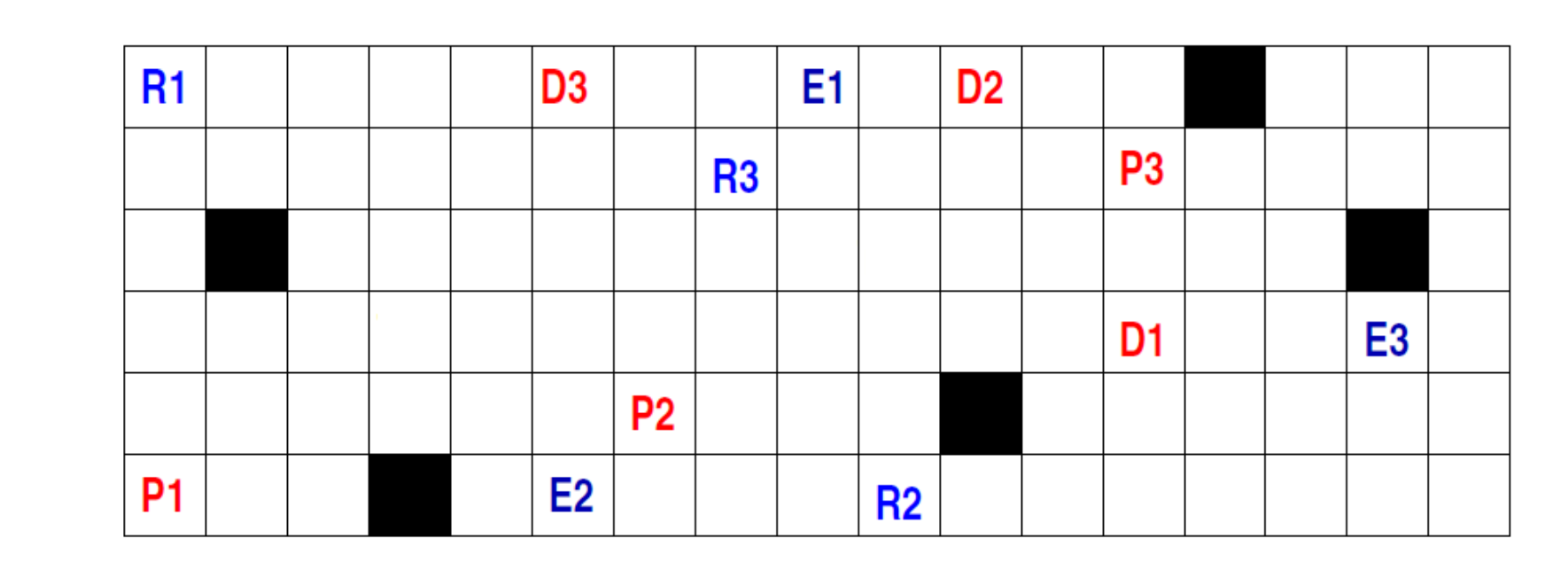
Problem Statement

A certain shop floor is represented as a 2-D grid of size n x m, as shown in the figure below. In the shop floor, robots are employed to perform shipment tasks, where a task involves shipping a component from a designated pick-up cell location to a destination cell. There are ‘k’ robots and ‘k’ shipment tasks. Each robot performs a single task. A robot can move at most one cell (either vertically or horizontally) at a time step. A cell cannot be simultaneously occupied by more than one robot. Black coloured cells are obstacles and cannot be traversed through. All robots start at the same time. The ith robot starts from cell location Ri, picks up one the components (say) at location Pj, delivers it in cell Ej and finally moves to cell Di. The objective is to choose the appropriate task for each robot and determine its corresponding travel path, so that overall completion time for the work schedule involving all tasks is minimised.



