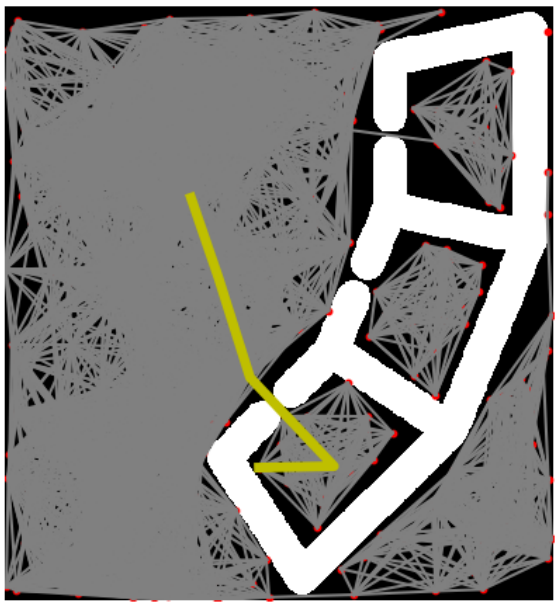
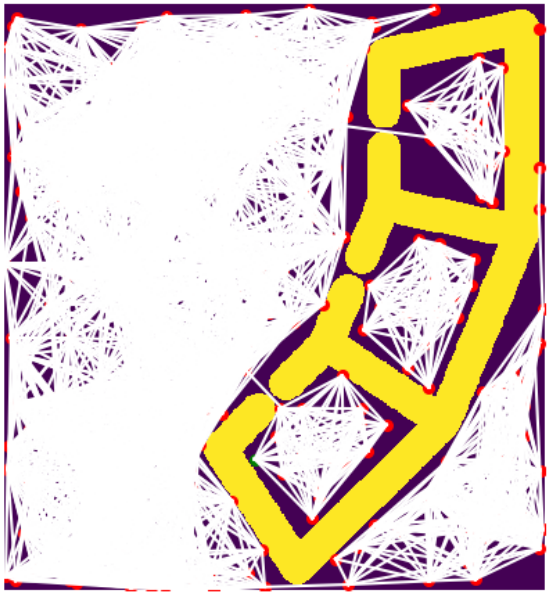
2.1 Deliverables (10 trials):

Part 1, Heuristics:





Heuristic weight = 1.0:

Success rate: 40%

Average path length = 257

Average A* runtime = 0.0019

Success 1

Path length 260

A* runtime 0.0021

Success 2

Path length 274

A* runtime 0.0024

Success 3

Path length 254

A* runtime 0.0017

Success 4

Path length 241

A* runtime 0.0013

Heuristic weight = 50.0:

Success rate: 30%

Average A* runtime: 0.0029

Average length: 324

Success 1

Path length 378

A* runtime 0.0037

Success 2

Path length 251

A* runtime 0.0016

Success 3

Path length 342

A* runtime 0.0033

Heuristic weight = 20.0:

Success rate: 50%

Average A* runtime: 0.00164

Average length: 256

Success 1

Path length 254

A* runtime 0.0017

Success 2

Path length 260

A* runtime 0.0018

Success 3

Path length 269

A* runtime 0.0017

Success 4

Path length 240

A* runtime 0.0013

Success 5

Path length 252

A* runtime 0.0017

There is no obvious change in average path length and planning time, partly due to limited sampling size, but also likely because heuristic weight doesn't have any notable impact on the metrics.

Part 2, connection radius:

Connection radius = 90

Success rate = 40%

Average A* planning time = 0.0020

Average path length = 272

Connection radius = 200

Success rate = 100%

Average A* planning time = 0.0059

Average path length = 286

The graph creation time greatly increased, but the A* planning time only saw a small increase, which makes sense as increasing connection radius should increase the number of edges created, which should have a small effect on A* search. We assumed path length would be shorter overall with a larger connection radius, as there should be more paths to compare, but the data shows different results. This might just be due to a limited data sample size. Increased success rate is likely due to more complete connections between nodes, since the number of nodes every node is connected to increases when the connection radius is increased.

