COMP 560 Artificial Intelligence: Rat in a Maze (Assignment 1)

Ben Hu,* Brian Luong,† and Stephen Yan‡
(Dated: September 15, 2015)

We discuss the relationships between breadth-first, depth-first, greedy best-first, and A* search. We use the Manhattan Distance as the heuristic for the latter of the two. We discuss their implementations in Java to solve a variety of mazes created from text files and detail their performance in terms of path cost and node expansion (accuracy and difficulty), ranging in size and difficulty. We also discuss maze design and its relationship to A* and greedy best first search performance. Lastly we discuss a new subproblem where there are "cheeses" located in the maze that must be eaten in place of reaching a specific end goal coordinate where we design a new A* search algorithm to obtain the optimal path where all the cheese is eaten.

I. SEARCH ALGORITHM DESIGN AND ANALYSIS

First, we want to represent the maze as a connected graph of coordinates within the maze that are not walls in the form of an adjacency list. Additionally, in each of these search algorithms, we will define a state to be just the location of the coordinate, row `i` and column `j`.

The maximum branching factor `b` is 4, and the maximum number of adjacent coordinates to any coordinate in the maze is 4. Thus, the maximum depth `d` of the optimal solution is `m * n`, where `m` and `n` are the length and width respectively. This is under the assumption that there are no repeated states. The maximum length of any path in the state space is also `m * n`.

In all of these search algorithm implementations, we will not allow a state to be added on the frontier or explored set if it already exists, effectively eliminating repeated states.

In all of our algorithms, we implemented a class `Index`, which has the instance variable `prev`, which points to the predecessor node (node that expanded it). This way, we can easily trace our solution path. Every time we expand a node, we modify its successor states' `prev` to point to the expanded node.

```
for (Index i : adjNodes) {
   if (!expanded.contains(i) &&
        !frontier.contains(i)) {
      i.prev = expand;
   }
}
```

A. Search Implementations

a. Breadth First Search - We will expand nodes in the frontier in the order they are added, thus we will represent our frontier as a FIFO queue.

* bxhu@live.unc.edu
† bluong@email.unc.edu
‡ sjyan@cs.unc.edu

```
%%%%%%%%%%%%%%%%%%%%%%
%S                    %
%.                    %
%.                    %
%.                    %
%..................G%
%%%%%%%%%%%%%%%%%%%%%%
Total Path Cost: 24
Total Nodes Expanded: 98
```

FIG. 1. Sparse Maze

b. Depth First Search - We will expand nodes in the frontier that are most recently added, thus we will represent our frontier as a LIFO stack.

c. Greedy Best First Search - This is an instance of an informed search, which means we have an approximation of the "goodness" of a state with a heuristic function `h(n)`. We will expand the lowest value of `h(n)`. In the maze search algorithm case, our heuristic function `h(n)` is the Manhattan Distance between the current state and the goal state.

d. A Search* - This is also an instance of an informed search. Our function to evaluate the "goodness" of a state is defined by `f(n) = g(n) + h(n)`, where `g(n)` is the current path cost to that state and `h(n)` is the heuristic fucntion, which we defined as the Manhattan Distance from the current state to the goal.

When we did not allow repeated states, A* would run in an infinite loop. This is because the frontier would become much too large, causing the evaluation `f(n)` to become slower. This is especially evident in a sparse graph represented below in `FIG. 1`

By allowing repeated states, the algorithm would finish in 6 seconds. Disallowing repeated states would allow it to run significantly faster.

Not surprisingly, Greedy Best First Search runs infinitely in this case. For the other two algorithms, BFS and DFS was unaffected time-wise because the frontiers just remove either the first or last node, an `O(1)` operation. However, the memory increased as repeated states were added onto the frontier. As a result, the number of expanded nodes increased.

II. MAZE DESIGN

In a broad sense, a maze that is difficult for A* is very akin to a maze that is difficult for a human. We can liken it to a game (RPG) that provides us different options at a crossroads. At each next stage, we're given even more options. Our priority is to make sure we've evaluated the optimal way to go and to take that path, even if that means exploring all dead ends. What differentiates A* from Greedy Best First Search is its consideration of both proximity to goal and current path cost from the start. At some point in our exploration of an endless tunnel, we are likely to stop and reevaluate that maybe "this is getting us nowhere." GBFS doesn't care so long as the current path takes us closer to the end goal.

