**CMPT 417 Report**

Just a side note that some future code is included in screenshots since I did not know I had to take them until I was nearly done the project.

1. **Implementing Space-Time A\***
   1. **Searching in the Space-Time Domain**

I simply followed the instructions for this step as they were quite explicit. To make the agent wait, I changed the loop from range(4) to range(5) and added a [0,0] coordinate in the move function.

Found a solution!

CPU time (s): 0.00

Sum of costs: 6

\*\*\*Test paths on a simulation\*\*\*

COLLISION! (agent-agent) (0, 1) at time 3.4

COLLISION! (agent-agent) (0, 1) at time 3.5

COLLISION! (agent-agent) (0, 1) at time 3.6

COLLISION! (agent-agent) (0, 1) at time 3.7

COLLISION! (agent-agent) (0, 1) at time 3.8

COLLISION! (agent-agent) (0, 1) at time 3.9

COLLISION! (agent-agent) (0, 1) at time 4.0

COLLISION! (agent-agent) (0, 1) at time 4.1

COLLISION! (agent-agent) (0, 1) at time 4.2

COLLISION! (agent-agent) (0, 1) at time 4.3

COLLISION! (agent-agent) (0, 1) at time 4.4

COLLISION! (agent-agent) (0, 1) at time 4.5

COLLISION! (agent-agent) (0, 1) at time 4.6

* 1. **Handling Vertex Constraints**

I checked for constraints each loop and continued the loop if it was true. The constraint table is built at the start of A\* call. If the agent was in another agent’s next location at the same timestep, that agent would not make that move.

Found a solution!

CPU time (s): 0.00

Sum of costs: 7

[[(1, 1), (1, 2), (1, 3), (1, 4), (1, 4), (1, 5)], [(1, 2), (1, 3), (1, 4)]]

\*\*\*Test paths on a simulation\*\*\*

COLLISION! (agent-agent) (0, 1) at time 3.4

COLLISION! (agent-agent) (0, 1) at time 3.5

COLLISION! (agent-agent) (0, 1) at time 3.6

COLLISION! (agent-agent) (0, 1) at time 3.7

COLLISION! (agent-agent) (0, 1) at time 3.8

COLLISION! (agent-agent) (0, 1) at time 3.9

COLLISION! (agent-agent) (0, 1) at time 4.0

COLLISION! (agent-agent) (0, 1) at time 4.1

COLLISION! (agent-agent) (0, 1) at time 4.2

COLLISION! (agent-agent) (0, 1) at time 4.3

COLLISION! (agent-agent) (0, 1) at time 4.4

COLLISION! (agent-agent) (0, 1) at time 4.5

COLLISION! (agent-agent) (0, 1) at time 4.6

COLLISION! (agent-agent) (0, 1) at time 4.7

COLLISION! (agent-agent) (0, 1) at time 4.8

COLLISION! (agent-agent) (0, 1) at time 4.9

COLLISION! (agent-agent) (0, 1) at time 5.0

COLLISION! (agent-agent) (0, 1) at time 5.1

COLLISION! (agent-agent) (0, 1) at time 5.2

COLLISION! (agent-agent) (0, 1) at time 5.3

COLLISION! (agent-agent) (0, 1) at time 5.4

COLLISION! (agent-agent) (0, 1) at time 5.5

COLLISION! (agent-agent) (0, 1) at time 5.6

* 1. **Adding Edge Constraints**

Similar to vertex but an agent has to specifically move from one location to another for the move to be pruned.

CPU time (s): 0.00

Sum of costs: 7

[[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 2), (1, 3), (1, 4)]]

\*\*\*Test paths on a simulation\*\*\*

COLLISION! (agent-agent) (0, 1) at time 1.4

COLLISION! (agent-agent) (0, 1) at time 1.5

COLLISION! (agent-agent) (0, 1) at time 1.6

COLLISION! (agent-agent) (0, 1) at time 1.7

COLLISION! (agent-agent) (0, 1) at time 1.8

COLLISION! (agent-agent) (0, 1) at time 1.9

COLLISION! (agent-agent) (0, 1) at time 2.0

COLLISION! (agent-agent) (0, 1) at time 2.1

COLLISION! (agent-agent) (0, 1) at time 2.2

COLLISION! (agent-agent) (0, 1) at time 2.3

COLLISION! (agent-agent) (0, 1) at time 2.4

COLLISION! (agent-agent) (0, 1) at time 2.5

COLLISION! (agent-agent) (0, 1) at time 2.6

COLLISION! (agent-agent) (0, 1) at time 2.7

COLLISION! (agent-agent) (0, 1) at time 2.8

COLLISION! (agent-agent) (0, 1) at time 2.9

COLLISION! (agent-agent) (0, 1) at time 3.0

COLLISION! (agent-agent) (0, 1) at time 3.1

COLLISION! (agent-agent) (0, 1) at time 3.2

COLLISION! (agent-agent) (0, 1) at time 3.3

COLLISION! (agent-agent) (0, 1) at time 3.4

COLLISION! (agent-agent) (0, 1) at time 3.5

COLLISION! (agent-agent) (0, 1) at time 3.6

COLLISION! (agent-agent) (0, 1) at time 3.7

COLLISION! (agent-agent) (0, 1) at time 3.8

COLLISION! (agent-agent) (0, 1) at time 3.9

COLLISION! (agent-agent) (0, 1) at time 4.0

COLLISION! (agent-agent) (0, 1) at time 4.1

COLLISION! (agent-agent) (0, 1) at time 4.2

COLLISION! (agent-agent) (0, 1) at time 4.3

COLLISION! (agent-agent) (0, 1) at time 4.4

COLLISION! (agent-agent) (0, 1) at time 4.5

COLLISION! (agent-agent) (0, 1) at time 4.6

* 1. **Handling Goal Constraints**

I modified the goal test by looking for the maximum timestep in the dictionary of constraints for the given agent, and only allowing the algorithm to return if we are at a timestep larger than this.