Part 3, vertices:

Vertices = 280

Success rate = 30%

Average A* planning time = 0.0017

Average path length = 270

Vertices = 600

Success rate = 90%

Average A* planning time = 0.0060

Average path length = 256

More nodes means more possible routes, with an increase in number of vertices, there are more paths to consider hence planning time increases and with more nodes the probability of getting the most optimal path increases hence path length decreases. With more nodes the success rate goes up because the probability of always finding a path between the given nodes increases because the free space is more thoroughly sampled.

3.1 Deliverables

Map1

40 vertices, 15 connection radius, heuristic weight 1.0

Non lazy A*

Graph creation time = 0.06673

Edges evaluated by while making graph = 780

Path length = 13.02

Plan time = 0.0004

Lazy A*

Graph creation time = 0.02937

Edges evaluated during Lazy A* = 365

Path length = 12.69

Plan time = 0.0283

Map2

300 vertices, 200 connection radius, heuristic weight 1.0

Non lazy A*

Graph creation time = 2.572 seconds

Edges evaluated by graph _maker = 1816

Path length = 243

Plan time = 0.0035

Lazy A*

Graph creation time = 0.844

Edges evaluated during Lazy A* = 1126

Path length = 256

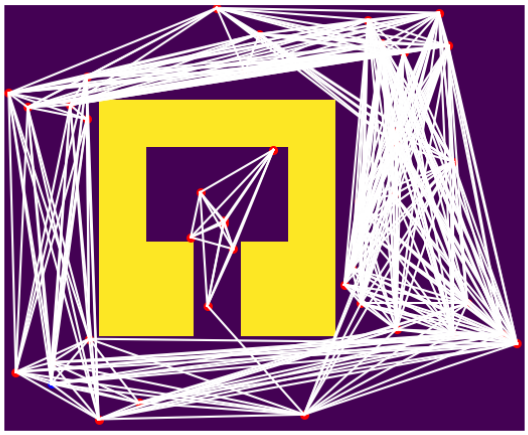
Plan time = 0.7804

Lazy A* avoids evaluating a LOT of edges, clearly saving time compared to non-lazy A*. The lazy A* graph has edges that go through the obstacles and a lower construction time as it does not evaluate the edges between the nodes. The A* graph construction takes a lot more time because the graph maker evaluates every possible edge between the nodes and only creates those that are valid. Lazy A* evaluates the edges as its finding the optimal path, hence only those edges that it considers while generating the optimal path get evaluated. The final paths between lazy A* and A* are not different for map1, they are slightly different in map2 because of

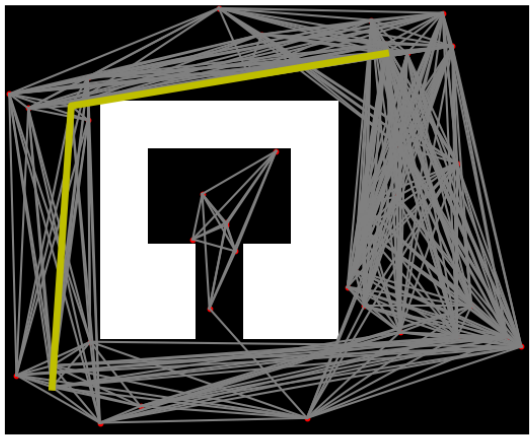
sampling differences. Given the same exact nodes, lazy A* and A* should produce the same path.