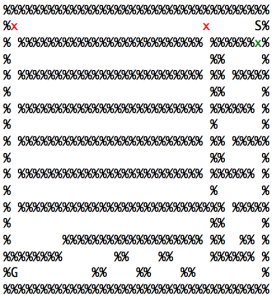


FIG. 2. A*-difficult (Red 'x' denotes suboptimal path. Green denotes optimal path)

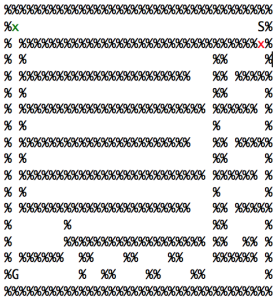


FIG. 3. GBFS-difficult.

A. Implementation

We used `mediumMaze.txt` given in the assignment as a basis for the new maze designs.

e. A-difficult maze* In order to design a maze difficult for A* in terms of node exploration, we designed it with many promising intersections. At the start of the A-Star difficult maze, it is given the choice of going down and left toward the goal as it is inclined to based on the heuristic as well as left all the way and down. The initial inclination for both search algorithms leads directly to the goal node, which is ideal for GBFS. However, A* has options to explore more. It can choose to explore through the middle or the leftmost pathway, both of which however lead to a suboptimal path. However, all three choices (left, middle, and right) are similar in optimality. A* search expanded `3.51` times as many nodes than GBFS here.

f. GBFS-difficult maze Designing a maze that is difficult for GBFS requires that the inclined path be long and a dead end. In this maze, there are essentially only two ways to proceed toward the goal (as opposed to the many in the A*Difficult maze). The first (inclined) path leads close to the goal but is a dead end. It also ends up exploring many nodes through the middle but reaches dead ends every time. GBFS explores the most inclined path virtually all the way through before exploring the easy, straight-line path. GBFS expanded `4.93` as many nodes than A* search here.

III. CHEESE-EATING OPTIMIZATION

We started off by adding methods for the new conditions we had to consider. Since we no longer had one goal to reach, we had to add a way to handle backtracking. We did this by keeping the previous node on the adjacency list instead of removing it. We also had to create new nodes for the adjacent nodes because otherwise they would refer to themselves, causing infinite loops. Keeping the previous node on the path caused problems with repeated nodes being added exponentially, causing a huge run time.

We attempted to remove nodes that did not move any closer to the nearest cheese but it would get stuck going back and forth. We then removed nodes on the frontier with the exact same state (same location and same cheeses visited). This helped the node expansions but still did not give us a solution.

We decided that we had to change the heuristic function or the weights for the heuristic function. The weights that we could change were: path cost, distance to cheese cost, and number of current cheeses. Changing the weights showed no improvement to finding a solution. We hypothesized that this was because the Manhattan distance was not a good representation for non-direct paths to cheeses. That is, paths that required the mouse to move away from the cheese first.

Our solution to this was to aggregate the Manhattan distance for every cheese in the maze. This heuristic function finally gave us a solution in a reasonable amount of time. However, the solution it gave was not very optimal. In fact it was usually double or more the optimal solution. We found that we were still weighing the cheeses from the previous attempts. The aggregate distance already gave the cheeses weight because removing one cheese would reduce the value by that distance. After removing the weight to cheese we got solutions that were fairly close to optimal in a reasonable amount of time. We used this for our final version.

We found that the reason the aggregate distance did not find that the most optimal solution is that it would heavily favor moving towards groups of cheese. This is because moving towards higher number of cheeses would reduce the cost for every cheese as opposed to moving towards a small number of cheeses. This would cause problems where the mouse would ignore a single cheese close to it for a group of cheeses further away or move non-optimally towards groups of cheese.

IV. FUTURE IMPROVEMENTS

One improvement would be to remove corridors. Corridors would be defined as nodes with only two adjacent nodes. This is because the only route of traversal through these nodes is straight through. A node with one adjaceny represents a dead end where the mouse would have to turn around. Nodes with greater than two adjacencies represent junctions where the mouse has to decide which path to take.

Removing corridors would help improve runtime and memory simply by removing unnecessary nodes. Implementing this for the cheese mazes would require calculating the path cost and/or turning point in corridors that have cheese but not for every coordinate in the corridor.

