more challenging search problem. In the following 4 questions, 2 new problems were formulated for Q5 6 and Q8 respectively. Several heuristic functions were designed for Q6 and Q7 to gain better performance.

Q5: Corners Problem: Representation

The new search problem is to find the shortest path through the maze that touches all four corners. This problem requires a state representation that encodes all the information necessary to detect whether all four corners have been reached. The definitions are as below

- **States** : A nested tuple $((x,y),T,T,T,T)$. The first tuple (x,y) where x,y are integers is the current position of Pac-man, and the following $boolean$ variables are flags that denote whether the ith corner is visited.
- **InitialState** : $((x_0,y_0),F,F,F,F)$, Where (x_0,y_0) is the starting position of Pac-man.
- **GoalTest** : Whether the flags in the tuple are all $True$

It should be mentioned that the number and order of corners' flags depend on the built-in variable, $self.corners$, which is (bottom-left, top-left, bottom-right, top-right) for Q5 6 specifically, that is already defined in class $CornersProblem$ by the instructors. For example, $((1,1),True,True,False,False)$ means for now Pac-man is at postion $(1,1)$, with bottom-left and top-left corners already visited. The results of running BFS on the newly formulated problem is shown below.

Table 6. BFS on 3 Corners Mazes

Maze	Cost	Expanded	Score
tinyCorners	28	243	512
mediumCorners	106	1921	434
bigCorners	162	7862	378

Q6: Corners Problem: Heuristic

For the problem formulated in Q5, some heuristic functions were put forward to boost performance. However, to maintain the optimality requires not only admissibility but also consistency when it comes to graph search problems. The results below show the difference between 3 heuristic functions, which will be introduced in details.

Table 7. Different Heuristic Functions for CornersProblem

Heutistic	Maze	Cost	Expanded	Time
Method1	medium	106	1136	0.0s
Method2	medium	106	383	0.7s
Method3	medium	106	**365**	0.5s

Admissible heuristics are usually also consistent, especially if they are derived from problem relaxations. Therefore it is usually easiest to start out by brainstorming admissible heuristics with relaxed problems.

Method 1

This method will return maximum Manhattan distance from current position to the untouched corners as the heuristic value.

$$H_1 = \max\left\{\text{manHattan}(P,C_i)\right\},\ C_i \in \mathbb{C} \tag{1}$$

where P denotes the current position, $\mathbb{C}$ the (x_{axis}, y_{axis}) set of all untouched corners.Problem relaxation includes two steps.

1. All the walls were ignored (Manhattan distance)
2. Consider only the farthest untouched corner (max)

Notice that the real distance between two position on the maze is surely larger than or equal to their Manhattan distance, which also indicates that heuristic "arc" cost $\leq$ actual cost for each arc(arc: the path from one successor to another successor), thus guaranteeing the **Consistency**. Therefore, it's also **Admissible**. Besides, only consider the distance between current position and the farthest corner also contributes to the admissibility and consistency of this function. Finally, choosing *farthest* rather than *nearest* unvisited corner gives out a cost relatively closer to the actual goal cost, making the function an **Effective** one.

Method 2

This method will return maximum actual distance between untouched corners plus the minimum actual distance from current position to one of the corners as the heuristic value.

$$H_2 = \max\left\{g(C_i,\ C_j)\right\} + \min\left\{g(P,\ C_k)\right\},\ i \neq j \tag{2}$$

where $C_i, C_j.C_k \in \mathbb{C}$, $\mathbb{C}$ and P are defined as above. g is the function that calculates the least cost(here is the same as length of actual shortest path) from one given position to the other, and when $|\mathbb{C}| \leq 1$, $g(C_i,\ C_j) = 0$, with $g(P,\ C_k) = 0$ if $|\mathbb{C}| = 0$. The problem relaxation here is that this function only considers the maximum distance between corners(when No. of untouched corners ≥ 2) and the least cost from current position to any unvisited corner, rather than taking all unvisited corners into account. The proof for the admissibility and consistency are as follow.

Admissibility *Proof*. The scenarios can be discussed one by one. **Firstly**, suppose all the corners are visited and it's obvious that the heuristic value is 0. **Secondly**, suppose there's one corner left, then $H_2 = \min\{g(P,\ C_k)\}$, which is exactly the actual cost from current position to the goal. **Thirdly**, suppose there's more than one corner left to be visited. Since for the corner problem, Pac-man must visit all the corners to get to the goal, so $\max\{g(C_i,\ C_j\}$ is sure to be part of the actual cost. And Pacman must select a way from current position to one of the unvisited corners, so the actual cost must be larger than or equal to $\min\{g(P,\ C_k)\}$. Thus it's admissible.

Consistency *Proof*. Suppose P_a is a successor better than P_b, and we want to prove $H_2(P_a) - H_2(P_b) \leq g(P_b) - g(P_a)$. **Firstly**, consider the tricky scenario that P_a is exactly a corner. Then after stepping into P_a, how much decreases in $\max\{g(C_i,\ C_j\}$ will finally increase in $\min\{g(P,\ C_k)\}$, just sparing few costs no more than actual movement. **Secondly**, suppose neither of P_a nor P_b are corners, than previous inequality can be rewritten into $\min\{g(P_a,\ C_k)\} - \min\{g(P_b,\ C_k)\} \leq g(P_b) - g(P_a)$, since $\max\{g(C_i,\ C_j\}$ is the same for both P_a and P_b. For simplicity we only consider the straight line distance between nodes during proof. If they have the same C_k, then according to some basic geometry-the difference between the two sides of a triangle is greater than the third side-the inequality obviously holds. If they have different C_ks, this is due to their relatively center positions and we get $\min\{g(P_a,\ C_k)\} - \min\{g(P_b,\ C_k)\} = g(P_b) - g(P_a)$. Thus it's consistent.