MAP 1

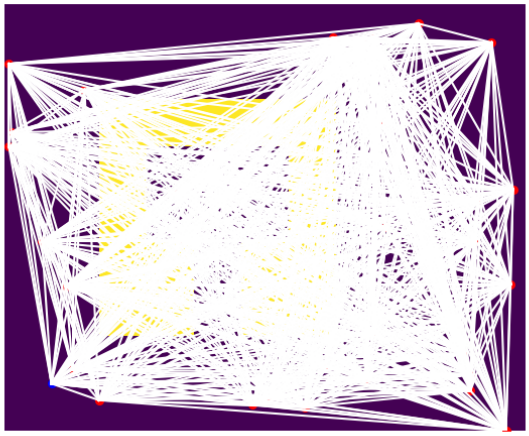
Not lazy graph



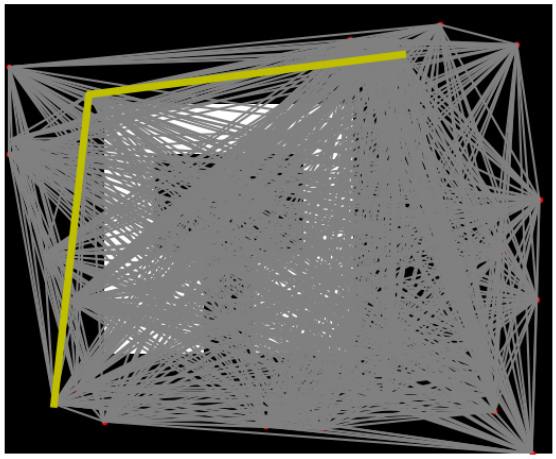
Not lazy plan



Lazy graph



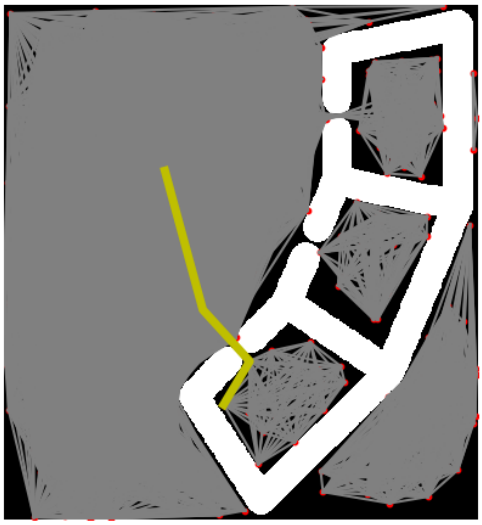
Lazy plan



MAP 2

Not lazy graph

Not lazy plan



Lazy graph

Lazy plan

4.1 Deliverables

Q) Discuss how the generated paths are different qualitatively as well as in terms of path length.

A) As the curvature parameter increases, the turns become tighter. This minimizes the cost for turning, generally decreasing overall path length. The length of the curved segments decreases and the length of the straight segments increases when curvature increases. The path length decreases as curvature increases.

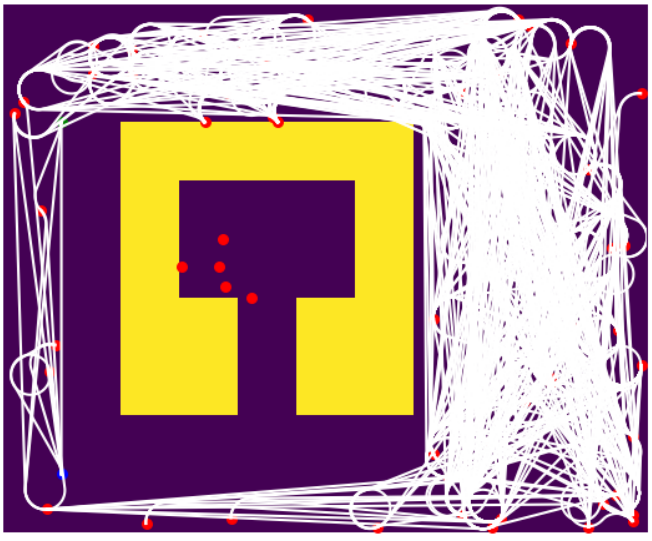
Q) Report the tuned parameters. You may find it necessary to change the parameter setup per curvature.

A) Starting at 60 vertices often left “holes” in the graph, with some nodes struggling to reach others, or the nodes that did reach others were limited, allowing for less variety for paths. Increasing vertices to 200 works for all curvature values we tested, the free space in the graph is almost completely sampled and every node seemed reachable from every other node.