A step-wise depiction of the solution approach

To write the CBS function, we followed the underlying steps:

1. We first defined a function named *cbs* that takes the input parameters given problem : grid size, the number of agents, the locations of obstacles, and the start, pick-up, delivery, and end locations for each agent.
2. Then we created a root node with no parent or constraint and initialised it’s paths with A\* search for each agent without any constraint.
3. Then we created a priority queue for storing nodes in the constraint tree and pushed the root node to it.
4. Created an empty list for storing the solution paths.
5. While the priority queue was not empty, we removed the node with the lowest cost from it.
6. If the node was a goal node, we assigned its paths to the solution list and broke the loop.
7. Otherwise, detected any conflicts between the agents’ paths in the node and chose one conflict to resolve.
8. For each agent involved in the conflict,we created a child node with a new constraint that prevented the agent from being at the conflict location at the conflict time.
9. For each child node,we updated its paths with A\* search for the constrained agent with the new constraint and kept the other agents’ paths unchanged.
10. Then we pushed each child node to the priority queue.
11. Finally, we returned the solution list or None if no solution is found.

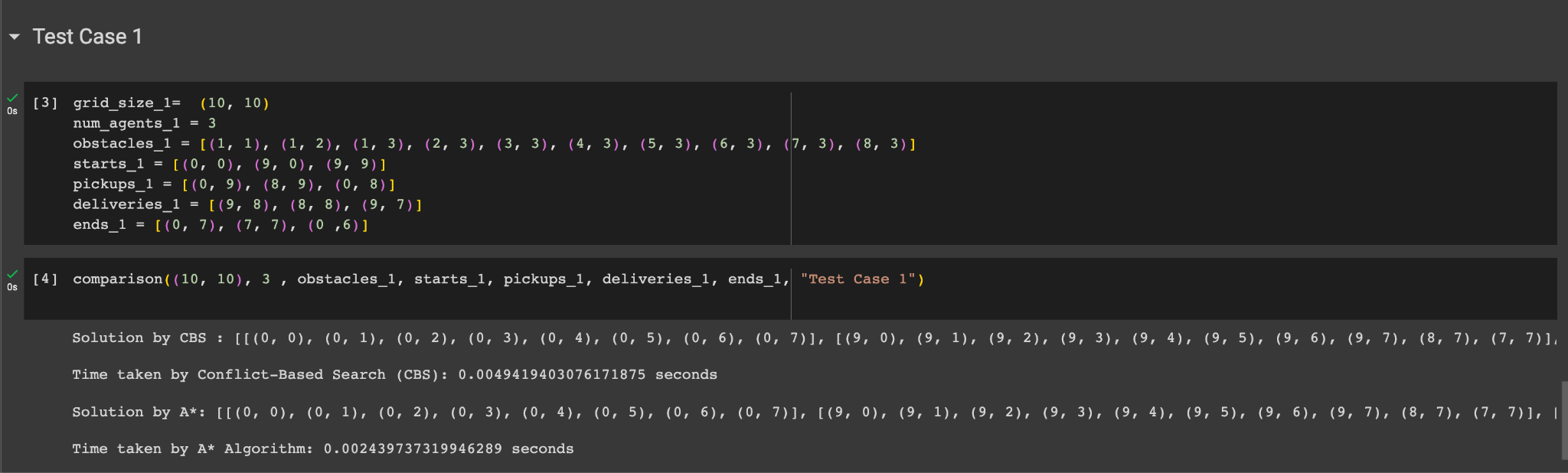
Screenshots inputs and outputs for various test cases :

Below figures contains both inputs and outputs :

