Path Planning over Grid Graphs using Dijkstra's and Bresenham's algorithms

ENGS 104: Final Project Presentation

DARTMOUTH

November 10, 2021

Spencer Bertsch

DARTMOUTH ENGINEERING

Agenda

- Background and Applications
- Graph Design and Setup
- Dijkstra& Bresenham Results and Analysis
- Simulation Results and Analysis
- Live Demo

Robot path planning has applications in UAVs and drone motion planning, autonomous driving, and underwater robotics

- This article from Duke discusses how ultrafast motion planning, similar to shortest path planning in our case, could make autonomous vehicles safer.
- As the vehicle considers which action to choose as it explores the environment, quickly calculating the optimal path to the goal state is very important.

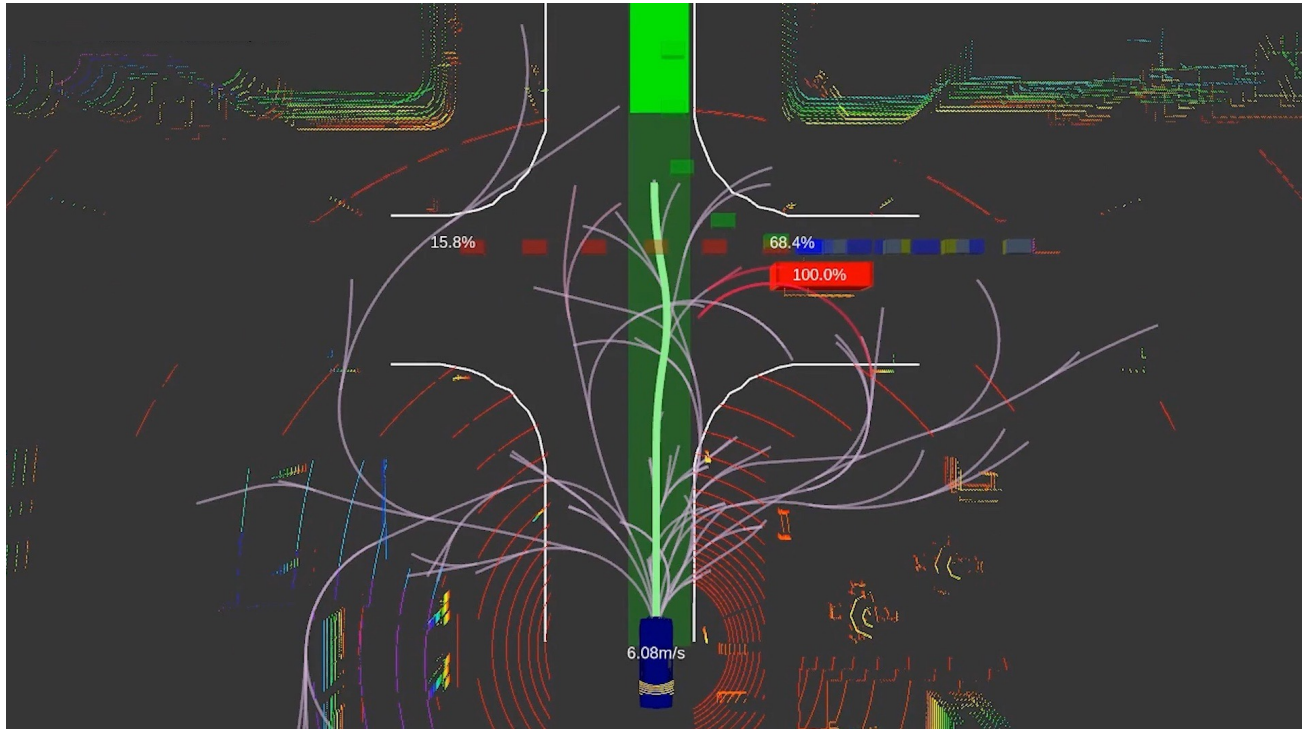


Image Source: otc.duke.edu

Robot path planning has applications in UAVs and drone motion planning, autonomous driving, and underwater robotics

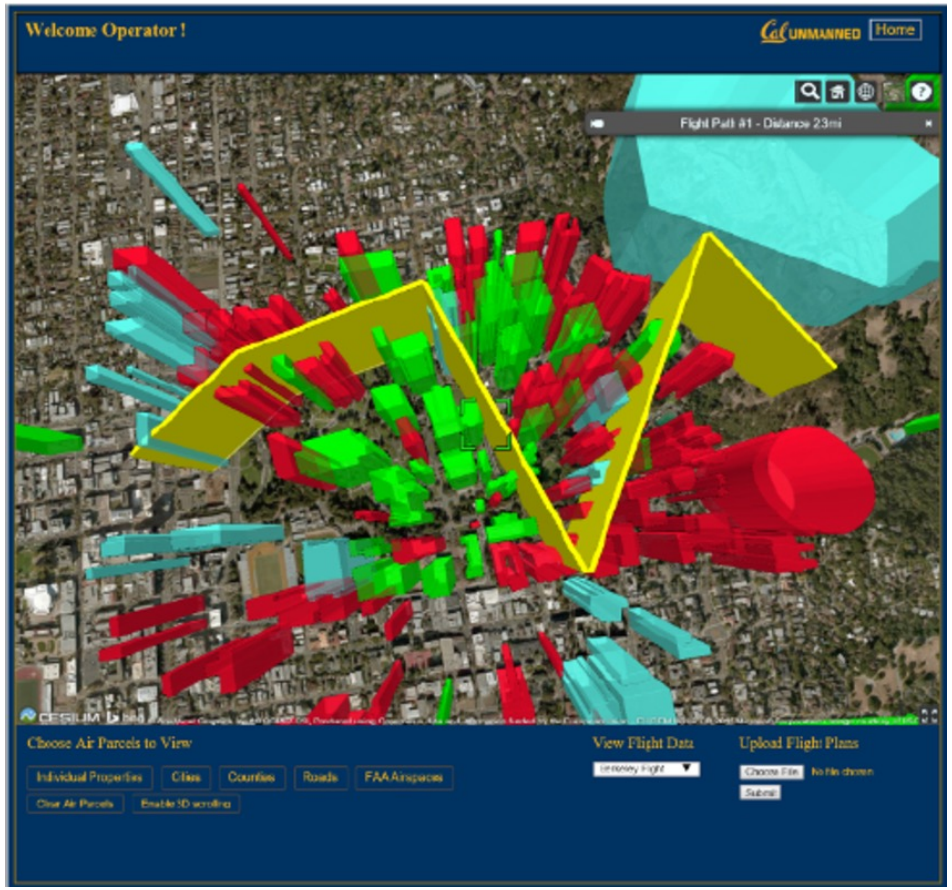


Image Source: researchgate.net

- Drone motion planning is a problem in three-dimensional space, but this problem can be converted to a simpler two-dimensional robot path planning problem by constraining the drone to remain at a fixed height and still avoid obstacles.
- As startups offer more capabilities for drone deliveries from prescription medicine to other small packages, drone motion planning in cities will require great topographical maps of urban landscapes, and highly performant motion planning algorithms.

Grid graphs and shortest path (SP) algorithms can be used for optimal robot motion/path planning

- Robot path planning generally considers a robot and a known or unknown environment in which the robot needs to reach a goal while minimizing cost.
- The cost function can depend on many things including the time taken, gasoline or electricity consumed, number of turns made by the robot, amount of breaking and accelerating, etc.
- For the analysis in this project, the cost is the amount of fuel consumed when the robot moves from one node in the graph to another node.
- The fuel consumed is linearly proportional to either the Euclidian or Manhattan distance

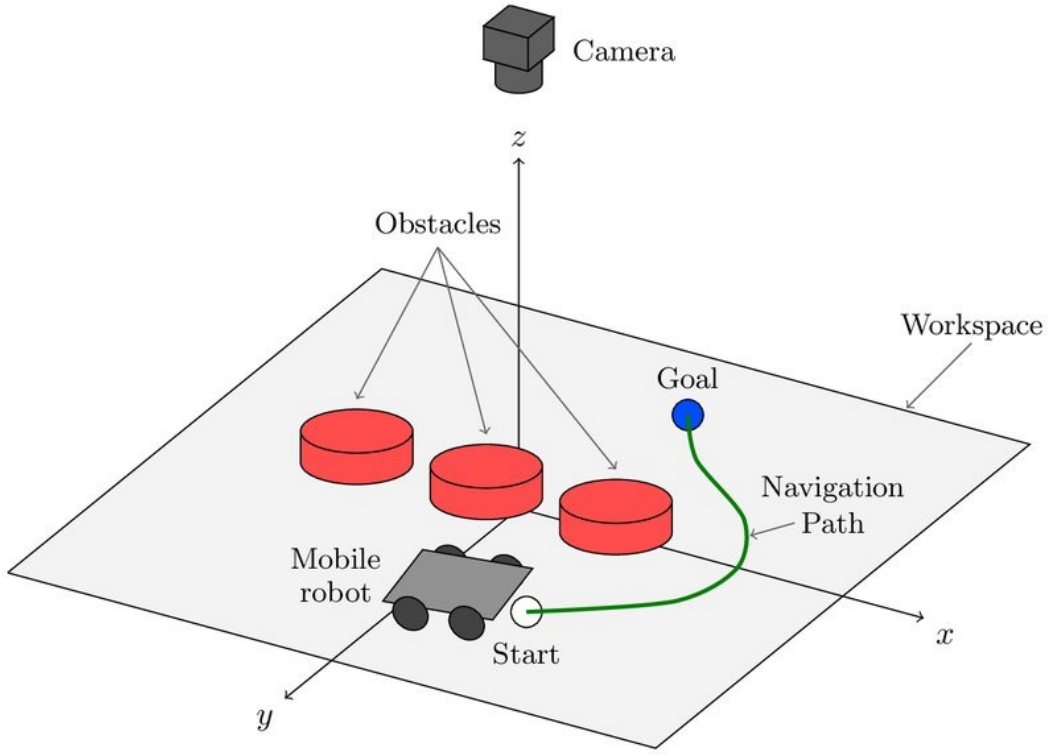
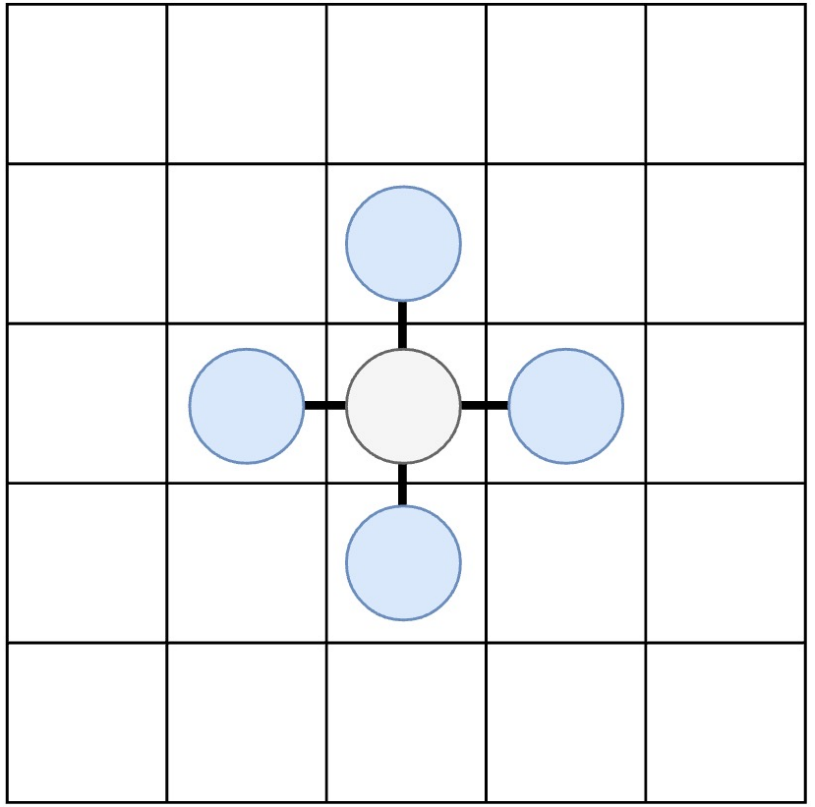


Image Source: spiedigitallibrary.org

Grid graphs can be constructed using grids filled with open cells and blocked cells.

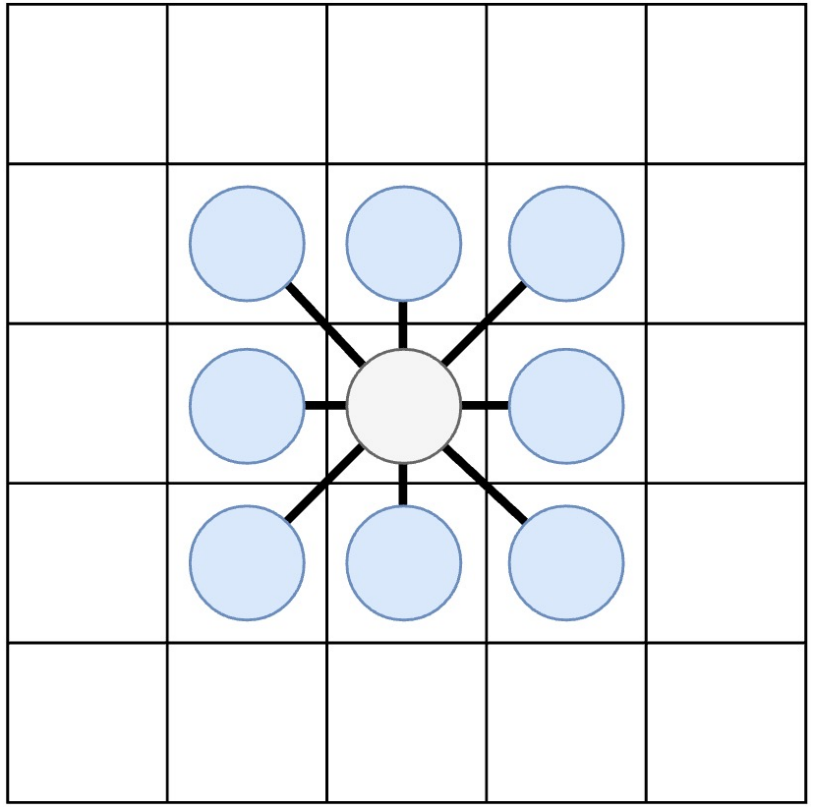
- The graph structure depends on several things including the neighbor model: 4 neighbors or 8 neighbors
- The graph structure also depends on the maximum allowed distance of connections in the grid. Distance (d) has a min value of 1, and a max value of n where n is the distance of the longest side of the grid. For us, (n) = side length because we use square grids.



