**CSE 537 - Artificial Intelligence**

**Report: Project 1**

**Khan Mostafa**

109365509

**Abhijit Betigeri**

109229784

{khan.mostafa, abhijit.betigeri}@stonybrook.edu

Department of Computer Science

Stony Brook University

# Finding Fixed Food dots using Search Algorithms

## Q1. DFS

**Methodology Used:** A generic search function using stack as the fringe. DFS starts by expanding from the root and goes all the way down to leaf nodes for search.

**Execution Details**

Set 1:

python pacman.py -l tinyMaze -p SearchAgent

*Function*: depthFirstSearch

*Problem Class*: PositionSearchProblem

*Total Cost for the Path Found*: 10

*Search Nodes Expanded*: 15

Set 2:

python pacman.py -l mediumMaze -p SearchAgent

*Function:* depthFirstSearch

*Problem Class:* PositionSearchProblem

*Total Cost for the Path Found:* 130

*Search Nodes Expanded:* 146

Set 3:

python pacman.py -l bigMaze -z .5 -p SearchAgent

*Function:* depthFirstSearch

*Problem Class:* PositionSearchProblem

*Total Cost for the Path Found:* 210

*Search Nodes Expanded:* 390

## Q2. BFS

**Methodology Used:** A generic search function with queue as fringe. BFS works by expanding each level one by one and return a successful result when a goal state is found in one level. It returns optimal, in term of length, path to result.

**Execution Details**

Set 1:

python pacman.py -l tinyMaze -p SearchAgent –a fn=bfs

*Function*: breadthFirstSearch

*Problem Class*: PositionSearchProblem

*Total Cost for the Path Found*: 8

*Search Nodes Expanded*: 15

Set 2**:**

python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs

*Function:* breadthFirstSearch

*Problem Class:* PositionSearchProblem

*Total Cost for the Path Found:* 68

*Search Nodes Expanded:* 269

Set 3**:**

python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5

*Function:* breadthFirstSearch

*Problem Class:* PositionSearchProblem

*Total Cost for the Path Found:* 210

*Search Nodes Expanded:* 620

Note: python eightpuzzle.py works correctly with same generic function.

# Varying the Cost Function

MediumDottedMaze & MediumScaryMazes are used – here the cost function is varied taking into consideration dangerous steps & food rich areas.

## Q3. Uniform Cost Search

**Methodology Used**: Same generic function as BFS and DFS is also used for UCS with a PriorityQueue as the fringe. For each expandable node in fringe, an associated path cost is also added as priority key in the fringe. Lower cost nodes are expanded first.

**Execution Details:**

Set 1:

python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

*Function: UniformCostSearch (ucs)*

*Problem Class: PositionSearchProblem*

*Total Cost for the Path Found: 68*

*Search Nodes Expanded: 269*

**Execution Set 2:** python pacman.py mediumDottedMaze -p StayEastSearchAgent

**Total Cost for the Path Found:** 1

**Search Nodes Expanded:** 186

**Execution Set 3:** python pacman.py -l mediumScaryMaze -p StayWestSearchAgent

**Total Cost for the Path Found:** 17183894840

**Search Nodes Expanded:** 169

<explaination about the results???>

**A\* Search:**

**Question 4: A\* Search**

Methodology Used: A generic search function with priority queue as the fringe is used for search. As the priority key, a cost function f(n) = g(n) + h(n) associated with the state (node) is used. Here g(n) is the actual cost to reach to that node, and h(n) is an estimate found using a heuristic function.

Execution: python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

**Function:** aStarSearch

**Problem Class:** PositionSearchProblem

**Total Cost for the Path Found:** 210

**Search Nodes Expanded:** 549

(v/s) – Comparison with ucs – astar gives a better optimal solution:

Execution: python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=ucs

**Function:** UCS

**Problem Class:** PositionSearchProblem

**Total Cost for the Path Found:** 210

**Search Nodes Expanded:** 620

Here search nodes expanded is 620 compared to 549 in astar.

**Finding All Corners:**

**Finding whether all the four corners have been reached.**

**Question 5: Detection of all four corners reached**

**Methodology Used:** WeUse the bitmap structure to keep track of all the corners visited.

when we update to get the next set of successors, the corner vectors are also updated accordingly.

**Execution Set:** python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem

**Function:** BFS

**Problem Class:** CornersProblem

**Total Cost for the Path Found:** 29

**Search Nodes Expanded:** 277

**Execution Set:** python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem

**Function:** BFS

**Problem Class:** CornersProblem

**Total Cost for the Path Found:** 107

**Search Nodes Expanded:** 2006

**Question 6: Heuristic for Corners Problem in cornersheuristic**

Heuristic explanation: We are computing the cost of the nearest corner plus cost to visit the rest of the remaining corners. The distance function is manhattanDistance.

