Final project report - analysis of multiple algorithms and heuristics for solving the Rush Hour puzzle

Search in artificial intelligence course

Aviv Rovshitz
ID - 307974162
Phone - 0525294795
Email - avivrov@post.bgu.ac.il

Leor Ariel Rose
ID - 208373365
Phone - 0503992002
Email - leorro@post.bgu.ac.i

Software and Information Systems Engineering
Ben-Gurion University of the Negev
Israel

08.03.23

1 Introduction

"Rush Hour" is a timeless single-player sliding block puzzle game, first introduced by renowned puzzle designer, Nob Yoshigahara, in the 1970s (Wikipedia contributors, 2022c). The objective of the game is to manoeuvre the red car to the exit of a 6x6 grid (Fig. 1). A few cars and trucks are scattered around this grid, among 12 different colours of cars and four different colours of trucks. The cars occupy 2 squares each, while the trucks occupy 3 squares, adding to the puzzle's complexity. In the game, players must complete 40 challenging card levels. To get the red car to the exit, the player must slide the other vehicles horizontally or vertically. Finding the best sequence of moves to accomplish this task while avoiding dead ends and ensuring that the red car does not get stuck is the challenge. Over four decades, "Rush Hour" has tested players' problem-solving skills and strategic thinking, making it a classic game enjoyed by individuals of all ages.

Figure 1: Rush hour game board with all the vehicles placed on it and level cards.

The following **algorithms** we used in our experiments:

1. **Breadth First Search (BFS)** - uninformed graph traversal algorithm that explores all the vertices of a graph or all the nodes of a tree data structure in order of their depth from the root, i.e., it visits all the vertices at distance 1 from the source vertex first, then all the vertices at distance 2, and so on. To keep track of the vertices that need to be visited, the algorithm uses a first-in, first-out queue data structure (Wikipedia contributors, 2022a).

2. **Depth First Search (DFS)** - uninformed graph traversal algorithm that explores vertices in depth-first order, i.e., it visits a vertex and then recursively visits all its unvisited children before backtracking. It is implemented using a last in first out stack data structure or recursion (Wikipedia contributors, 2023b).

3. **A Star (A*)** - informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs: starting from a specific starting node of a graph, it aims to find

a path to the given goal node having the smallest cost (least distance travelled, shortest time, etc.). A* uses a priority queue to keep track of the nodes to visit and it visits the node with the lowest f(n) = g(n) + h(n), where g(n) is the cost of reaching the node n and h(n) is the estimated distance from the node n to the goal node. This way, it tries to find the shortest path by exploring nodes that are more likely to lead to the goal (Wikipedia contributors, 2023a).

4. **Iterative deepening A* (IDA*)** - informed search algorithm and a variant of the A* algorithm that combines the benefits of iterative deepening DFS and A*. It explores nodes in depth first, but it also uses a heuristic function like A* to estimate the distance to the goal node. The key idea behind IDA* is to limit the depth of the search, and incrementally increase the limit after each iteration until the goal node is found or the limit exceeds the estimated cost of reaching the goal node. Since it is a depth-first search algorithm, its memory usage is lower than in A*, but unlike ordinary iterative deepening search, it concentrates on exploring the most promising nodes and thus does not go to the same depth everywhere in the search tree. Unlike A*, IDA* does not utilise dynamic programming and therefore often ends up exploring the same nodes many times. In IDA*, a threshold value is set for the maximum f(n) = g(n) + h(n) value that a node can have and still be considered for expansion. Nodes with f(n) exceeding the threshold are discarded, and the search continues with the next node. If no solution is found, the threshold is increased and the search is repeated. This process continues until a solution is found or the threshold exceeds the estimated cost of reaching the goal (Wikipedia contributors, 2022b).

The following **heuristics** were used in our experiments:

1. **Zero/Null/Trivial** - a heuristic function that always returns a value of 0 for any given node. In other words, it provides no additional information or estimates about the distance to the goal node. When used in a heuristic search algorithm like A*, the zero heuristic effectively turns the algorithm into a uniform-cost search, which explores all the nodes in increasing order of their cost, without making any assumptions or estimations about the distance to the goal. The number of actions we need to do in order to get to a goal state is at least zero distance from the red car to the exit. Therefore, the heuristic doesn't overestimate the actual number of actions and thus is admissible.

2. **Manhattan Distance** - a heuristic function that returns the manhattan distance of the red car from the exit. The number of actions we need to do in order to get to a goal state is at least the Manhattan distance of the red car from the exit (maybe more if additional vehicles need to be moved). Therefore, the heuristic doesn't overestimate the actual number of actions and thus is admissible.

3. **Blocking Vehicles** - a heuristic function that returns the number of vehicles blocking the car from the exit. The number of actions we need to do in order to get to a goal state is at least the number of vehicles blocking the car from the exit (maybe more if additional vehicles need to be moved). Therefore, the heuristic doesn't overestimate the actual number of actions and thus is admissible.

4. **Improved Blocking Vehicles** - Same as the "Blocking Vehicles" heuristic, the only improvement is that blocked vehicles gain another point. Each vehicle blocking the car from the exit that is also blocked means we need to make (at least) another move to unblock it so the improvement doesn't make this heuristic overestimate the actual cost function, and thus is admissible.

5. **Distance Improved Blocking Vehicles** - a combination of the "Improved Blocking Vehicles" and "Manhattan distance" heuristics. The number of actions we need to do in order to get to a goal state is at least the number of vehicles blocking the car from the exit. Each vehicle blocking the car from the exit that is also blocked means we have to make (at least) another move to unblock it. After we deal with all the blocking vehicles we still need to get the red car to the exit. This is the Manhattan distance of the red car from the exit. Therefore, this heuristic doesn't overestimate the actual cost function, and thus is admissible.

2 Experiment Objective

The goal of this project is to comprehensively evaluate and compare several algorithms (BFS, DFS, A*, IDA*) and heuristics (Zero, Manhattan distance, Blocking vehicles, Improved blocking vehicles, Distance improved blocking vehicles) for solving the classic puzzle game, Rush Hour. The objective is to determine the most efficient and effective method of solving this game. To achieve this objective, the project will conduct a thorough analysis and testing of each algorithm and heuristic. This will take into consideration various performance metrics such as solution time, solution length, and expanded nodes (complexity). Our hypothesis for this experiment is that the most efficient algorithm is A*, followed by BFS, DFS and that the poorest results are achieved by IDA*. Heuristically speaking, we believe the "distance improved blocking vehicle" will be the best since it has the most reliable estimation, but we're not sure how the other heuristics will stand up to themselves.

3 Experiment Description

The experiments for solving the Rush Hour puzzle game using various algorithms and heuristics were conducted using the Python programming language (Full code at appendix A.2). We created the Rush Hour puzzle game, algorithms, and heuristics mentioned in the introduction. We extracted for each algorithm and heuristic combination their solution times, solution lengths, and expanded nodes from a set of 40 problems. Of these problems, 36 were solvable and 4 were non-solvable. We conducted the experiments using a controlled environment to ensure that the results are consistent and reliable. A high-performance PC workstation was used. The workstation was equipped with an 11th Gen Intel(R) Core(TM) i7-11700KF @ 3.60GHz 3.60 GHz processor, 64 GB of RAM, and a 64-bit Windows operating system. Additionally, the workstation was equipped with a powerful Nvidia GeForce RTX 3070 Ti GPU, providing ample computational resources for the experiment.

4 Results

4.1 Analysis of algorithms by nodes expanded

Comparison of the number of expanded nodes in each algorithm is a crucial aspect. Our experiment aimed to analyze this factor and the results were quite revealing. As shown in (Fig. 2), it was evident that the A* algorithm had the lowest number of expanded nodes among all the algorithms tested. This indicates that the A* algorithm is more efficient in terms of node expansion and that it is able to find solutions with fewer nodes being explored. It is interesting to note that the BFS algorithm performed relatively well in terms of node expansion. However, it still lags behind the A* algorithm, with a slightly higher number of expanded nodes. On the other hand, the DFS algorithm had a relatively high number of expanded nodes compared to the A* and BFS algorithms. Finally, the IDA* algorithm had the highest number of expanded nodes among all the algorithms tested. In fact, the number of expanded nodes in IDA* was more than 10 times higher than that of A*. This can be attributed to the fact that IDA* combines the depth-first search approach of DFS with the added constraint of limited memory, which results in a higher number of node expansions.

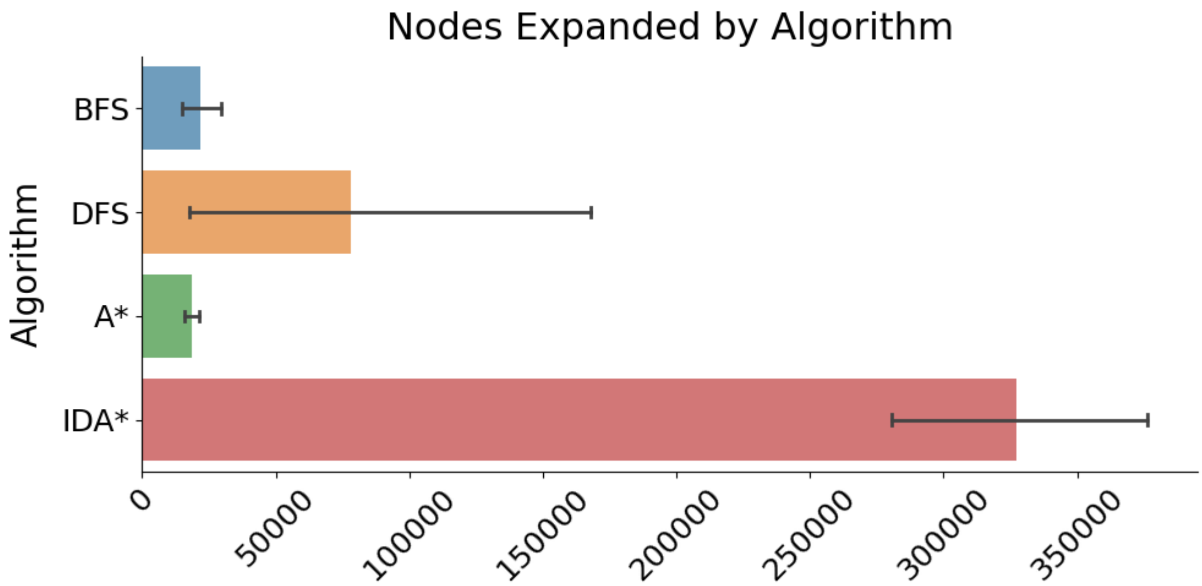

Figure 2: Amount of nodes expanded by each algorithm for all 36 solvable problems. (Blue) BFS algorithm, (Orange) DFS algorithm, (Green) A* algorithm, (Red) IDA* algorithm.

In addition to the results for solvable problems, it is also critical to examine the performance of the algorithms in unsolvable problems (Fig. 3). Upon examining the number of expanded nodes in unsolvable problems, a similar trend was observed. This trend was observed with the A* algorithm having the lowest number of expanded nodes, followed by BFS and DFS. However, the IDA* algorithm could not be compared in this instance as it exhausted all resources within the 48-hour time frame without finding a solution. This highlights the limitations of the IDA* algorithm in situations where the problem may be unsolvable or has a very large search space.