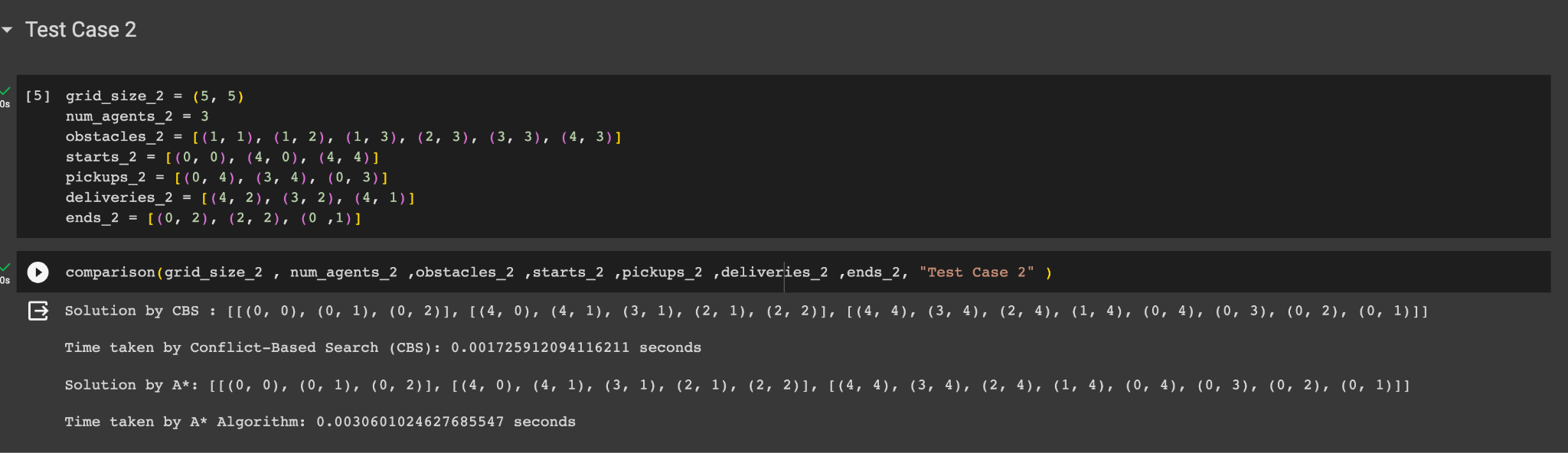
Inputs : grid size, obstacles, starts, pickups, deliveries, ends

Outputs : The path traversed, time taken

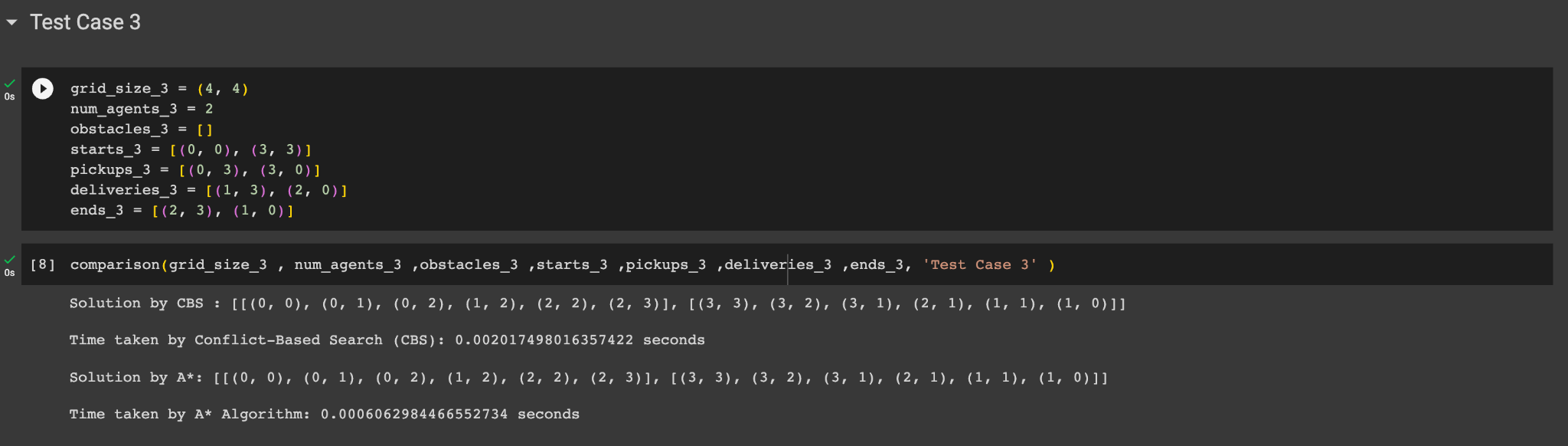
1. Test Case 1 :