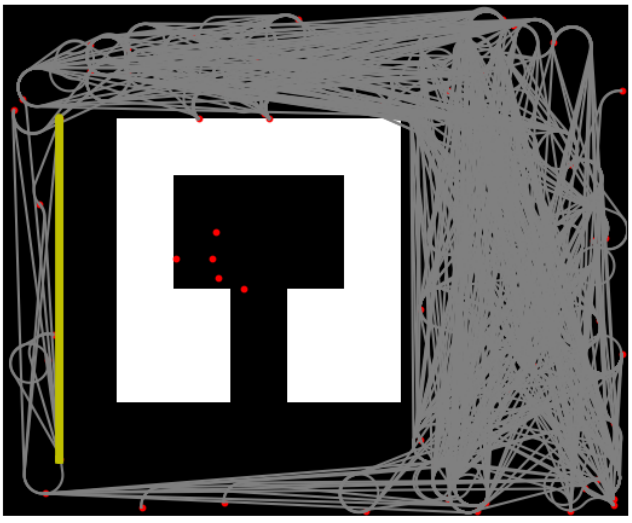
Final tuned parameters are 200 vertices, 15 connection radius.

Curvature = 3.0

graph

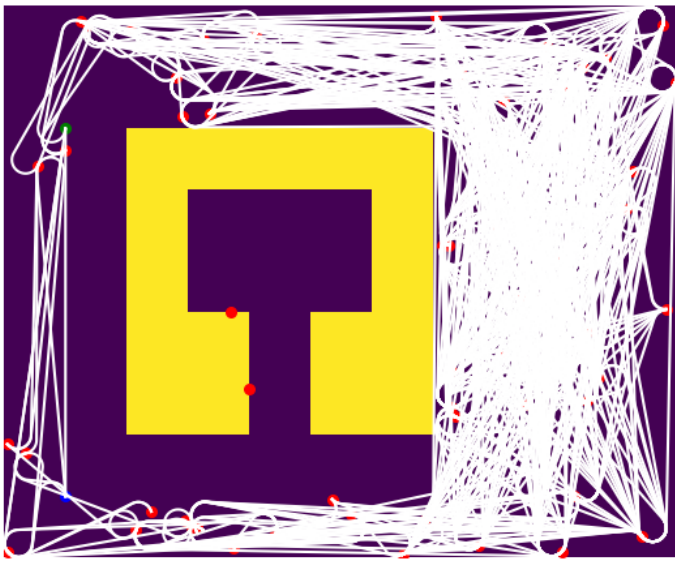


plan

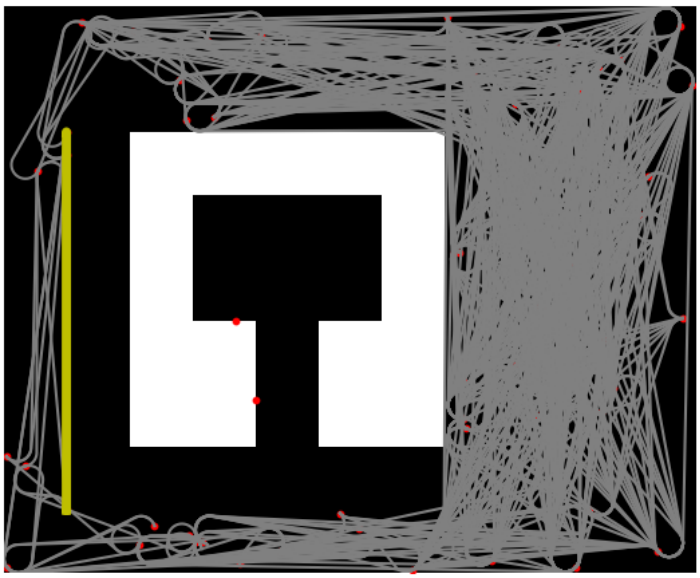


Curvature = 4.5

graph

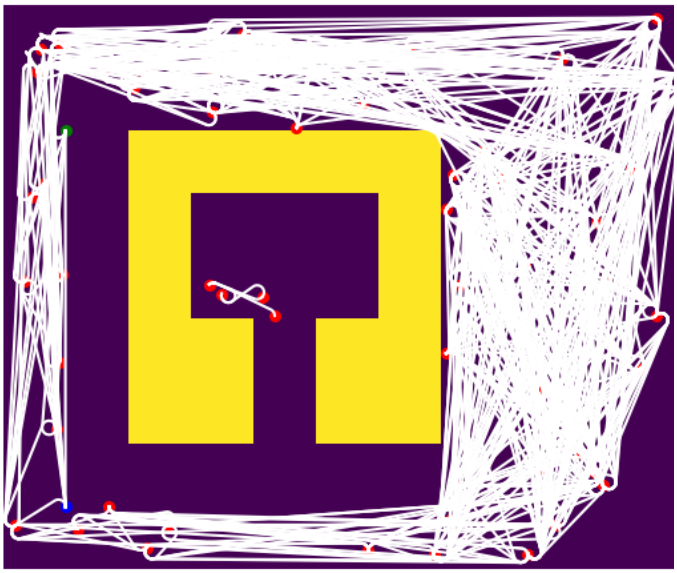


plan

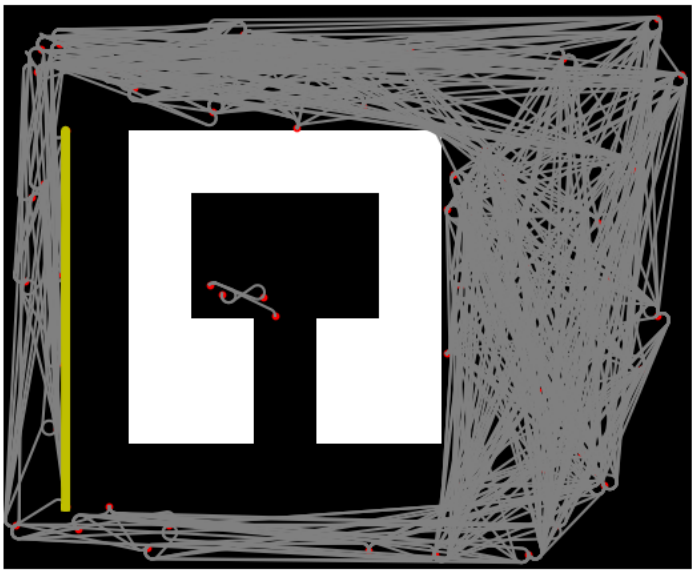


curvature = 9.0

graph



plan



5.1 Deliverables

Q) Compare the path length, total planning time, and post processing time for map2.txt with

and without shortcut.

A) I) I ran the regular path planning algorithm (not dubin) and used the shortcut method. The parameters were:

1) map: map2.txt

2) Connection radius = 100

3) Number of vertices = 300

The results were:

A* planning time = 0.00157

Path length = 408.972

Shortcut planning time = 0.04488

Shortcut path length = 406.546