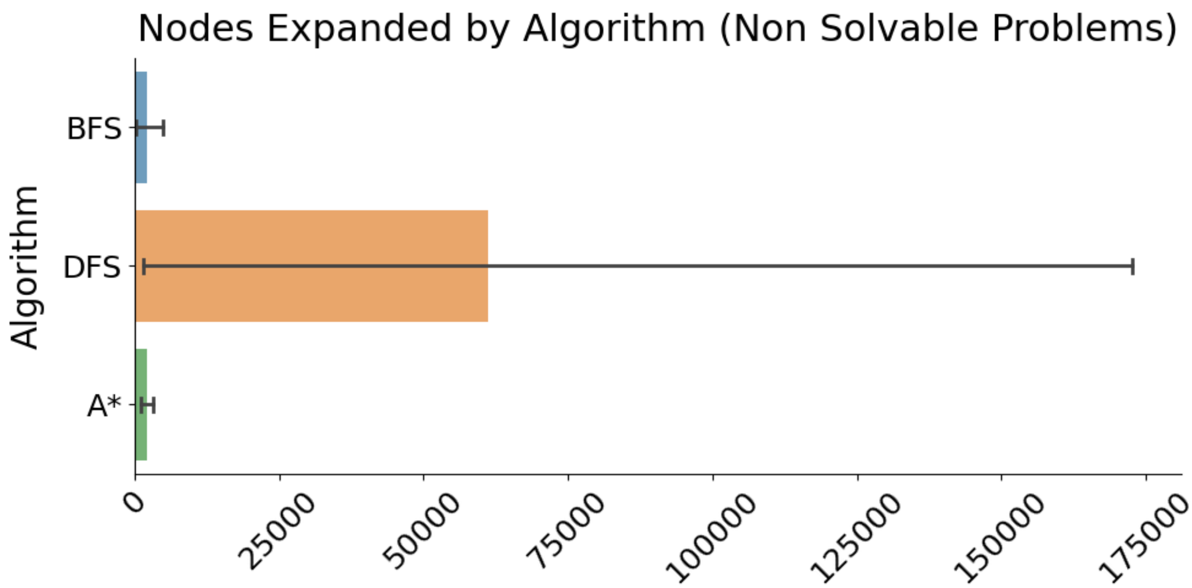

Figure 3: Amount of nodes expanded by each algorithm (excluding IDA* becuase of its incompleteness) for all 4 unsolvable problems. (Blue) BFS algorithm, (Orange) DFS algorithm, (Green) A* algorithm.

4.2 Analysis of heuristics by nodes expanded

In the same manner we compared the number of expanded nodes in each heuristic (Fig. 4, 5)t. It was evident that the Distance Improved Blocking Vehicles heuristic had the lowest number of expanded nodes among all the heuristics tested. This indicates that the Distance Improved Blocking Vehicles heuristic is more efficient in terms of node expansion and that it helps to find solutions with fewer nodes being explored. Manhattan distance, blocking vehicles, and improved blocking vehicles heuristics were almost all the same in the number of nodes expanded. Finally, the zero heuristic had the highest number of expanded nodes among all the algorithms tested. This can be attributed to the fact that it does not add any information to the search space.

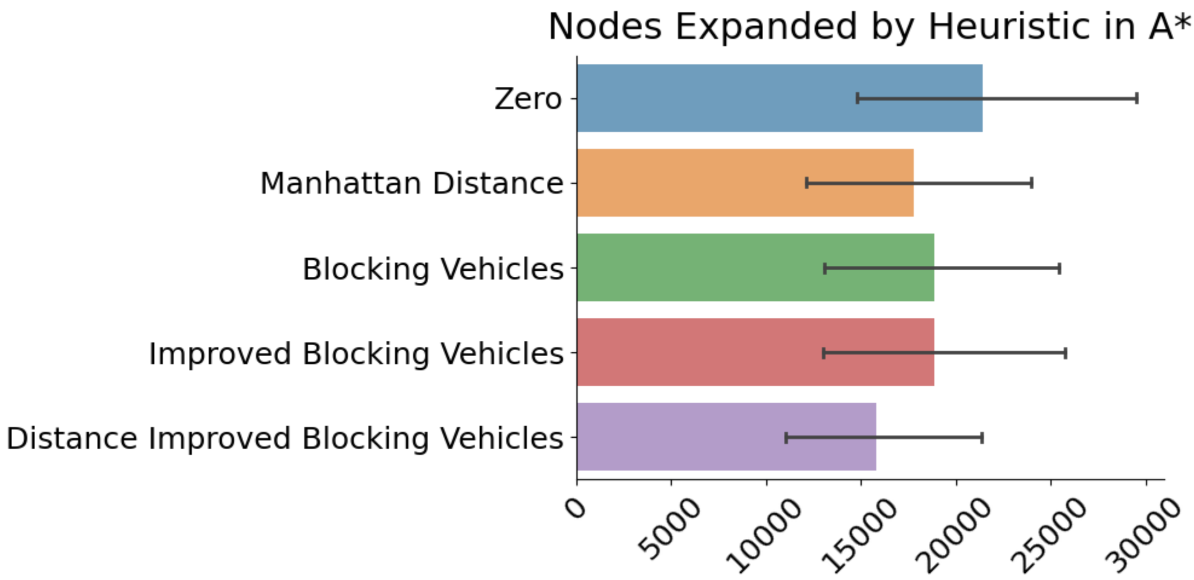

Figure 4: Amount of nodes expanded by each heuristic in A* algorithm for all 36 solvable problems. (Blue) Zero heuristic, (Orange) Manhattan Distance heuristic, (Green) Blocking Vehicles heuristic, (Red) Improved Blocking Vehicles heuristic, (Purple) Distance Improved Blocking Vehicles heuristic.

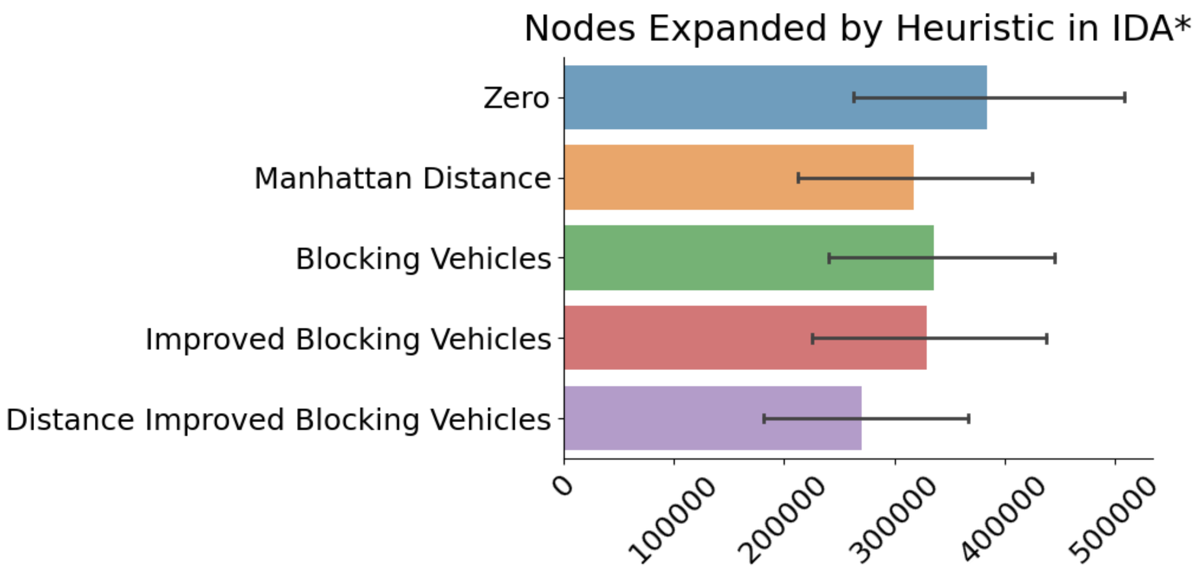

Figure 5: Amount of nodes expanded by each heuristic in IDA* algorithm for all 36 solvable problems. (Blue) Zero heuristic, (Orange) Manhattan Distance heuristic, (Green) Blocking Vehicles heuristic, (Red) Improved Blocking Vehicles heuristic, (Purple) Distance Improved Blocking Vehicles heuristic.

In addition to the results for solvable problems, it is also important to examine the performance of the algorithms in unsolvable problems (Fig. 6). Upon examining the number of expanded nodes in unsolvable problems, a similar trend was observed. This trend was observed within the A* algorithm due to the fact that the IDA* exhausted all resources within the 48-hour time frame without finding a solution.

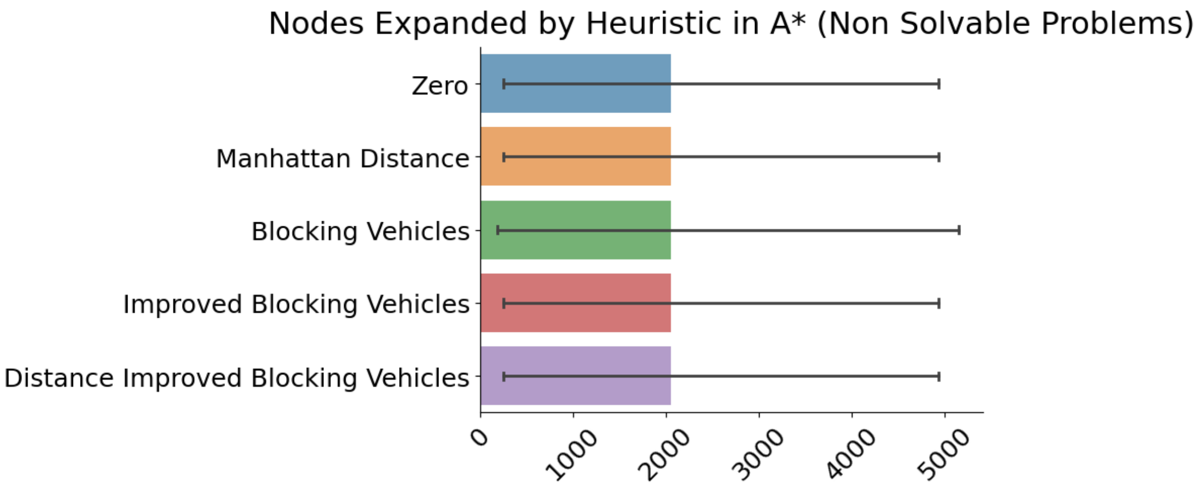

Figure 6: Amount of nodes expanded by each heuristic in A* algorithm for all 4 unsolvable problems. (Blue) Zero heuristic, (Orange) Manhattan Distance heuristic, (Green) Blocking Vehicles heuristic, (Red) Improved Blocking Vehicles heuristic, (Purple) Distance Improved Blocking Vehicles heuristic.