* 1. **Designing Constraints**

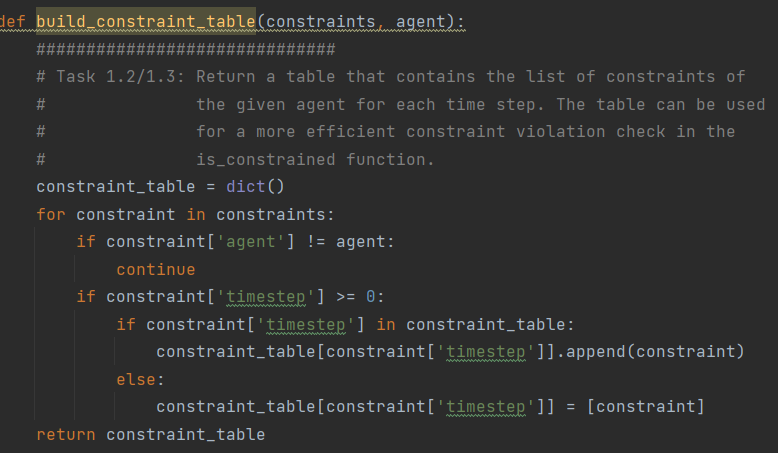
This was not hard as all that needed to be done was move agent 1 down to [2,3] to make room for agent 0 to get past.

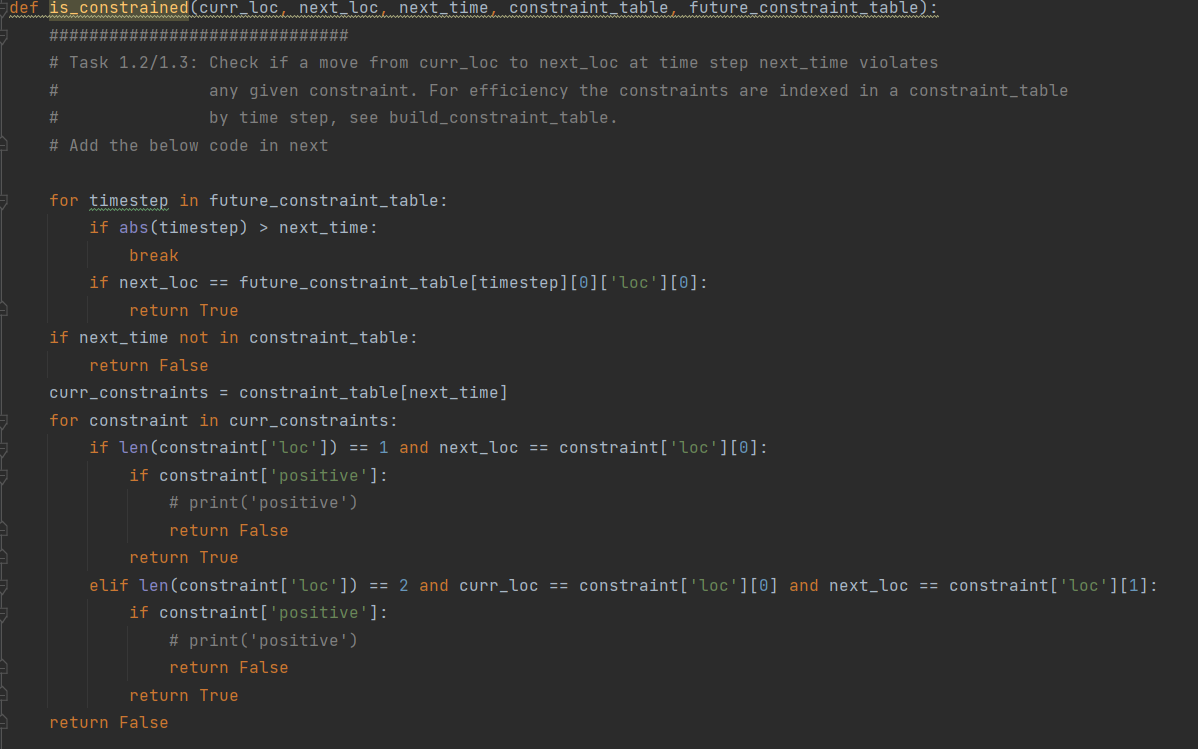
{'agent': 1, 'loc': [(1,3), (2,3)], 'timestep': 3}

