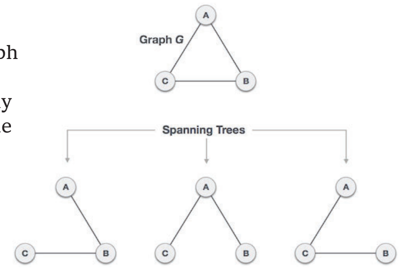
The A-Maze-ing Race

Generating and solving different mazes based on differerent search algortihms and comparing their effectiveness against each other

By Michael Ku, Enzo Smajlaj, and Yunzhu Chen

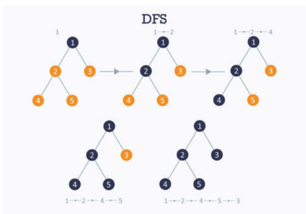
What is a Spanning Tree?

A spanning tree is a subset of a connected graph that includes all its vertices and just enough edges to keep it connected without forming any cycles. In simpler terms, it connects every node in the graph using the minimum number of edges possible, which is always one less than the number of vertices (V−1).



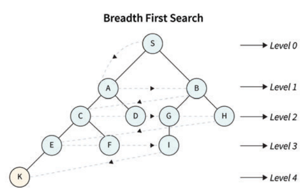
Depth First Search

Depth-First Search (DFS) is an algorithm that explores a graph or tree by starting at a root node and following one branch as far as possible before backtracking. It uses a stack (either explicit or via recursion) to remember which nodes to visit next.



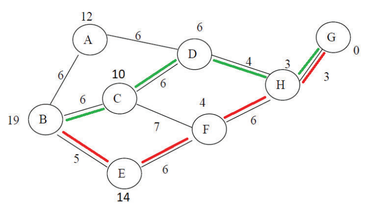
Breadth First Search

Breadth-First Search (BFS) is an algorithm that explores a graph or tree level by level, visiting all neighboring nodes before moving on to the next depth. It uses a queue to keep track of nodes to visit in order.



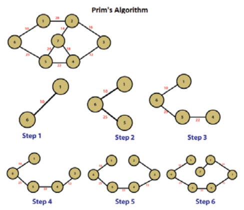
Greedy Search

Greedy Search is an algorithm that makes the locally optimal choice at each step based on a heuristic, aiming to find a global optimum efficiently. It prioritizes nodes that appear closest to the goal, often using a heuristic function to guide its decisions.



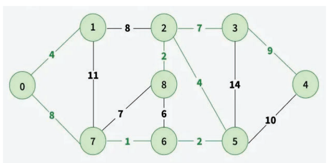
Prims Algorithm

Prim's Algorithm is a greedy method used to find the Minimum Spanning Tree (MST) of a weighted, connected graph. It builds the tree by starting from one vertex and repeatedly adding the smallest edge that connects a vertex in the tree to a vertex outside it.



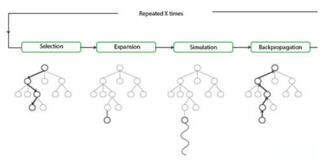
Kruskals Algorithm

Kruskal's Algorithm is a greedy method for finding the Minimum Spanning Tree (MST) of a weighted, connected graph. It works by sorting all edges by weight and adding them one by one to the tree, skipping any that would create a cycle.



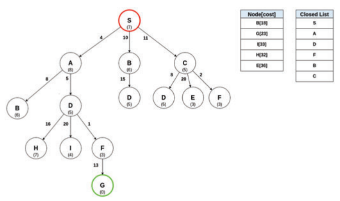
Monte Carlo

The Monte Carlo algorithm is a computational method that uses random sampling and statistical modeling to approximate solutions to complex problems. It's often used when exact solutions are difficult to compute, such as in optimization, integration, or probabilistic simulations.



A*

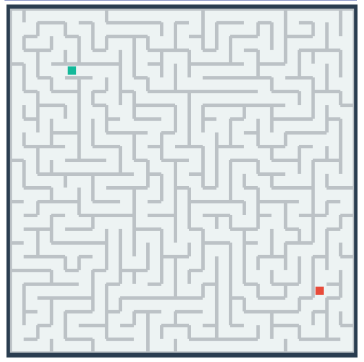
The A* Search Algorithm is an informed pathfinding method for finding the optimal path in a weighted graph. It guides its search by combining the known cost from the start, g(n), with an admissible heuristic estimate to the goal, h(n).