4.3 Analysis of algorithms by run time

Comparison of the run time of each algorithm also a crucial aspect. Not surprisingly the same trend appears here. As shown in (Fig. 7), it was evident that the A* algorithm had the shortest run time among all the algorithms tested. This indicates that the A* algorithm is more efficient in terms of run time and that it is able to find solutions faster. It is interesting to note that the BFS algorithm performed relatively well in terms of run time. However, it still lags behind the A* algorithm, with a slightly longer run time. On the other hand, the DFS algorithm had a relatively long run time compared to the A* and BFS algorithms. Finally, the IDA* algorithm had the longest run time among all the algorithms tested. In fact, the run time of IDA* was more than 10 times longer than that of A*. This can be attributed to the fact that IDA* combines the depth-first search approach of DFS with the added constraint of limited memory, which results in a longer run time.

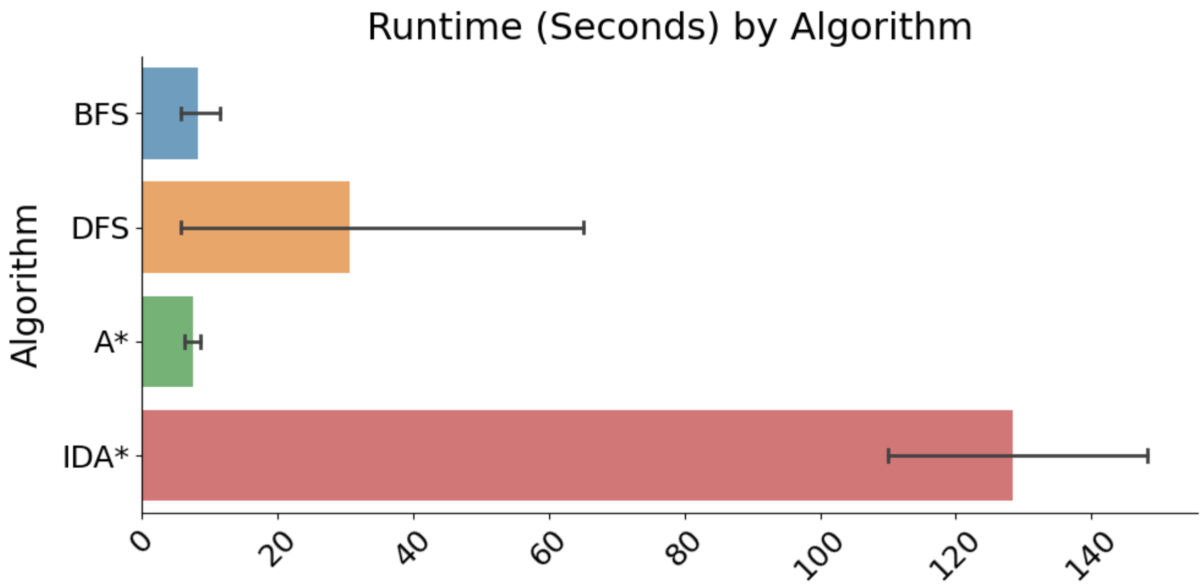

Figure 7: Run time in seconds by each algorithm for all 36 solvable problems. (Blue) BFS algorithm, (Orange) DFS algorithm, (Green) A* algorithm, (Red) IDA* algorithm.

In terms of runtime, the results for unsolvable problems (Fig. 8) showed that the A* algorithm had the shortest runtime among all algorithms, followed by BFS and DFS. However, the IDA* algorithm could not be compared as it was unable to find a solution within the given time frame. This highlights the limitations of the IDA* algorithm in cases where the problem may be unsolvable or has a very large search space.

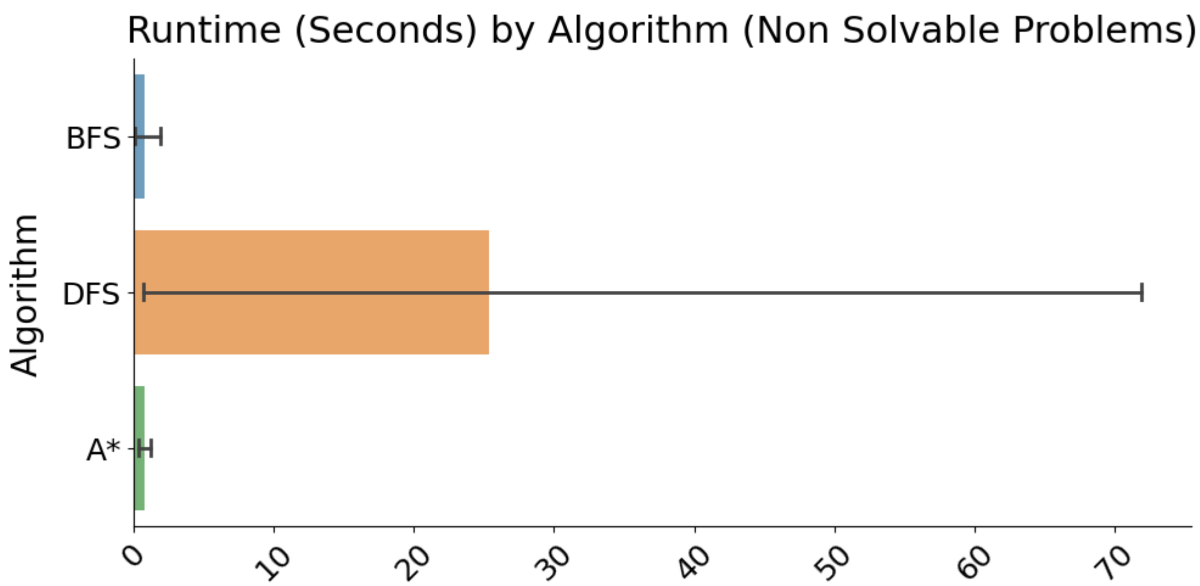

Figure 8: Run time in seconds by each algorithm (excluding IDA* becuase of its incompleteness) for all 4 unsolvable problems. (Blue) BFS algorithm, (Orange) DFS algorithm, (Green) A* algorithm.

4.4 Analysis of heuristics by run time

In the same manner we compared the run time for each heuristic (Fig. 9, 11). As shown in Fig. 2, it was evident that the Distance Improved Blocking Vehicles heuristic had the shortest run time among all the heuristics tested. This indicates that the Distance Improved Blocking Vehicles heuristic is more efficient in terms of run time and that it helps to find solutions faster. The Manhattan distance, blocking vehicles, and improved blocking vehicles heuristics had almost similar run times. Finally, the zero heuristic had the longest run time among all the heuristics tested. This can be attributed to the fact that it does not add any information to the search process.

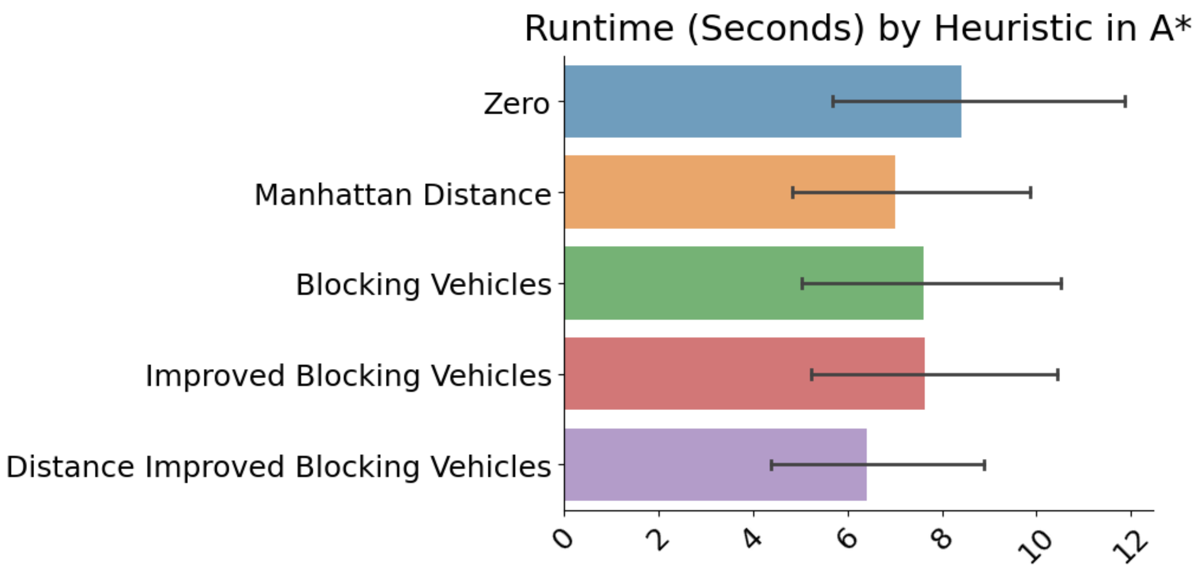

Figure 9: Run time in seconds by each heuristic in A* algorithm for all 36 solvable problems. (Blue) Zero heuristic, (Orange) Manhattan Distance heuristic, (Green) Blocking Vehicles heuristic, (Red) Improved Blocking Vehicles heuristic, (Purple) Distance Improved Blocking Vehicles heuristic.

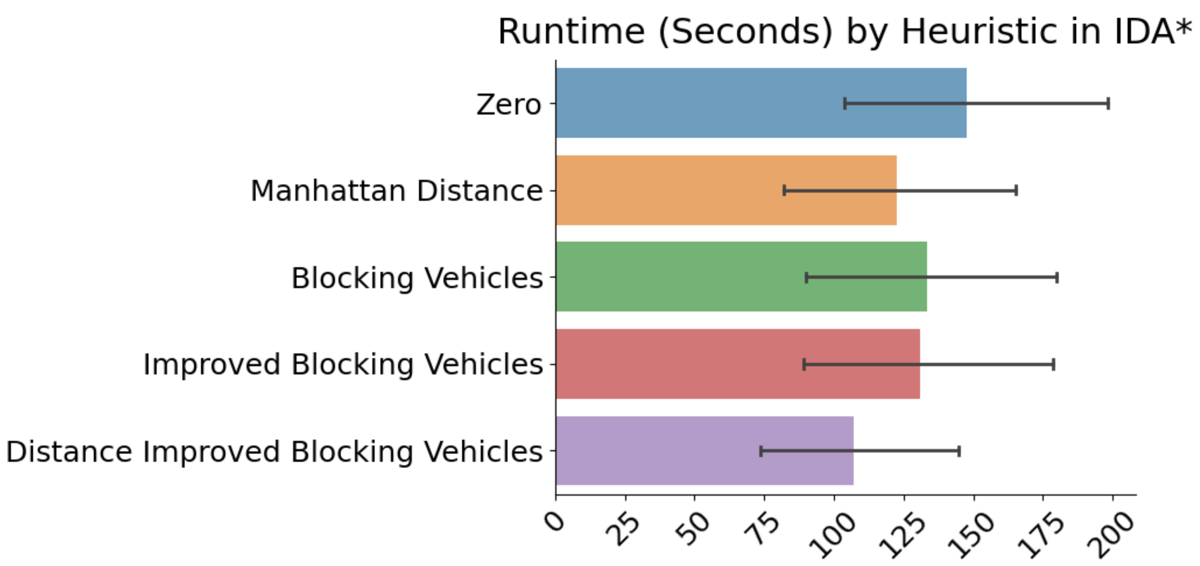

Figure 10: Run time in seconds by each heuristic in IDA* algorithm for all 36 solvable problems. (Blue) Zero heuristic, (Orange) Manhattan Distance heuristic, (Green) Blocking Vehicles heuristic, (Red) Improved Blocking Vehicles heuristic, (Purple) Distance Improved Blocking Vehicles heuristic.