- 4-Neighbor model graph
- Distance (d) is set to 1
- There are 0 blocked cells

Grid graphs can be constructed using grids filled with open cells and blocked cells.

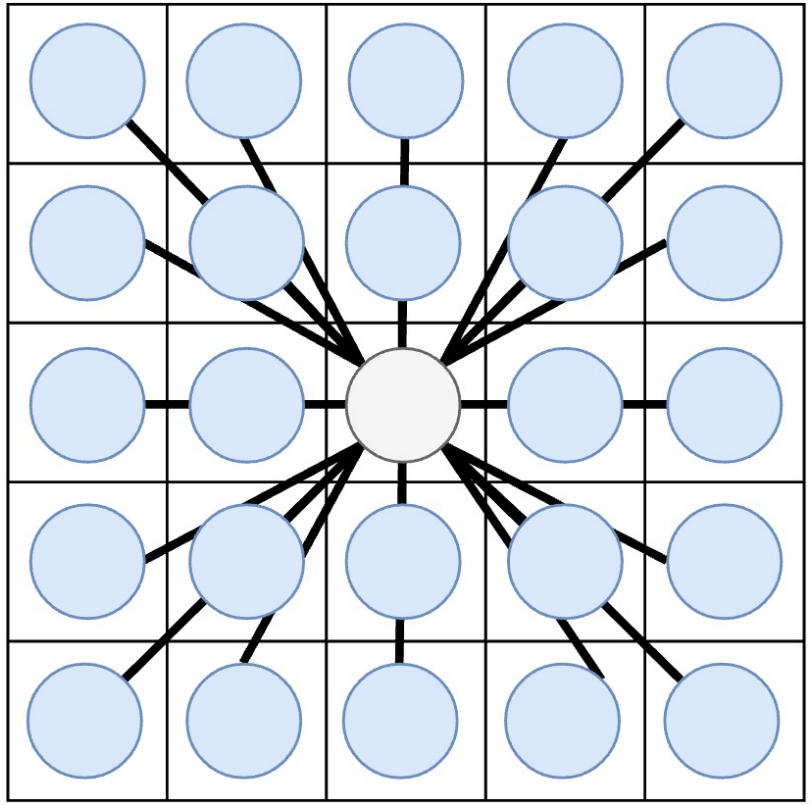
- The graph structure depends on several things including the neighbor model: 4 neighbors or 8 neighbors
- The graph structure also depends on the maximum allowed distance of connections in the grid. Distance (d) has a min value of 1, and a max value of n where n is the distance of the longest side of the grid. For us, (n) = side length because we use square grids.
- We can see in the graph here that we have four additional neighbor nodes connected to the cell (2, 2) where the robot currently sits



- 8-Neighbor model graph
- Distance (d) is set to 1
- There are 0 blocked cells

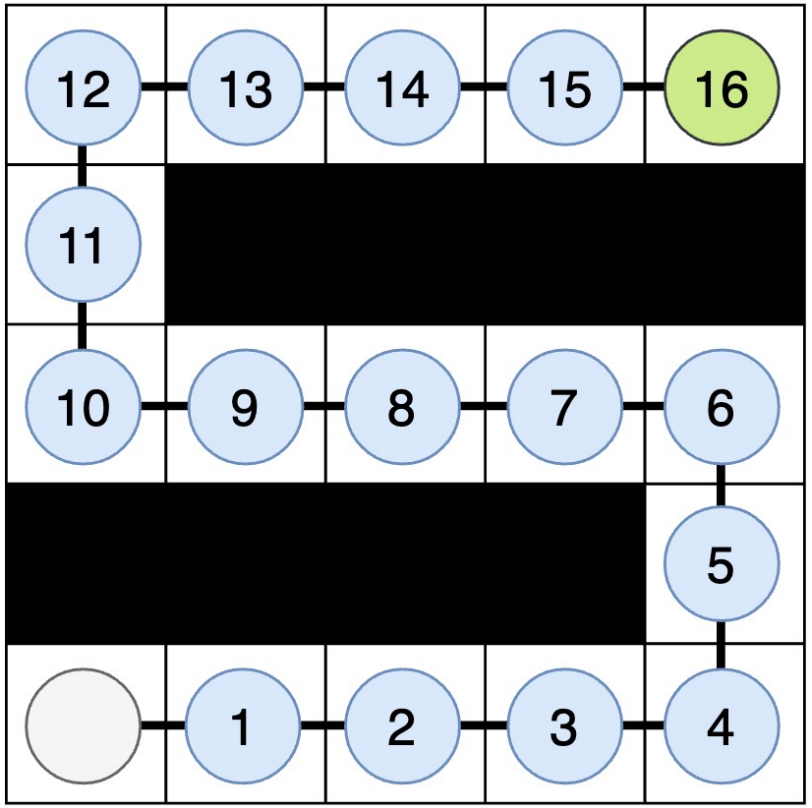
Grid graphs can be constructed using grids filled with open cells and blocked cells.

- The graph structure depends on several things including the neighbor model: 4 neighbors or 8 neighbors
- The graph structure also depends on the maximum allowed distance of connections in the grid. Distance (d) has a min value of 1, and a max value of n where n is the distance of the longest side of the grid. For us, (n) = side length because we use square grids.
- Here we can see that when we use the 8-Neighbor model and set distance to 2, then the robot can have at most 24 neighbor nodes, assuming no cells are blocked and the robot is at least 2 cells away from walls



- 8-Neighbor model graph
- Distance (d) is set to 2
- There are 0 blocked cells

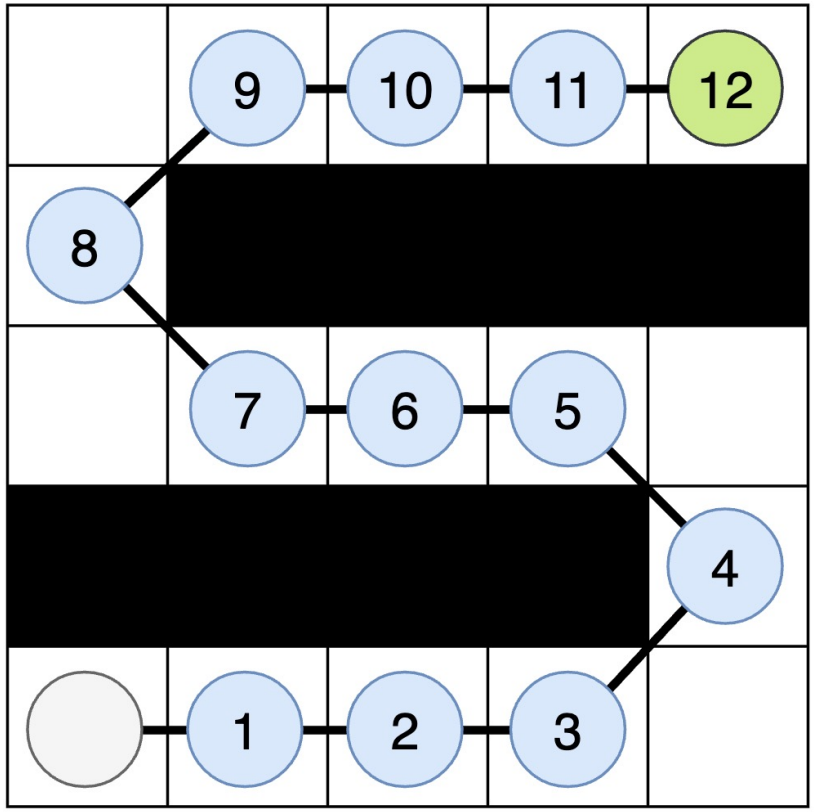
Activating different graph building and search conditions has a large impact on the resulting optimal path weight



- 4-Neighbor model graph
- Distance (d) is set to 1
- There are 8 blocked cells

- Take this maze for example, when the graph was constructed using the 4-neighbor model and the distance is set to 1, then every node needs to be explored to reach the goal, and the path .
- What would the graph look like if we were to use the 8-Neighbor model instead of the 4-Neighbor model? How many nodes would exist on the optimal path?

Activating different graph building and search conditions has a large impact on the resulting optimal path weight

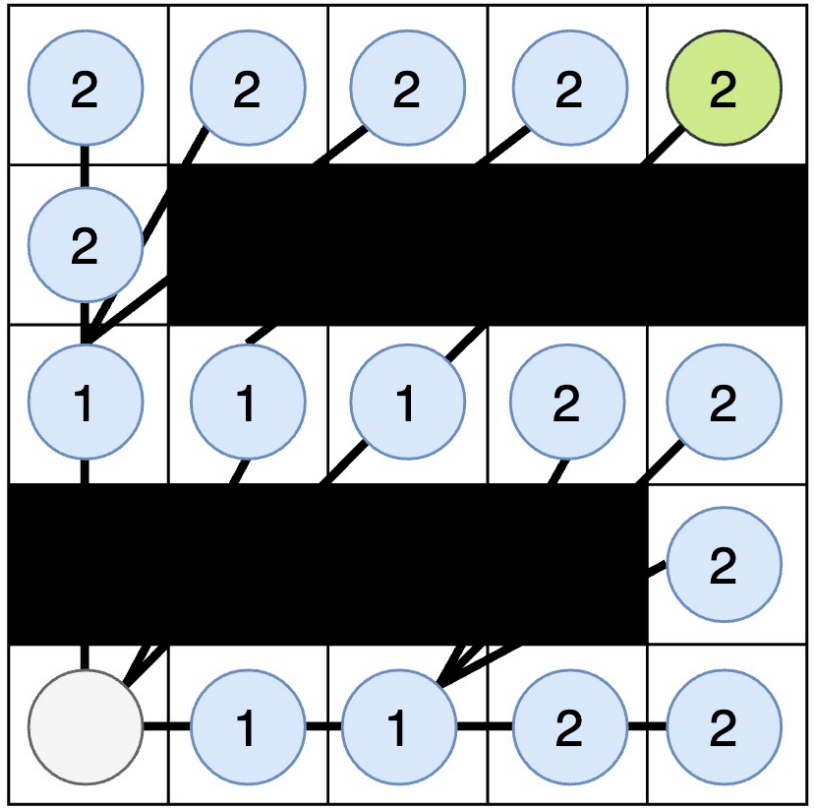


- 8-Neighbor model graph
- Distance (d) is set to 1
- There are 8 blocked cells

- Now only 12 nodes need to be expanded to reach the goal! In addition the weight of the final path is slighlt decreased because four coners could be cut on the way to the goal node, using 1.41 units of fuel instead of 2 in each instance.

- What would the graph look like if we were to use the 8-Neighbor model and set the Distance (d) to 2? How many nodes would lie on the optimal path?

Activating different graph building and search conditions has a large impact on the resulting optimal path weight

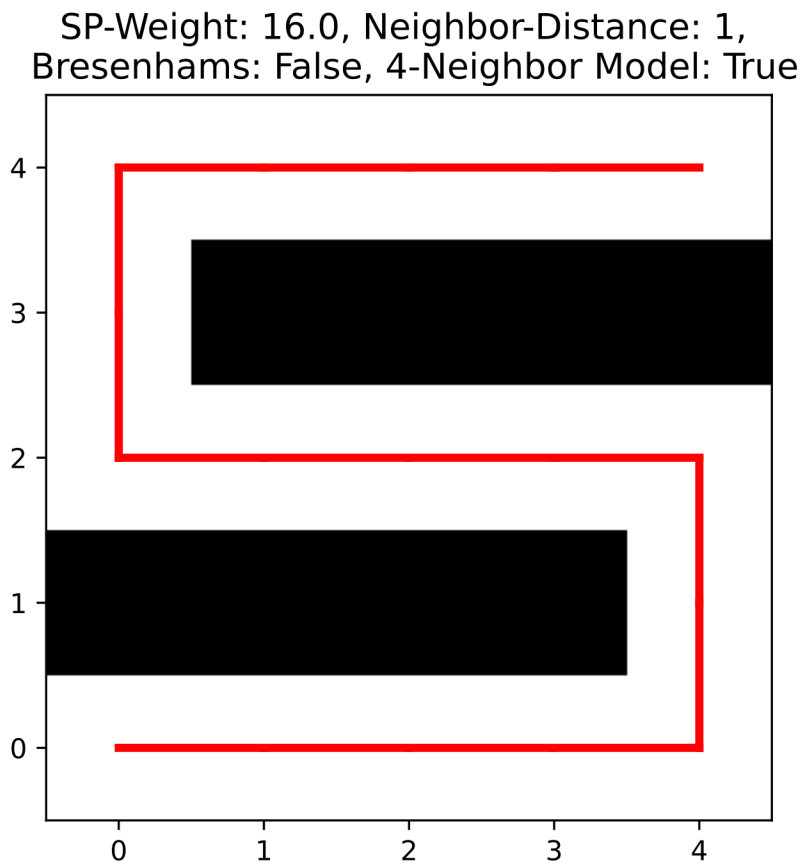


- 8-Neighbor model graph
- Distance (d) is set to 1
- There are 8 blocked cells

- Only 2 nodes would be on the optimal path from the starting state to the goal node! Because we set distance to 2 and the 8-Neighbor model is turned on, the robot can move diagonally two cells away, jumping over blocked cells.
- So now we know what the graphs look like under different conditions, but what are their optimal path weights?
- Here we can begin to consider the results of the Dijkstra-Bresenham framework implemented in Python

