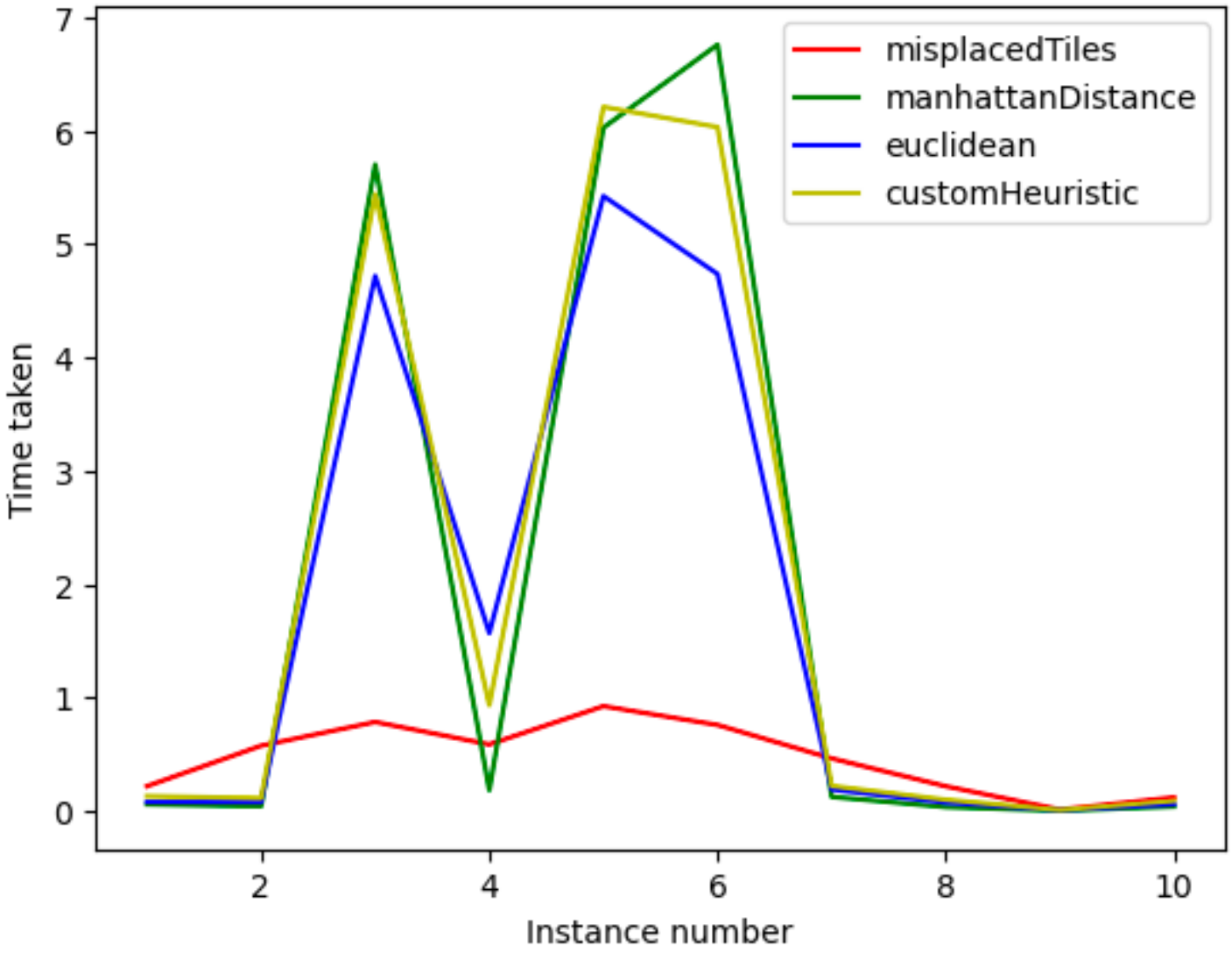
A Star search for 8 puzzle

Lab Report
Date: 1st Feb, 2023

Heuristics used for estimation:

- **Misplaced tiles** heuristic.
- **Manhattan distance** heuristic.
- **Euclidean distance** heuristic.
- A **custom heuristic** which is a linear combination of the three of the above heuristics. The linear combination I used was 0.6*Manhattan + 0.1*MisplacedTiles + 0.3*Euclidean.

