project1: The Maze is on Fire

Bingyu Xin

September 2020

1 Generate Maze

The initial mazes we generate can satisfy two requirements: a) current maze is solvable, b) there is a path from the agent to the initial fire source. We implement it algorithmically as Algorithm 1. There are mainly two steps when generating a maze with fire: 1) generating maze with random occupied cells, check if it is solvable 2) adding fire to the generated maze, check if it is solvable from agent to the fire source.

We show some generated samples with different dimensions as Fig 1, and we compare the average time cost of different dimensions of mazes using this maze generation algorithm on our machine as Table 1, we calculate the average value with 100 random samples for each dimension setting.

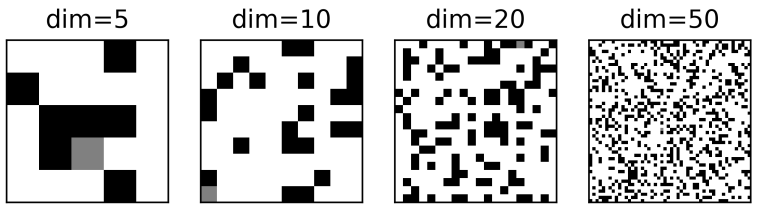


Figure 1: The samples of generated mazes with different dimensions when the obstacle density p is 0.3. The white cell is empty, black cell is occupied and gray cell is on fire

Table 1: Time cost when generating mazes with different dimensions, $p = 0.3$

dim	5	10	20	50
time(ms)	10.7	26.8	70.8	1894.5

2 Strategy 1

Strategy 1 ignores the advancements of the fire and decides the final path to the goal point at the beginning. We implement DFS, BFS and A^* Search as shown in Fig 2, we compare their average time cost on 100 random maze samples as Table 2, and finally we choose A^* Search as our basic search fuction for its speed and shortest path. We describe Strategy 1 in Algorithm 2

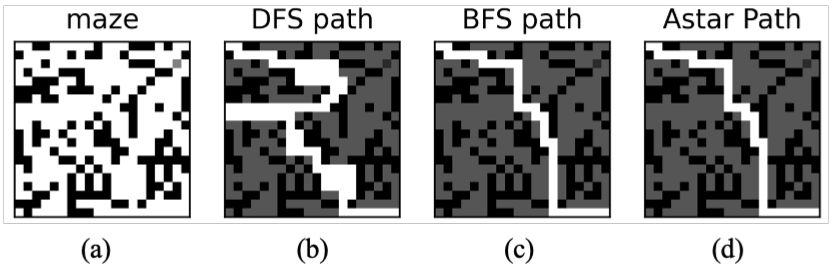


Figure 2: Path found by different search methods, (a)maze, (b)DFS, (c)BFS,(d)$A^*search$, $dim = 20$, $p = 0.3$, $q = 1$.

Table 2: Time cost(ms) of different search methods, $dim = 20$, $p = 0.3$

methods	DFS	BFS	A^*
time(ms)	2.8	5.9	2.8

3 Strategy 2

The fire advances at every time step with flammability q. Fig 3 shows the process when $q = 1$.

Strategy 2 re-computes the shortest path from the agent's current position to the goal at every time step based on the current state of maze and fire. We describe it in Algorithm 3

4 Strategy 3

As shown in the Fig 4, with flammability $q > 0$, the current fire have potential influence on the other empty cells in the maze.

Strategy 3 is an improvement of Strategy 2 in terms of survival rate. It considers the future state (one step further) of the maze and fire while searching the path from current position to the goal. We describe it in Algorithm 4

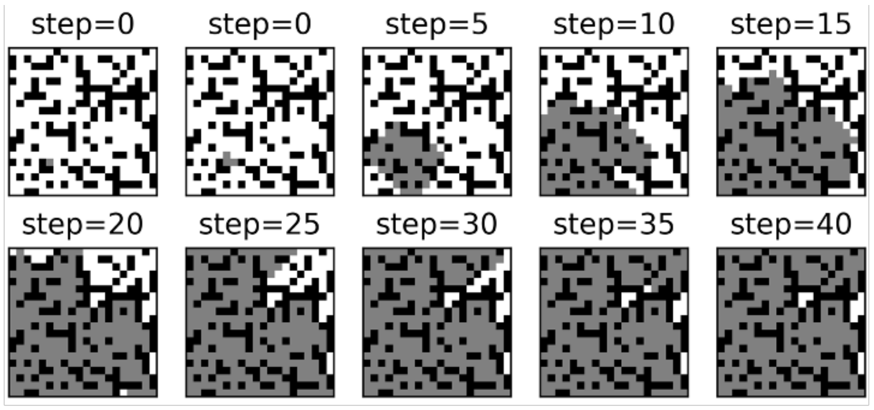


Figure 3: The advancement of fire in a maze, $dim = 20$, $p = 0.3$, $q = 1$.