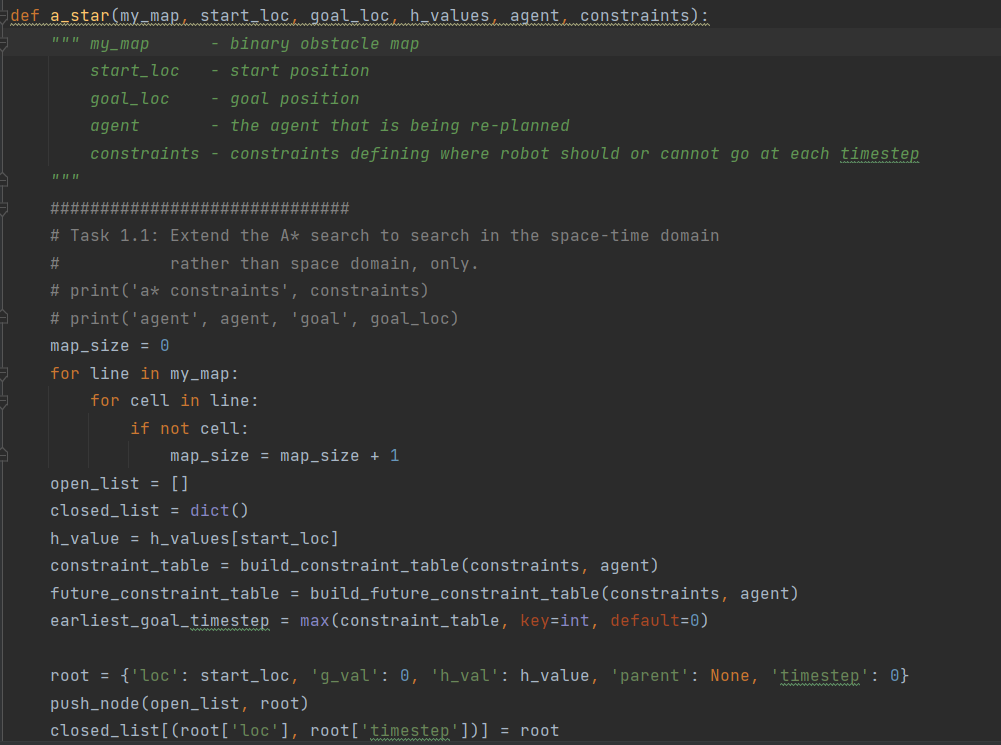
CPU time (s): 0.00

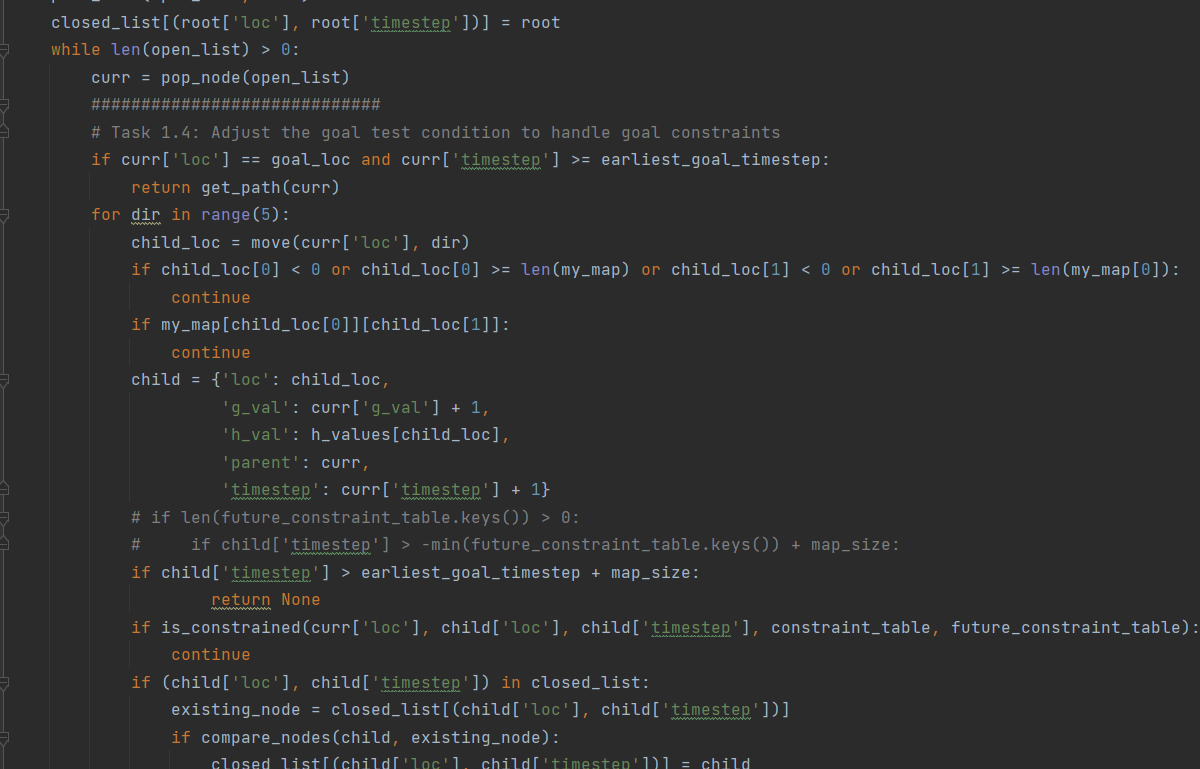
Sum of costs: 8

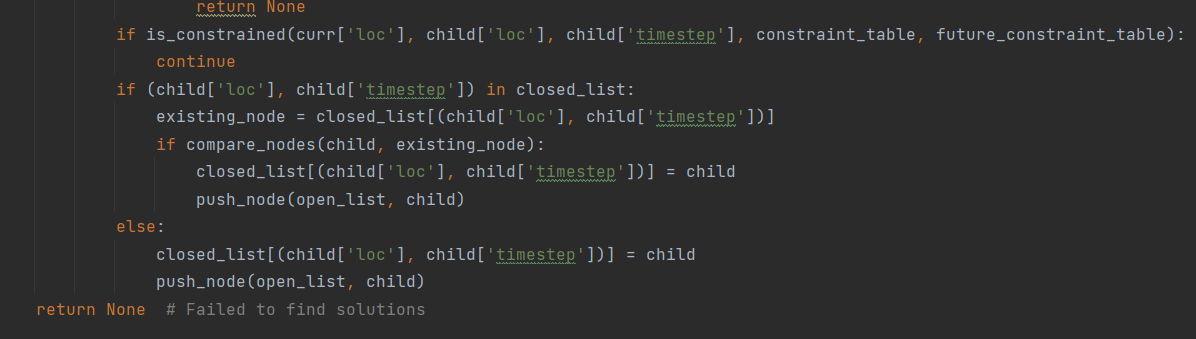
[[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3), (2, 3), (1, 3), (1, 4)]]