Method 3

This method will return maximum actual distance between untouched corners plus the minimum actual distance from current position to one of these 2 corners as the heuristic value. It is inspired by Q7 and will be introduced in details later for its university.

Q7: Eating All The Dots

This question requires eating all the Pac-man food in as few steps as possible in a newly defined problem. Some results of consistent heuristic functions are listed below.

Table 8. Results of Some Consistent Methods

Heutistic	Maze	Cost	Expanded	Time
Method0	TrickySearch	60	9551	1.3s
Method1	TrickySearch	60	4137	0.7s
Method2	TrickySearch	60	719	0.3s
Method3	TrickySearch	60	**168**	**0.1**s
Method4	TrickySearch	60	376	**0.1**s

Method 0 is the same as Method 1 in Q6, and it doesn't performed effectively enough. Method 1 returns the maximum actual distance from current position to one of the unvisited food. Its admissibility and consistency is obvious, and since it's a tighter lower bound for real cost so it's also efficient enough to get full credits.

To further reduce the No. of nodes expanded and the time spent, some other methods were put forward.

Method 2

This method is the general form of Method 2 in Q6. The only difference is it calculates the maximum actual distance from food to food rather than corner to corner. Notice it really reduces the nodes expanded greatly.

$$H_2 = \max\left\{g(F_i,\ F_j)\right\} + \min\left\{g(P,\ F_k)\right\},\ i \neq j \qquad (3)$$

where $F_i, F_j.F_k \in \mathbb{F}$, the set of unvisited food, and P and g are defined as above.

Admissibility *Proof*. The only distinction here is for the food eating problem, $\max\{g(F_i,\ F_j)\}$ isn't sure to be part of the actual cost, since the goal in this problem doesn't have the same special feature as in the corner problem. However, with the same characteristic that all food should be eaten and again the knowledge from elementary geometry(playing with the sides of one triangle), it's easy to find that $\max\{g(F_i,\ F_j)\}$ is always less than or equal to the actual cost because Pac-man will either follow this path or take a detour to get to the other food once it has eaten one of them. So it's admissible.

Consistency *Proof*. The proof for Consistency is similar to the proof for Method 2 in Q6 once substituting all C_i, C_j, C_k for F_i, F_j, F_k. Thus it's consistent.

Method 4 This method is really similar to Method 2, except that it will consider smaller actual distance between current position to one of the food that have the maximum actual distance. Since it's a tighter lower bound for the actual cost, it's pretty efficient, and its admissibility and consistency are also obvious. Also notice that this implementation makes it do less calculations of distance each time than Method 2.

Method 3

This method is an extension of Method 2, since it only additionally considers the minimum actual distance from food to food. It will lose its admissibility once the distribution of food is extremely special, like all the food lining up in a row etc.. Luckily, for mazes with a size smaller than mediumSearch, none of them have the special cases. It's better to check whether tricky cases have happened, however due to the reason mentioned this was omitted for simplicity.

Inconsistent Methods for mediumSearch

Unluckily, none of the methods above could solve the mediumSearch, for the solution would increase exponentially with its relatively large space size. However, some inconsistent methods were developed just for exploration.

Method 2.5 is an extension to Method 2, and it will replace the maximum actual distance from food to food with the sum of all the actual distances between food. **Method 5** wants to solve mediumSearch through dividing this problem into smaller sub-problems. It picks out some "Tricky Points(TPs)" which is defined as the points that have 3 walls around it, and send them into cornersProblem to solve(the space state in cornersProblem has already been modified for this purpose; Note that the function *getStartState* in corners problem will reset *self.StartState* each time it's called).

Table 9. Results of 2 Inconsistent Methods

Heutistic	Maze	Cost	Expanded	Score	Time
Method2.5	medium	159	159	1421	2.9s
Method5	medium	157	34846	1423	27.2s
Method2.5	big	292	299	2418	48.8s
Method5	big	Fails	Fails	Fails	Fails

Q8: Suboptimal Search

In this problem Pac-man will find a reasonably good path(but may not be optimal), quickly. That is, it always greedily eats the closest dot. *ClosestDotSearchAgent* will find the closest dot with BFS, UCS, A*(manHattan) or A*(euclidean), and use *AnyFoodSearchProblem* to solve it. The state space and goal test for it are the same as in the PositionSearchProblem. The results are listed below.

Table 10. Results of Q8

Maze	Cost	Score
medium	165	1415
bigSearch	334	2376

Since the optimal solution to sub-problem may not be included in the optimal solution to the whole problem in this scenario, Greedy Algorithm fails to find the best solution. One can also find that it's impossible for the greedy agent to find the optimal solution in the specific case below.

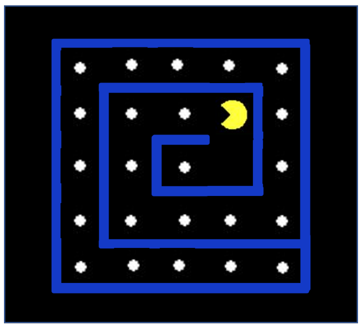

Figure 1. A Terrible Case for Greedy Algorithm