CSE537 Artificial Intelligence

Assignment-4 Report

DFS, BFS and A-Star in Prolog and Corners Problem

Authors:

Vishal Nayak: 109892702: vnayak@cs.stonybrook.edu

Ashish Chaudhary: 109770154: ashchaudhary@cs.stonybrook.edu

Contents

1. Intro and details

1.1. Compilation, machine details and commands

Compiler: ***python 2.7.6***

Machine: Ubuntu 14.04 64-bit

***Note*: Wherever code is invoking XSB, there are comments.** Please update the location of the binary there for the below commands to work properly.

- *python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch*
- *python pacman.py -l tinyMaze -p SearchAgent -a fn=depthFirstSearch*
- *python pacman.py -l mediumMaze -z .7 -p SearchAgent --frameTime=0.015 -a fn=depthFirstSearch*
- *python pacman.py -l bigMaze -z .3 -p SearchAgent --frameTime=0.01 -a fn=depthFirstSearch*
- *python pacman.py -l tinyMaze -p SearchAgent -a fn=breadthFirstSearch*
- *python pacman.py -l mediumMaze -z .7 -p SearchAgent --frameTime=0.015 -a fn=breadthFirstSearch*
- *python pacman.py -l bigMaze -z .5 -p SearchAgent --frameTime=0.01 -a fn=breadthFirstSearch*
- *python pacman.py -l tinyMaze -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic*
- *python pacman.py -l mediumMaze -z .7 -p SearchAgent --frameTime=0.015 -a fn=astar,heuristic=manhattanHeuristic*
- *python pacman.py -l bigMaze -z .4 -p SearchAgent --frameTime=0.01 -a fn=astar,heuristic=manhattanHeuristic*
- *python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem*

1.2. Approach

Create the maze.P file generically for PositionSearchProblem such that it can be utilized by all the searches: DFS, BFS and A-Star. At every valid position, all the four directions are checked. If the neighboring location is also a valid location, then an edge clause is added for the same.

After all the edge clauses are populated, the goal clause is added at the end for the search procedures to terminate.

Later, this file is loaded to XSB along with the respective search strategy file and the query is invoked to get the path.

All the while adding the edge clauses, a direction_list is maintained. This helps in reverse mapping the result of prolog, the path to goal, into the game state directions. The list of directions are returned.

2. DFS Search

Depth First Search is a search algorithm which explores the items in the fringe list in an order such that the last entered node in the fringe is popped out at the earliest.

Analysis:

This procedure fits the iterative paradigm of prolog and works with the terminal condition of fringe being drained out and the path containing the goal node as the head.

3. BFS Search

Breadth First Search is a search algorithm which explores the nodes at constant distant distances in increases order from the current node. The maze.P created beforehand will have all the clauses required. The python code will invoke the search with the start node to receive the path required to reverse map and compute the actions to be taken.

Analysis:

In prolog, the implementation is slightly complicated than DFS. Here a list of paths needs to be maintained all the time and one for each spine of the search tree. Goal test comparison should take care of the list of list head comparison instead of just the head comparison in case of DFS.

4. A* Search

A* is a combination of both best-first and uniform-cost search. In the case of pacman problem, since the cost from each state to another is just one, the A* f(n) will greatly rely on the heuristic chosen to reach the goal. Along with maze.P, we have created the heuristic information in python and are supplying the heuristics to prolog through astar_heuristic.P file. Both the files are loaded along with the astar.P to get the path.

Analysis:

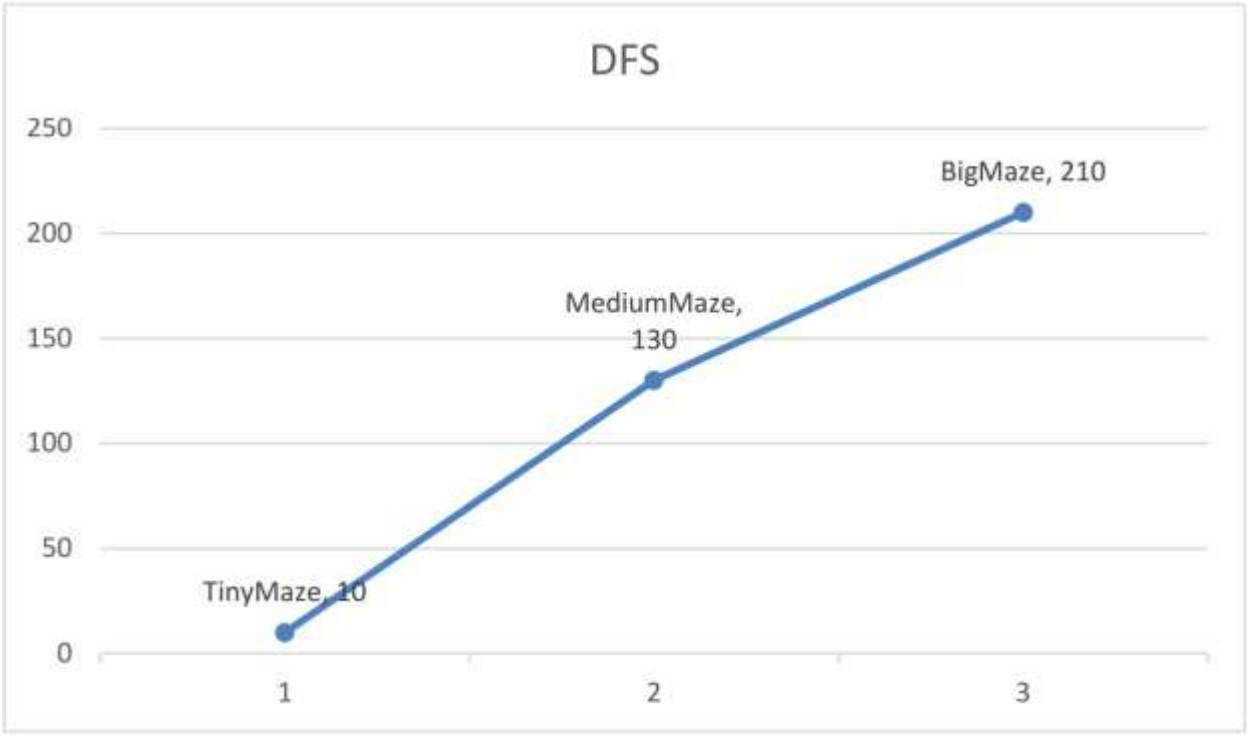
In prolog, the neighbors of each node have to be found and its relative f(n) value is coupled with each of the neighbors. The 'findall' method is used to find such instances. Then 'keysort' is used to pick the fringe element with least f(n) value and the unification is continued.

5. Corners Problem

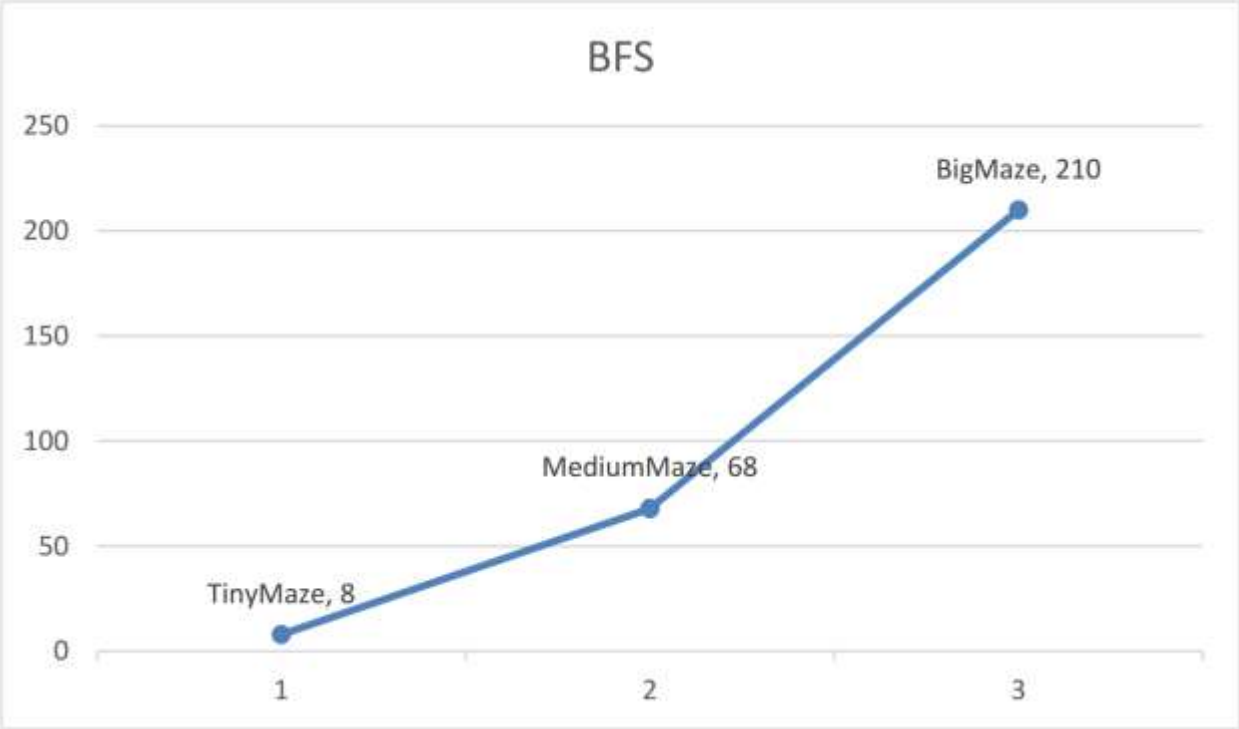
The approach for in solving this problem is slightly different than the other search strategies. Instead of adding the goal clause in the maze.P file, we have created separate files, one for each corner to hold the goal clause. In python, four different queries are done one with each corner as the goal by loading different prolog corner clause file. The query will take the goal of previous query as the start position of the current query. The results of all the queries are put in a merged list. Later the merged list is reverse mapped to the result list using the precomputed directions list. This is effectively solving the corners problem.