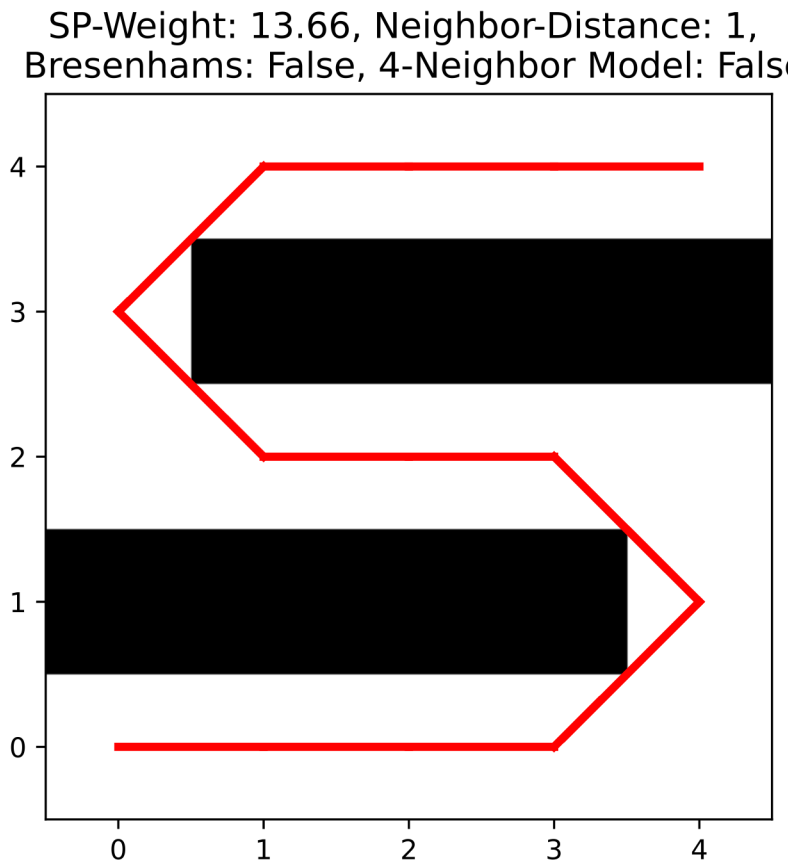
Here we use the Shortest Path (SP)-Weight as the metric to minimize for our two-dimensional robot path planning.

- Remember here that the the SP-Weight can be calculated using either the Euclidian distance or the Manhattan distance. Because we will consider diagonal robot movement, Euclidian distance will be used for the remainder of this presentation.
- We can see from the example to the right that we get the exact path weight we expect! The robot needed to burn 16 units of fuel to reach the goal with this graph/search configuration. How much fuel would we save if we changed the graph to an 8-Neighbor model?



Here we use the Shortest Path (SP)-Weight as the metric to minimize for our two-dimensional robot path planning.

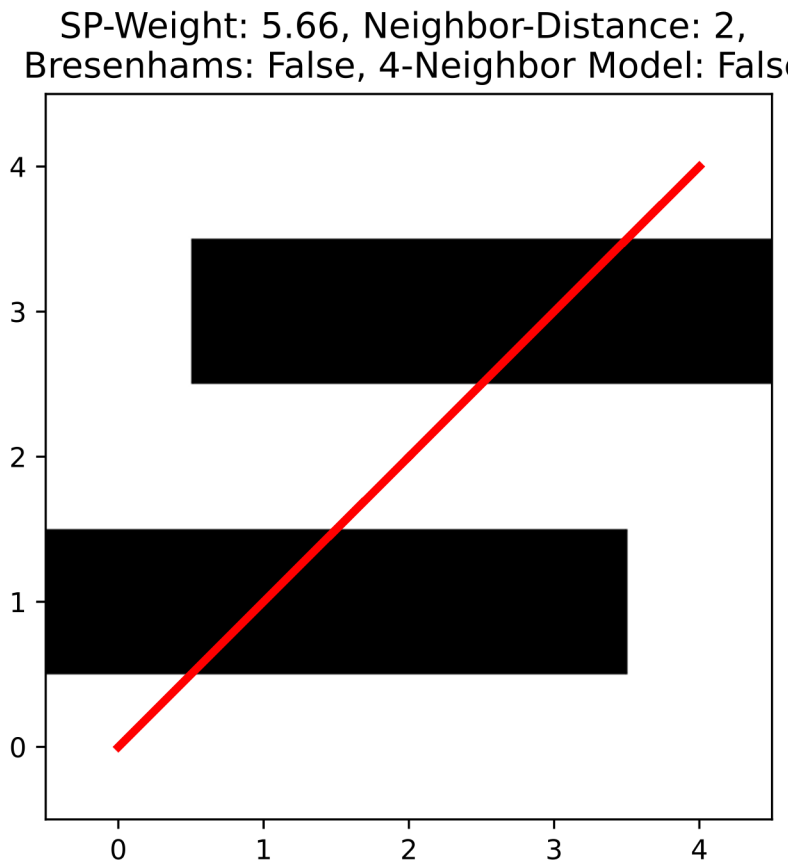
- We were able to reduce the fuel consumed by 2.33 units! This makes sense because we use ~1.41 units of fuel instead of 2 units each time we cut the corner around a blocked wall in the maze.
- Finally, how much fuel does it take to reach the goal if we set the Distance (d) to 2? What is the minimum possible fuel that could be used to reach the goal state from the starting state?



Here we use the Shortest Path (SP)-Weight as the metric to minimize for our two-dimensional robot path planning.

- We were able to reduce the fuel consumed all the way down to a mere 5.66 units! This happens to be the optimal solution for this maze because:

$$Optimal\ Length = sqrt(4^2 + 4^2) = 5.66$$

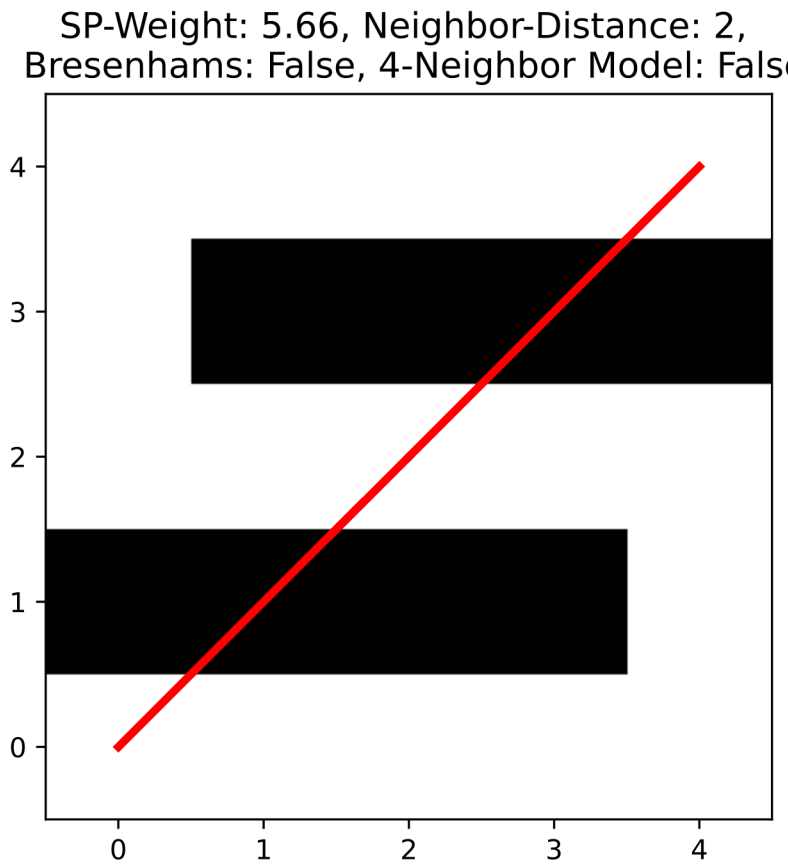


Here we use the Shortest Path (SP)-Weight as the metric to minimize for our two-dimensional robot path planning.

- We were able to reduce the fuel consumed all the way down to a mere 5.66 units! This happens to be the optimal solution for this maze because:

$$Optimal\ Length = sqrt(4^2 + 4^2) = 5.66$$

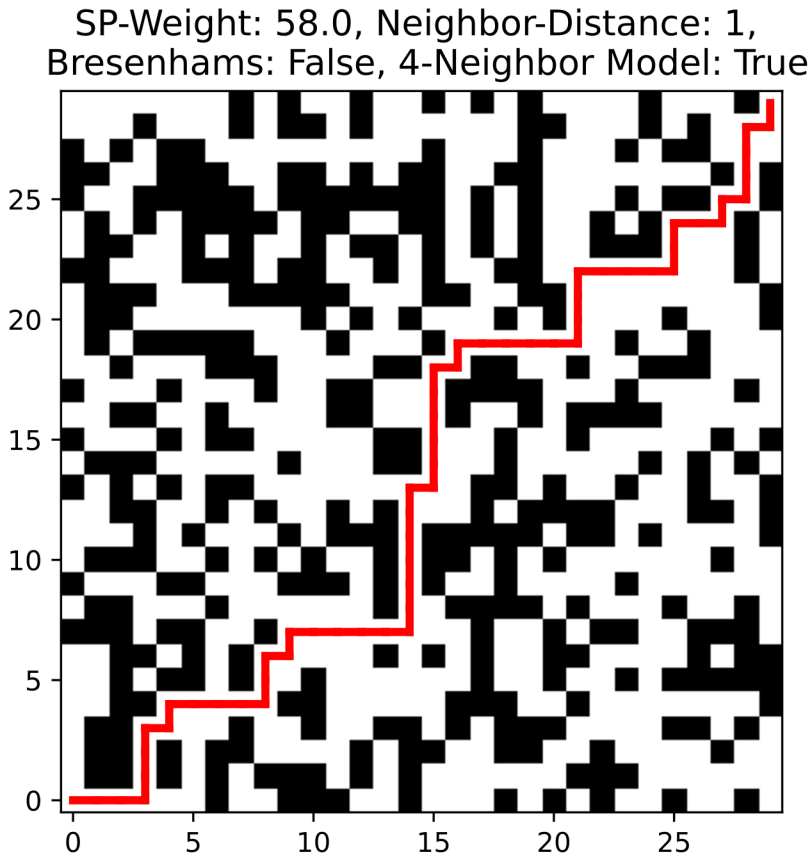
- So now we know how the setup works. But how can we use this framework to learn more about path planning and motion planning? Next, we look at randomly generated images of grid graphs for analysis.



This same framework can be applied to simulated grids that each have a predetermined probability of blocked cells

- Here we can see a grid graph with an edge length of 30 cells. The optimal path found by Dijkstra's algorithm happens to be a solution in which 58 units of fuel are burned, representing the optimal solution for a square graph with side-length 30.

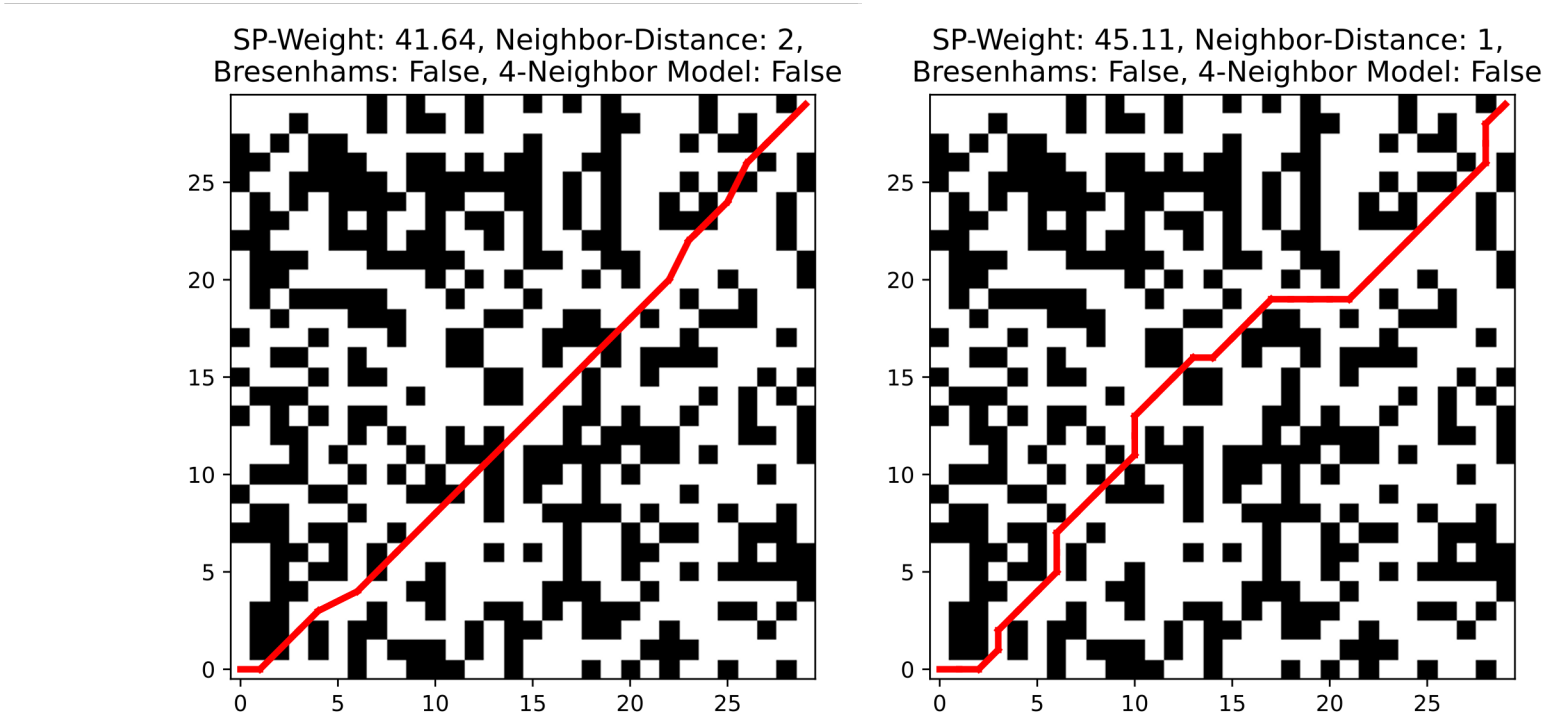
- How much fuel can we save by changing the model used? How much fuel can be saved by allowing the robot to jump over obstacles (where Distance is set to 2)?