Generated Mazes

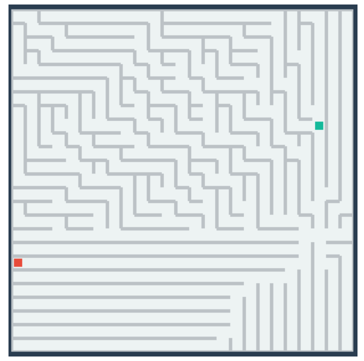
(25x25, no weight, no added loops)

Depth First Search

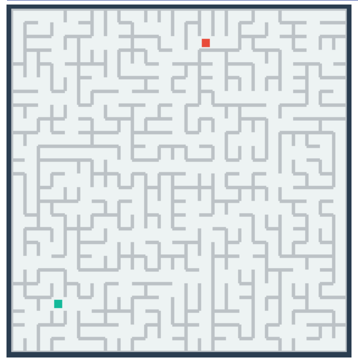


Breadth First Search

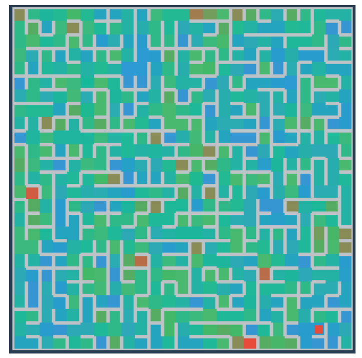
Greedy Search



Prims Algorithm

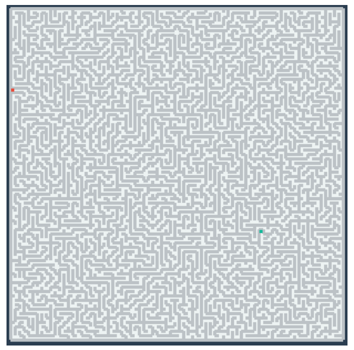


Kruskals Algorithm

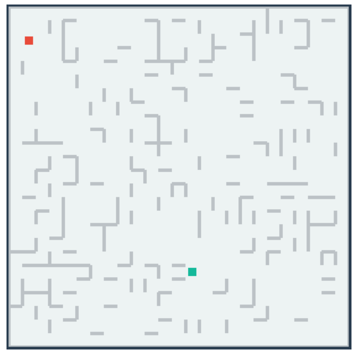


Maze Variables (Size, Loops, Rewards, Weights)

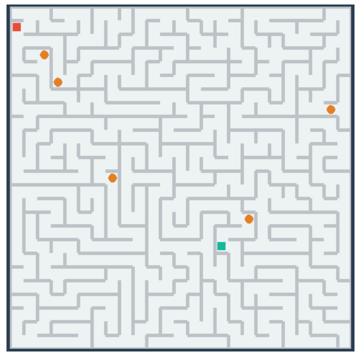
Width & Height: Control the width and height of the maze in terms of the reachable cells.



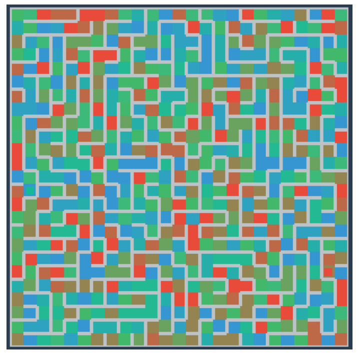
Loop_Percent: Controls the density of cycles by removing dead-ends.



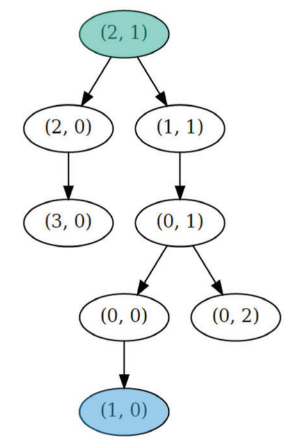
Num_Rewards: Specifies the number of reward cells that must be visited before the goal (and randomly places them).



Max_Weight: Sets the maximum cost for single cells post-generation that will be randomly distributed.