Another improvement would be improving the heuristic function. We would have to implement something similar to an algorithm that solves the traveling salesman problem. This might include a minimum spanning tree or other sorts of searches.

V. INDIVIDUAL CONTRIBUTIONS

g. Hu, B. Responsible for design for cheese-eating algorithm, report formatting, styling

h. Luong, B. Responsible for project structure, search algorithm design, cheese-eating algorithm design, styling, setting up source control, print and extra-credit animation

i. Yan, S. Responsible for maze design, aggregate heuristic design for cheese-eating algorithm test design, sparse maze testing for A* search revision, report formatting, styling

Appendix

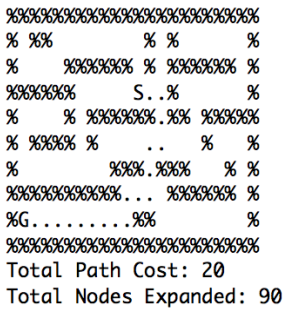
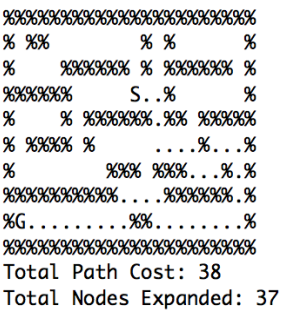
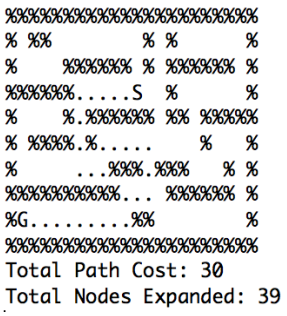
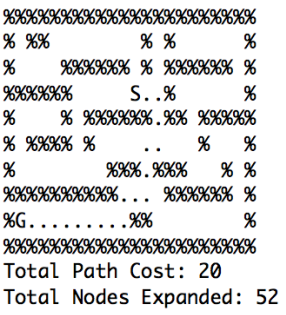
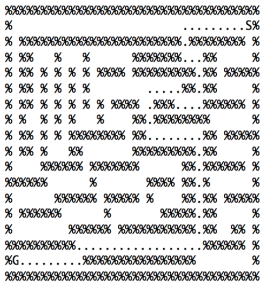
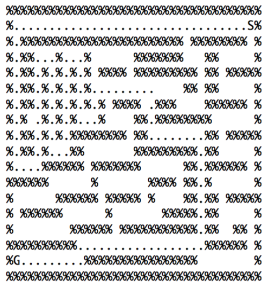
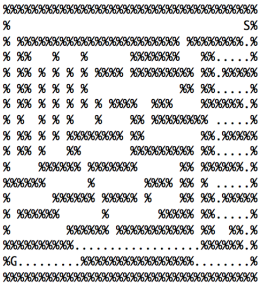
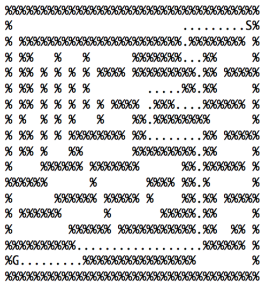
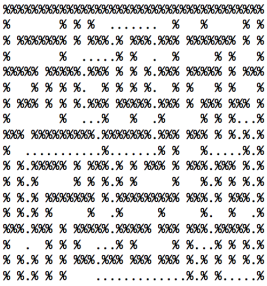
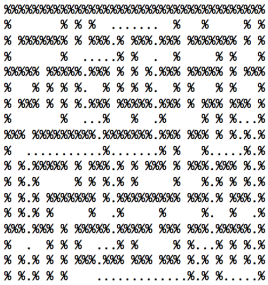
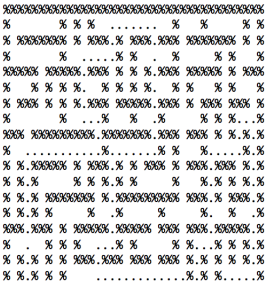
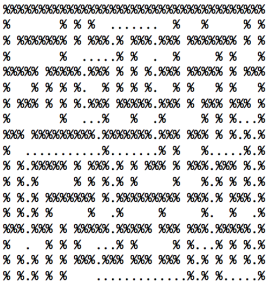
	BFS	DFS	GBFS	A*
Small	 Total Path Cost: 20 Total Nodes Expanded: 90	 Total Path Cost: 38 Total Nodes Expanded: 37	 Total Path Cost: 30 Total Nodes Expanded: 39	 Total Path Cost: 20 Total Nodes Expanded: 52
Medium	 Total Path Cost: 69 Total Nodes Expanded: 267	 Total Path Cost: 131 Total Nodes Expanded: 144	 Total Path Cost: 75 Total Nodes Expanded: 78	 Total Path Cost: 69 Total Nodes Expanded: 221
Large	 Total Path Cost: 211 Total Nodes Expanded: 617	 Total Path Cost: 211 Total Nodes Expanded: 387	 Total Path Cost: 211 Total Nodes Expanded: 466	 Total Path Cost: 211 Total Nodes Expanded: 549