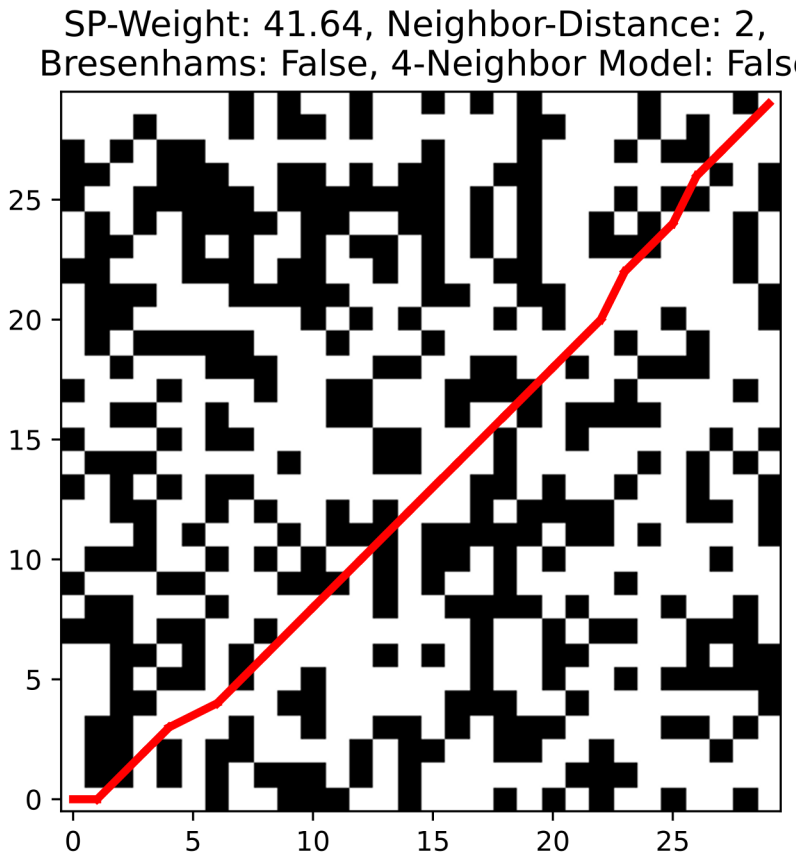
This same framework can be applied to simulated grids that each have a predetermined probability of blocked cells

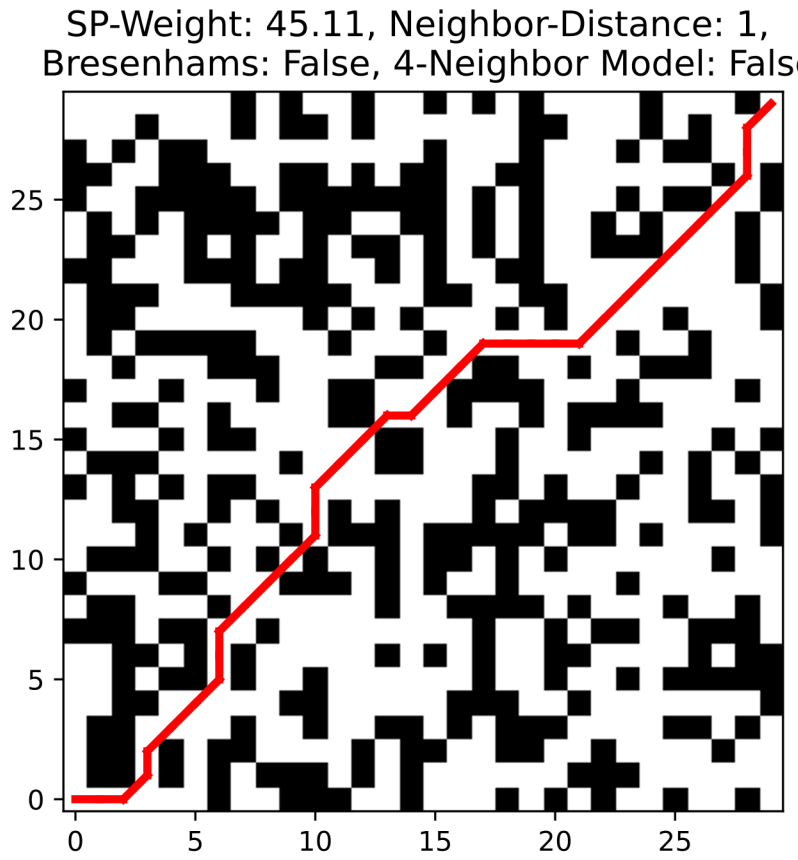
- Here we can see the large impact of shifting from the 4-Neighbor model to the 8-Neighbor model, and by increasing the Neighbor Distance metric to 2, allowing the robot to jump over single blocked cells.

- What happens as we increase Neighbor Distance even more? How will that impact the fuel consumed by the robot?

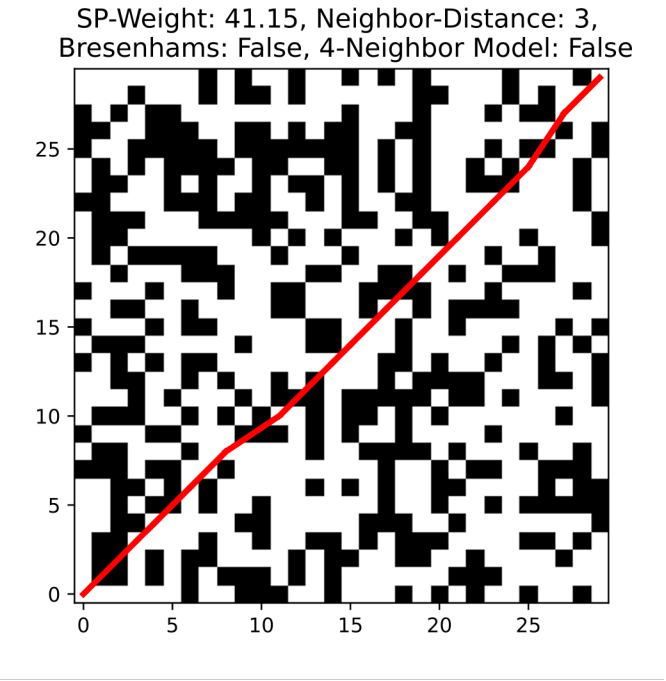


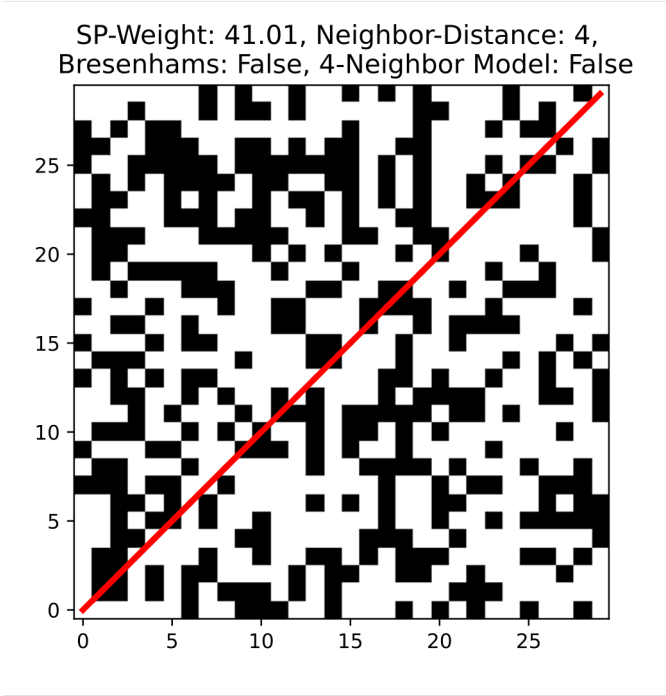
Allowing the robot to move between corners of blocked cells decreases fuel consumed to 45.11 units and allowing the robot to jump over blocked cells further reduces consumption to 41.64 units.

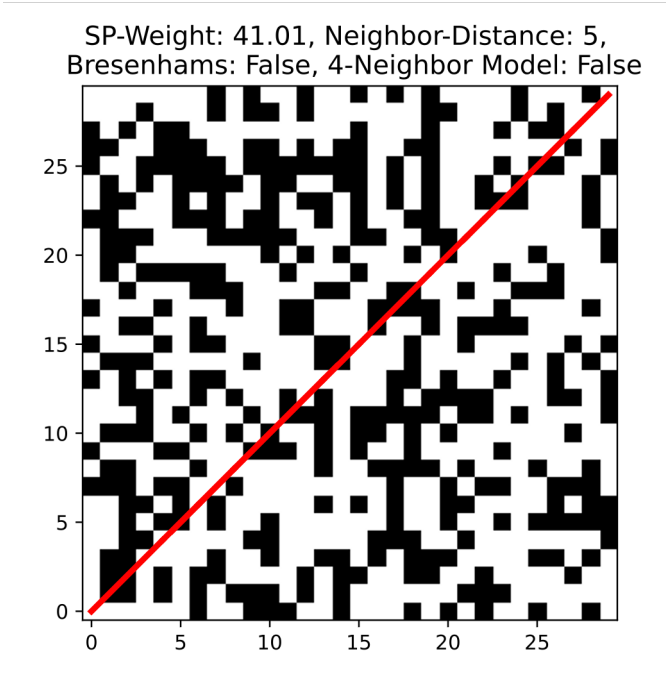




As we increase the distance of allowed neighbors in the graph, the SP solution weight decreases and eventually reaches optimality

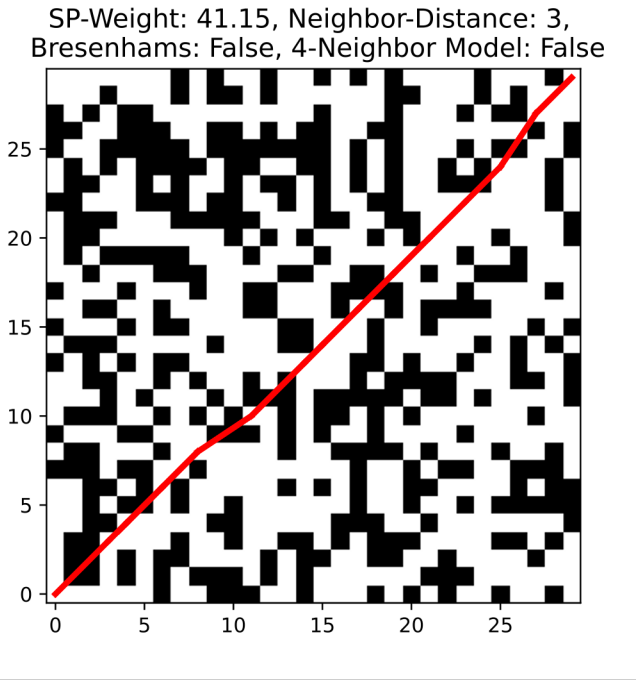


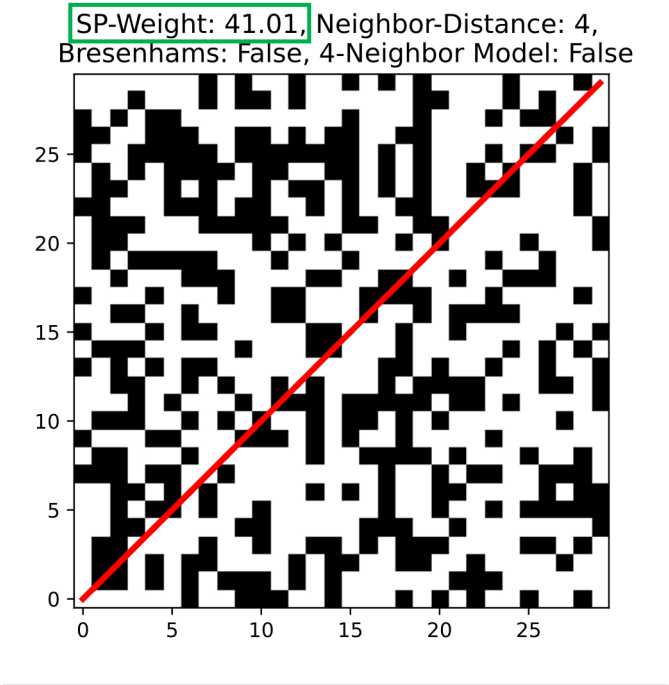


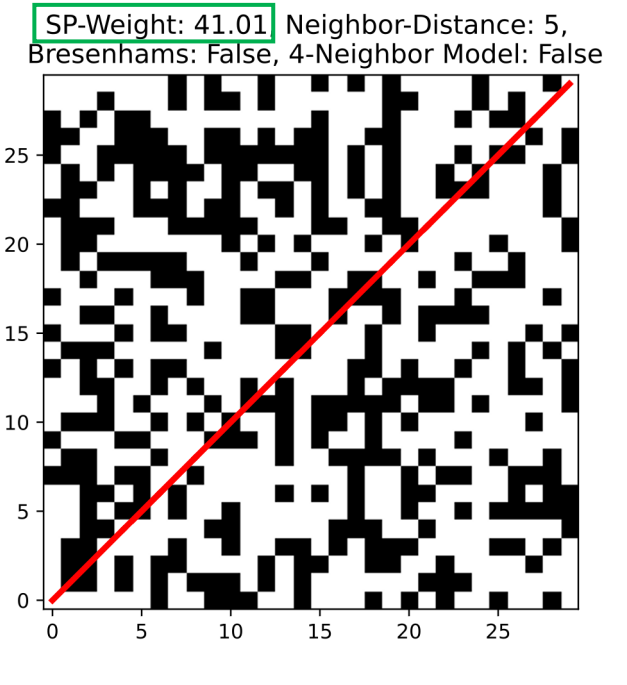


As we increase the distance of allowed neighbors in the graph, the SP solution weight decreases and eventually reaches optimality