Decision Tree



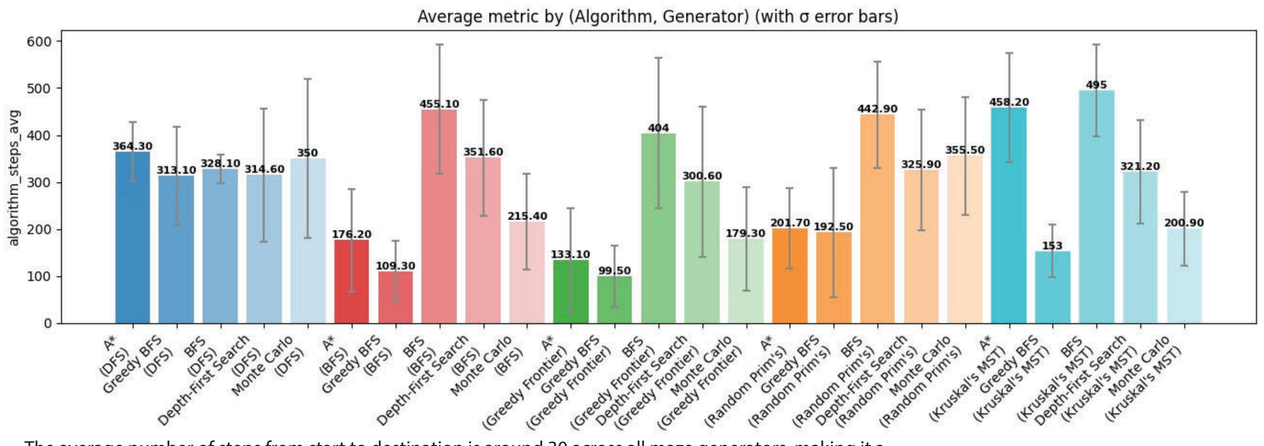
A simple example of decision tree where the robot starts at (2,1) and ends at (1,0)

Results of Simulations

Figure 1 (Right): The average algorithm steps for each solving and generation algorithm pair for a 25x25 maze with added variations after 10 simulations. The generation algorithms are color coded with the solving algorithms light coded. The standard deviation is visualized with an error bar.

Metrics

unique_explored: The number of unique cells visited by the algorithm during path-finding.
min_path_cost: The cost of the shortest path found by the algorithm.
frontier_max: The number of look-ahead nodes that are kept in the algorithm's memory.
algorithm_steps: The number of iterations necesarry for the algorithm until finished



The average number of steps from start to destination is around 30 across all maze generators, making it a fair comparison for most algorithms -- except for Kruskal's, which requires noticeably more steps.
When there are more open loops (i.e., multiple possible solutions), BFS takes significantly more steps to finish the maze, but its consistency is also the best.
In terms of finding the shortest path, Greedy BFS performs the worst significantly when there are more loops.
The efficient performance of the greedy algorithm is interesting as we believe this to be most similiar to how a human would intuitively solve a maze. We look at the end and start and determine a line that connects this, which is an interpretation of our greedy algorithm.
We note that many of the metrics are positively correlated, as exploring more nodes does require additional algorithmic iterations.

General Conclusion

We originally theorized that an algorithm would be efficient at solving a maze if it also generated that maze. Looking at the results of Figure 1, we see this is not always true. The BFS algorithm produces very narrow hallow-like mazes, which means that, comparatively, switching between the first sections of each hallway is inefficient compared to other algorithms. We also note that there is a lot of deviation regarding the average algorithm steps for BFS generated mazes, as there are large differences for near and far reward locations since the walls have minimal impact. On the other hand, the greedy generation creates an easy greedy application as it guides the user to the end, which is in line with the greedy approach.
They do better on mazes generated with Prim's algorithm than with Kruskal's. This makes sense because Kruskal's algorithm produces its MST in a more scattered way, making the maze harder to solve.

Resources

https://www.geeksforgeeks.org/dsa/depth-first-search-or-dfs-for-a-graph/
https://www.geeksforgeeks.org/dsa/breadth-first-search-or-bfs-for-a-graph/
https://www.geeksforgeeks.org/dsa/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/
https://www.caktusgroup.com/blog/2015/09/24/introduction-monte-carlo-tree-search-1/
https://en.wikipedia.org/wiki/A*_search_algorithm
https://algs4.cs.princeton.edu/43mst/