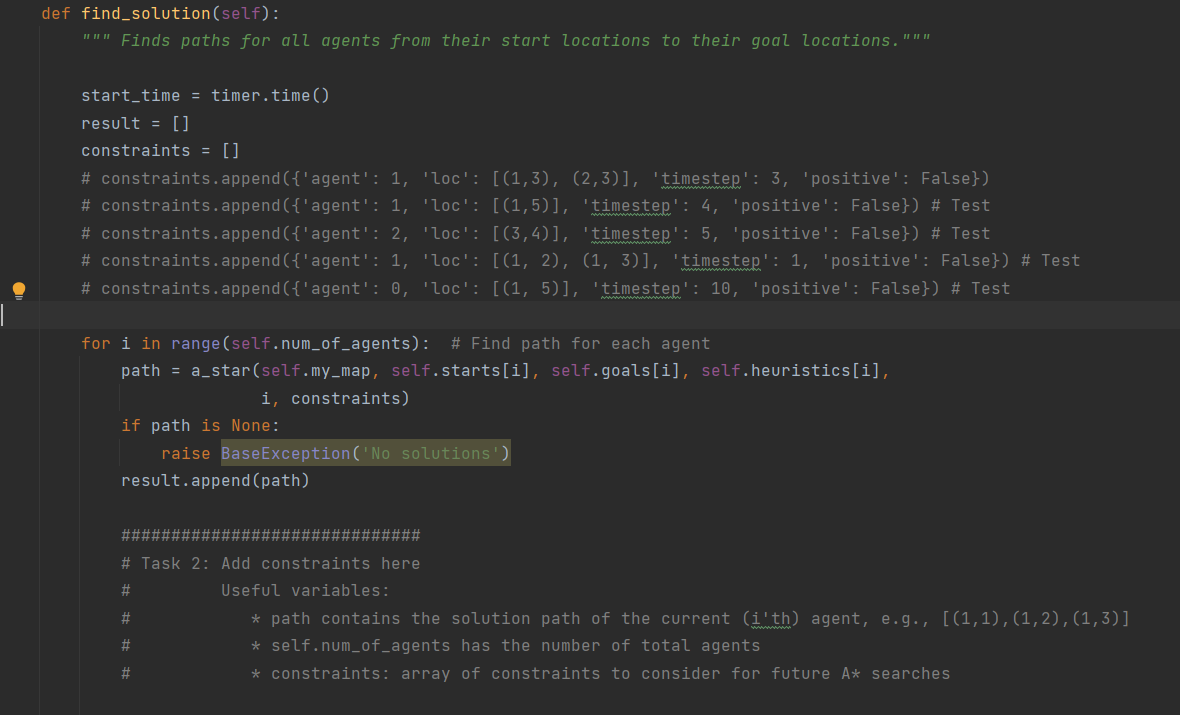












1. **Implementing Prioritized Planning**
   1. **Adding Vertex Constraints**

I added one loop to go over the current agent’s path, and another to apply constraints for every agent after the current agent, since they would have the same constraints at that point.

* 1. **Adding Edge Constraints**

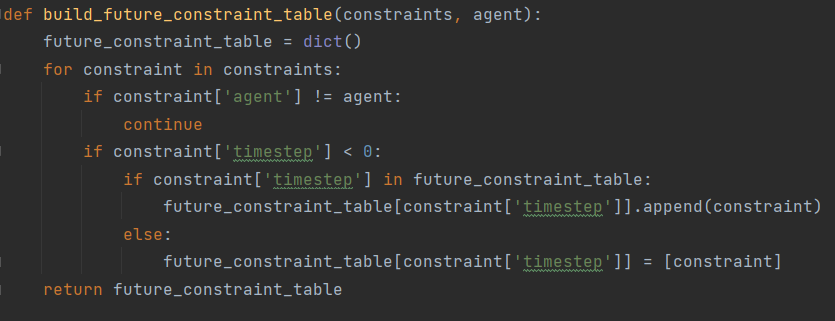
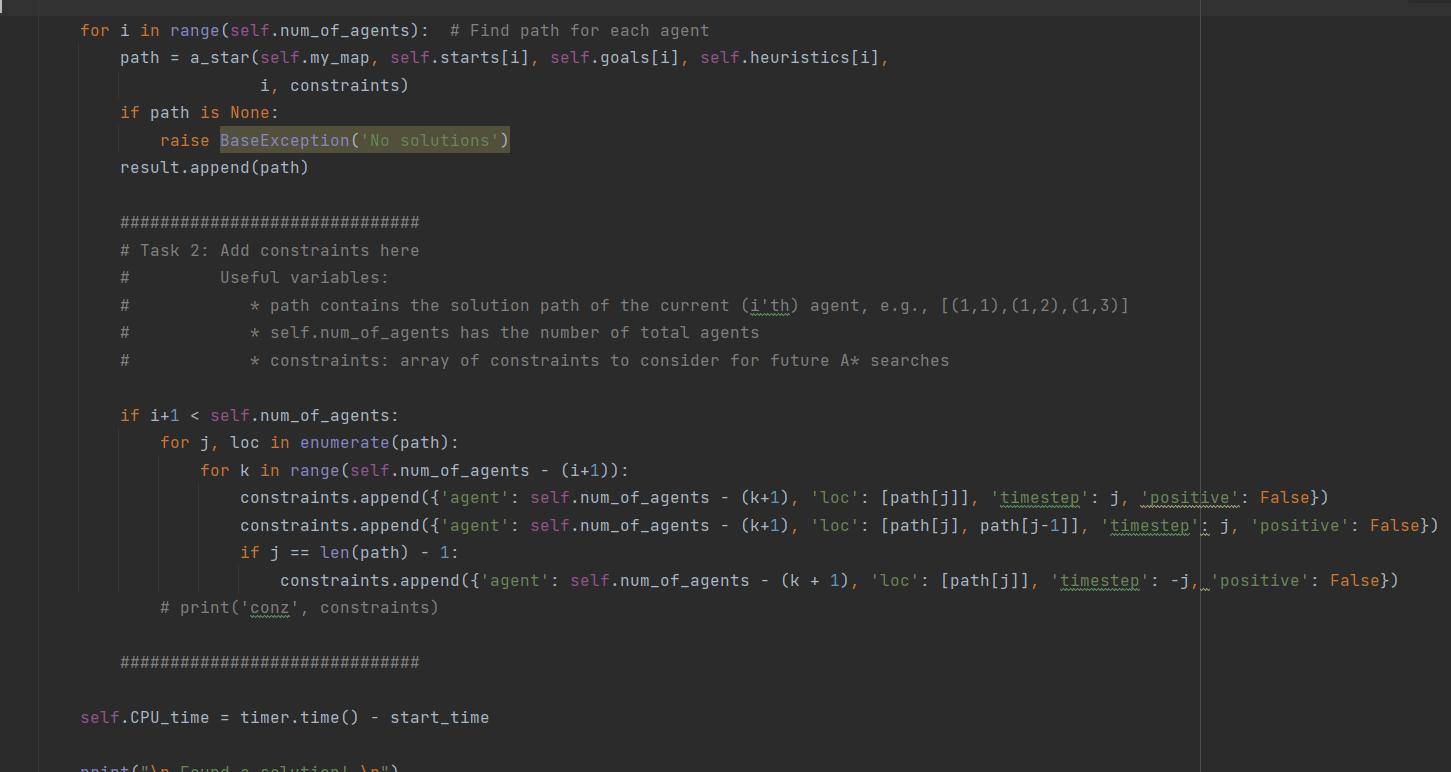
If agent 0 goes from location a to b in timestep t, then any agent after cannot go from b to a in timestep t.