$$sqrt(29^2 + 29^2) = 41.01$$

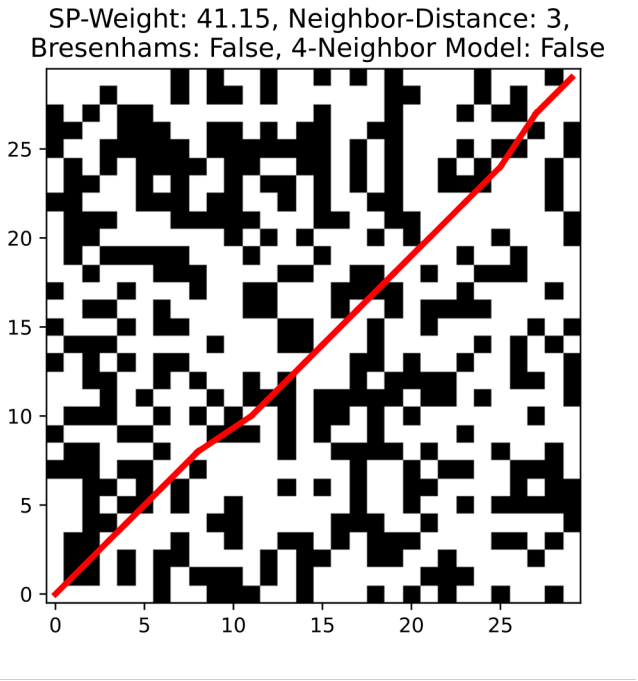


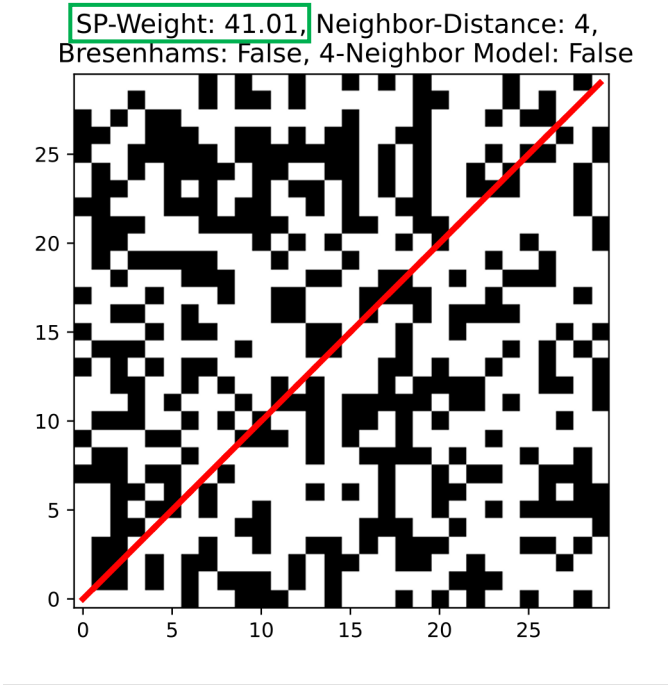


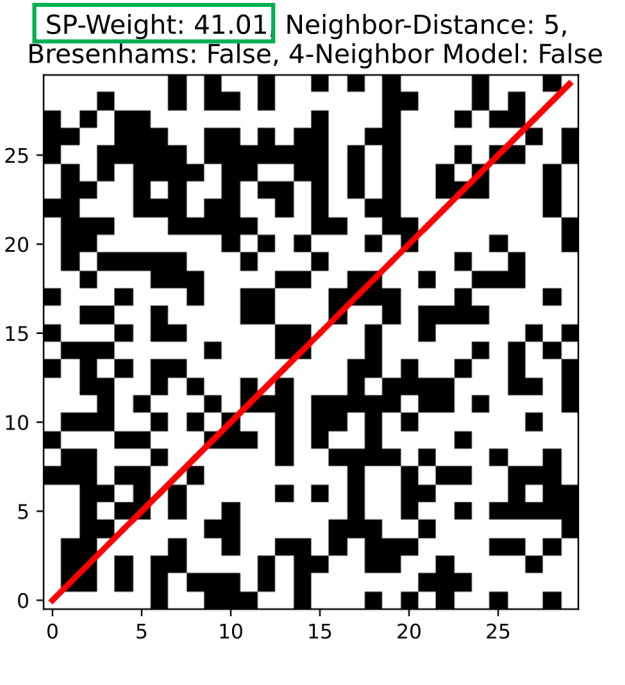


As we increase the distance of allowed neighbors in the graph, the SP solution weight decreases and eventually reaches optimality

$$sqrt(29^2 + 29^2) = 41.01$$

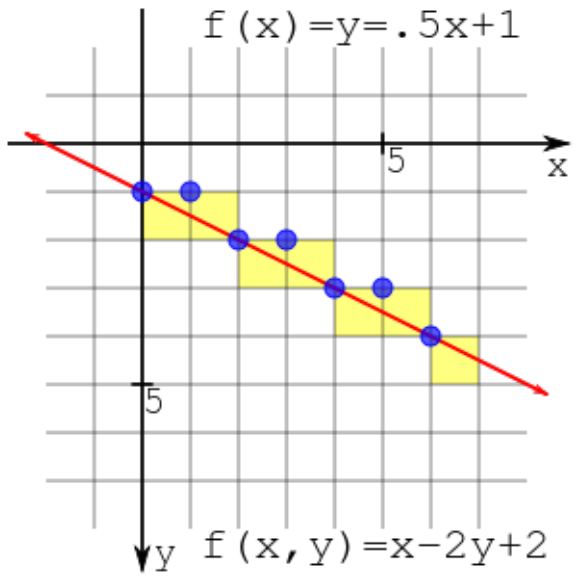






- How does turning on the Bresenham line algorithm impact fuel burned?

The Bresenham line algorithm is a way of finding the cells that are intersected by a line created by connecting two points [x1, y1] and [x2, y2].



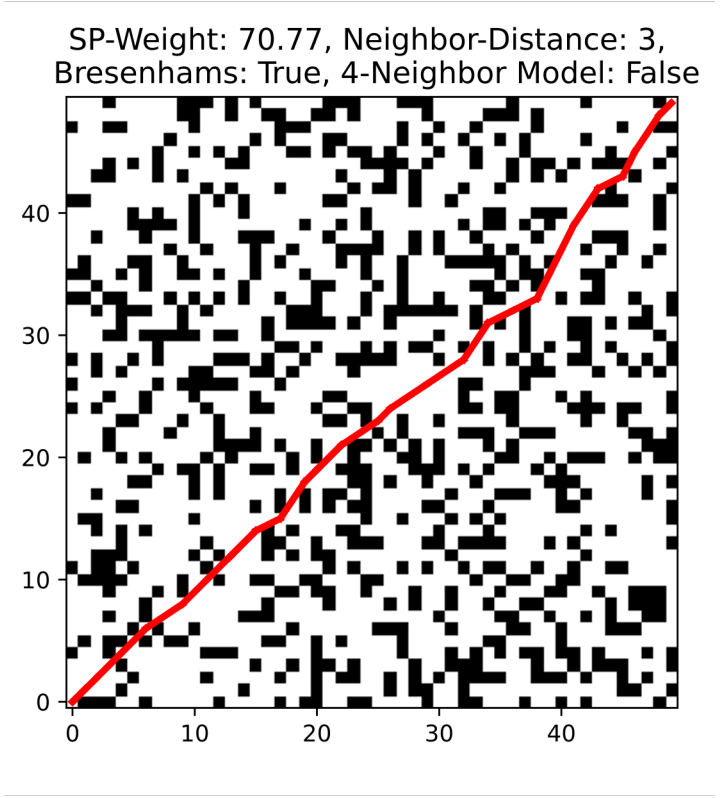
upload.wikimedia.org

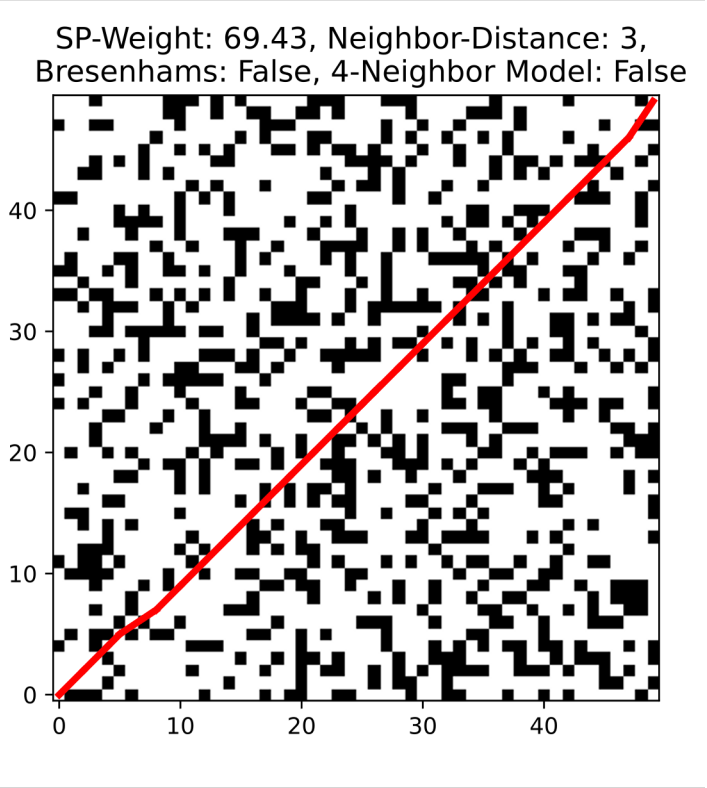
- In our grid graph implementation, applying the Bresenham algorithm has the effect of reducing the number of neighbor nodes to only those that don't require the robot to move through a line of obstacles.
- There are several possible implementations of the Bresenham line algorithm, but the one used allows the robot to move between the corners of blocked cells.
- The affect that the Bresenham algorithm has on the robot's efficiency is most interesting when we increase the Neighbor Distance to at least 5, which would usually allow the robot to reach optimality (a straight line) when reaching the goal node.

Restricting the graph constriction using the Bresenham line algorithm increases the fuel consumed even as Neighbor Distance increases.

- Here we can see that an additional 1.34 units of fuel were expended on the optimal path to the goal when the graph was constructed using the Bresenham line algorithm.

- Although this algorithm still allows the robot to move between blocked cells, the robot must go around solid lines of blocked cells, even with the Neighbor Distance is large enough for the robot to jump over the obstacle.

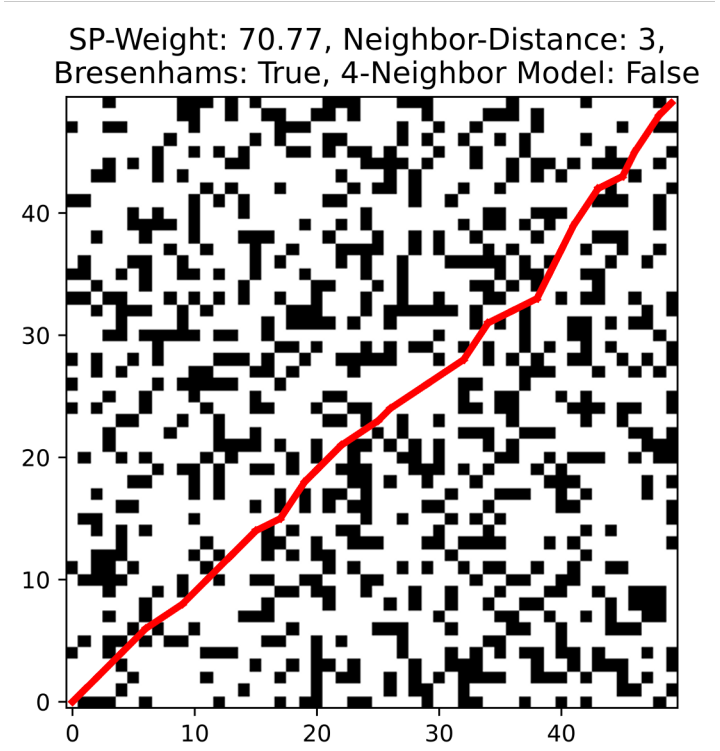


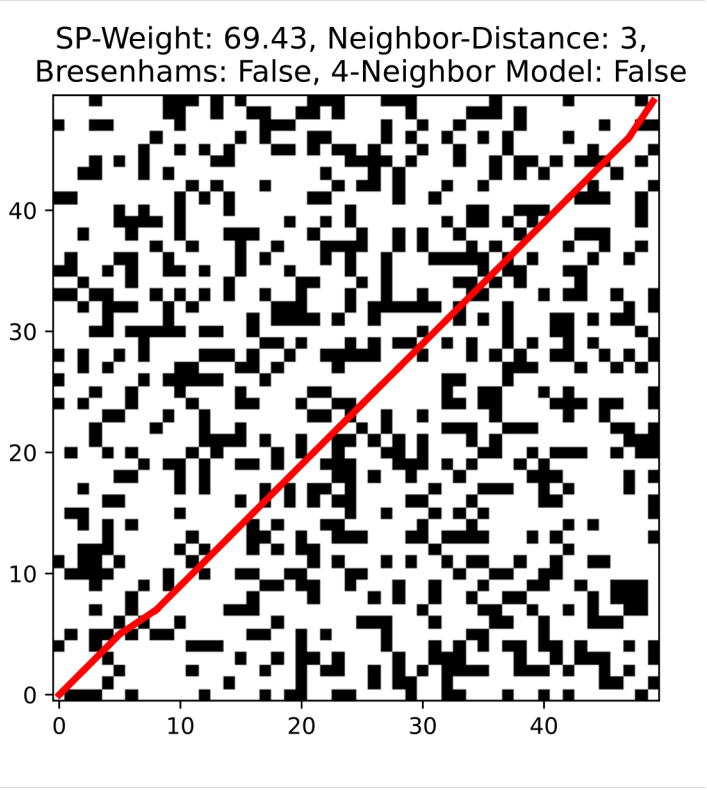


Restricting the graph constriction using the Bresenham line algorithm increases the fuel consumed even as Neighbor Distance increases.

