Report for Search in Pacman

Ziqin Luo 18307130198

October 11, 2020

Contents

1 Framework of Search Algorithms and Details of the Implementation

Since the differences among DFS, BFS, UCS and A* only exist in the way of how to pop out the nodes from fringe, all of these four algorithms have similar programming structure. Thus, I will briefly describe the framewokr of the four algorithms I wrote in `search.py`

Algorithm 1 Framework of Search Algorithms in PacMan Game

Input: *problem*: specific problem to be resolved; h: Heuristic function (just used in A*);
Output: P: a list of actions that will lead the agent from the start to the goal

define a data structure to represent the states in *problem*
according to *problem*, build a appropriate container *Fringe* (Stack, Queue or Priority Queue) and a set *explored* to record the visited states
get the start state from *problem* and push it into *Fringe*
while *Fringe* is not empty **do**
    pop out the best state s to be visited/expanded
    if s is visited before **then**
        ignore s (never expand a state **twice**)
    else
        add s into *explored*
    end if
    if s is Goal **then return** the target path P by s
    end if
    push s's unvisited adjacent states and their satellite data into *Fringe*
end while

I want to list some common variables or structures in my implementations:
Node: denotes state and store its satellite data.
Fringe: store the nodes (or states) have been observed but not visited.
explored: store the nodes (or states) have been visited. (To implement the graph algorithm)

Graph algorithm will check whether the state we focus on is in ***explored*** both in the procedures of pop and push. For the check in pop, we avoid to expand a state twice. For the check in push, we avoid to push the visited state into ***Fringe***. Although the second one can be omitted, it can help us do less redundant operations. We should note only when a state is poped out for the first time, can we add it to ***explored*** because that's why the search criterions (depth, path cost, f value in A*, etc) holds.

As for the heuristic h, as long as it is **admissible** and **consistent**, then Graph Search will hold the optimality if the search algorithm is optimal itself.

From the overlay of states explored show in PacMan board, we can see that **DFS can't guarantee the optimality** especially in the **openMaze** case. In this case, PacMan only get 212 points using the DFS search strategy while the score becomes 456 when using BFS,UCS or A* That's because DFS always try to find the " left" most solutin regardless of the corresponding depth or cost. In contrast to DFS, under the settings of PacMan, BFS, UCS and A* can hold the optimality.

2 Settings of Corners Problem and Suboptimal Search

To complete the definition of the Corners Problem, we have to select a suitable state space and successor function for it. Now that we care both the position of PacMan and the state of each corner, we can put them together and form a tuple with two elements corresponding to position and states respectively.
like this:

$$\text{state} = \left((x, y), (\text{state of } (1,1), \text{state of } (1, top), \text{state of } (right, 1), \text{state of } (right, top))\right)$$

Under this sort of setting, we can easily redefine the *getStartState*, *isGoalState* and *getSuccessors* methods to fit our search problem.

The Suboptimal Search problem is solved by solving its subproblem *AnyFoodSearchProblem* repeatedly until every food dot has been visited. As long as we use BFS to solve *AnyFoodSearchProblem*, it will always return a list of actions which lead PacMan to the closest food dot. Therefore we use the code below to define the *findPathToClosestDot* method:

```
1    "*** YOUR CODE HERE ***"
2    return search.breadthFirstSearch(problem)
```

3 Heuristics in Corners Problem and Food Search Problem

For Corners problem, we have heuristic like:

Algorithm 2 Corners Heuristic

Input: s: The current search state; *problem*: A CornersProblem instance for this layout.
Output: h: a number that is a lower bound on the shortest path from the state to a goal of the problem;

```
if s is Goal state then return 0
else
    compute the correct minimum Manhattan distance from PacMan's position to the unvisited corners
    compute the minimum total path length to go through all the unvisited corners along with the edges of PacMan's world
    return the sum of minimum Manhattan distance and the minimum total path length
end if
```

For concrete definition of **the correct minimum Manhattan distance** and **minimum total path length**, please refer to the source code in `searchAgents.py` which is quite easy to understand. Note that there are 4 situations corresponding to the number of corners unvisited respectively. We should pay attention to the situation when there are 3 corners unvisited. In this kind of situation, PacMan need to get to the colsest corners on the diagonal first.("closest" means the minimum Manhattan distance.)