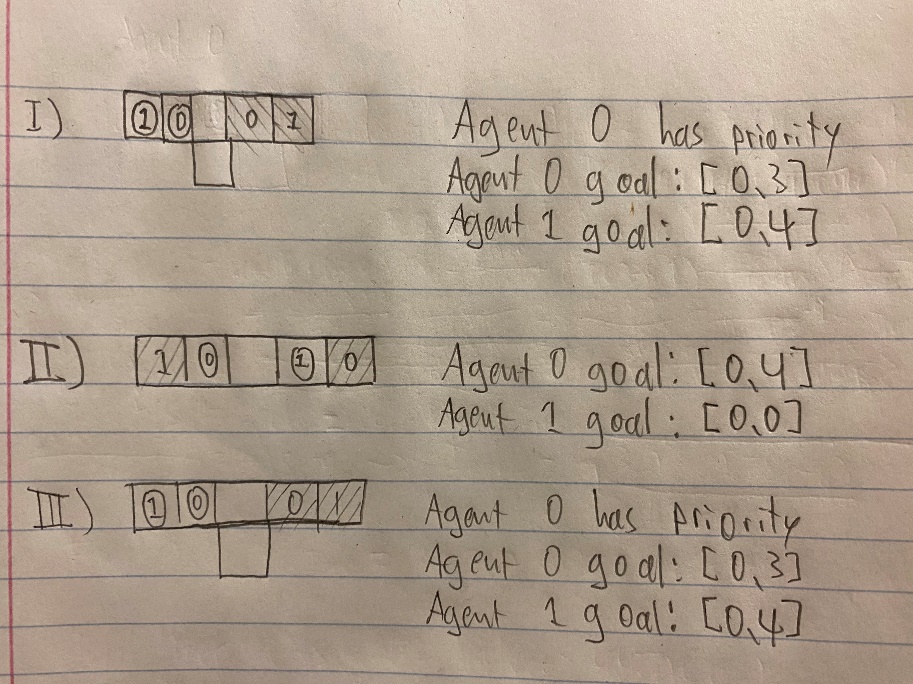
* 1. **Adding Additional Constraints**

I used a marker at the end of the agent’s path in the constraints to indicate that they have reached their goal by pacing a negative value for that agent’s goal timestep. That is, if the agent finishes at timestep 10, then the constraint contains -10 for the timestep of the agent. I made a separate constraint table and builder function, future\_constraint\_table and build\_future\_constraint\_table to handle these negative timestep values, and create constraints for all timesteps after at the given location.

* 1. **Addressing Failures**

In Exp2\_3, agent 1 has higher priority than agent 0, which means that his path is planned first. Agent 1’s path is first made according to A\*, with no constraints, but when agent 1 reaches his goal, he completely blocks agent 0’s path to his respective goal. Agent 1 has already reached his goal and his search has terminated, so he cannot move, meaning that Agent 0 is stuck in an infinite loop. I changed the code to have an upper bound on the current agent equal to the timestep which the previous agent was at when he reached his goal plus the size of the map, since it would not make sense for the current agent to traverse a length greater than the size of the map after surpassing the timestep of the previous agent.



* 1. **Showing that Prioritized Planning is Incomplete and Suboptimal**

1. **Implementing Conflict-Based Search (CBS)**
   1. **Detecting Collisions**

I called detect\_collision in detect\_collisions for each agent pair. Collision detection was quite similar to constraint building for both vertex and edge collisions.

* 1. **Converting Collisions to Constraints**

This was quite straightforward. I simply use the collision value to create a constraint as it has all the necessary information.

* 1. **Implementing the High-Level Search**

I basically just followed the pseudocode. The algorithm will keep splitting on collisions and run A\* for every constraint generated from collisions in the open list.

Generate node 0

Expand node 0

expanded node: {'cost': 6, 'constraints': [], 'paths': [[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3), (1, 4)]], 'collisions': [{'a1': 0, 'a2': 1, 'loc': [(1, 4)], 'timestep':

3}]}

Generate node 1

Generate node 2

Expand node 1

expanded node: {'cost': 7, 'constraints': [{'agent': 0, 'loc': [(1, 4)], 'timestep': 3}], 'paths': [[(1, 1), (1, 2), (1, 3), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3), (1, 4)]], 'collision

s': [{'a1': 0, 'a2': 1, 'loc': [(1, 4)], 'timestep': 4}]}

Generate node 3

Generate node 4

Expand node 2

expanded node: {'cost': 8, 'constraints': [{'agent': 1, 'loc': [(1, 4)], 'timestep': 3}], 'paths': [[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3), (1, 4), (1, 3), (1, 4)]], 'c

ollisions': [{'a1': 0, 'a2': 1, 'loc': [(1, 3), (1, 4)], 'timestep': 3}]}

Generate node 5

Generate node 6

Expand node 3

expanded node: {'cost': 8, 'constraints': [{'agent': 0, 'loc': [(1, 4)], 'timestep': 3}, {'agent': 0, 'loc': [(1, 4)], 'timestep': 4}], 'paths': [[(1, 1), (1, 2), (1, 3), (1, 3), (1, 3),

