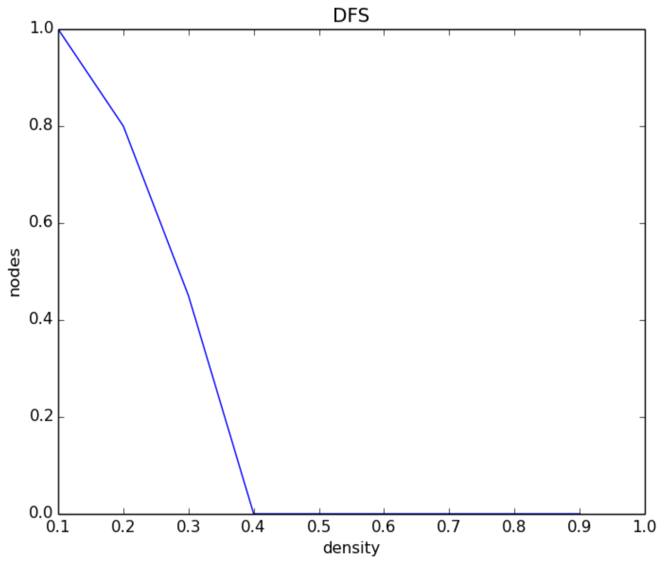
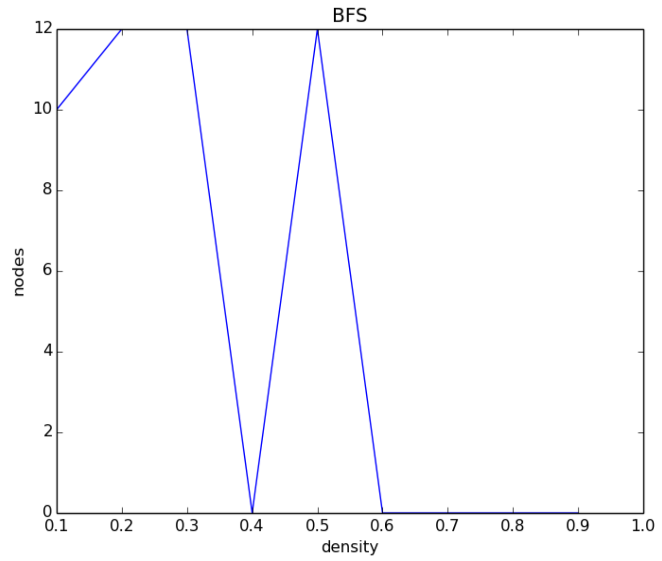
Maze on Fire

January 2, 2024

We implemented DFS and BFS algorithm using Stacks and Queues respectively to solve the static maze. Between the two DFS is a better choice since it consumes very less memory space and will reach the goal node in a less time period than BFS if it traverses in a right path. Unlike BFS it does not require extra memory space to store the nodes at every branch, which can be very high in problems like finding a path in a maze with higher dimensions.



Plot for DFS



Plot for BFS

If there is no path from S to G the nodes explored by A* will be more than the node than BFS.

Largest dimension that can be solved using DFS at p = 0.3 in less than a minute – low mid 3700's
Largest dimension that can be solved using BFS at p = 0.3 in less than a minute – 7000 - 8000
Largest dimension that can be solved using A* at p = 0.3 in less than a minute – less than 50

Our strategy 3 introduces a heuristic which will allow the agent to determine which tile to go to based off of distance from the goal and the distance from fire. It will choose the point that minimizes distance from goal and maximizes distance from fire. The formula goes as follows, where G is distance from goal and F is distance from fire:

$$h = G - \alpha * F$$

The value α will represent how much the agent prioritizes fire distance–the larger α is, the more the formula will prioritize fire. If α is too large, the agent will then develop an increasingly bigger tendency to get itself cornered into a wall, failing to solve the maze.

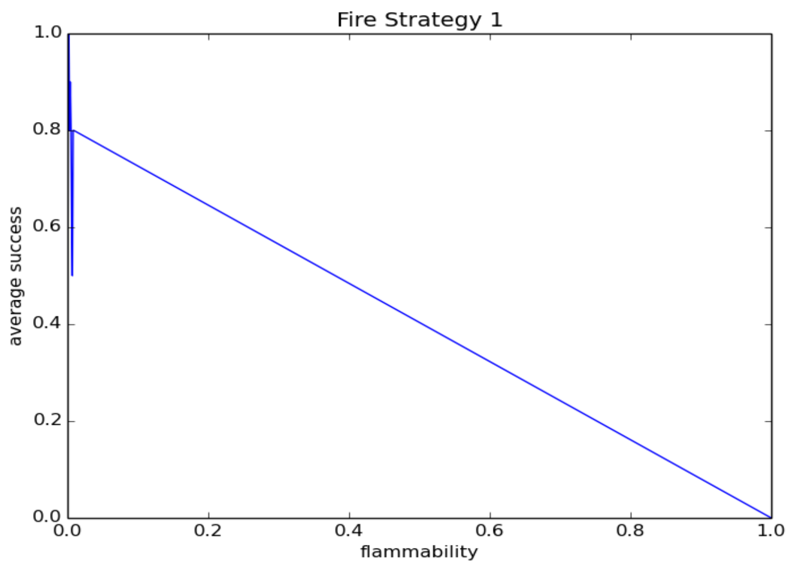
The strategy is along the same line as DFS but instead of a stack we are using a Priority Queue. Nodes are pushed into this priority queue and popped out based on the values calculated by the heuristic where the smallest value has the highest priority.