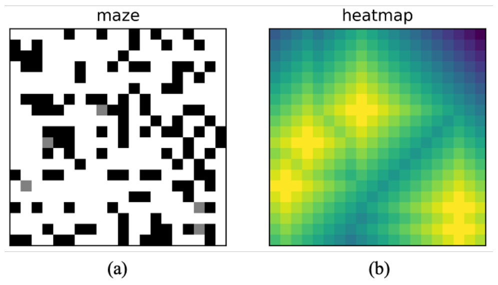
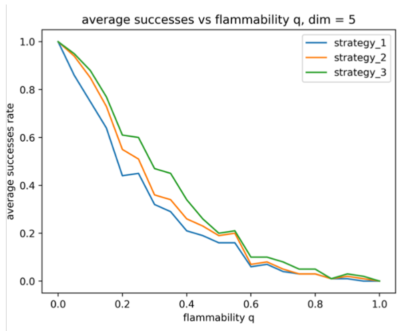


Figure 4: The potential influence heatmap of the current fire in a maze, $dim = 30$, $p = 0.3$, $q = 0.5$.

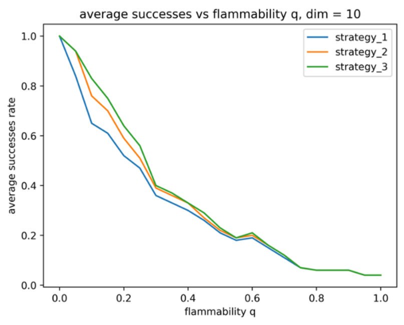
5 Analysis of three Strategies

Our first comparison of three Strategies is generating plots 'average successes vs flammability q' of different dimension as Fig 5, for every point shown in the figure, we calculate it with 10 mazes, restarting each 10 times with new initial fire locations, to get fair results. As we expected, the survival rate of strategy 2 is higher than strategy 1, because strategy 2 takes current state of maze and fire into consideration; and strategy 3 is better than strategy 2, because our strategy further consider the future state of the environment.

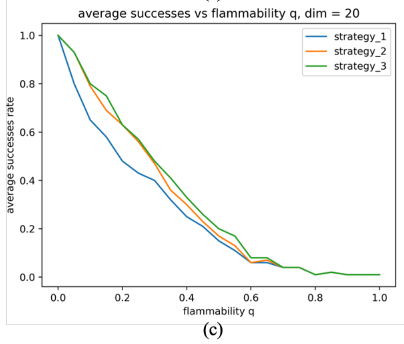
Then we analysis the time cost of each strategy with different dimensions when $p = 0.3$, and $q = 0.5$ as Table 3. Though our strategy can improve survival rate, it also requires more time to generate results. In a real burning maze, when the time is very limited, our strategy may not be the best choice.



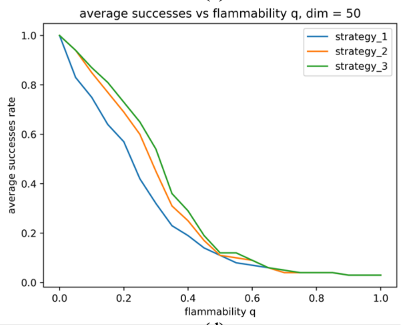
(a)



(b)



(c)



(d)

Figure 5: 'average successes vs flammability q' of three strategies, with different dimensions when the obstacle density p is 0.3. (a) $dim = 5$, (b) $dim = 10$, (c) $dim = 20$, (d) $dim = 50$,

Table 3: Time cost(ms) of each strategy with different dimensions, $p = 0.3$, $q = 0.5$

dim	5	10	20	50
strategy 1	0.2	0.7	2.9	35.4
strategy 2	0.5	3.1	26.1	839.1
strategy 3	1.3	6.0	839.1	1253.3