(1, 4), (1, 5)], [(1, 2), (1, 3), (1, 4)]], 'collisions': [{'a1': 0, 'a2': 1, 'loc': [(1, 4)], 'timestep': 5}]}

Generate node 7

Generate node 8

Expand node 6

expanded node: {'cost': 8, 'constraints': [{'agent': 1, 'loc': [(1, 4)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 4), (1, 3)], 'timestep': 3}], 'paths': [[(1, 1), (1, 2), (1, 3), (1, 4),

(1, 5)], [(1, 2), (1, 3), (1, 3), (1, 3), (1, 4)]], 'collisions': [{'a1': 0, 'a2': 1, 'loc': [(1, 3)], 'timestep': 2}]}

Generate node 9

Generate node 10

Expand node 10

expanded node: {'cost': 8, 'constraints': [{'agent': 1, 'loc': [(1, 4)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 4), (1, 3)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 3)], 'timestep': 2}

], 'paths': [[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3), (1, 4), (1, 5), (1, 4)]], 'collisions': [{'a1': 0, 'a2': 1, 'loc': [(1, 4), (1, 5)], 'timestep': 4}]}

Generate node 11

Generate node 12

Expand node 5

expanded node: {'cost': 9, 'constraints': [{'agent': 1, 'loc': [(1, 4)], 'timestep': 3}, {'agent': 0, 'loc': [(1, 3), (1, 4)], 'timestep': 3}], 'paths': [[(1, 1), (1, 2), (1, 3), (1, 3),

(1, 4), (1, 5)], [(1, 2), (1, 3), (1, 4), (1, 3), (1, 4)]], 'collisions': [{'a1': 0, 'a2': 1, 'loc': [(1, 3)], 'timestep': 3}]}

Generate node 13

Generate node 14

Expand node 7

expanded node: {'cost': 9, 'constraints': [{'agent': 0, 'loc': [(1, 4)], 'timestep': 3}, {'agent': 0, 'loc': [(1, 4)], 'timestep': 4}, {'agent': 0, 'loc': [(1, 4)], 'timestep': 5}], 'path

s': [[(1, 1), (1, 2), (1, 3), (1, 3), (1, 3), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3), (1, 4)]], 'collisions': [{'a1': 0, 'a2': 1, 'loc': [(1, 4)], 'timestep': 6}]}

Generate node 15

Generate node 16

Expand node 9

expanded node: {'cost': 9, 'constraints': [{'agent': 1, 'loc': [(1, 4)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 4), (1, 3)], 'timestep': 3}, {'agent': 0, 'loc': [(1, 3)], 'timestep': 2}

], 'paths': [[(1, 1), (1, 2), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3), (1, 3), (1, 3), (1, 4)]], 'collisions': [{'a1': 0, 'a2': 1, 'loc': [(1, 3)], 'timestep': 3}]}

Generate node 17

Generate node 18

Expand node 11

expanded node: {'cost': 9, 'constraints': [{'agent': 1, 'loc': [(1, 4)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 4), (1, 3)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 3)], 'timestep': 2}

, {'agent': 0, 'loc': [(1, 4), (1, 5)], 'timestep': 4}], 'paths': [[(1, 1), (1, 2), (1, 3), (1, 4), (1, 4), (1, 5)], [(1, 2), (1, 3), (1, 4), (1, 5), (1, 4)]], 'collisions': [{'a1': 0, 'a

2': 1, 'loc': [(1, 4)], 'timestep': 4}]}

Generate node 19

Generate node 20

Expand node 12

expanded node: {'cost': 9, 'constraints': [{'agent': 1, 'loc': [(1, 4)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 4), (1, 3)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 3)], 'timestep': 2}

, {'agent': 1, 'loc': [(1, 5), (1, 4)], 'timestep': 4}], 'paths': [[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3), (1, 2), (1, 3), (1, 4), (1, 4)]], 'collisions': [{'a1': 0, 'a

2': 1, 'loc': [(1, 2), (1, 3)], 'timestep': 2}]}

Generate node 21

Generate node 22

Expand node 14

expanded node: {'cost': 9, 'constraints': [{'agent': 1, 'loc': [(1, 4)], 'timestep': 3}, {'agent': 0, 'loc': [(1, 3), (1, 4)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 3)], 'timestep': 3}

], 'paths': [[(1, 1), (1, 2), (1, 3), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3), (1, 4), (1, 5), (1, 4)]], 'collisions': [{'a1': 0, 'a2': 1, 'loc': [(1, 4)], 'timestep': 4}]}

Generate node 23

Generate node 24

Expand node 18

expanded node: {'cost': 9, 'constraints': [{'agent': 1, 'loc': [(1, 4)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 4), (1, 3)], 'timestep': 3}, {'agent': 0, 'loc': [(1, 3)], 'timestep': 2}

, {'agent': 1, 'loc': [(1, 3)], 'timestep': 3}], 'paths': [[(1, 1), (1, 2), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3), (1, 4), (1, 5), (1, 4)]], 'collisions': [{'a1': 0, 'a2': 1, '

loc': [(1, 4)], 'timestep': 4}]}

Generate node 25

Generate node 26

Expand node 22

expanded node: {'cost': 9, 'constraints': [{'agent': 1, 'loc': [(1, 4)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 4), (1, 3)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 3)], 'timestep': 2}

, {'agent': 1, 'loc': [(1, 5), (1, 4)], 'timestep': 4}, {'agent': 1, 'loc': [(1, 3), (1, 2)], 'timestep': 2}], 'paths': [[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 2), (1, 2),

