Search

Sai Srinadhu Katta & Venkata Sainath Thota

1 Introduction

In this project we implemented different search strategies like DFS, BFS, UCS, A* Search in Pacman world in Python. This report presents the heuristic functions used for A* Search along with the task specific observations made during implementation of the project.

2 Observations

2.1 Finding a Fixed Food Dot using Depth First Search

In this task DFS Algorithm is implemented. The solution found by our DFS algorithm for mediumMaze had a length of 130 when successsors are pushed reversly and had a length of 246 when successors are pushed normally. This is not a least cost solution because the Breadth first Search (BFS) in the next question gave a solution with cost 68 which is less than 130 and since BFS is guaranteed to give least cost solution in finite state spaces. The reason is that DFS will go deep into the state space until there is no successor or the goal is reached, If there is no goal then it retraces its path and goes in another direction deep which shows that we may not always find the least cost or shallower goals always but there's a high chance of finding a goal in deeper depths if present which will have cost greater than least cost possible. The exploration order of squares was as expected for DFS but for optimality better can be done and it doesn't go to all explored squares on it's way to goal.

```
def recursive_dfs(state,problem,explored,reverse):
 if (reverse):  #Add the sucessors in the reverse order
    reverse_insertion = list(problem.getSuccessors(state))
    reverse_insertion.reverse()
    for successor,direction,cost in reverse_insertion: #All successors
      if successor not in explored:
        action= recursive_dfs(successor,problem,explored,reverse)

        if (action != False):
          return [direction] + action

    return False
```

Code Snippet of DFS

2.2 Breadth First Search

Breadth first search is implemented in this task. We observed that the Breadth first search gives Least cost solution than dfs as it should, with a cost of 68 and the number of nodes expanded is 269.

```
def breadthFirstSearch(problem):
"""children"""
        for successor, action, stepCost in  problem.getSuccessors(state):
            if successor not in explored + frontier.list:   # Add to ↩
    frontier if not in explored or frontier
                frontier.push(successor)
                parent[successor] = (action, state)
```

Code Snippet from BFS

2.3 Varying the Cost Function

In this we implemented Uniform Cost search (UCS) using next task i.e A* Search. Since UCS is nothing but A* search with the heuristic value 0 for all states.

```
 def uniformCostSearch(problem):
  """Search the node of least total cost first."""

  return aStarSearch(problem, nullHeuristic)  #since UCS is same as A* ↩
    Search with nullHeuristic.

def nullHeuristic(state, problem=None):
    """
    A heuristic function estimates the cost from the current state to the ↩
    nearest
    goal in the provided SearchProblem.  This heuristic is trivial.
    """
    return 0

```

Code Snippet of UCS

2.4 A* search

Implemented A* search in this task. We expanded the child state which has the least combined cost i.e true cost of reaching this child state from start state + estimated cost of this child state to goal state(heurisitic value). Please refer to astar function in search.py for code.

Comparing all the searches for openMaze		
Algorithm	Nodes Expanded	Cost
DFS	576	298
BFS	682	54
UCS	682	54
A* (manhattan)	535	54

A* (manhattan) is little better compared to UCS Search.

2.5 Finding All the Corners

Task is to find the shortest path through the maze that touches all four corners irrespective of anything. The state needs information about pacman's position, visited/not visited for all corners and so is defined as the pacman position and the location of the four corners i.e,

State is a tuple (pacmanposition, cornersMap)
pacmanposition: (x,y)
cornersMap: a tupple mapping for each corner in corners to boolen for visited

```
# Code in __init__(self, startingGameState) in CornersProblem class.
self.cornersMap = tuple([False for x in self.corners])
self.start = (self.startingPosition, self.cornersMap)

# Code in getSuccessors(self, state) function.
if not self.walls[nextx][nexty]:
    nextcornerMap = list(state[1])
    neighbour = (nextx, nexty)
    for i in range(0,len(self.corners)):
        if neighbour == self.corners[i]:
            nextcornerMap[i] = True
    successors.append(((neighbour,tuple(nextcornerMap)),action, 1))
```

Code Snippet for Finding all corners

2.6 Corners Problem: Heuristic

In this the heurisitic was defined as follows: For any state first we calculated the minimum manhattan distance of unvisited corners from pacman position(min_cost) and from this pacman position we calculated the maximum manhattan distance to remaining unvisited corners (max_corner_dist). Our heuristic value for a state is min_cost + max_corner_dist. For code refer to cornersHeuristic(state, problem) in searchAgents.py file.

For goal state the heuristic returns 0 value. This heuristic is admissible since it never overestimates the true cost which can be seen intuitively. For consistency it's easy to see that, decrease is not more than true cost since we are using manhattan distance. This heuristic resulted in expansion of 783 nodes.

2.7 Eating All The Dots

In this task the search problem is eating all the pacman food in as few steps as possible. The heuristic, we used was maximum mazeDistance of food from pacman position. It's easy to see it's admissibility. If no food is there i.e, GoalState will have 0 heurisitic value. For consistency it's easy to see that it decreases at most by action cost and so our heurisitic is consistent. This heuristic has expanded 4137 nodes.

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

  position, foodGrid = state #Start positon
    food_coordinates = foodGrid.asList()  #list of food coordinates.

  if (len(food_coordinates) == 0): #Goal State
    return 0;

  else:
    max_distance = 0
    for food in food_coordinates:
      distance = mazeDistance(position, food, problem.startingGameState)
      if (max_distance < distance):
        max_distance = distance
        return max_distance  #returns the maximum mazeDistance.
```

Code Snippet for foodHeuristic

2.8 Suboptimal Search

We wrote an agent which greedily eats the closest dot. For goal test it's just if the food is present in that position it's goal else it's not. Then for finding path to closest food dot we used the BFS search strategy. This won't always find the shortest possible path through the maze, for example consider a maze in which they are no walls, food dots and agent are in horizontal line. There are three food dots, one left at 11 distance, one right at 10 distance, other right at 10 distance. The optimal least cost is 42 but by this greedy agent,the cost obtained is 51 which proves our above claim.

```
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 self.food[x][y]
```

Code Snippet for GoalTest

```
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.bfs(problem)
```

Code Snippet for findPathToClosestdot

References

[1] UC Berkeley CS 188 Intro to AI – Course Materials,
`http://ai.berkeley.edu/search.html`

[2] LaTeX Templates for Laboratory Reports,
`https://github.com/mgius/calpoly_csc300_templates`