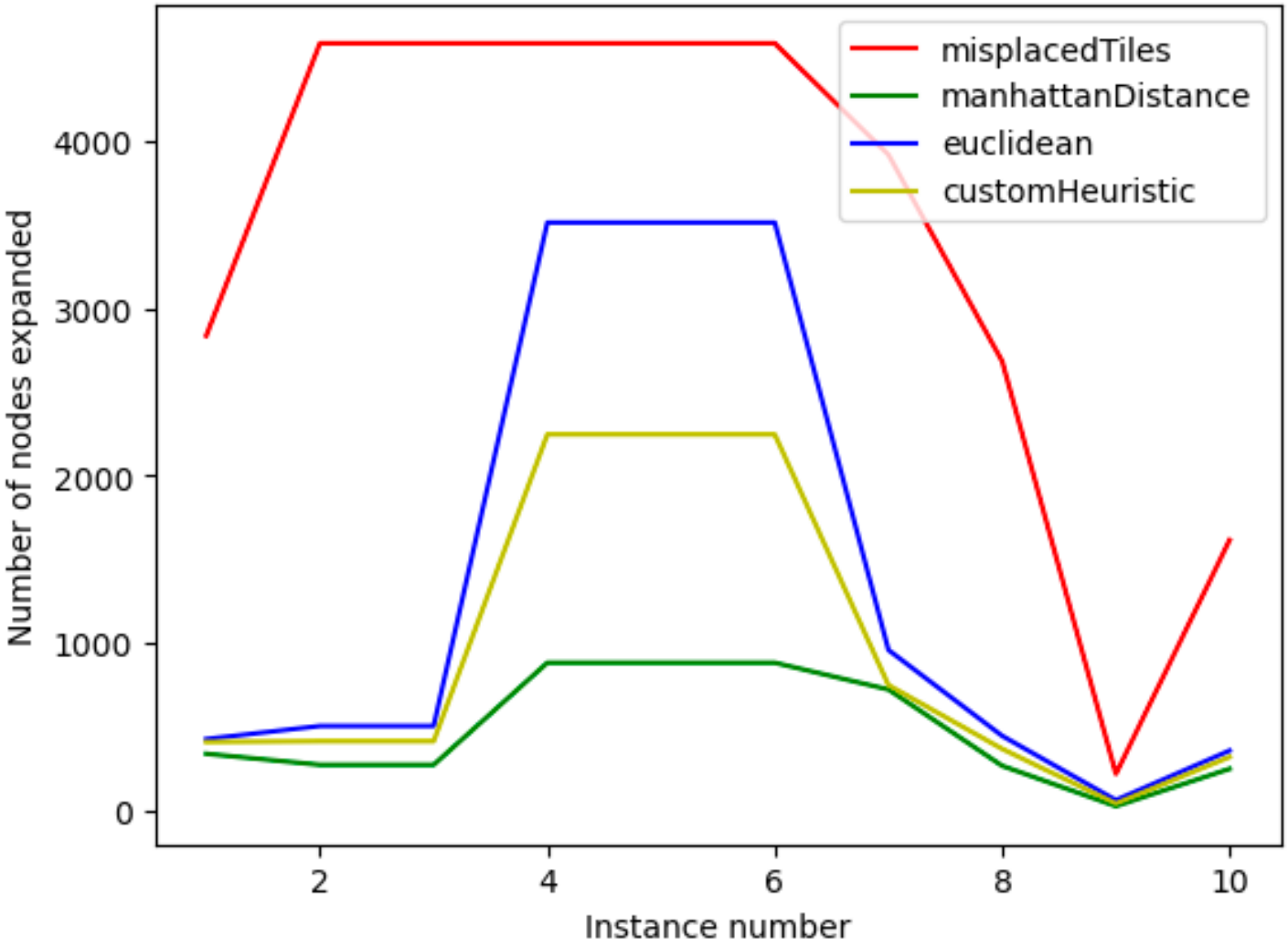
Time taken by the algorithm:



In the first two instances, 4th and the last 4 instances, **Manhattan distance turns out to be taking lesser time** than the other Heuristics. Euclidean comes next followed by custom and then misplaced tiles heuristic.