- Here we can see that an additional 1.34 units of fuel were expended on the optimal path to the goal when the graph was constructed using the Bresenham line algorithm.

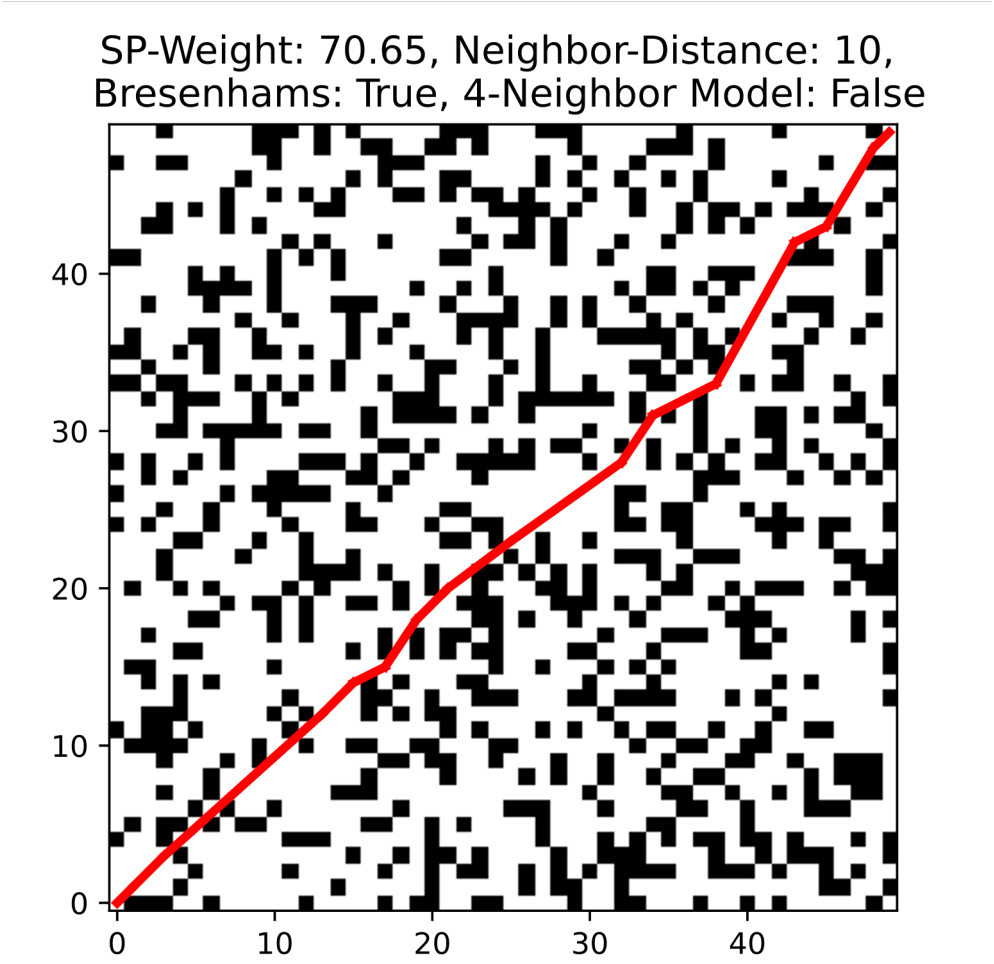
- Although this algorithm still allows the robot to move between blocked cells, the robot must go around solid lines of blocked cells, even with the Neighbor Distance is large enough for the robot to jump over the obstacle.





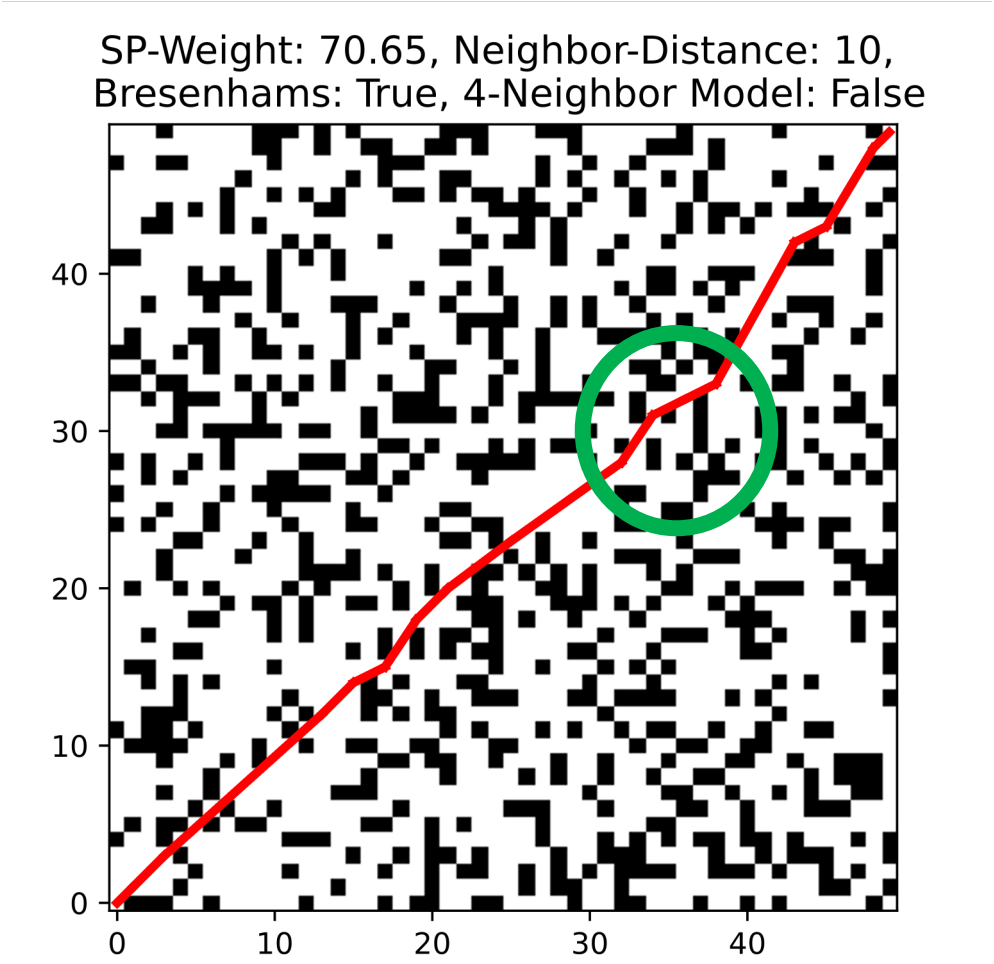
- What is Neighbor Distance was increased to 10?

Restricting the graph constriction using the Bresenham line algorithm increases the fuel consumed even as Neighbor Distance increases.



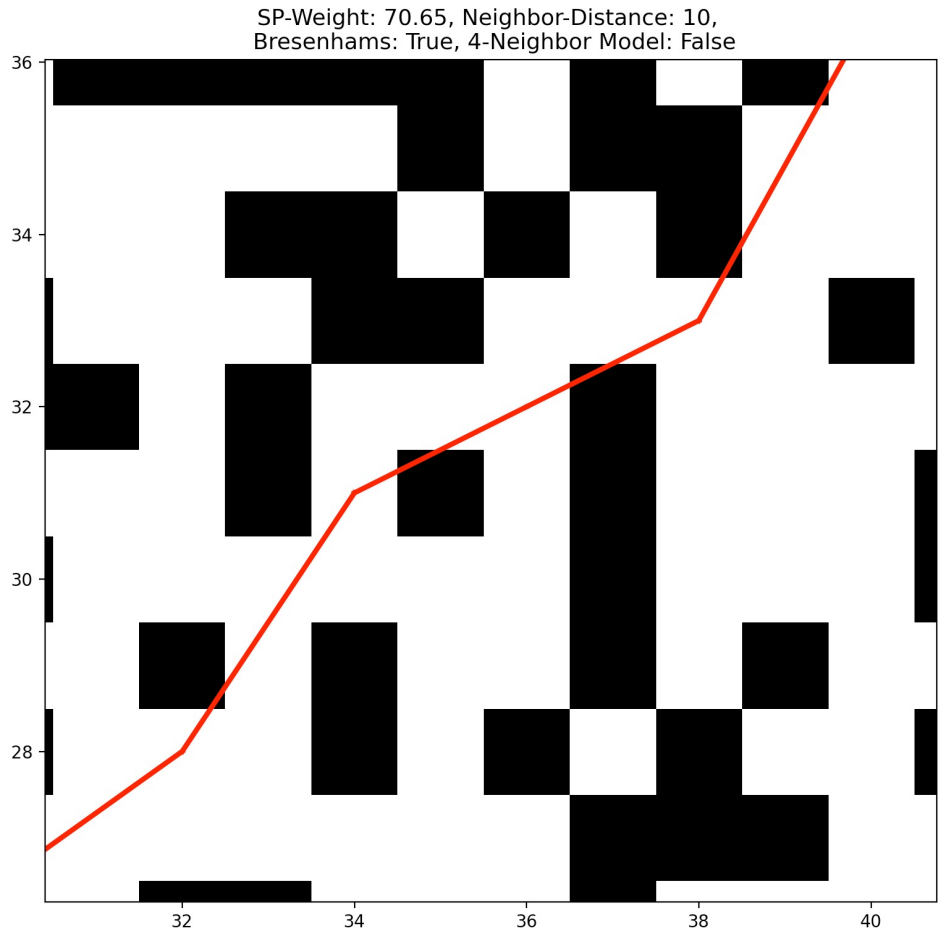
- Even with the Neighbor distance is increased to 10, the Bresenham line algorithm restricts the motion of the robot through the maze.
- Although this algorithm still allows the robot to move between blocked cells, the robot must go around solid lines of blocked cells, even with the Neighbor Distance is large enough for the robot to jump over the obstacle.

Restricting the graph constriction using the Bresenham line algorithm increases the fuel consumed even as Neighbor Distance increases.



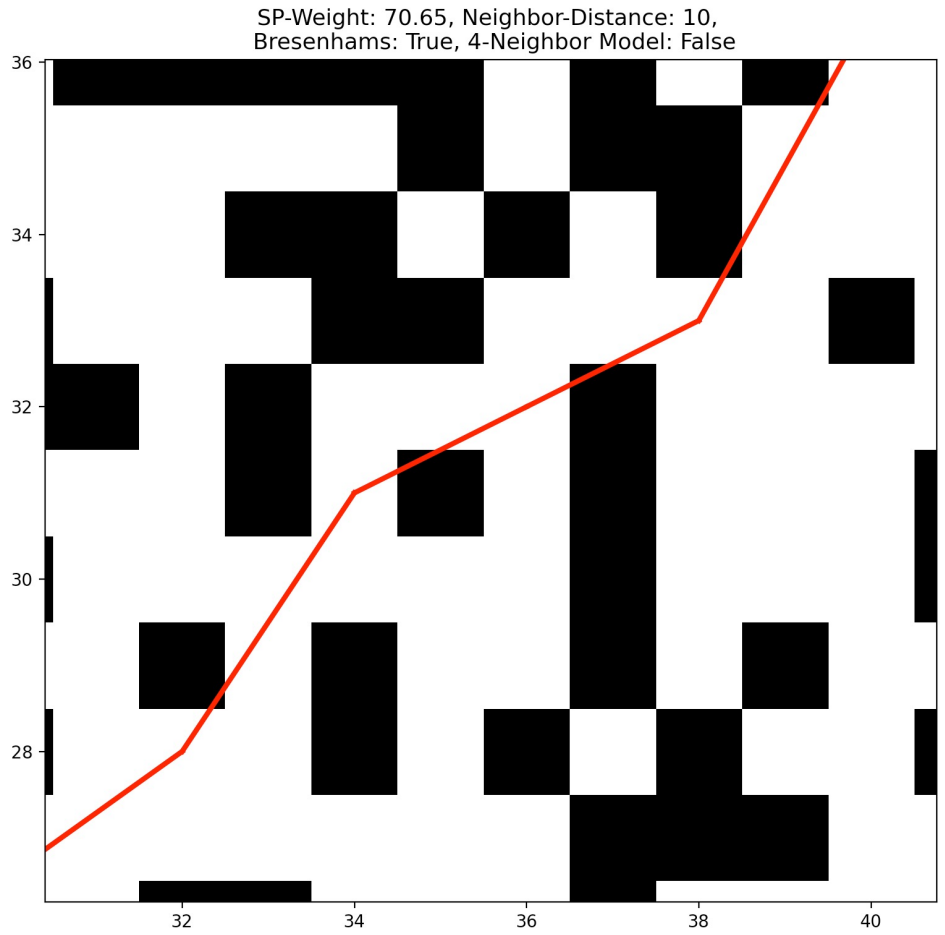
- Even with the Neighbor distance is increased to 10, the Bresenham line algorithm restricts the motion of the robot through the maze.
- Although this algorithm still allows the robot to move between blocked cells, the robot must go around solid lines of blocked cells, even with the Neighbor Distance is large enough for the robot to jump over the obstacle.

Restricting the graph constriction using the Bresenham line algorithm increases the fuel consumed even as Neighbor Distance increases.



- By zooming in we can explore the behavior of the robot when constrained by the Bresenham line algorithm.
- Although this algorithm still allows the robot to move between blocked cells, the robot must go around solid lines of blocked cells, even with the Neighbor Distance is large enough for the robot to jump over the obstacle.