In addition to the results for solvable problems, it is also important to examine the performance of the algorithms in unsolvable problems (Fig. 3). Upon examining the run time in unsolvable problems, a similar trend was observed for the A* algorithm, with the IDA* algorithm unable to find a solution within the 48-hour time frame. This highlights the limitations of the IDA* algorithm in situations where the problem may be unsolvable or has a very large search space.

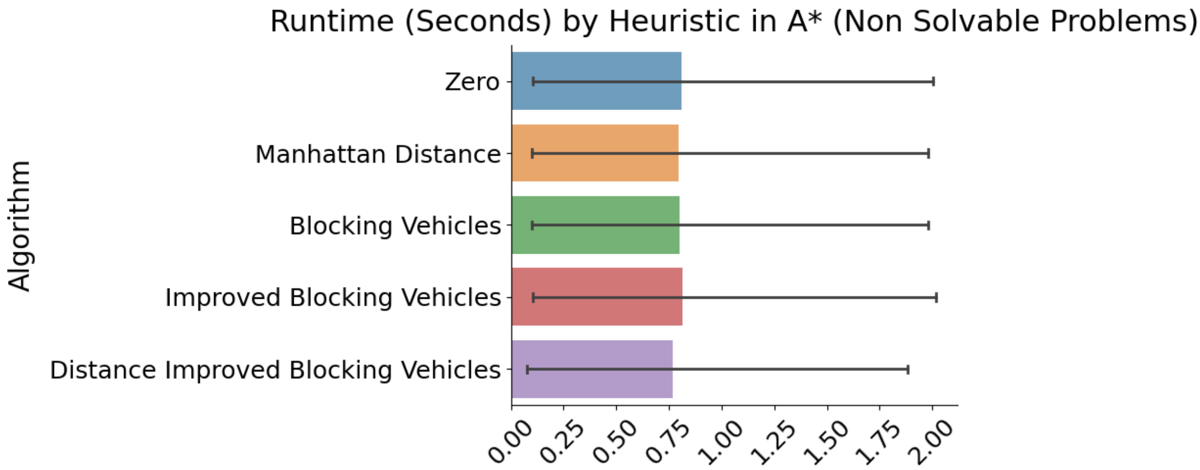

Figure 11: Run time in seconds by each heuristic in A* algorithm for all 4 unsolvable problems. (Blue) Zero heuristic, (Orange) Manhattan Distance heuristic, (Green) Blocking Vehicles heuristic, (Red) Improved Blocking Vehicles heuristic, (Purple) Distance Improved Blocking Vehicles heuristic.

5 Conclusion

We comprehensively evaluated and compared several algorithms (BFS, DFS, A*, IDA*) and heuristics (Zero, Manhattan distance, Blocking vehicles, Improved blocking vehicles, Distance improved blocking vehicles) for solving the classic puzzle game, Rush Hour. As we can see from the results in the previous chapter, our hypothesis was correct. The most efficient algorithm is A*, followed by BFS, DFS and that the poorest results are achieved by IDA* both in node expansion and in run time. Based on heuristic analysis, "distance improved blocking vehicle" is the most effective, and all other heuristic options except zero heuristic are similar.

References

Wikipedia contributors. 2022a. Breadth-first search — Wikipedia, the free encyclopedia. [Online; accessed 4-February-2023].

Wikipedia contributors. 2022b. Iterative deepening a* — Wikipedia, the free encyclopedia. [Online; accessed 4-February-2023].

Wikipedia contributors. 2022c. Rush hour (puzzle) — Wikipedia, the free encyclopedia. [Online; accessed 4-February-2023].

Wikipedia contributors. 2023a. A* search algorithm — Wikipedia, the free encyclopedia. [Online; accessed 4-February-2023].

Wikipedia contributors. 2023b. Depth-first search — Wikipedia, the free encyclopedia. [Online; accessed 4-February-2023].

A Appendix

A.1 Full results table

Problem	Heuristic	Algorithm	Goal Tests	Nodes Expanded	Nodes Evaluated	Solution Cost	Runtime (Seconds)

p1	Zero	BFS	1072.0	7357.0	0.0	16	2.3828
p1	Zero	DFS	1313.0	8809.0	0.0	246	2.9563
p1	Zero	A*	1069.0	7343.0	1075.0	16	2.3574
p1	Manhattan Distance	A*	1021.0	7080.0	1050.0	16	2.2892
p1	Blocking Vehicles	A*	1057.0	7285.0	1069.0	16	2.4997
p1	Improved Blocking Vehicles	A*	1052.0	7257.0	1071.0	16	2.5558
p1	Distance Im- proved Blocking Vehicles	A*	894.0	6271.0	1003.0	16	2.2167
p1	Zero	IDA*	9898.0	69251.0	10975.0	16	22.3448
p1	Manhattan Distance	IDA*	6727.0	47316.0	7802.0	16	15.2501
p1	Blocking Vehicles	IDA*	7986.0	55884.0	9064.0	16	18.3739
p1	Improved Blocking Vehicles	IDA*	7461.0	52478.0	8807.0	16	18.2292
p1	Distance Im- proved Blocking Vehicles	IDA*	4297.0	30510.0	5617.0	16	10.4818
p2	Zero	BFS	2919.0	22540.0	0.0	14	8.4457
p2	Zero	DFS	1657.0	13283.0	0.0	1627	4.6692
p2	Zero	A*	3074.0	23777.0	3579.0	14	9.0742
p2	Manhattan Distance	A*	1658.0	12870.0	2022.0	14	4.8315
p2	Blocking Vehicles	A*	1903.0	14731.0	2264.0	14	5.6513
p2	Improved Blocking Vehicles	A*	1740.0	13470.0	2173.0	14	5.1975
p2	Distance Im- proved Blocking Vehicles	A*	872.0	6745.0	1318.0	14	2.6129
p2	Zero	IDA*	16110.0	125442.0	19707.0	14	45.4084
p2	Manhattan Distance	IDA*	7580.0	59463.0	10017.0	14	21.6121
p2	Blocking Vehicles	IDA*	9085.0	70487.0	11715.0	14	26.2021

p2	Improved Blocking Vehicles	IDA*	8251.0	64012.0	11408.0	14	23.8896
p2	Distance Improved Blocking Vehicles	IDA*	2926.0	22735.0	4923.0	14	8.5166
p3	Zero	BFS	824.0	4837.0	0.0	33	1.4123
p3	Zero	DFS	643.0	3841.0	0.0	180	1.1498
p3	Zero	A*	812.0	4768.0	825.0	33	1.7247
p3	Manhattan Distance	A*	656.0	3813.0	761.0	33	1.357
p3	Blocking Vehicles	A*	776.0	4564.0	816.0	33	1.3929
p3	Improved Blocking Vehicles	A*	770.0	4538.0	818.0	33	1.3817
p3	Distance Improved Blocking Vehicles	A*	477.0	2676.0	601.0	33	0.8468
p3	Zero	IDA*	12233.0	70771.0	13069.0	33	20.2167
p3	Manhattan Distance	IDA*	9576.0	54885.0	10727.0	33	15.7297
p3	Blocking Vehicles	IDA*	10882.0	62628.0	11782.0	33	19.1416
p3	Improved Blocking Vehicles	IDA*	10234.0	58989.0	11248.0	33	18.0391
p3	Distance Improved Blocking Vehicles	IDA*	7770.0	44294.0	8762.0	33	13.4484
p4	Zero	BFS	7.0	18.0	0.0	3	0.0058
p4	Zero	DFS	6.0	15.0	0.0	5	0.0045
p4	Zero	A*	7.0	18.0	10.0	3	0.0061
p4	Manhattan Distance	A*	4.0	9.0	8.0	3	0.0028
p4	Blocking Vehicles	A*	7.0	18.0	10.0	3	0.006
p4	Improved Blocking Vehicles	A*	7.0	18.0	10.0	3	0.0072

p4	Distance Improved Blocking Vehicles	A*	4.0	9.0	8.0	3	0.0033
p4	Zero	IDA*	16.0	41.0	26.0	3	0.0135
p4	Manhattan Distance	IDA*	4.0	9.0	11.0	3	0.0031
p4	Blocking Vehicles	IDA*	16.0	41.0	26.0	3	0.014
p4	Improved Blocking Vehicles	IDA*	16.0	41.0	26.0	3	0.014
p4	Distance Improved Blocking Vehicles	IDA*	4.0	9.0	11.0	3	0.0035
p5	Zero	BFS	2607.0	19331.0	0.0	18	7.0873
p5	Zero	DFS	258.0	1856.0	0.0	245	0.6773
p5	Zero	A*	2648.0	19604.0	2714.0	18	7.2042
p5	Manhattan Distance	A*	2208.0	16569.0	2358.0	18	6.1435
p5	Blocking Vehicles	A*	2284.0	17090.0	2442.0	18	6.4017
p5	Improved Blocking Vehicles	A*	2192.0	16490.0	2407.0	18	6.2143
p5	Distance Improved Blocking Vehicles	A*	1450.0	10971.0	1896.0	18	4.1844
p5	Zero	IDA*	21789.0	163229.0	24505.0	18	60.5985
p5	Manhattan Distance	IDA*	14038.0	105532.0	16539.0	18	39.2335
p5	Blocking Vehicles	IDA*	15168.0	114104.0	18494.0	18	43.4456
p5	Improved Blocking Vehicles	IDA*	12735.0	95825.0	16464.0	18	36.5835
p5	Distance Improved Blocking Vehicles	IDA*	6129.0	45885.0	9221.0	18	17.592
p6	Zero	BFS	2081.0	14303.0	0.0	17	5.3961
p6	Zero	DFS	864.0	5644.0	0.0	677	2.1651

p6	Zero	A*	1991.0	13757.0	2120.0	17	5.0398
p6	Manhattan Distance	A*	1556.0	11084.0	1720.0	17	4.0895
p6	Blocking Vehicles	A*	1662.0	11767.0	1832.0	17	4.4492
p6	Improved Blocking Vehicles	A*	1590.0	11339.0	1807.0	17	4.2873
p6	Distance Im-proved Blocking Vehicles	A*	1061.0	7837.0	1331.0	17	2.9561
p6	Zero	IDA*	17336.0	124989.0	19546.0	17	46.3702
p6	Manhattan Distance	IDA*	10537.0	77458.0	12689.0	17	28.8341
p6	Blocking Vehicles	IDA*	11643.0	85033.0	13864.0	17	32.3781
p6	Improved Blocking Vehicles	IDA*	10681.0	78558.0	13590.0	17	30.0588
p6	Distance Im-proved Blocking Vehicles	IDA*	5116.0	38456.0	7425.0	17	14.7741
p7	Zero	BFS	4434.0	35939.0	0.0	21	12.1688
p7	Zero	DFS	337.0	2425.0	0.0	332	0.7864
p7	Zero	A*	3767.0	30127.0	4616.0	21	9.9461
p7	Manhattan Distance	A*	2083.0	15441.0	2578.0	21	5.1154
p7	Blocking Vehicles	A*	2539.0	19399.0	3293.0	21	6.5992
p7	Improved Blocking Vehicles	A*	2539.0	19399.0	3293.0	21	6.616
p7	Distance Im-proved Blocking Vehicles	A*	1742.0	12669.0	1978.0	21	4.2866
p7	Zero	IDA*	28286.0	216170.0	33574.0	21	72.3435
p7	Manhattan Distance	IDA*	17067.0	126346.0	21028.0	21	42.3853
p7	Blocking Vehicles	IDA*	20385.0	152410.0	25102.0	21	52.8401