A Algorithms

Algorithm 1: Generate Maze

```
Input: $\boldsymbol{dim}$: dimension of the maze
       $\boldsymbol{p}$: obstacle density, probability of a cell being occupied
       $\boldsymbol{DFS(maze, start, goal)}$: return true if it is solvable from
       $start$ point to $goal$ point using Depth First Search
       $\boldsymbol{Generate\_Maze\_without\_Fire(dim, p)}$: generate maze
       with random occupied cells
       $\boldsymbol{Add\_Fire(maze)}$: add one random fire to the maze
Output: $\boldsymbol{maze}$: the random generated maze
1  Initialize $maze = None$ ;
2  while $True$ do
       /* generate occupied maze without fires */
3      while $True$ do
4          if $DFS(maze, [0, 0], [dim - 1, dim - 1])$ then
5              break
6          else
7              $maze\_nofire = Generate\_Maze\_without\_Fire(dim, p)$
8          end
9      end
       /* add fire to the maze */
10     for $i \in [1, tolerance]$ do
11         $maze, fire = Add\_Fire(maze\_nofire)$
            if $DFS(maze, [0, 0], fire)$ then
12              return maze
13         end
14     end
15 end
```

Algorithm 2: Strategy 1

```
  Input: $\boldsymbol{maze\_init}$: input maze
         $\boldsymbol{q}$: flammability rate
         $\boldsymbol{Generate\_Advanced\_Maze(maze\_init, q)}$: generate
         advanced maze at each step
         $\boldsymbol{Astar(maze, start, goal)}$: return the path from $start$ point
         to $goal$ point using $A^*$ Search
  Output: $\boldsymbol{survive}$: return True if survive
1 Initialize $shortest\_path = Astar(maze\_init, [0, 0], [dim - 1, dim - 1])$
   $advance\_maze = Generate\_Advanced\_Maze(maze\_init, q)$
   $max\_step = len(shortest\_path)$
   for $i \in [0, max\_step]$ do
2      $advanced\_maze\_step = advance\_maze[i]$
       $position\_step = shortest\_path[i]$
       if $advanced\_maze\_step[position\_step]$ is on fire then
3          return False
4      end
5 end
6 return True
```

Algorithm 3: Strategy 2

```
  Input: maze: input maze
         q: flammability rate
         Generate_Advanced_Maze(maze_init, q): generate
         advanced maze at each step
         Astar(maze, start, goal): return the path from start point
         to goal point using $A^*$ Search
  Output: survive: return True if survive
1 Initialize $advance\_maze = Generate\_Advanced\_Maze(maze, q)$
   current_start = [0,0]
   current_step = 0
   while True do
2     $shortest\_path = Astar(maze, current\_start, [dim-1, dim-1])$
      if $len(shortest\_path) == 0$ then
3         return False
4     end
5     $current\_step += 1$
      $advanced\_maze\_step = advance\_maze[current\_step]$
      $position\_step = shortest\_path[0]$
      if $advanced\_maze\_step[position\_step]$ is on fire then
6         return False
7     end
8     $maze = advanced\_maze\_step$
      current_start = position_step

9 end
```

Algorithm 4: Strategy 3

```
   Input: $\boldsymbol{maze}$: input maze
          $\boldsymbol{q}$: flammability rate
          $\boldsymbol{Generate\_Advanced\_Maze(maze\_init, q)}$: generate
          advanced maze at each step
          $\boldsymbol{Astar(maze, start, goal)}$: return the path from $start$ point
          to $goal$ point using $A^*$ Search
   Output: $\boldsymbol{survive}$: return True if survive
1  Initialize $advance\_maze = Generate\_Advanced\_Maze(maze, q)$
   current_start = [0,0]
   current_step = 0
   while $True$ do
2      $current\_step += 1$
       $advanced\_maze\_step = advance\_maze[current\_step]$
       $shortest\_path =$
       $Astar(advanced\_maze\_step, current\_start, [dim - 1, dim - 1])$
       if $len(shortest\_path) == 0$ then
3          $shortest\_path = Astar(maze, current\_start, [dim - 1, dim - 1])$
           if $len(shortest\_path) == 0$ then
4              return False
5          end
6          return False
7      end
8      $position\_step = shortest\_path[0]$
       if $advanced\_maze\_step[position\_step]$ $is$ $on$ $fire$ then
9          return False
10     end
11     $maze = advanced\_maze\_step$
       current_start = position_step
12 end
```