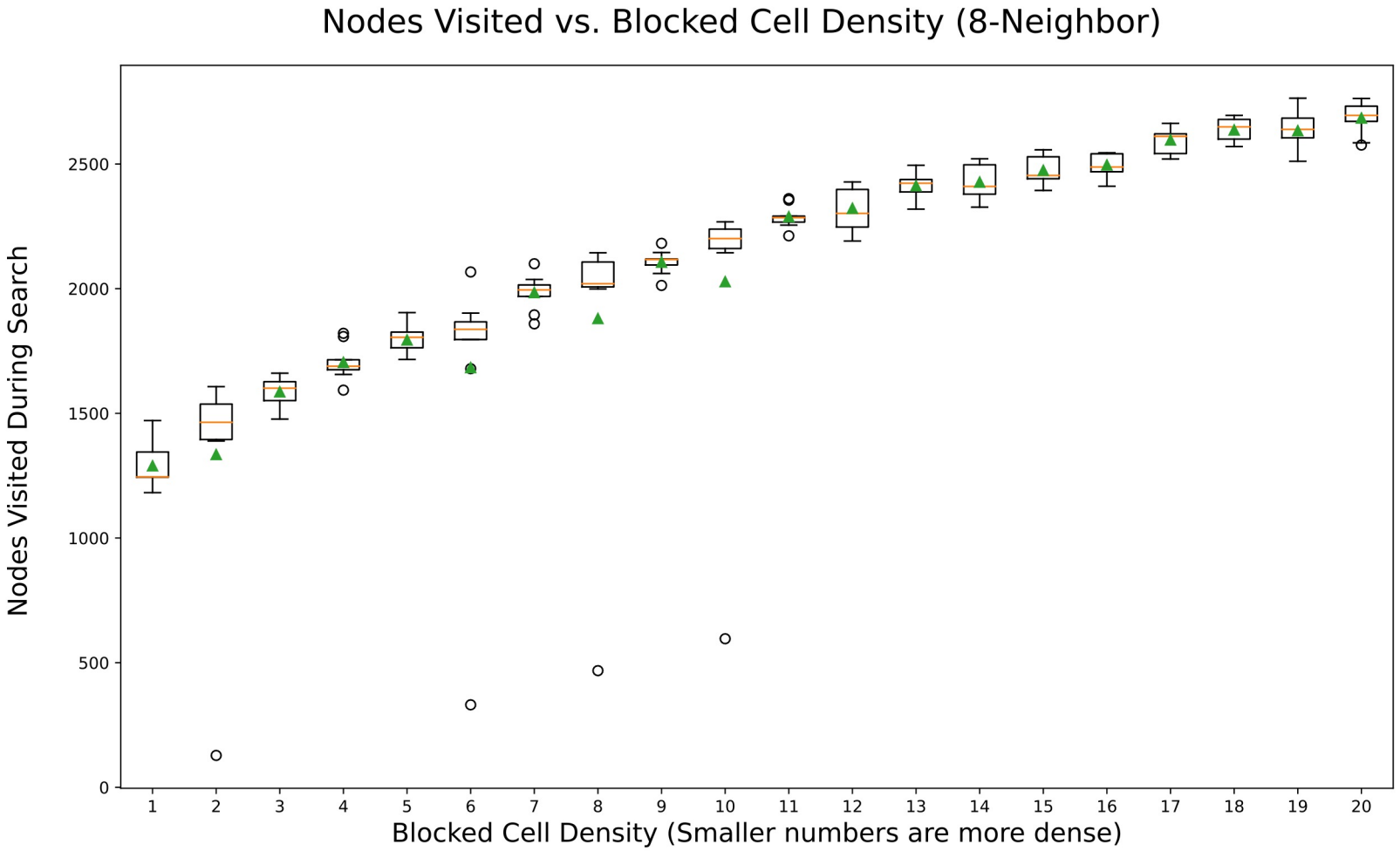
Restricting the graph constriction using the Bresenham line algorithm increases the fuel consumed even as Neighbor Distance increases.



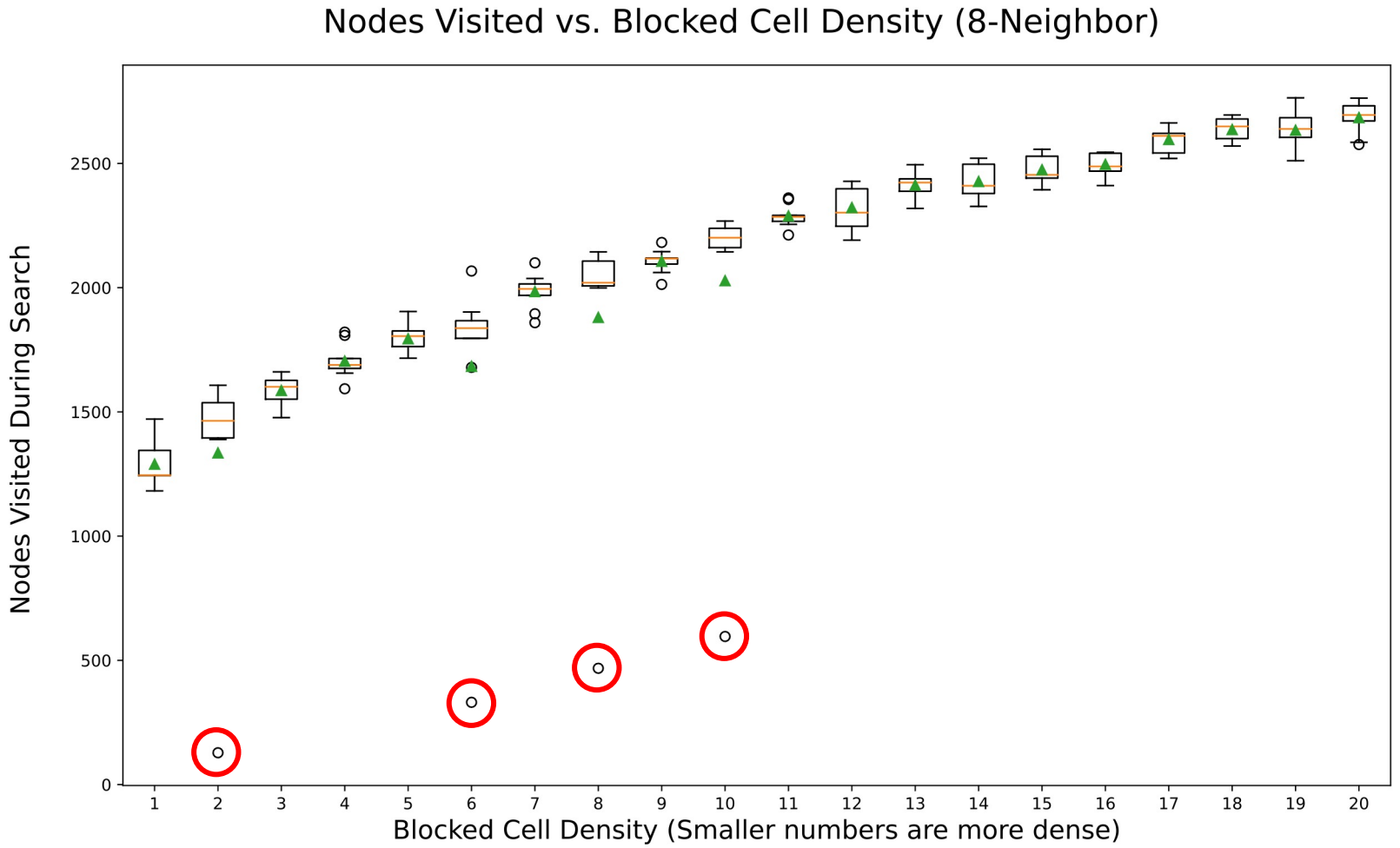
- By zooming in we can explore the behavior of the robot when constrained by the Bresenham line algorithm.
- Although this algorithm still allows the robot to move between blocked cells, the robot must go around solid lines of blocked cells, even with the Neighbor Distance is large enough for the robot to jump over the obstacle.

What can we learn by running sequences of searches?

As expected, as the density of blocked cells in the graph increases, the number of graph nodes in the 8-Neighbor model that need to be explored before reaching the goal state decreases.



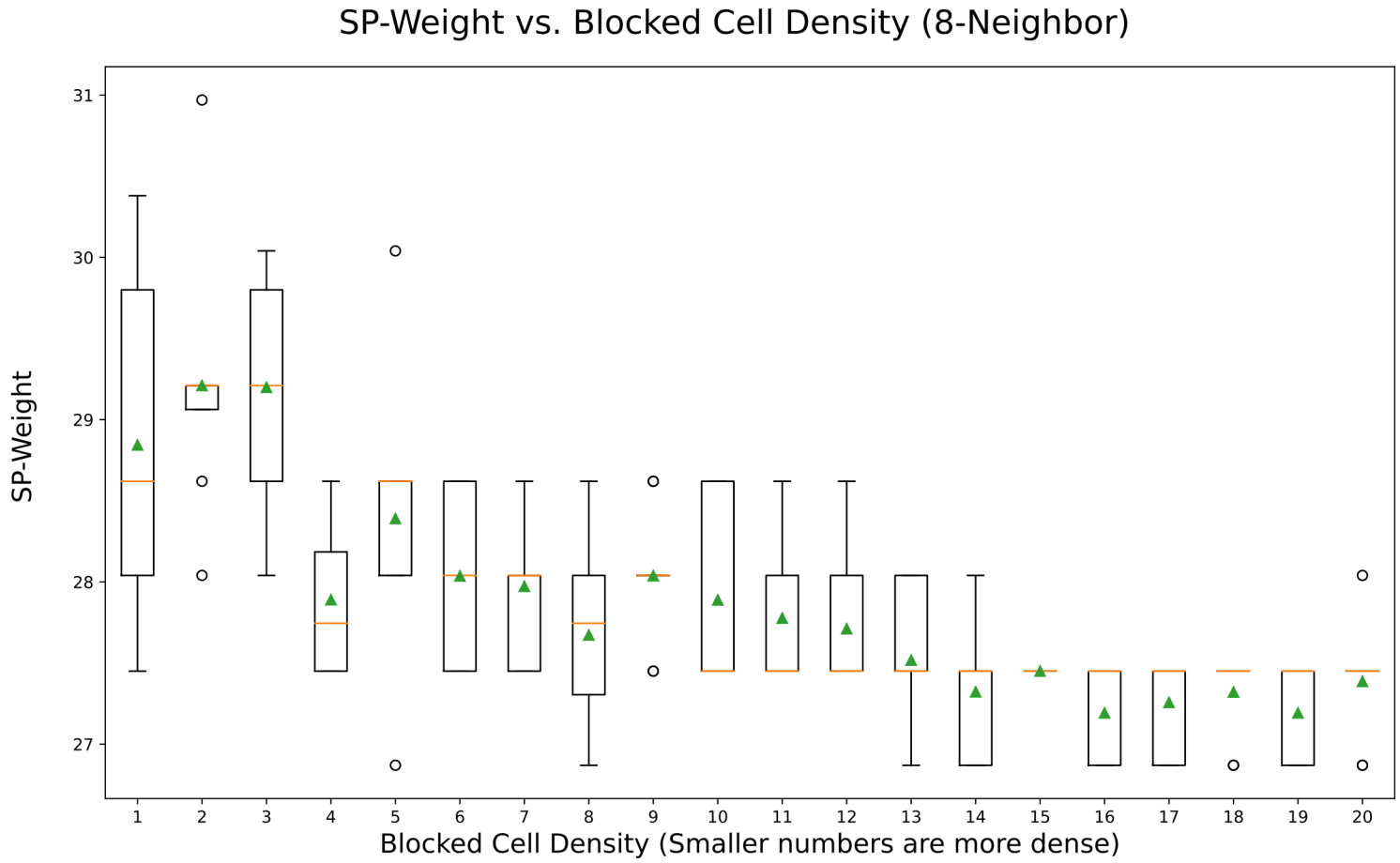
As expected, as the density of blocked cells in the graph increases, the number of graph nodes in the 8-Neighbor model that need to be explored before reaching the goal state decreases.



- As density increases, the likelihood of finding the goal node after much less exploration also increases.
- We can see that in some cases, the goal node was found after exploring less than 200 nodes in the graph. The boxplot tells us that these cases are outliers, but still worth noting.

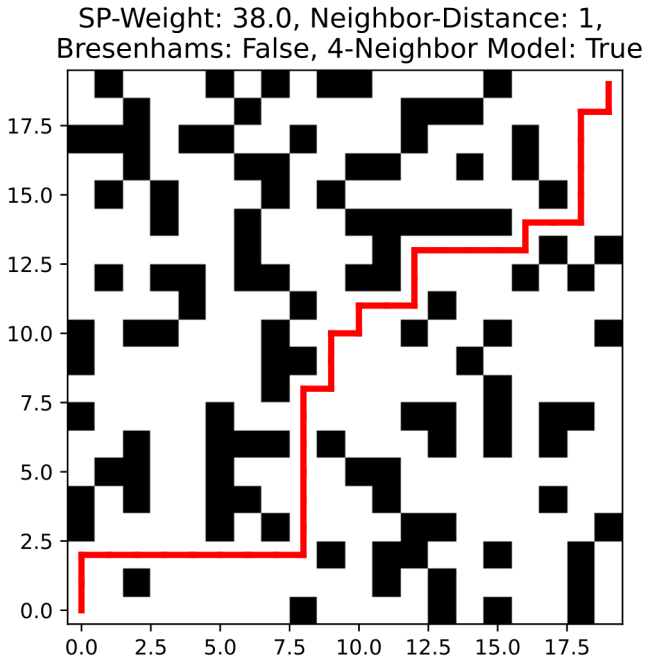
As the density of blocked cells in the graph increases, the amount of fuel needed to reach the goal increases after a certain threshold is reached.