TABLE I. Maze Solutions

Small	Tricky
%%%%%%%%%%%%%%%%%%%% %g 321S d% %h%%b%%a%%c%%4%% %e% % %% %98765 %f% %%%%%%%%%%%%%%%%%%%% Total Path Cost: 59 Total Nodes Expanded: 170	%%%%%%%%%%%%%%%%%%%% %3 67% % %4%%2%%1%%5%%8%% % % % S % % %%%%%%%%%%%%%%%%%% % %dcba9 % %%%%%%%%%%%%%%%%%%%% Total Path Cost: 61 Total Nodes Expanded: 5404

TABLE II. Cheese Solutions

	A*-Difficult	GBFS-Difficult
A*	%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% % S% % %%%%%%%%%%%%%%%%%%%%%%%% %%%%%%.% % %%.....% % %%%%%%%%%%%%%%%%%%%%%%%% %%.%%%%% % %%.....% % %%%%%%%%%%%%%%%%%%%%%%%% %%%%%%.% % % .....% % %%%%%%%%%%%%%%%%%%%%%%%% %%.%%%%% % %%.....% % %%%%%%%%%%%%%%%%%%%%%%%% %%%%%%.% % % .....% % %%%%%%%%%%%%%%%%%%%%%%%%%%.%%%%% % %%.....% % %%%%%%%%%%%%%%%%%%%% %% %%.% %%%%%%%% ....%%....%%.... %%%%%%.% %G..........%%....%%....%%.........% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Total Path Cost: 79 Total Nodes Expanded: 299	%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %.....S% %.%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% % %.% %% % %.%%%%%%%%%%%%%%%%%%%%%%%% %% %%%%% %.% %% % %.%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%% % %.% % % %.%%%%%%%%%%%%%%%%%%%%%%%% %% %%%%% %.% %% % %.%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%% % %.% % % %.%%%%%%%%%%%%%%%%%%%%%%%% %% %%%%% %. % %% % %. %%%%%%%%%%%%%%%%%%%% %% %% % %.%%%%%% %% %% %% %% %%%%%% % %G % %% %% %% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Total Path Cost: 49 Total Nodes Expanded: 56
GBFS	%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% % S% % %%%%%%%%%%%%%%%%%%%%%%%% %%%%%%.% % %%.....% % %%%%%%%%%%%%%%%%%%%%%%%% %%.%%%%% % %%.....% % %%%%%%%%%%%%%%%%%%%%%%%% %%%%%%.% % % .....% % %%%%%%%%%%%%%%%%%%%%%%%% %%.%%%%% % %%.....% % %%%%%%%%%%%%%%%%%%%%%%%% %%%%%%.% % % .....% % %%%%%%%%%%%%%%%%%%%%%%%%%%.%%%%% % %%.....% % %%%%%%%%%%%%%%%%%%%% %% %%.% %%%%%%%% ....%%....%%.... %%%%%%.% %G..........%%....%%....%%.........% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Total Path Cost: 79 Total Nodes Expanded: 82	%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %.....S% %.%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% % %.% %% % %.%%%%%%%%%%%%%%%%%%%%%%%% %% %%%%% %.% %% % %.%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%% % %.% % % %.%%%%%%%%%%%%%%%%%%%%%%%% %% %%%%% %.% %% % %.%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%% % %.% % % %.%%%%%%%%%%%%%%%%%%%%%%%% %% %%%%% %. % %% % %. %%%%%%%%%%%%%%%%%%%% %% %% % %.%%%%%% %% %% %% %% %%%%%% % %G % %% %% %% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Total Path Cost: 49 Total Nodes Expanded: 276

TABLE III. Comparing A* and GBFS difficulty on different mazes