6. Statistics

6.1. DFS



6.2. BFS


BFS
250
200
150
100
50
0
1
2
3
BigMaze, 210
MediumMaze, 68
TinyMaze, 8

6.3. ASTAR


ASTAR
250
200
150
100
50
0
1
2
3
BigMaze, 210
MediumMaze, 152
TinyMaze, 10

7. Traces of commands

7.1. TinyMaze

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

```
[SearchAgent] using function tinyMazeSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:       502
Win Rate:     1/1 (1.00)
Record:       Win
```

7.2. DFS: TinyMaze, MediumMaze, BigMaze

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

```
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
This is DFS
Path found with total cost of 10 in 0.8 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores:       500
Win Rate:     1/1 (1.00)
Record:       Win
```

python pacman.py -l mediumMaze -z .7 -p SearchAgent --frameTime=0.015 -a fn=depthFirstSearch

```
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
This is DFS
Path found with total cost of 130 in 0.8 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores:       380
Win Rate:     1/1 (1.00)
Record:       Win
```

python pacman.py -l bigMaze -z .3 -p SearchAgent --frameTime=0.01 -a fn=depthFirstSearch

```
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
This is DFS
Path found with total cost of 210 in 0.8 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 300
Average Score: 300.0
```

Scores: 300
Win Rate: 1/1 (1.00)
Record: Win

7.3. BFS: TinyMaze, MediumMaze, BigMaze

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

[SearchAgent] using function breadthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
This is BFS!
Path found with total cost of 8 in 0.7 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores: 502
Win Rate: 1/1 (1.00)
Record: Win

python pacman.py -l mediumMaze -z .7 -p SearchAgent --frameTime=0.015 -a fn=breadthFirstSearch

[SearchAgent] using function breadthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
This is BFS!
Path found with total cost of 68 in 0.8 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores: 442
Win Rate: 1/1 (1.00)
Record: Win

python pacman.py -l bigMaze -z .5 -p SearchAgent --frameTime=0.01 -a fn=breadthFirstSearch

[SearchAgent] using function breadthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
This is BFS!
Path found with total cost of 210 in 0.8 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores: 300
Win Rate: 1/1 (1.00)
Record: Win

7.4. A-Star: TinyMaze, MediumMaze, BigMaze

python pacman.py -l tinyMaze -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
This is A*!

Path found with total cost of 10 in 1.0 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores: 500
Win Rate: 1/1 (1.00)
Record: Win

python pacman.py -l mediumMaze -z .7 -p SearchAgent --frameTime=0.015 -a fn=astar,heuristic=manhattanHeuristic

[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
This is A*!
Path found with total cost of 152 in 1.1 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 358
Average Score: 358.0
Scores: 358
Win Rate: 1/1 (1.00)
Record: Win

python pacman.py -l bigMaze -z .4 -p SearchAgent --frameTime=0.01 -a fn=astar,heuristic=manhattanHeuristic

[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
This is A*!
Path found with total cost of 210 in 1.0 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores: 300
Win Rate: 1/1 (1.00)
Record: Win

7.5. CornersProblem: TinyCorners

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

[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
This is BFS!
Path found with total cost of 35 in 3.4 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 505
Average Score: 505.0
Scores: 505
Win Rate: 1/1 (1.00)
Record: Win

7.6. CornersProblem: MediumCorners

We ran the prolog code for medium corners independently and it is yielding results. The integration is not resulting in solution. We believe that this is some problem of the software.