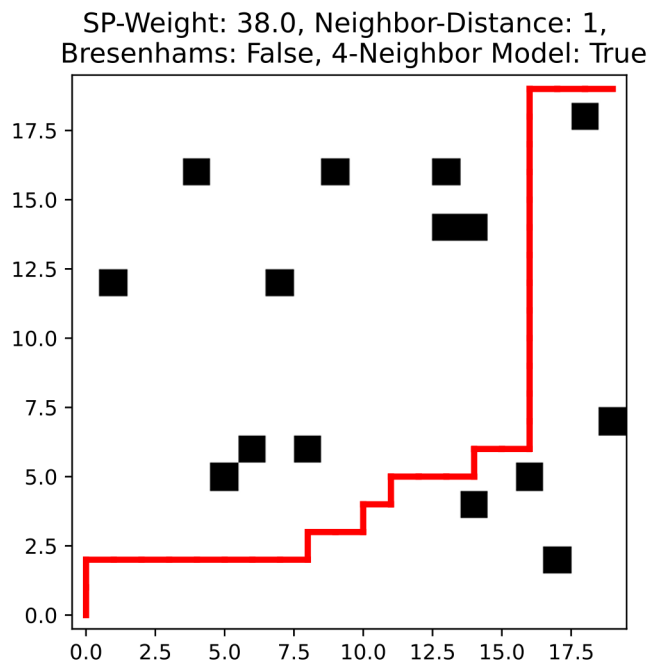
- In this case, the SP-Weight (our stand-in metric for fuel consumed by the robot) was calculated using an 8-Neighbor model graph and a Neighbor Distance of 1.
- What happens to the relationship between graph density and fuel consumed when we switch to the 4-Neighbor model for graph building?



One interesting result was that if the 4-Neighbor model is turned on, then increasing the density of the blocked cells often does not change the amount of fuel burned until the density becomes very high.

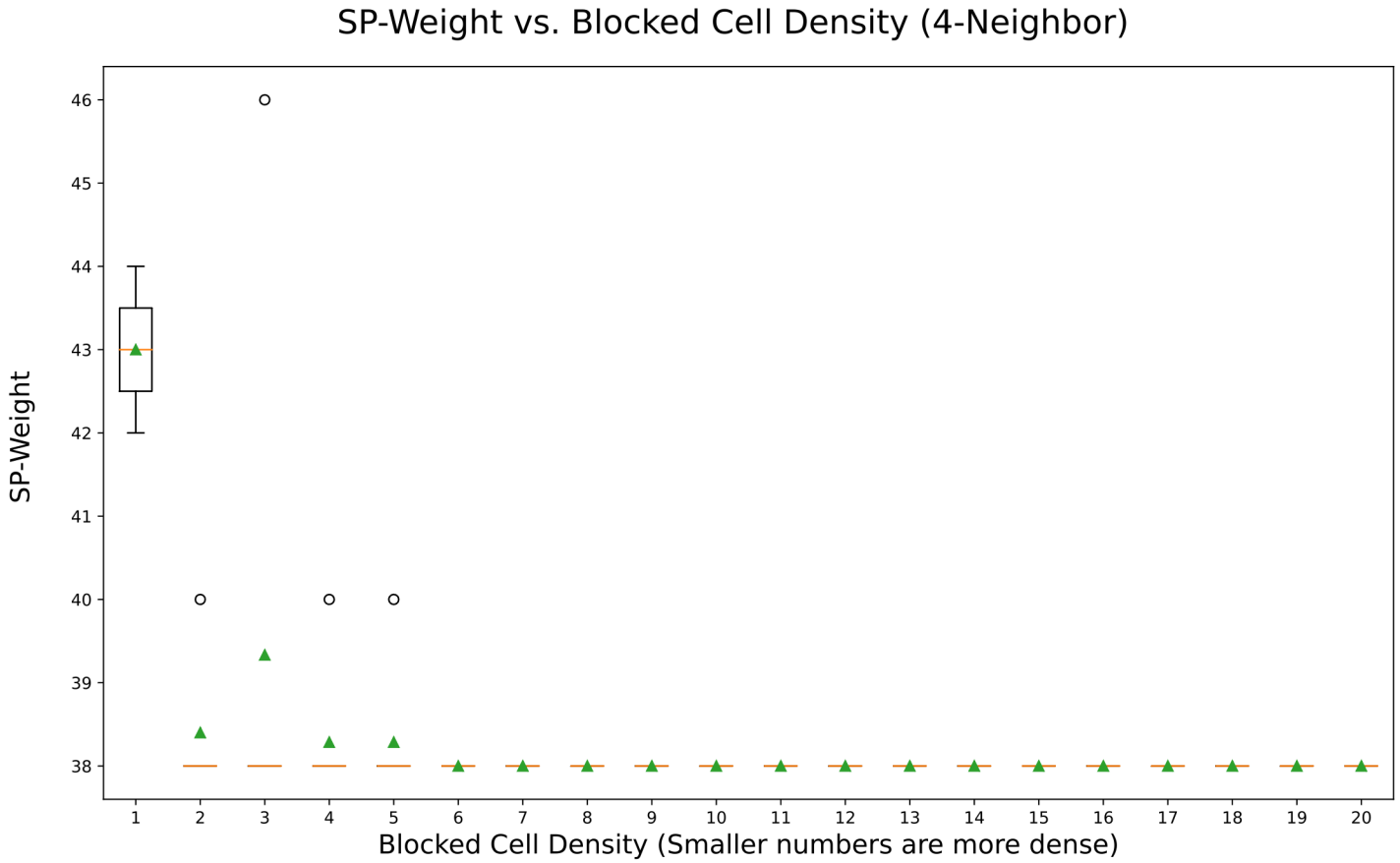
- Here we can see that 4-Neighbor model graphs with both high and low density can lead to the optimal solution. 38 units of fuel was consumed by the robot for the optimal solution for both graphs.
- This generally does not occur for the 8-Neighbor model, as seen on previous slides.



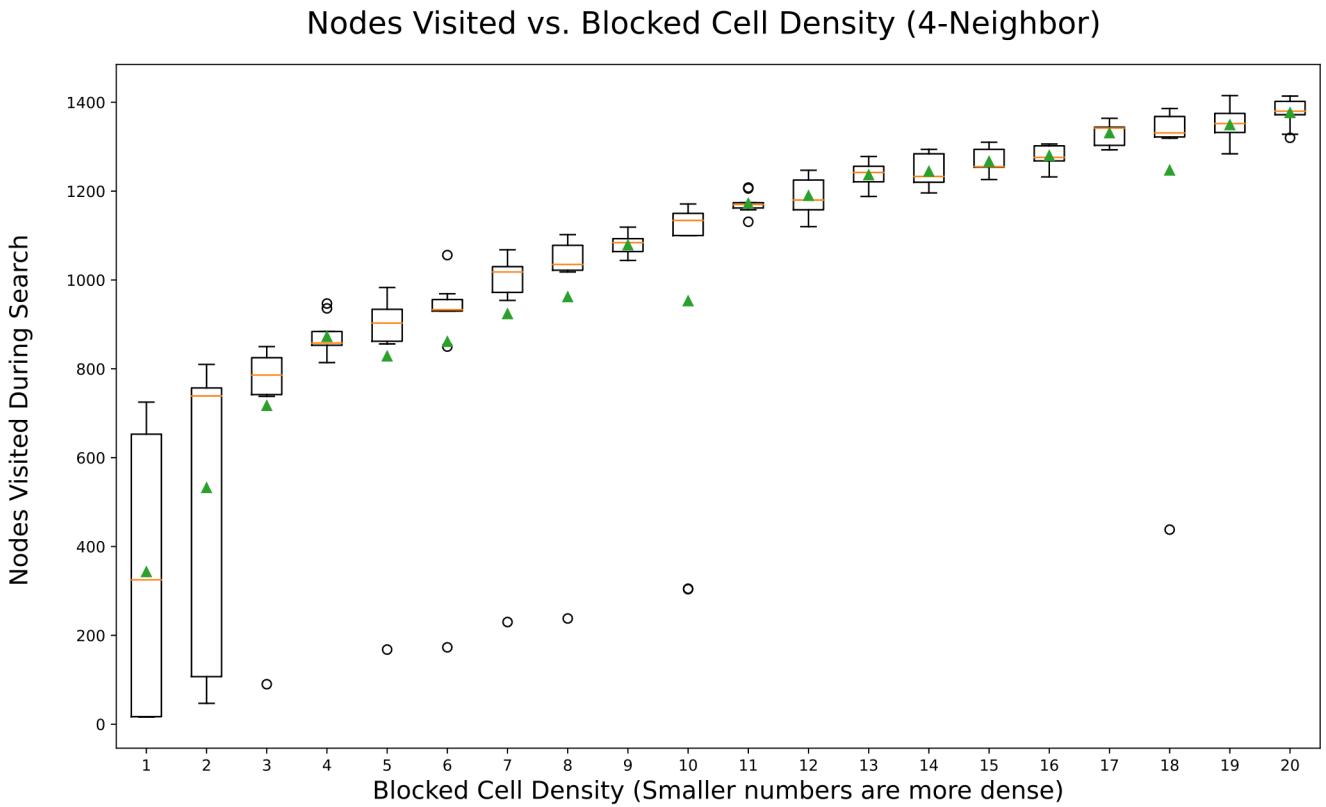


As the density of blocked cells in the graph increases, the amount of fuel needed to reach the goal increases after a certain threshold is reached.

- In this case, the SP-Weight (our stand-in metric for fuel consumed by the robot) was calculated using an 8-Neighbor model graph and a Neighbor Distance of 1.
- What happens to the relationship between graph density and fuel consumed when we switch to the 4-Neighbor model for graph building?



The number of nodes explored in the 4-Neighbor model graph decreases much more quickly than the 8-Neighbor as the density of blocked cells increases.

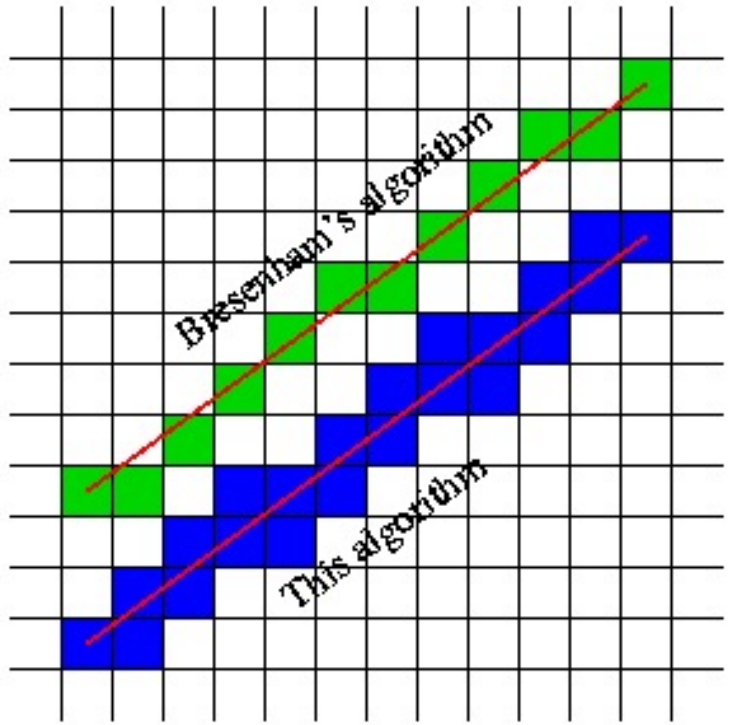


- The reason for this behavior is that the robot is limited during its search – it can't jump between corners of blocked cells and therefore the increased density of blocked cells has a much more limiting effect.
- How does changing to the 4-Neighbor model impact the fuel burned?

Future areas of exploration could include several things such as altering the line algorithm used or running simulations comparing the size of the maze and the number of nodes explored

- Custom line algorithms could be written and compared that reduced the number of neighbor nodes to those that did not intersect any blocked nodes. This would reduce the robots ability to move between the corners of blocked nodes and might represent a more realistic view of the ability for a physical robot to search a two-dimensional space.

- Other future work could involve empirically deriving the relationship between the number of nodes explored and the edge length of the square grid graph. This would (hopefully) reveal the time complexity of Dijkstra's to be O(v^2) with our implementation.



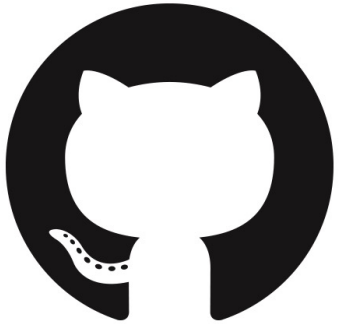
https://i.stack.imgur.com/j19KD.png

Live Demo

We can now generate new custom or randomly generated mazes or run the simulations that produced the plots seen in this presentation. New ideas are welcome!

Feel free to clone the Github repo for more!

If you're interested in running these simulations, generating more mazes, or testing the implementation of Dijkstra's or Bresenham's algorithms, just clone the Github repo! Feel free to give the repo a star if you like the code ☺



github.com/spencerbertsch1/ENGS104/tree/main/final-project

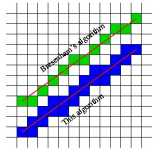
Thank You!

Image Source: brand.thayer.dartmouth.edu

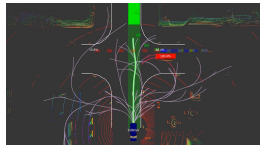
Appendix

Image Sources

- Full image sources are also listed in the notes section of the slides they appear on. If this slide deck is being viewed as a PDF or in printed format, then readers are encouraged to reference image sources below.

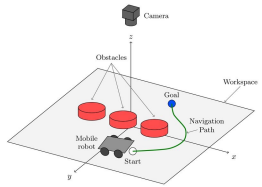


https://i.stack.imgur.com/j19KD.png



https://otc.duke.edu/wp-content/uploads/2019/05/Realtime-Robotics-Motion-Planning.jpeg

https://www.researchgate.net/figure/The-yellow-item-visualizes-a-drone-flight-path-It-shows-drone-flight-data-uploaded-to_fig2_287492026



spiedigitallibrary.org/ContentImages/Proceedings/9970/99700X/FigureImages/00027_psisdg9970_99700x_page_2_1.jpg

https://github.githubassets.com/images/modules/logos_page/GitHub-Mark.png