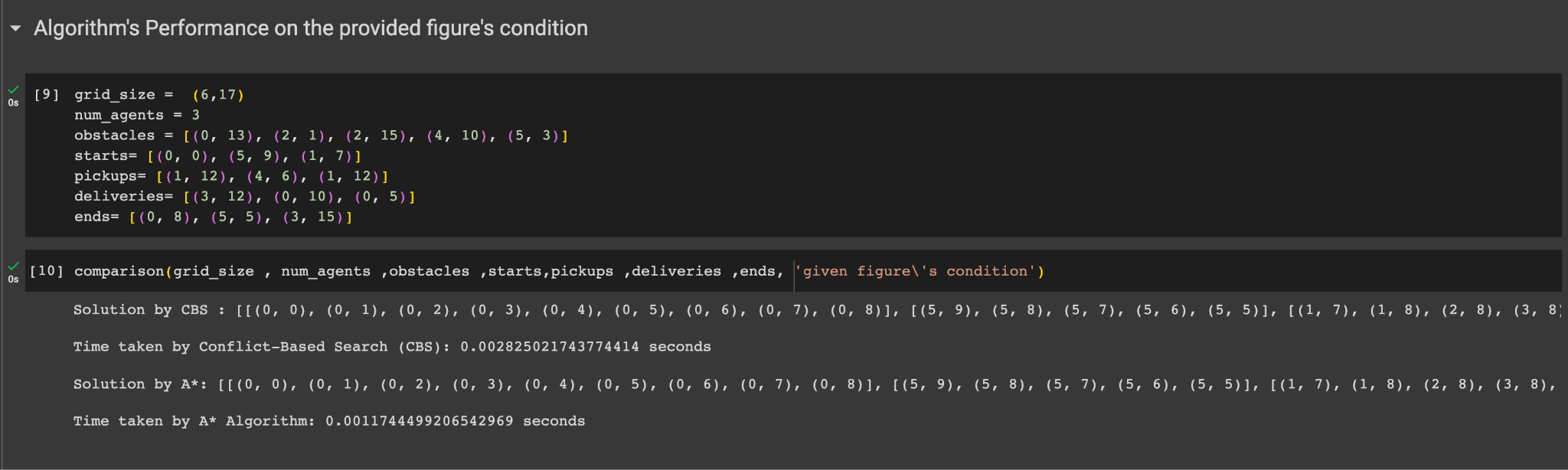
1. Test Case 2 :



1. Test Case 3 :



### Algorithm's Performance on the provided figure's condition



Analysis:

We implemented the Conflict Based Search (CBS) algorithm and a simple A\* based algorithm for the multi-agent path finding problem on the shop floor. We used Google Colab as the programming environment and Python as the programming language and tested our algorithms on different grid sizes, number of agents, and obstacle densities. Then measured the execution time and the solution quality of both algorithms.

The CBS algorithm is a two-level algorithm that searches over a constraint tree, where each node represents a set of constraints on the agents’ paths, and uses A\* search to find paths for each agent that satisfy the constraints. It may guarantee optimal solutions, but it may require a large amount of memory and time to explore the constraint tree.

The A\* based algorithm is a simple algorithm that runs A\* search on each agent’s path independently, without considering any conflicts or constraints. It does not guarantee optimal solutions, but it is faster and more memory-efficient than CBS.

Discussion:

We compared the performance of CBS and A\* based algorithms on different scenarios. We found that CBS outperformed A\* based algorithm in terms of the execution time and solution quality, as it always found optimal solutions, while A\* based algorithm often found optimal. However, CBS also had a higher time overhead than A\* based algorithm, as it had to search over a large constraint tree and resolve many conflicts. The time overhead of CBS increased exponentially with the number of agents and the obstacle density, while the time overhead of A\* based algorithm increased linearly.

We also observed that CBS was more robust and scalable than A\* based algorithm, as it could handle complex scenarios with many agents, obstacles, and different start and goal locations, while A\* based algorithm often failed to find any solution or produced paths with collisions. CBS was able to avoid or resolve conflicts by imposing constraints on the agents’ paths, while A\* based algorithm did not have any mechanism to deal with conflicts.

We conclude that CBS is a more suitable algorithm for multi-agent path finding on the shop floor, as it can find optimal solutions that minimizes the overall completion time for the work schedule involving all tasks. However, CBS also has some limitations and challenges, such as high memory and time requirements, dependency on heuristic functions, and sensitivity to conflict resolution strategies. Some directions for future research can be , such as improving the efficiency and effectiveness of CBS by using heuristic pruning, symmetry breaking, meta-agent formation, or machine learning techniques.

Group Participants : 1. Ashish Kumar : 20BT3AI05

2. Debajyoti Roy : 20BT3AI06

3. Rohan Goel : 20QM30011

4. Arunesh Manna : 20IE10053