p7	Improved Blocking Vehicles	IDA*	20385.0	152410.0	25102.0	21	52.4388
p7	Distance Improved Blocking Vehicles	IDA*	12885.0	94912.0	16033.0	21	32.7049
p8	Zero	BFS	951.0	5535.0	0.0	22	2.4126
p8	Zero	DFS	3251.0	18885.0	0.0	218	8.6068
p8	Zero	A*	951.0	5535.0	951.0	22	2.3326
p8	Manhattan Distance	A*	950.0	5532.0	951.0	22	2.36
p8	Blocking Vehicles	A*	957.0	5562.0	957.0	22	2.3995
p8	Improved Blocking Vehicles	A*	961.0	5591.0	961.0	22	2.4711
p8	Distance Improved Blocking Vehicles	A*	923.0	5418.0	959.0	22	2.3422
p8	Zero	IDA*	11647.0	68751.0	12597.0	22	29.1038
p8	Manhattan Distance	IDA*	8352.0	49073.0	9301.0	22	20.8531
p8	Blocking Vehicles	IDA*	9115.0	53878.0	10124.0	22	23.4227
p8	Improved Blocking Vehicles	IDA*	8549.0	50484.0	9558.0	22	22.0932
p8	Distance Improved Blocking Vehicles	IDA*	5247.0	30813.0	6289.0	22	13.5765
p9	Zero	BFS	631.0	4293.0	0.0	17	1.7054
p9	Zero	DFS	170.0	1145.0	0.0	154	0.4365
p9	Zero	A*	669.0	4558.0	792.0	17	1.7579
p9	Manhattan Distance	A*	348.0	2279.0	465.0	17	0.8775
p9	Blocking Vehicles	A*	410.0	2713.0	538.0	17	1.071
p9	Improved Blocking Vehicles	A*	410.0	2713.0	538.0	17	1.0743

p9	Distance Improved Blocking Vehicles	A*	229.0	1442.0	335.0	17	0.5727
p9	Zero	IDA*	2720.0	17469.0	3522.0	17	6.7978
p9	Manhattan Distance	IDA*	1338.0	8182.0	1917.0	17	3.1915
p9	Blocking Vehicles	IDA*	1652.0	10281.0	2307.0	17	4.1118
p9	Improved Blocking Vehicles	IDA*	1652.0	10281.0	2307.0	17	4.1271
p9	Distance Improved Blocking Vehicles	IDA*	724.0	4196.0	1132.0	17	1.719
p10	Zero	BFS	2138.0	14001.0	0.0	32	5.4881
p10	Zero	DFS	493.0	2912.0	0.0	395	1.1167
p10	Zero	A*	2106.0	13816.0	2251.0	32	5.4124
p10	Manhattan Distance	A*	1689.0	10876.0	1845.0	32	4.477
p10	Blocking Vehicles	A*	1866.0	12133.0	2040.0	32	5.1095
p10	Improved Blocking Vehicles	A*	1828.0	11877.0	2003.0	32	5.1705
p10	Distance Improved Blocking Vehicles	A*	1491.0	9476.0	1592.0	32	4.022
p10	Zero	IDA*	27377.0	174729.0	29772.0	32	72.0877
p10	Manhattan Distance	IDA*	22637.0	143463.0	25232.0	32	56.754
p10	Blocking Vehicles	IDA*	24508.0	155737.0	26737.0	32	63.6184
p10	Improved Blocking Vehicles	IDA*	23645.0	150194.0	25877.0	32	63.3069
p10	Distance Improved Blocking Vehicles	IDA*	19797.0	125180.0	22133.0	32	52.4136
p11	Zero	BFS	849.0	4368.0	0.0	56	1.4593
p11	Zero	DFS	599.0	2938.0	0.0	209	0.9555

p11	Zero	A*	846.0	4352.0	854.0	56	1.4335
p11	Manhattan Distance	A*	825.0	4255.0	840.0	56	1.4523
p11	Blocking Vehicles	A*	844.0	4344.0	853.0	56	1.5156
p11	Improved Blocking Vehicles	A*	845.0	4349.0	854.0	56	1.5537
p11	Distance Im-proved Blocking Vehicles	A*	811.0	4191.0	837.0	56	1.4573
p11	Zero	IDA*	24396.0	127167.0	25256.0	56	42.1246
p11	Manhattan Distance	IDA*	21645.0	112960.0	22508.0	56	37.2606
p11	Blocking Vehicles	IDA*	23098.0	120377.0	23995.0	56	41.2569
p11	Improved Blocking Vehicles	IDA*	22406.0	116767.0	23304.0	56	40.5538
p11	Distance Im-proved Blocking Vehicles	IDA*	19748.0	103103.0	20651.0	56	35.619
p12	Zero	BFS	980.0	5654.0	0.0	33	1.7881
p12	Zero	DFS	200.0	1134.0	0.0	116	0.3574
p12	Zero	A*	1032.0	5971.0	1130.0	33	2.1086
p12	Manhattan Distance	A*	677.0	3782.0	755.0	33	1.3159
p12	Blocking Vehicles	A*	735.0	4133.0	832.0	33	1.4943
p12	Improved Blocking Vehicles	A*	654.0	3655.0	736.0	33	1.2927
p12	Distance Im-proved Blocking Vehicles	A*	523.0	2895.0	583.0	33	1.0904
p12	Zero	IDA*	10681.0	60203.0	11825.0	33	20.8168
p12	Manhattan Distance	IDA*	7646.0	42570.0	8477.0	33	14.1259
p12	Blocking Vehicles	IDA*	8522.0	47528.0	9511.0	33	16.0368

p12	Improved Blocking Vehicles	IDA*	7648.0	42637.0	8537.0	33	14.5617
p12	Distance Improved Blocking Vehicles	IDA*	5673.0	31493.0	6374.0	33	10.7554
p13	Zero	BFS	10502.0	77955.0	0.0	32	31.5384
p13	Zero	DFS	3081.0	20344.0	0.0	1811	8.2884
p13	Zero	A*	10700.0	79668.0	11021.0	32	34.8318
p13	Manhattan Distance	A*	9633.0	71145.0	10100.0	32	30.5498
p13	Blocking Vehicles	A*	9835.0	72622.0	10251.0	32	31.6342
p13	Improved Blocking Vehicles	A*	9835.0	72623.0	10253.0	32	32.1338
p13	Distance Improved Blocking Vehicles	A*	8080.0	59293.0	8949.0	32	25.8423
p13	Zero	IDA*	129105.0	937668.0	140149.0	32	384.722
p13	Manhattan Distance	IDA*	99069.0	713704.0	110303.0	32	284.523
p13	Blocking Vehicles	IDA*	103130.0	742295.0	114762.0	32	301.882
p13	Improved Blocking Vehicles	IDA*	100224.0	721744.0	112287.0	32	295.637
p13	Distance Improved Blocking Vehicles	IDA*	71566.0	508854.0	83287.0	32	208.361
p14	Zero	BFS	14426.0	116330.0	0.0	34	44.8037
p14	Zero	DFS	2529.0	21084.0	0.0	2416	8.0097
p14	Zero	A*	14292.0	114910.0	16349.0	34	46.843
p14	Manhattan Distance	A*	8501.0	67360.0	9723.0	34	28.2664
p14	Blocking Vehicles	A*	10316.0	82318.0	12128.0	34	34.5071
p14	Improved Blocking Vehicles	A*	10316.0	82318.0	12128.0	34	34.0423

p14	Distance Improved Blocking Vehicles	A*	7282.0	57608.0	8487.0	34	23.5376
p14	Zero	IDA*	110054.0	841934.0	126708.0	34	318.809
p14	Manhattan Distance	IDA*	70663.0	527677.0	83423.0	34	200.643
p14	Blocking Vehicles	IDA*	83968.0	631818.0	98529.0	34	245.215
p14	Improved Blocking Vehicles	IDA*	83968.0	631818.0	98529.0	34	245.923
p14	Distance Improved Blocking Vehicles	IDA*	55034.0	402907.0	66215.0	34	157.279
p15	Zero	BFS	527.0	2759.0	0.0	32	1.1898
p15	Zero	DFS	195.0	920.0	0.0	84	0.3876
p15	Zero	A*	526.0	2754.0	531.0	32	1.2505
p15	Manhattan Distance	A*	522.0	2740.0	525.0	32	1.2045
p15	Blocking Vehicles	A*	523.0	2743.0	526.0	32	1.3612
p15	Improved Blocking Vehicles	A*	523.0	2743.0	526.0	32	1.3175
p15	Distance Improved Blocking Vehicles	A*	521.0	2737.0	525.0	32	1.2682
p15	Zero	IDA*	12142.0	65558.0	12673.0	32	27.2164
p15	Manhattan Distance	IDA*	10563.0	57279.0	11149.0	32	23.8073
p15	Blocking Vehicles	IDA*	10675.0	57861.0	11273.0	32	24.6564
p15	Improved Blocking Vehicles	IDA*	10475.0	56829.0	11212.0	32	24.3209
p15	Distance Improved Blocking Vehicles	IDA*	8903.0	48574.0	9665.0	32	20.8092
p16	Zero	BFS	2789.0	18293.0	0.0	41	6.9466
p16	Zero	DFS	311.0	2038.0	0.0	301	0.747

p16	Zero	A*	2635.0	17255.0	2794.0	41	6.3767
p16	Manhattan Distance	A*	2335.0	15319.0	2526.0	41	5.9135
p16	Blocking Vehicles	A*	2411.0	15788.0	2611.0	41	6.1978
p16	Improved Blocking Vehicles	A*	2234.0	14619.0	2441.0	41	5.7193
p16	Distance Improved Blocking Vehicles	A*	1969.0	12932.0	2118.0	41	5.0741
p16	Zero	IDA*	52144.0	343672.0	55093.0	41	124.565
p16	Manhattan Distance	IDA*	45105.0	298328.0	48202.0	41	108.391
p16	Blocking Vehicles	IDA*	47458.0	313072.0	50615.0	41	116.186
p16	Improved Blocking Vehicles	IDA*	44790.0	295719.0	47784.0	41	110.019
p16	Distance Improved Blocking Vehicles	IDA*	38585.0	255673.0	41526.0	41	95.158
p17	Zero	BFS	2162.0	14145.0	0.0	47	5.5241
p17	Zero	DFS	8248.0	54184.0	0.0	301	22.3681
p17	Zero	A*	2172.0	14180.0	2194.0	47	5.6767
p17	Manhattan Distance	A*	2070.0	13667.0	2109.0	47	5.4375
p17	Blocking Vehicles	A*	2131.0	13996.0	2158.0	47	6.0232
p17	Improved Blocking Vehicles	A*	2128.0	13982.0	2157.0	47	5.9551
p17	Distance Improved Blocking Vehicles	A*	2009.0	13360.0	2058.0	47	5.4051
p17	Zero	IDA*	47861.0	315879.0	50058.0	47	119.857
p17	Manhattan Distance	IDA*	39660.0	261794.0	41866.0	47	99.4535
p17	Blocking Vehicles	IDA*	42045.0	277642.0	44635.0	47	108.079