(1, 3), (1, 4), (1, 4)]], 'collisions': [{'a1': 0, 'a2': 1, 'loc': [(1, 2)], 'timestep': 1}]}

Generate node 27

Generate node 28

Expand node 28

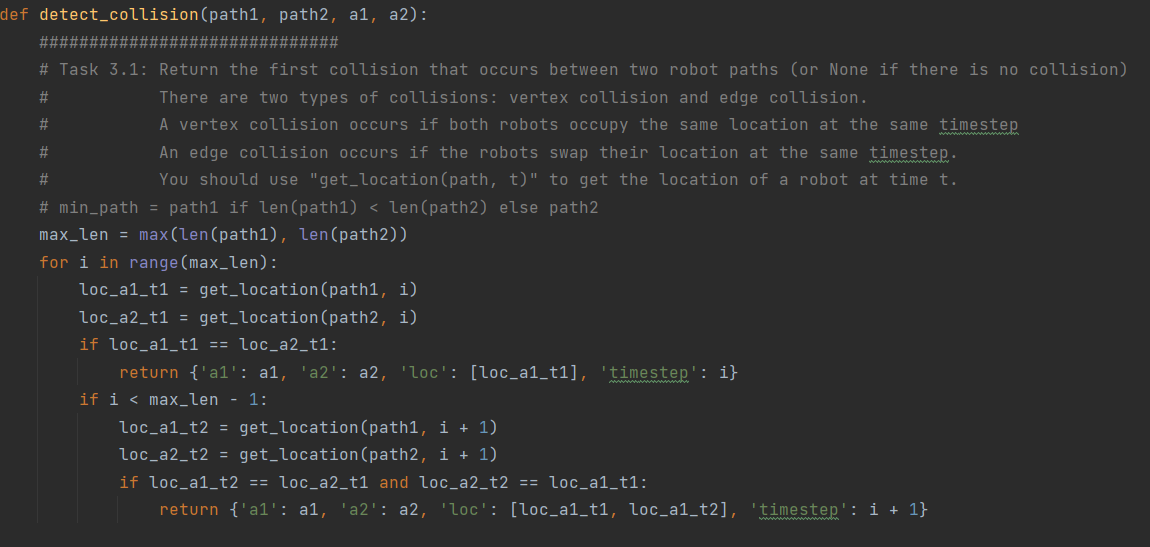
expanded node: {'cost': 9, 'constraints': [{'agent': 1, 'loc': [(1, 4)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 4), (1, 3)], 'timestep': 3}, {'agent': 1, 'loc': [(1, 3)], 'timestep': 2}

