CS 440 MP1
Section Q4

Christian Howard
howard28@illinois.edu

Luke Pitstick
pitstck2@illinois.edu

Liuyi Shi
liuyis2@illinois.edu

Abstract

Within this report, we investigate creating agents that use different path planning algorithms and do an analysis of how they compare for a set of sample problems. **Insert summary of results**. We later apply A^* algorithms to guide an agent to solving the Sokoban puzzle. **Insert summary of results**.

Contents

1 Part 1 - Multi-Goal Search

Within this section, we investigate creating an agent based on search algorithms to tackle single-goal and multi-goal objectives. To handle this problem, the provided maze is processed into a graph that can be used to figure out what actions a given agent can take and when they reach some goal node. To model the motion of some agent, we define the state, $\boldsymbol{x}$, as the following:

$$\boldsymbol{x} = (p, b_1, b_2, \cdots, b_n)$$

where p is an index representing the graph node the agent is on and b_k represents a Boolean variable that is 1 (true) when the agent has passed through the k^{th} goal point at some point in the past, 0 (false) otherwise. We know an agent has completed a maze when it reaches a state that satisfies the following:

$$b_1 \wedge b_2 \wedge \cdots \wedge b_n = 1$$

This check will be what is performed in the software implementation of the search algorithms to know we have found a goal state. The transition function for this agent is also the following:

```
Algorithm 1: Transition Function
  Data: State, $\boldsymbol{x}$
  Data: Action, $a$
  Data: Maze Graph, $G_m$
  Data: Goal Node Set, $V_g = \{p_1, p_2, \cdots, p_n\}$
  Result: New State, $\boldsymbol{x}_n$

  // Set the new state with the
  // current state's boolean configuration
  for $k \in \{1, 2, \cdots, |V_g|\}$ do
    $\boldsymbol{x}_n(b_k) = \boldsymbol{x}(b_k)$
  end

  // Update position in maze given the action $a$
  $\boldsymbol{x}_n(p) = G_m(\boldsymbol{x}(p), a)$

  // Check if new position is in goal node set
  // If so, find index associated with goal node and
  // set corresponding boolean to 1
  if $\boldsymbol{x}(p) \in V_g$ then
    Get $m \ni \boldsymbol{x}(p) = p_m$
    Set $\boldsymbol{x}(b_m) = 1$
  end

  return $\boldsymbol{x}_n$
```

Given the state representation and transition function described above, we are now able to implement and test our search-based agents. That said, we are now able to dive into solving each challenge of Part 1.

1.1 Part 1.1 - Basic Pathfinding

In this first challenge for Part 1, the goal is to implement search algorithms for navigating through a maze environment to a single goal location. For this challenge, we have implemented the following search algorithms:

- Depth-First Search
- Breadth-First Search
- Greedy Best-First Search
- A* Search

For Greedy and A*, we have also set their Heuristic Function as the Manhattan Distance metric, defined as the following:

$$d_M(u, v) = |u - v|_1$$

where $u, v \in \mathbb{R}^2$ represent positions in the maze we want the distance between and $|\cdot|_1$ is the L_1 norm. Using this formula, the heuristic function will be:

$$h(n) = d_M(n, g)$$

where n is the 2-D position of the current agent's position and g is the 2-D position of the goal point. Given the above search strategies and heuristic, Table 1 and Table 2 show the stats between each method for each of the provided mazes:

	Medium Maze	Big Maze	Open Maze
Depth-First Search	124	474	59
Breadth-First Search	68	148	45
Greedy Best-First Search	68	222	77
A* Search	68	148	45

Table 1: Solution Path Cost

As we can see in the table, Depth-First Search (DFS) tended to arrive at a solution to a given maze quickly but at the expense of a sub-optimal path cost. We can note that Breadth First Search (BFS) and A* both, as expected, produced optimal path costs but it is interesting to note that for the Big Maze and Open Maze, BFS actually reached the optimal solution faster. It seems

	Medium Maze	Big Maze	Open Maze
Depth-First Search	198	1029	319
Breadth-First Search	345	1259	523
Greedy Best-First Search	77	311	27761
A^* Search	202	1495	667

Table 2: Number of Nodes Expanded

feasible that the Manhattan distance heuristic would, at times, take A* in a path that would lead to dead ends, in turn requiring A* to expand more nodes than BFS would have to.

One other thing to note was the interesting characteristics of the Greedy Search. We can see that on one end, it reached an optimal result in a very low number of expanded nodes for the Medium Maze. On the other end, it reaches a sub-optimal path cost after expanding a huge number of states in the Open Maze. Greedy obviously lacks robustness and this can be suspected due to the fact it does not take into account how far it has come but only thinks about how far away it thinks it is. Given this one sided thinking, the agent can get too excited being "close" to the goal even though they may be winding through a lot of poor sub-paths.

1.2 Part 1.2 - Search with Multiple Dots

1.3 Part 1 Extra Credit - Suboptimal Search

2 Part 2 - Sokoban