p17	Improved Blocking Vehicles	IDA*	41728.0	275734.0	44507.0	47	107.631
p17	Distance Improved Blocking Vehicles	IDA*	33679.0	222356.0	36410.0	47	86.8522
p18	Zero	BFS	1623.0	9014.0	0.0	60	3.145
p18	Zero	DFS	15757.0	89775.0	0.0	206	32.7316
p18	Zero	A*	1613.0	8963.0	1626.0	60	3.0226
p18	Manhattan Distance	A*	1574.0	8772.0	1601.0	60	2.9564
p18	Blocking Vehicles	A*	1599.0	8891.0	1614.0	60	3.1343
p18	Improved Blocking Vehicles	A*	1598.0	8887.0	1614.0	60	3.1145
p18	Distance Improved Blocking Vehicles	A*	1541.0	8597.0	1577.0	60	3.1346
p18	Zero	IDA*	41385.0	232205.0	43020.0	60	76.5566
p18	Manhattan Distance	IDA*	35990.0	202082.0	37688.0	60	66.7077
p18	Blocking Vehicles	IDA*	39446.0	221381.0	41136.0	60	74.8669
p18	Improved Blocking Vehicles	IDA*	37951.0	212949.0	39647.0	60	72.3865
p18	Distance Improved Blocking Vehicles	IDA*	32591.0	182997.0	34341.0	60	62.2595
p19	Zero	BFS	74.0	460.0	0.0	3	0.1493
p19	Zero	DFS	6.0	31.0	0.0	5	0.0159
p19	Zero	A*	37.0	223.0	97.0	3	0.0781
p19	Manhattan Distance	A*	5.0	26.0	23.0	3	0.0084
p19	Blocking Vehicles	A*	23.0	153.0	84.0	3	0.0828
p19	Improved Blocking Vehicles	A*	23.0	153.0	84.0	3	0.0614

p19	Distance Improved Blocking Vehicles	A*	4.0	21.0	20.0	3	0.0097
p19	Zero	IDA*	72.0	442.0	206.0	3	0.1384
p19	Manhattan Distance	IDA*	6.0	32.0	32.0	3	0.01
p19	Blocking Vehicles	IDA*	38.0	254.0	148.0	3	0.0831
p19	Improved Blocking Vehicles	IDA*	38.0	254.0	148.0	3	0.0831
p19	Distance Improved Blocking Vehicles	IDA*	4.0	21.0	23.0	3	0.0075
p20	Zero	BFS	1862.0	11535.0	0.0	18	3.9485
p20	Zero	DFS	240.0	1407.0	0.0	231	0.4723
p20	Zero	A*	1924.0	11957.0	2273.0	18	4.3045
p20	Manhattan Distance	A*	843.0	4954.0	1140.0	18	1.7975
p20	Blocking Vehicles	A*	1084.0	6582.0	1432.0	18	2.4763
p20	Improved Blocking Vehicles	A*	1074.0	6528.0	1432.0	18	2.4489
p20	Distance Improved Blocking Vehicles	A*	485.0	2756.0	699.0	18	1.0635
p20	Zero	IDA*	9401.0	56451.0	11805.0	18	19.6488
p20	Manhattan Distance	IDA*	3785.0	21553.0	5160.0	18	7.5572
p20	Blocking Vehicles	IDA*	5435.0	31843.0	7364.0	18	11.4594
p20	Improved Blocking Vehicles	IDA*	5357.0	31440.0	7364.0	18	11.3703
p20	Distance Improved Blocking Vehicles	IDA*	1959.0	10863.0	2938.0	18	3.9567
p21	Zero	BFS	273.0	1288.0	0.0	Failed	0.421
p21	Zero	DFS	2806.0	12223.0	0.0	Failed	4.5647

p21	Zero	A*	273.0	1288.0	273.0	Failed	0.4091
p21	Manhattan Distance	A*	273.0	1288.0	273.0	Failed	0.4032
p21	Blocking Vehicles	A*	273.0	1288.0	273.0	Failed	0.4088
p21	Improved Blocking Vehicles	A*	273.0	1288.0	273.0	Failed	0.4083
p21	Distance Improved Blocking Vehicles	A*	273.0	1288.0	273.0	Failed	0.4145
p21	Zero	IDA*	inf	inf	inf	Failed	inf
p21	Manhattan Distance	IDA*	inf	inf	inf	Failed	inf
p21	Blocking Vehicles	IDA*	inf	inf	inf	Failed	inf
p21	Improved Blocking Vehicles	IDA*	inf	inf	inf	Failed	inf
p21	Distance Improved Blocking Vehicles	IDA*	inf	inf	inf	Failed	inf
p22	Zero	BFS	4752.0	34234.0	0.0	46	12.9534
p22	Zero	DFS	139118.0	872902.0	0.0	543	359.719
p22	Zero	A*	4814.0	34650.0	4943.0	46	13.5949
p22	Manhattan Distance	A*	3871.0	27564.0	4234.0	46	10.8144
p22	Blocking Vehicles	A*	4352.0	31299.0	4648.0	46	12.8212
p22	Improved Blocking Vehicles	A*	4353.0	31308.0	4649.0	46	12.7658
p22	Distance Improved Blocking Vehicles	A*	3453.0	24251.0	3877.0	46	10.549
p22	Zero	IDA*	55679.0	373427.0	60661.0	46	142.33
p22	Manhattan Distance	IDA*	40139.0	261419.0	45240.0	46	99.8777
p22	Blocking Vehicles	IDA*	47352.0	312982.0	52732.0	46	122.536

p22	Improved Blocking Vehicles	IDA*	46787.0	309391.0	52225.0	46	121.442
p22	Distance Improved Blocking Vehicles	IDA*	32805.0	209135.0	37748.0	46	82.2225
p23	Zero	BFS	2763.0	15204.0	0.0	49	5.2425
p23	Zero	DFS	376.0	1746.0	0.0	256	0.6137
p23	Zero	A*	2690.0	14742.0	2855.0	49	5.5462
p23	Manhattan Distance	A*	2037.0	11159.0	2224.0	49	4.0736
p23	Blocking Vehicles	A*	2285.0	12523.0	2478.0	49	4.6713
p23	Improved Blocking Vehicles	A*	2250.0	12346.0	2471.0	49	4.6487
p23	Distance Improved Blocking Vehicles	A*	1658.0	9036.0	1871.0	49	3.4702
p23	Zero	IDA*	31160.0	166018.0	34116.0	49	58.0666
p23	Manhattan Distance	IDA*	21732.0	113090.0	24131.0	49	39.7475
p23	Blocking Vehicles	IDA*	24872.0	130414.0	27690.0	49	47.0357
p23	Improved Blocking Vehicles	IDA*	24162.0	126664.0	27095.0	49	45.8865
p23	Distance Improved Blocking Vehicles	IDA*	16295.0	82405.0	18480.0	49	30.0294
p24	Zero	BFS	4269.0	30462.0	0.0	50	10.4193
p24	Zero	DFS	155383.0	1166735.0	0.0	598	429.427
p24	Zero	A*	4238.0	30260.0	4325.0	50	10.8895
p24	Manhattan Distance	A*	4125.0	29559.0	4192.0	50	10.821
p24	Blocking Vehicles	A*	4174.0	29867.0	4262.0	50	11.0132
p24	Improved Blocking Vehicles	A*	4174.0	29867.0	4262.0	50	10.8504

p24	Distance Improved Blocking Vehicles	A*	4044.0	29062.0	4126.0	50	10.5769
p24	Zero	IDA*	145768.0	1088865.0	150123.0	50	374.964
p24	Manhattan Distance	IDA*	133444.0	1002812.0	138677.0	50	346.134
p24	Blocking Vehicles	IDA*	140556.0	1052499.0	145215.0	50	370.331
p24	Improved Blocking Vehicles	IDA*	140556.0	1052499.0	145215.0	50	371.185
p24	Distance Improved Blocking Vehicles	IDA*	128321.0	966978.0	133859.0	50	341.09
p25	Zero	BFS	8865.0	67187.0	0.0	52	27.4252
p25	Zero	DFS	1926.0	13609.0	0.0	1538	5.5072
p25	Zero	A*	8827.0	66959.0	8879.0	52	27.2376
p25	Manhattan Distance	A*	8559.0	65339.0	8685.0	52	26.7716
p25	Blocking Vehicles	A*	8707.0	66277.0	8808.0	52	27.5612
p25	Improved Blocking Vehicles	A*	8697.0	66233.0	8802.0	52	28.0769
p25	Distance Improved Blocking Vehicles	A*	8131.0	62288.0	8436.0	52	27.2396
p25	Zero	IDA*	179207.0	1357459.0	188126.0	52	543.137
p25	Manhattan Distance	IDA*	152067.0	1150097.0	161719.0	52	480.791
p25	Blocking Vehicles	IDA*	157483.0	1190055.0	166525.0	52	527.092
p25	Improved Blocking Vehicles	IDA*	154603.0	1168306.0	163668.0	52	506.497
p25	Distance Improved Blocking Vehicles	IDA*	128048.0	964258.0	137785.0	52	405.865
p26	Zero	BFS	4851.0	33003.0	0.0	49	12.7613
p26	Zero	DFS	1008.0	6669.0	0.0	856	2.5819

p26	Zero	A*	4832.0	32904.0	4852.0	49	13.1911
p26	Manhattan Distance	A*	4617.0	31709.0	4726.0	49	12.4554
p26	Blocking Vehicles	A*	4714.0	32286.0	4806.0	49	13.1959
p26	Improved Blocking Vehicles	A*	4716.0	32320.0	4810.0	49	13.5781
p26	Distance Im- proved Blocking Vehicles	A*	4100.0	28324.0	4315.0	49	11.7178
p26	Zero	IDA*	105547.0	728284.0	110417.0	49	293.571
p26	Manhattan Distance	IDA*	90979.0	628777.0	96724.0	49	254.626
p26	Blocking Vehicles	IDA*	94250.0	651583.0	100182.0	49	268.766
p26	Improved Blocking Vehicles	IDA*	92724.0	641896.0	98701.0	49	264.809
p26	Distance Im- proved Blocking Vehicles	IDA*	78749.0	545486.0	84932.0	49	227.484
p27	Zero	BFS	2827.0	15593.0	0.0	57	5.4767
p27	Zero	DFS	2834.0	15039.0	0.0	638	5.5459
p27	Zero	A*	2922.0	16157.0	3019.0	57	5.9412
p27	Manhattan Distance	A*	2451.0	13455.0	2575.0	57	4.9317
p27	Blocking Vehicles	A*	2602.0	14326.0	2755.0	57	5.6037
p27	Distance Im- proved Blocking Vehicles	A*	2274.0	12544.0	2376.0	57	4.6214
p27	Zero	IDA*	51440.0	278330.0	54467.0	57	103.348
p27	Manhattan Distance	IDA*	41242.0	221378.0	43968.0	57	83.2403
p27	Blocking Vehicles	IDA*	44753.0	241163.0	47963.0	57	93.7743
p27	Improved Blocking Vehicles	IDA*	43161.0	233064.0	46523.0	57	91.9074