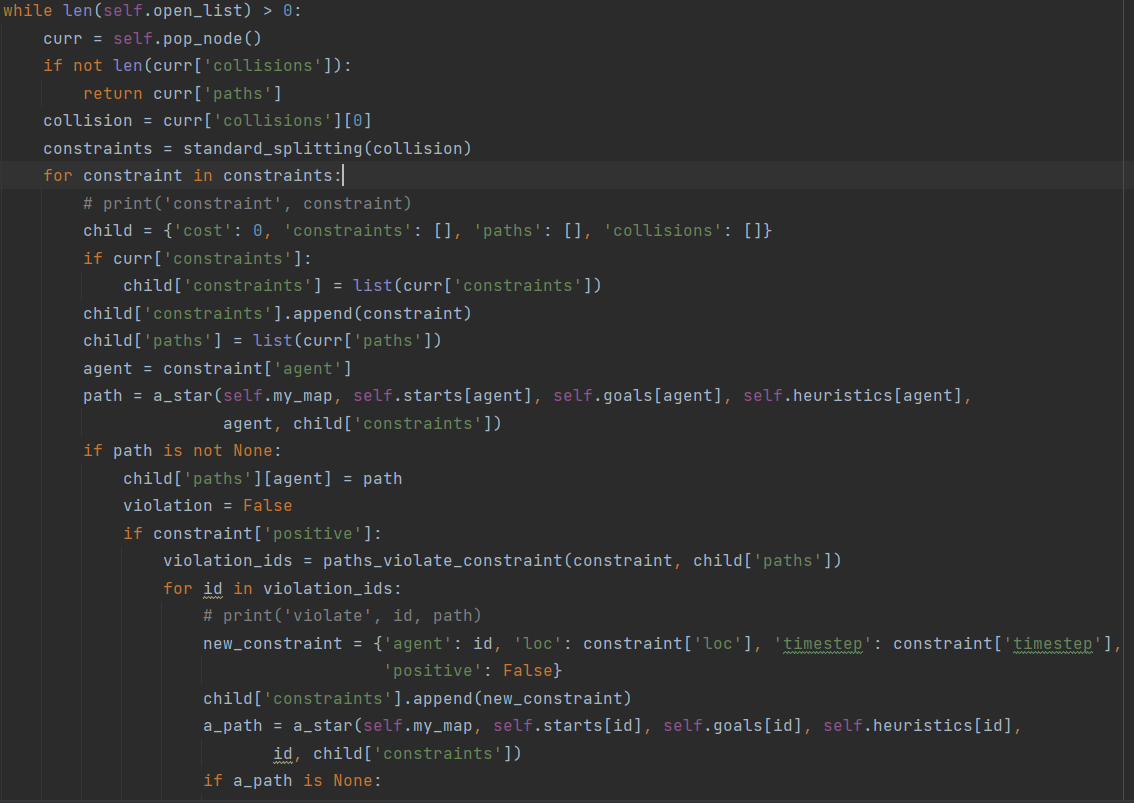
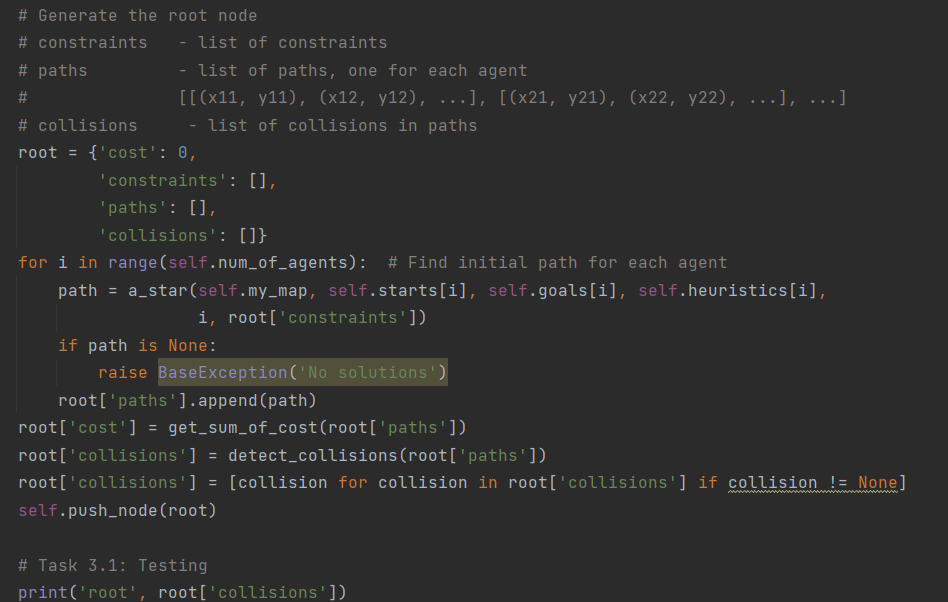
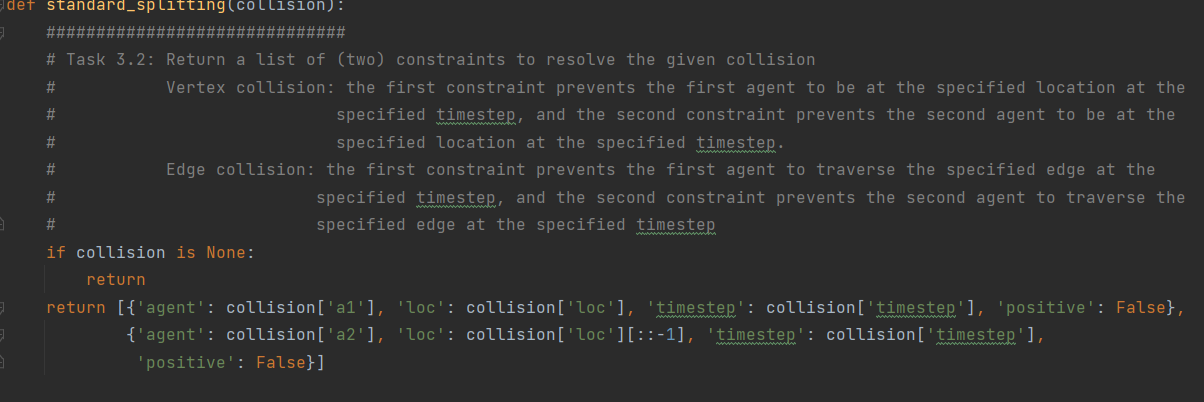
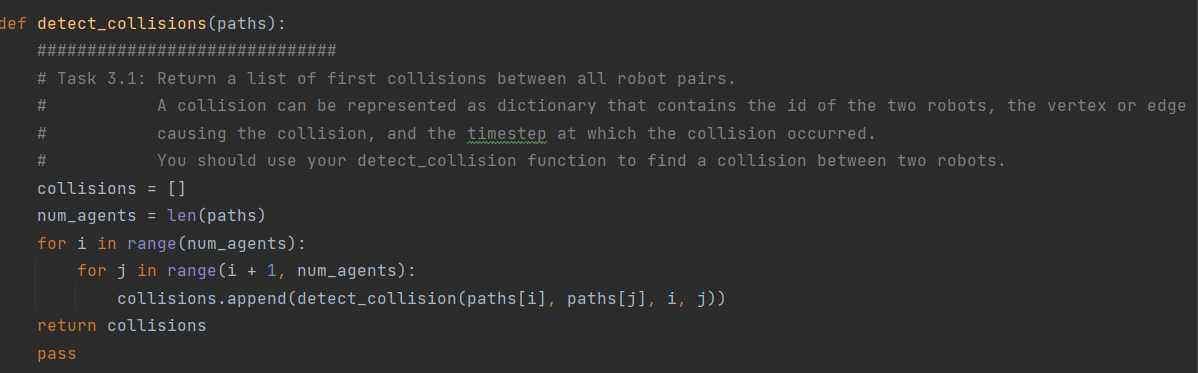
, {'agent': 1, 'loc': [(1, 5), (1, 4)], 'timestep': 4}, {'agent': 1, 'loc': [(1, 3), (1, 2)], 'timestep': 2}, {'agent': 1, 'loc': [(1, 2)], 'timestep': 1}], 'paths': [[(1, 1), (1, 2), (1,

3), (1, 4), (1, 5)], [(1, 2), (1, 3), (2, 3), (1, 3), (1, 4), (1, 4)]], 'collisions': [None]}

* 1. **Testing your Implementation**

I added some boundary checks in single\_agent\_planner since the algorithm was failing on some test instances where there are no obstacles on the borders.





1. **Implementing CBS with Disjoint Splitting**
   1. **Supporting Positive Constraints**

I added a positive field in the constraint dictionary for every instance of constraint across all of the files. In single\_agent\_planner, I handled positive constraints in is\_constrained and returned False if, for the given constraint, positive is True. This way, the agent will have this route in his path.

* 1. **Converting Collisions to Constraints**

My disjoint\_splitting contains these two constraints: agent n must go to the given location in the given timestep, or he must not go to the given location in the given timestep.

* 1. **Adjusting the High-Level Search**

My code expands 15 nodes for both implementations. It runs into an infinite loop on some instances. I’m not sure why. It worked fine at a point (still with no node generation improvement, though), then broke. I conducted A\* on each agent returned from paths\_violate\_constraint, and prune the node if there is no path.