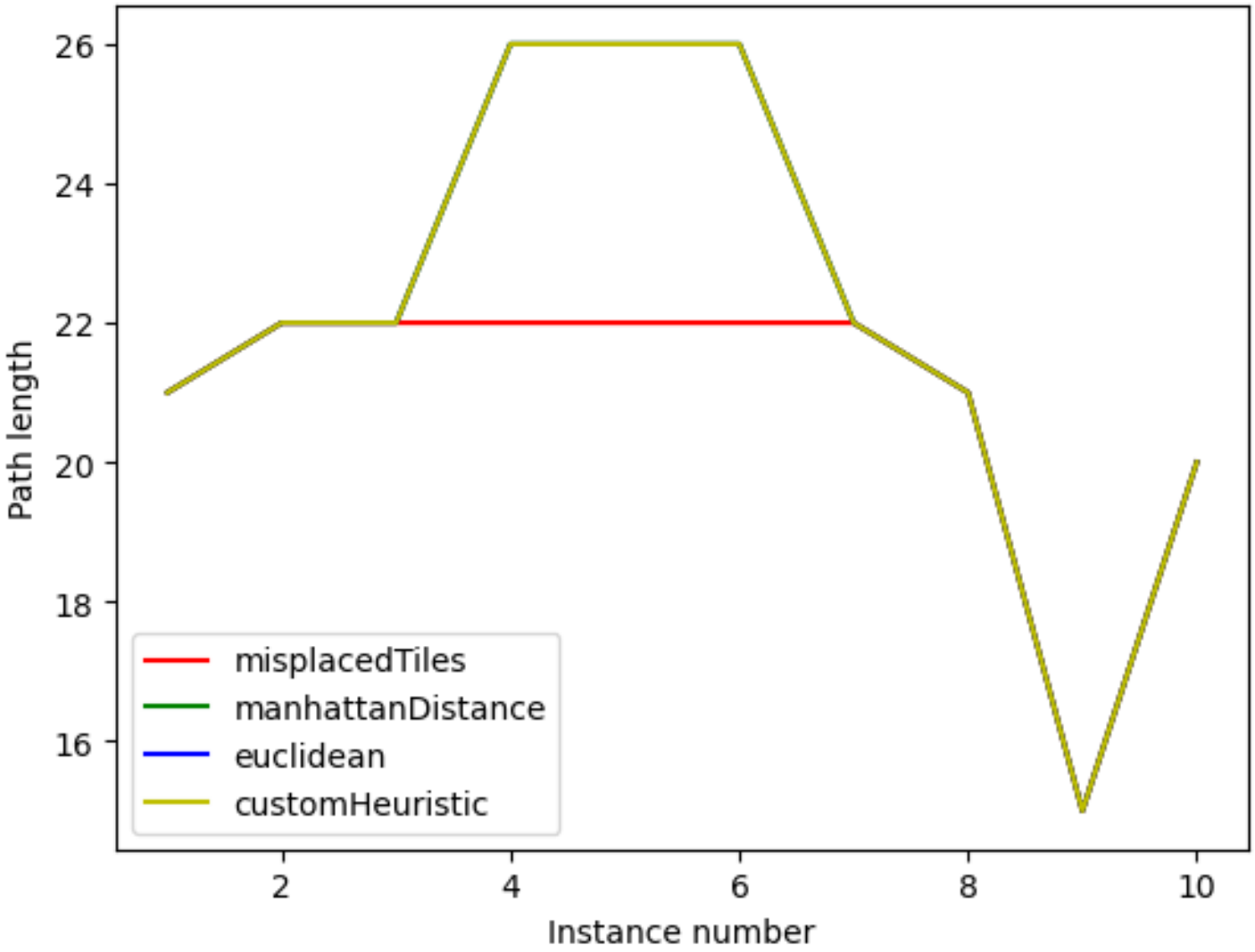
In the 3rd, 5th and 6th instance, Manhattan distance seems to be taking more time than the others. This is because these instances were terminated before completion due to the upper limit of iterations i.e. 10000, set by me to reduce the total time taken to run the program. As these instances couldn't go to completion, Manhattan distance and euclidean distance seem to be taking more time because of the **relatively higher per state computational cost** of these heuristics (i.e. **O((n)^2**) where n is the number of tiles) as compared with misplaced tiles where the cost is **O(n)**.

Number of nodes expanded:



This graph clearly shows how Manhattan distance is the better heuristic for solving 8 puzzle using A star algorithm. Number of nodes expanded by Manhattan distance is **marginally lower** than that by Misplaced tiles.

Path length:



All the three standard heuristics show **similar path lengths** as A star algorithm is meant to return the **optimal path** given that the heuristic does not overestimate the cost of the paths.

My own custom heuristic seems to be taking a longer path in the middle instances, which shows that for some instances, the custom heuristic overestimates the cost of the path length.

Best Heuristic so far: Manhattan distance

We can clearly conclude that **Manhattan distance is the better heuristic** among the ones we have tried so far. This is because Manhattan distance closely estimates the path cost by giving us the cost of moving a tile **perpendicularly** rather than directly which is used by Euclidean distance. This resembles and tries to mimic how the tiles are moved in the actual game, i.e. their actual path length. Misplaced tiles heuristic does not perform that good a task at estimating the cost as Manhattan distance because of the same reasons.

Search for a better heuristic:

Disjoint Pattern Database

I also tried to implement the disjoint pattern database heuristic which pre computes the costs of reaching subproblems or sub-patterns from the goal state and stores it in a database to be accessed later at O(1) cost and their sum used as a heuristic. This greatly improves the performance of the algorithm as the heuristic closely estimates the cost of solving one subproblem while ignoring the other parts of the problem and gives us a fairly optimal heuristic estimate. But I could not implement it within the given time as I kept running into an error or an infinite loop while using the heuristic, and could not quite locate the issue. Anyway I have also attached its code with my source code towards the end to identify what is causing the error or if my implementation is wrong.