p27	Distance Improved Blocking Vehicles	IDA*	34227.0	183030.0	37105.0	57	71.1664
p28	Zero	BFS	94.0	387.0	0.0	Failed	0.1534
p28	Zero	DFS	685.0	3016.0	0.0	Failed	1.2536
p28	Zero	A*	94.0	387.0	94.0	Failed	0.1548
p28	Manhattan Distance	A*	94.0	387.0	94.0	Failed	0.1488
p28	Blocking Vehicles	A*	94.0	387.0	94.0	Failed	0.1531
p28	Improved Blocking Vehicles	A*	94.0	387.0	94.0	Failed	0.1543
p28	Distance Improved Blocking Vehicles	A*	94.0	387.0	94.0	Failed	0.1543
p28	Zero	IDA*	inf	inf	inf	Failed	inf
p28	Manhattan Distance	IDA*	inf	inf	inf	Failed	inf
p28	Blocking Vehicles	IDA*	inf	inf	inf	Failed	inf
p28	Improved Blocking Vehicles	IDA*	inf	inf	inf	Failed	inf
p28	Distance Improved Blocking Vehicles	IDA*	inf	inf	inf	Failed	inf
p29	Zero	BFS	4313.0	28630.0	0.0	54	10.9435
p29	Zero	DFS	2922.0	16684.0	0.0	932	6.7562
p29	Zero	A*	4299.0	28550.0	4314.0	54	10.7976
p29	Manhattan Distance	A*	4295.0	28528.0	4314.0	54	10.8782
p29	Blocking Vehicles	A*	4300.0	28558.0	4319.0	54	11.1083
p29	Improved Blocking Vehicles	A*	4313.0	28633.0	4332.0	54	11.2124
p29	Distance Improved Blocking Vehicles	A*	4279.0	28432.0	4338.0	54	11.1261

p29	Zero	IDA*	123670.0	846379.0	127998.0	54	339.82
p29	Manhattan Distance	IDA*	107866.0	739855.0	112234.0	54	299.574
p29	Blocking Vehicles	IDA*	111128.0	761175.0	116065.0	54	315.135
p29	Improved Blocking Vehicles	IDA*	108713.0	744796.0	113657.0	54	317.555
p29	Distance Im-proved Blocking Vehicles	IDA*	93251.0	639923.0	98276.0	54	263.978
p30	Zero	BFS	1170.0	6334.0	0.0	55	2.2865
p30	Zero	DFS	285.0	1421.0	0.0	244	0.4893
p30	Zero	A*	1169.0	6330.0	1170.0	55	2.4025
p30	Manhattan Distance	A*	1161.0	6293.0	1168.0	55	2.4041
p30	Blocking Vehicles	A*	1169.0	6329.0	1171.0	55	2.4355
p30	Improved Blocking Vehicles	A*	1169.0	6329.0	1171.0	55	2.3968
p30	Distance Im-proved Blocking Vehicles	A*	1138.0	6172.0	1165.0	55	2.3876
p30	Zero	IDA*	25189.0	132067.0	26359.0	55	49.0852
p30	Manhattan Distance	IDA*	22534.0	117997.0	23943.0	55	43.3527
p30	Blocking Vehicles	IDA*	23569.0	123408.0	24795.0	55	45.5014
p30	Improved Blocking Vehicles	IDA*	22986.0	120174.0	24300.0	55	44.9938
p30	Distance Im-proved Blocking Vehicles	IDA*	20313.0	106011.0	21847.0	55	39.6581
p31	Zero	BFS	3930.0	23479.0	0.0	69	8.8077
p31	Zero	DFS	647.0	3655.0	0.0	507	1.391
p31	Zero	A*	3931.0	23481.0	4008.0	69	9.2214
p31	Manhattan Distance	A*	3803.0	22778.0	3896.0	69	8.8953
p31	Blocking Vehicles	A*	3855.0	23034.0	3917.0	69	9.1945

p31	Improved Blocking Vehicles	A*	3854.0	23031.0	3919.0	69	9.1345
p31	Distance Improved Blocking Vehicles	A*	3657.0	21921.0	3763.0	69	8.8528
p31	Zero	IDA*	112982.0	687203.0	117013.0	69	265.203
p31	Manhattan Distance	IDA*	99367.0	603876.0	103372.0	69	229.425
p31	Blocking Vehicles	IDA*	104206.0	632889.0	108279.0	69	240.397
p31	Improved Blocking Vehicles	IDA*	102812.0	624591.0	107272.0	69	236.494
p31	Distance Improved Blocking Vehicles	IDA*	89371.0	542147.0	93739.0	69	207.824
p32	Zero	BFS	616.0	2752.0	0.0	62	1.0318
p32	Zero	DFS	505.0	2137.0	0.0	140	0.8569
p32	Zero	A*	613.0	2737.0	627.0	62	1.0933
p32	Manhattan Distance	A*	579.0	2579.0	607.0	62	1.009
p32	Blocking Vehicles	A*	599.0	2666.0	612.0	62	1.0629
p32	Improved Blocking Vehicles	A*	599.0	2666.0	612.0	62	1.1041
p32	Distance Improved Blocking Vehicles	A*	523.0	2303.0	567.0	62	0.9443
p32	Zero	IDA*	20165.0	90326.0	20794.0	62	33.6455
p32	Manhattan Distance	IDA*	18058.0	80790.0	18704.0	62	30.272
p32	Blocking Vehicles	IDA*	18829.0	84287.0	19512.0	62	32.2171
p32	Improved Blocking Vehicles	IDA*	18515.0	82993.0	19209.0	62	31.8892

p32	Distance Improved Blocking Vehicles	IDA*	16482.0	73879.0	17178.0	62	28.5601
p33	Zero	BFS	4037.0	25427.0	0.0	77	9.9924
p33	Zero	DFS	699.0	3942.0	0.0	531	1.5271
p33	Zero	A*	3956.0	24935.0	4072.0	77	10.1257
p33	Manhattan Distance	A*	3484.0	21925.0	3628.0	77	8.8509
p33	Blocking Vehicles	A*	3639.0	22932.0	3789.0	77	9.6319
p33	Improved Blocking Vehicles	A*	3488.0	21922.0	3656.0	77	9.205
p33	Distance Improved Blocking Vehicles	A*	2784.0	17047.0	3039.0	77	7.1194
p33	Zero	IDA*	105259.0	651314.0	109413.0	77	256.639
p33	Manhattan Distance	IDA*	90953.0	559150.0	94761.0	77	222.521
p33	Blocking Vehicles	IDA*	93290.0	572877.0	97585.0	77	235.46
p33	Improved Blocking Vehicles	IDA*	90089.0	552402.0	94483.0	77	235.03
p33	Distance Improved Blocking Vehicles	IDA*	77840.0	473423.0	81645.0	77	192.965
p34	Zero	BFS	4408.0	28182.0	0.0	71	10.9399
p34	Zero	DFS	24292.0	160438.0	0.0	764	66.611
p34	Zero	A*	4410.0	28193.0	4421.0	71	11.4238
p34	Manhattan Distance	A*	4380.0	28056.0	4408.0	71	11.3758
p34	Blocking Vehicles	A*	4396.0	28137.0	4416.0	71	11.5995
p34	Improved Blocking Vehicles	A*	4401.0	28177.0	4422.0	71	11.5814
p34	Distance Improved Blocking Vehicles	A*	4322.0	27761.0	4401.0	71	11.5135

p34	Zero	IDA*	138844.0	861432.0	143272.0	71	340.825
p34	Manhattan Distance	IDA*	123825.0	764089.0	128509.0	71	305.285
p34	Blocking Vehicles	IDA*	126603.0	781210.0	131388.0	71	322.394
p34	Improved Blocking Vehicles	IDA*	125395.0	774217.0	130569.0	71	319.014
p34	Distance Im-proved Blocking Vehicles	IDA*	110577.0	677808.0	115996.0	71	279.304
p35	Zero	BFS	3908.0	23020.0	0.0	75	8.5397
p35	Zero	DFS	6137.0	30606.0	0.0	818	12.2986
p35	Zero	A*	3901.0	22963.0	3970.0	75	8.7233
p35	Manhattan Distance	A*	3855.0	22712.0	3912.0	75	8.6268
p35	Blocking Vehicles	A*	3844.0	22643.0	3891.0	75	8.8463
p35	Improved Blocking Vehicles	A*	3828.0	22571.0	3872.0	75	9.0137
p35	Distance Im-proved Blocking Vehicles	A*	3819.0	22509.0	3856.0	75	8.9251
p35	Zero	IDA*	158090.0	917246.0	162079.0	75	350.429
p35	Manhattan Distance	IDA*	143818.0	831606.0	147744.0	75	317.64
p35	Blocking Vehicles	IDA*	147479.0	852985.0	152092.0	75	340.754
p35	Improved Blocking Vehicles	IDA*	143539.0	830055.0	148287.0	75	325.954
p35	Distance Im-proved Blocking Vehicles	IDA*	129599.0	746051.0	134319.0	75	295.697
p36	Zero	BFS	930.0	6438.0	0.0	Failed	2.4874
p36	Zero	DFS	33479.0	229131.0	0.0	Failed	95.3851
p36	Zero	A*	930.0	6438.0	930.0	Failed	2.6173
p36	Manhattan Distance	A*	930.0	6438.0	930.0	Failed	2.5886
p36	Blocking Vehicles	A*	930.0	6438.0	930.0	Failed	2.588

p36	Improved Blocking Vehicles	A*	930.0	6438.0	930.0	Failed	2.6408
p36	Distance Improved Blocking Vehicles	A*	930.0	6438.0	0.0	Failed	2.4609
p36	Zero	IDA*	inf	inf	inf	Failed	inf
p36	Manhattan Distance	IDA*	inf	inf	inf	Failed	inf
p36	Blocking Vehicles	IDA*	inf	inf	inf	Failed	inf
p36	Improved Blocking Vehicles	IDA*	inf	inf	inf	Failed	inf
p36	Distance Improved Blocking Vehicles	IDA*	inf	inf	inf	Failed	inf
p37	Zero	BFS	1951.0	12507.0	0.0	65	5.2264
p37	Zero	DFS	9721.0	63900.0	0.0	402	27.4135
p37	Zero	A*	1950.0	12501.0	1955.0	65	5.2535
p37	Manhattan Distance	A*	1936.0	12429.0	1950.0	65	5.235
p37	Blocking Vehicles	A*	1945.0	12483.0	1951.0	65	5.4376
p37	Improved Blocking Vehicles	A*	1949.0	12503.0	1955.0	65	5.4124
p37	Distance Improved Blocking Vehicles	A*	1906.0	12231.0	1950.0	65	5.3043
p37	Zero	IDA*	42010.0	252544.0	43966.0	65	104.702
p37	Manhattan Distance	IDA*	35850.0	212664.0	37818.0	65	87.2393
p37	Blocking Vehicles	IDA*	36694.0	217997.0	38670.0	65	91.756
p37	Improved Blocking Vehicles	IDA*	35975.0	214084.0	37995.0	65	90.5507