For Food Search problem, we have heuristic like:

Algorithm 3 Food Heuristic

Input: s: The current search state; *problem*: A FoodSearchProblem instance for this layout.
Output: h: a number that is a lower bound on the shortest path from the state to a goal of the problem;

```
if s is Goal state then return 0
else
    compute all the real distances from current position to all the food dots
    return the maximum value of all the real distances
end if
```

4 Analysis of Heuristics

Properties of a good heuristic function:

$$\textbf{Admissibility}: h(\text{node A}) \leq \text{ actual cost from A to G} \tag{1}$$

$$\textbf{Consistency}: h(\text{node A}) - h(\text{A's successor node C}) \leq cost(\text{A to C}) \tag{2}$$

For Corners Heuristic,
1. the heuristic returns a solution to its relaxed problem in which **all the walls are gone and PacMan only need to find a path which passes the four corners with the least cost**. It means **Admissibility** holds.
2. On the other hand, for simplicity, firstly, we assume that node A is directly the father

node of node C. The cost of every movement in CornersProblem is always **1** and the total path length won't change when PacMan is moving, which means the RHS in (2) is always **1** and the change of heuristic h is only determined by the change of minimum Manhattan distance from PacMan's position to the unvisited corners. It can be easily verified that the change of minimum Manhattan distance in every movement is $\mathbf{-1}, \mathbf{0}$ or $\mathbf{1}$ and it means in (2) LHS $\in \{-1, 0, 1\} \leq$ RHS $= 1$. That is to say **Consistency** holds **in every movement (or action)**. When node A is node C's ancestor node (may not be C's father node), we can always find a path from A to C (denoted by $\boldsymbol{P}$) and assume that $(\mathrm{X}_1, \mathrm{X}_2, ..., \mathrm{X}_m)$ are a series of nodes in $\boldsymbol{P}$ and they connect A and C. From the statements above, we know every pair of nodes $(\mathrm{X}_i, \mathrm{X}_{i+1})$,$i = 1, 2, ..., m-1$ holds **consistency**. Then we have:

$$
\begin{aligned}
h(\mathrm{A}) - h(\mathrm{X}_1) &\leq cost(\mathrm{A} \text{ to } \mathrm{X}_1) \\
h(\mathrm{X}_1) - h(\mathrm{Y}_2) &\leq cost(\mathrm{X}_1 \text{ to } \mathrm{X}_2) \\
h(\mathrm{X}_2) - h(\mathrm{Y}_3) &\leq cost(\mathrm{X}_2 \text{ to } \mathrm{X}_3) \\
&\cdots\cdots \\
h(\mathrm{X}_m) - h(\mathrm{C}) &\leq cost(\mathrm{X}_m \text{ to } \mathrm{C}) \\
\overset{\text{add them all up}}{\overset{\frown}{\leftrightarrow}} \quad h(\mathrm{A}) - h(\mathrm{C}) &\leq cost(\mathrm{A} \text{ to } \mathrm{C}) \qquad (3)
\end{aligned}
$$

(3) means the transitivity of **consistency** holds. Thus, we know (2) still holds for the pair of A and C. So we can conclude that Corners Heuristic is both **Admissible** and **Consistent**.

Note that a sudden large change of the heuristic value won't happen when the PacMan visit some corner. That's because the heuristic is equal to the sum of **minManhattan** and **total path length**. When PacMan visit some corner, the second part will decrease but the first part will update and increase. In the end, the heuristic value actually don't change a lot.

For Food Heuristic, it ignores all the food but the food with the maximum real distance to PacMan. It is a subproblem of the primal problem and its solution is definitely less than or equal to the primal solution. So it is **Admissible**. As for its **Consistency**, similarly, let's consider that A is directly the father node of C, which means in (2) RHS $= cost(\mathrm{A} \text{ to } \mathrm{C}) = \mathbf{1}$. It is obvious that in (2) LHS $\in \{-1, 0, 1\}$ (according to the positions of walls) So in every movement (or action), **Consistency** holds. Then similar to the previous statements in Corners Heuristic, (2) still holds whether A and C is adjacent or not. Thus, Food Heuristic is both **Admissible** and **Consistent**.

To make it more clear, consider the current position of PacMan is (x, y) and the corresponding farthest food is $\boldsymbol{F}$. Then after a movement, if $\boldsymbol{F}$ is still the farthest food, then the change of heuristic is in $\{-1, 0, 1\}$, else the new farthest food is $\boldsymbol{F^{new}}$ and the change of heuristic of the real distance from PacMan to $\boldsymbol{F^{new}}$ is in $\{0, 1\}$. Both two situations don't lead to a sudden large change of the heuristic value. Thus **Consistency** truely holds.