So while heap is not empty pop the node with highest priority from the queue and check if it is that node is on fire, if it is that means agent was burned before they could reach the destination. If the node is not on fire check to see if it is equal to the destination node it is a success and the agent reached its destination goal. If not reached the destination look for all the possible neighbors for that node. Check if they are valid (not outside the range of the grid, not a wall or on fire or visited before). Mark that node as visited and push it on the queue and continue.

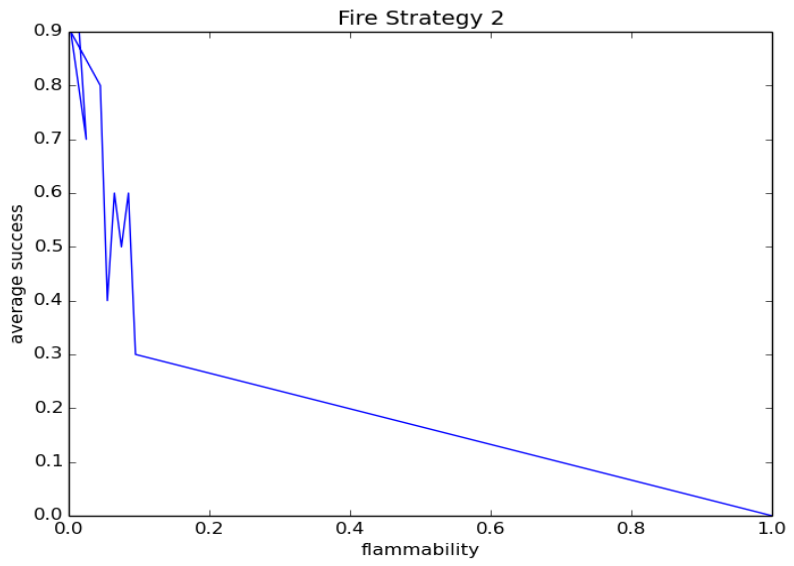
As the flammability increases the the average success rates decreases gradually for all the three strategies.

They are different in the average time it takes for each strategy to find a path from start to goal. Strategy 3 takes the longest since there are more computations involved with each step compared to strategy 2 and strategy 1 where it is only checking for a cell with fire and is not concerned with future state of fire.

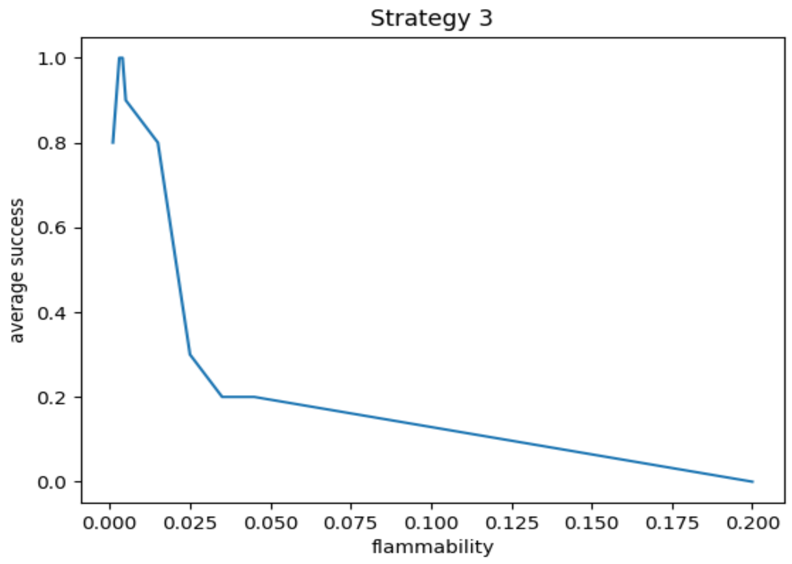
Plot for Strategy 1 (agent does not check updates for fire):



Plot for Strategy 2 (will now check for fire):



Plot for Strategy 3 (our own strategy):



The drawback of strategy 3 is that if we increase the alpha the agent can corner itself to a wall failing to solve the maze. So if we had unlimited computational resources we would introduce β to out original heuristic formula. The beta will represent how much the agent prioritizes distance from the wall. The value of β should balance the value chosen for α in order to make the agent less greedy. The new formula will be:

$$h = \beta * G - \alpha * F$$

If we could only take 10 seconds using heuristics will not work since calculating them takes longer than 10 seconds. Instead we could give the agent a maximum sight range where it can see the fire and if there is no fire greater than,say 10 blocks then continue with strategy 2.