The post-processing time is the time it takes to make the shortcut, without the shortcut the post processing time is 0. Hence post processing time with shortcut is 0.044883

Total time without shortcut = 14.68 (graph creation time) + 0.00157 = 14.68157

Total time with shortcut = 14.68 (graph creation time) + 0.00157 + 0.04488 = 14.72645

II) I ran the dubins path planning algorithm with the following parameters:

1) Map = map2.txt

2) Connection radius = 100

3) Number of vertices = 300

4) Curvature = 3

The results were:

A* planning time = 0.1245

Path length = 398.57

Shortcut planning time = 0.0138039

Shortcut path length = 392.576

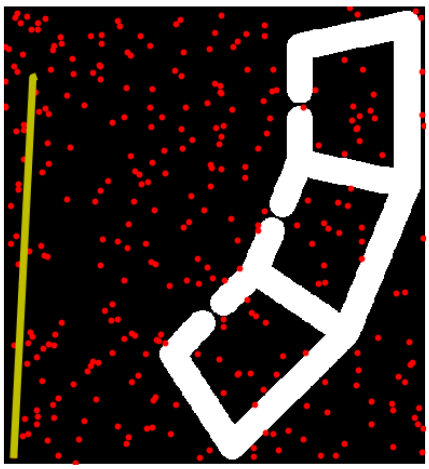
The post-processing time is the time it takes to make the shortcut, without the shortcut the post processing time is 0. Hence post processing time with shortcut is 0.0138039