**Execution Details:**

python pacman.py -l mediumCorners -p SearchAgent –a fn=aStar,prob=CornersProblem,heuristic=cornersHeuristic

**Function:** astar and cornersHeuristic

**Problem Class:** CornersProblem

**Total Cost for the Path Found:** 107

**Search Nodes Expanded:** 709

**Question 7: Eating all the dots:**

Checking of the code with testSearch for FoodSearchProblem.

**Execution Set:**

python pacman.py -l testSearch -p AStarFoodSearchAgen

(or)

python pacman.py -l testSearch -p SearchAgent -a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic

**Function:** astart and FoodHeuristic

**Problem Class:** FoodSearchProblem

**Total Cost for the Path Found:** 7

**Search Nodes Expanded:** 12

**UCS algorithm with tinySearch Layout – very Slow:**

**Execution Set:**

python pacman.py -l testSearch -p SearchAgent -a fn=ucs,prob=FoodSearchProblem

**Function:** UCS

**Problem Class:** FoodSearchProblem

**Total Cost for the Path Found: 2**7

**Search Nodes Expanded:** 5057

**Question 7:**

**Methodology used to solve the problem:**

The foods are visualized to be on the circumference of the convex hull. The convex hull is constructed taking into consideration several foods and such that no food falls outside convex hull but there might be food inside convex hull. Then nearest food (in terms of the location) present on the convex hull from the pacman is considered – that food is traversed and so on.

Food locations can be considered as points on a plane. Based on this model two heuristics are used. For the heuristic, we relax the problem to have no internal walls and try to estimate a lower bound of shortest path to visit all these points. An actual estimate will be as hard as traveling salesman problem. So, two safe and faster estimates are used as heuristic function to solve the problem:

convexArchLenPlusDistance: It finds a convex hull of all points. If the largest arm of the hull is taken out, we get an arch, what we can call the convex arch. This arch includes all points on the hull. The length of the arch is the cost to visit all points on the convex hull. This is a lower bound on visiting all points. To better fit out heuristic to actual cost, we also consider the cost of Pacman to move onto the arch. It is apparent that, the cost for Pacman to reach out to the arch will be no lesser than the minimum distance between Pacman and any point on the arch.

quadrantExtremesDistance: The grid can be divided into four natural quadrants with respect to Pacamans location. We can find farthest point in each quadrant. It is apparent that, reaching all these extremes is a lower bound to actual cost. We estimate the cost to reach to the closest of the extremes plus the rest of the extremes thereon.

Max of the two heuristics is used for better estimate.

**Execution Set: Our Heuristic expands within 7000 nodes!!....**

python pacman.py -l trickySearch -p SearchAgent -a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic

**Function:** astart and foodHeuristic

**Problem Class:** FoodSearchProblem

**Total Cost for the Path Found:** 60

**Search Nodes Expanded:** 6945

We just expand 6945 nodes to find optimal path but whereas with null heuristic/bfs the nodes expanded are 16688.

A sanity check of the heuristic can be done with easySearch, which is done on same grid as trickySearch, with all internal walls removed. In which a BFS finds the optimal path of cost 29.

**Greedy search non-optimal solution for MediumMaze**

Solving MediumMaze optimally within reasonable time with A\* search seems to be a hard problem. However, greedy search (where f(n)=h(n)) can find a (non-optimal) path. A greedy search with the same foodHeuristic finds a path of cost 210 by expanding over 35 thousand nodes. However, a more naïve and greedier estimate pointsLeftEstimate (which considers only how many foods are left) yields a path of cost 184 by expanding a little over 300 nodes.

python pacman.py -l mediumSearch -p SearchAgent –a fn=greedy,prob=FoodSearchProblem,heuristic=pointsLeftHeuristic

**Analysis:**

Usage of BFS vs DFS depends heavily on the structure of the tree, number of solutions etc. If the solution is not far from the root of the tree, then bfs might be better. If the tree is very deep, solutions are rare, then dfs might take long time, and bfs could be faster. If the tree is wide, then bfs might need a lot of memory. And if the solutions are frequent but located deep in the tree, BFS could be impractical. Deeper search its better off using DFS. DFS is space efficient compared to BFS.

A\* (astar) algorithm can be tweaked efficiently with other path finding algorithms by analyzing how it evaluates and what type of heuristics it uses. A\* need not give best solution to all the problems but is very effective versatile tool with our admissible, consistent heuristic can give optimal solutions to complex problems frequently.