Total time without shortcut = 8.627 (graph creation time) + 0.1245 = 8.7515

Total time with shortcut = 8.627(graph creation time) + 0.1245 + 0.0138039 = 8.7653

dubin's path without shortcut

dubin's path with shortcut



The non-shortcut path is longer and has more turns, this is because the goal node is outside the connection radius of the start node, hence even though there was an optimal path between them, one could not be found due to the fact that there was not an edge between them. The shortcut algorithm determined that the start and goal nodes could be connected by a single Dubin's path edge hence this path is much shorter and has less turns.

6.1 Deliverables:

Number of vertices = 500

Connection radius = 100

Curvature = 0.04

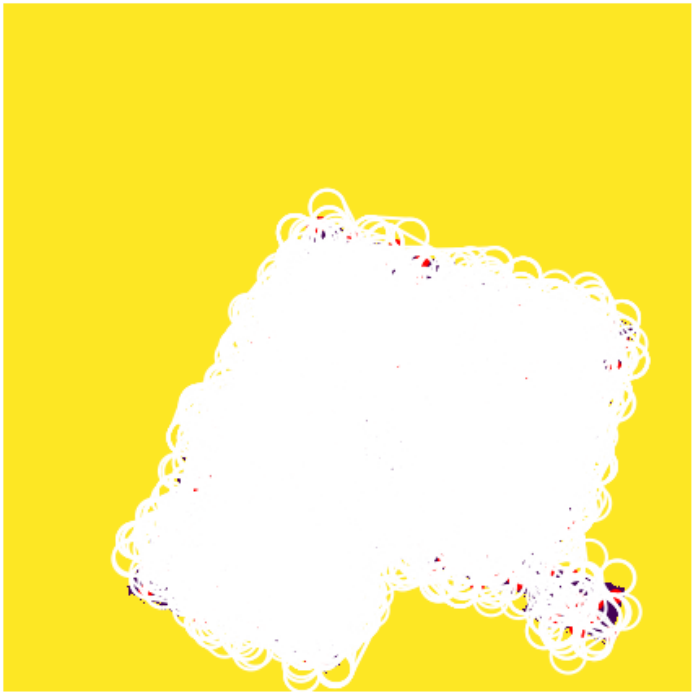
Graph construction time = 8.53

Edges evaluated by lazy A* = 2170

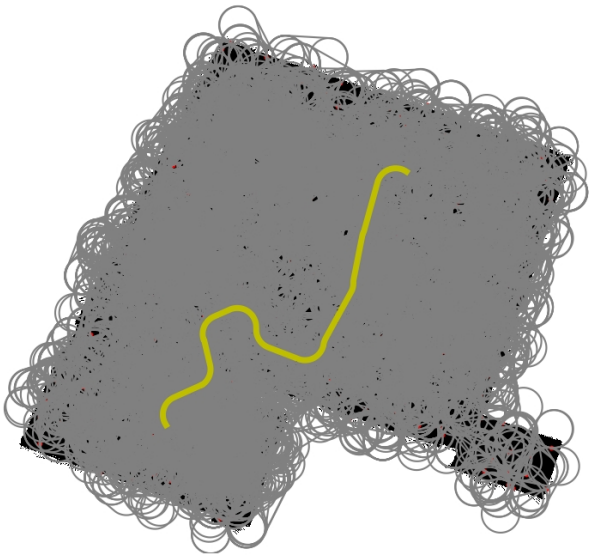
Planning time = 4.0002

Post Processing time = 0.8457

Generated graph



Generated plan without shortcut



Generated plan with shortcut