p37	Distance Improved Blocking Vehicles	IDA*	29940.0	174734.0	31996.0	65	74.3291
p38	Zero	BFS	3942.0	22128.0	0.0	77	7.9613
p38	Zero	DFS	4614.0	22720.0	0.0	743	8.898
p38	Zero	A*	3913.0	21869.0	4028.0	77	8.4854
p38	Manhattan Distance	A*	3402.0	18881.0	3562.0	77	7.4364
p38	Blocking Vehicles	A*	3610.0	20189.0	3784.0	77	8.0159
p38	Improved Blocking Vehicles	A*	3605.0	20161.0	3785.0	77	8.0111
p38	Distance Improved Blocking Vehicles	A*	3181.0	17597.0	3352.0	77	6.7948
p38	Zero	IDA*	86400.0	479862.0	90501.0	77	181.458
p38	Manhattan Distance	IDA*	73353.0	405050.0	77109.0	77	154.233
p38	Blocking Vehicles	IDA*	77854.0	430749.0	82097.0	77	172.688
p38	Improved Blocking Vehicles	IDA*	76772.0	424638.0	81207.0	77	167.715
p38	Distance Improved Blocking Vehicles	IDA*	64769.0	355783.0	68700.0	77	142.498
p39	Zero	BFS	3892.0	20864.0	0.0	82	8.2664
p39	Zero	DFS	27798.0	170406.0	0.0	732	71.6572
p39	Zero	A*	3875.0	20747.0	4000.0	82	8.0677
p39	Manhattan Distance	A*	3744.0	19991.0	3861.0	82	7.7473
p39	Blocking Vehicles	A*	3766.0	20042.0	3860.0	82	8.0061
p39	Improved Blocking Vehicles	A*	3762.0	20018.0	3856.0	82	8.0034
p39	Distance Improved Blocking Vehicles	A*	3610.0	19196.0	3696.0	82	7.7019

p39	Zero	IDA*	163747.0	897052.0	167806.0	82	362.94
p39	Manhattan Distance	IDA*	152456.0	836537.0	156482.0	82	337.77
p39	Blocking Vehicles	IDA*	156412.0	856845.0	160550.0	82	355.845
p39	Improved Blocking Vehicles	IDA*	154848.0	848362.0	159206.0	82	353.438
p39	Distance Im-proved Blocking Vehicles	IDA*	143833.0	789269.0	148143.0	82	326.86
p40	Zero	BFS	30.0	118.0	0.0	Failed	0.0499
p40	Zero	DFS	56.0	222.0	0.0	Failed	0.0926
p40	Zero	A*	30.0	118.0	30.0	Failed	0.0508
p40	Manhattan Distance	A*	30.0	118.0	30.0	Failed	0.0491
p40	Blocking Vehicles	A*	30.0	118.0	30.0	Failed	0.05
p40	Improved Blocking Vehicles	A*	30.0	118.0	30.0	Failed	0.051
p40	Distance Im-proved Blocking Vehicles	A*	30.0	118.0	30.0	Failed	0.0498
p40	Zero	IDA*	inf	inf	inf	Failed	inf
p40	Manhattan Distance	IDA*	inf	inf	inf	Failed	inf
p40	Blocking Vehicles	IDA*	inf	inf	inf	Failed	inf
p40	Improved Blocking Vehicles	IDA*	inf	inf	inf	Failed	inf
p40	Distance Im-proved Blocking Vehicles	IDA*	inf	inf	inf	Failed	inf

A.2 Code

Full code can be found in our GitHub repository https://github.com/leorrose/AI-Rush-Hour but we added the crucial files here.

A.2.1 Rush hour board

```python
""" This module contains RushHourBoard class for rush hour game.

"""

from __future__ import annotations
import numpy as np
from rush_hour.vehicle import Vehicle
from typing import Tuple, List, Generator
import copy

# Set the goal vehicle
GOAL_VEHICLE_SYMBOL = 'X'
GOAL_VEHICLE = Vehicle(GOAL_VEHICLE_SYMBOL, 4, 2, 'H')


class RushHourBoard:
  """A class used to represent a rush hour board"""

  def __init__(self, vehicles: List[Vehicle]) -> None:
    """RushHourBoard initializer

    Args:
        vehicles (List[Vehicle]): List of vehicles in board.
    """
    self._vehicles = vehicles

  def __eq__(self, __o: object) -> bool:
    # Test if object is an instance of RushHourBoard
    if not isinstance(__o, RushHourBoard):
      return False
    # Check if object is equal
    sorted_vehicles = sorted(self.vehicles, key=lambda x: x.symbol)
    sorted_vehicles_o = sorted(__o.vehicles, key=lambda x: x.symbol)
    return sorted_vehicles == sorted_vehicles_o

  def __repr__(self) -> str:
    return self.get_board().__repr__()

  def __str__(self) -> str:
    return self.get_board().__str__()

  def __hash__(self) -> int:
    return hash(self.__repr__())

  @property
  def vehicles(self) -> List[Vehicle]:
    """Rush hour board vehicles"""
```

```python
        return self._vehicles

    def _get_empty_board(self) -> np.array:
        """Method to get empty rush hour board.

        Returns:
            np.array: Empty rush hour board
        """
        return np.asarray(
            [
                [' ', ' ', ' ', ' ', ' ', ' '],
                [' ', ' ', ' ', ' ', ' ', ' '],
                [' ', ' ', ' ', ' ', ' ', ' '],
                [' ', ' ', ' ', ' ', ' ', ' '],
                [' ', ' ', ' ', ' ', ' ', ' '],
                [' ', ' ', ' ', ' ', ' ', ' '],
            ]
        )

    def get_board(self) -> np.array:
        """Method to get current rush hour board.

        Returns:
            np.array: Rush hour board.
        """
        # Get empty board
        board = self._get_empty_board()
        # Fill board with vehicles
        for vehicle in self.vehicles:
            board[vehicle.get_location_indexes()] = vehicle.symbol
        return board

    def get_next_possible_states(
            self
    ) -> Generator[Tuple[Tuple[str, str], RushHourBoard], None, None]:
        """Method to get all possible next states from current state.

        Yields:
             Generator[Tuple[Tuple[str, str]], RushHourBoard]: A generator of
                actions and next possible states.
        """
        # Get current board
        board = self.get_board()
        # Go over all vehicles
        for vehicle_idx in range(0, len(self.vehicles)):
            # Get the vehicle
            vehicle = self.vehicles[vehicle_idx]
            # Move left or right
```

```python
        if vehicle.orientation == 'H':
          # Check left position is legal and empty
          if vehicle.can_move_vehicle("left") and board[vehicle.y,
                                                        vehicle.x - 1] == ' ':
            # Move vehicle in next state to not affect current state
            new_vehicles = copy.deepcopy(self.vehicles)
            new_vehicles[vehicle_idx].move_vehicle("left")
            yield (vehicle.symbol, "left"), RushHourBoard(new_vehicles)
          # Check left position is legal and empty
          if vehicle.can_move_vehicle("right") and board[vehicle.y,
                                                         vehicle.x_end +
                                                         1] == ' ':
            # Move vehicle in next state to not affect current state
            new_vehicles = copy.deepcopy(self.vehicles)
            new_vehicles[vehicle_idx].move_vehicle("right")
            yield (vehicle.symbol, "right"), RushHourBoard(new_vehicles)
        # Move down or up
        else:
          # Check up position is legal and empty
          if vehicle.can_move_vehicle("up") and board[vehicle.y - 1,
                                                      vehicle.x] == ' ':
            # Move vehicle in next state to not affect current state
            new_vehicles = copy.deepcopy(self.vehicles)
            new_vehicles[vehicle_idx].move_vehicle("up")
            yield (vehicle.symbol, "up"), RushHourBoard(new_vehicles)
          # Check down position is legal and empty
          if vehicle.can_move_vehicle("down") and board[vehicle.y_end + 1,
                                                        vehicle.x] == ' ':
            # Move vehicle in next state to not affect current state
            new_vehicles = copy.deepcopy(self.vehicles)
            new_vehicles[vehicle_idx].move_vehicle("down")
            yield (vehicle.symbol, "down"), RushHourBoard(new_vehicles)

  def get_distance_to_exit(self) -> int:
    """Method to get distance of red car to goal car

    Returns:
        int: Distance of red car to goal car
    """
    # Find the red car
    red_car = None
    for vehicle in self.vehicles:
      if vehicle.symbol == GOAL_VEHICLE_SYMBOL:
        red_car = vehicle
        break

    # Get distance
    distance = abs(red_car.x - GOAL_VEHICLE.x) + abs(red_car.y - GOAL_VEHICLE.y)
```

```python
        return distance

    def get_num_blocking_vehicles(self) -> int:
        """Method to get number of vehicle blocking the red car

        Returns:
            int: Number of vehicle blocking the red car.

        """
        # Get current board
        board = self.get_board()
        # Counter for number of blocking cars
        num = 0
        # Loop from exit until we get to the red car
        for i in range(5, -1, -1):
          # Get cell content
          cell = board[2, i]
          # Return number of blocking vehicle
          if cell == "X":
            break
          # No blocking vehicle to add
          elif cell == " ":
            continue
          # Blocking vehicle to add
          else:
            num += 1
        return num

    def get_improved_num_blocking_vehicles(self) -> int:
        """Method to get number of vehicle blocking the red car and check if these
            vehicles are also blocked.

        Returns:
            int: Number of vehicle blocking the red car and if a vehicle is blocked
                too than we count it as two instead of 1.

        """
        # Get current board
        board = self.get_board()
        # Counter for number of blocking vehicle
        num = 0
        # Define set of vehicles blocking
        vehicles_blocking = set()
        # Loop from exit until we get to the red car
        for i in range(5, -1, -1):
          # Get cell content
          cell = board[2, i]
          # Return number of blocking vehicle
```

```
        if cell == "X":
          break
        # No blocking vehicle to add
        elif cell == " ":
          continue
        # Blocking car to add
        else:
          # Add vehicle symbol
          vehicles_blocking.add(cell)
          num += 1

      # Loop over all blocking vehicle
      for vehicle in self.vehicles:
        if vehicle.symbol in vehicles_blocking:
          if (not vehicle.can_move_vehicle("up")
             ) and (not vehicle.can_move_vehicle("up")):
            num += 1
      return num

    def is_solved(self) -> bool:
      """Method to check if board is solved

      Returns:
          bool: board is solved indicator.
      """
      return GOAL_VEHICLE in self.vehicles
```

A.2.2 Rush hour vehicle

```
"""This module contains Vehicle class for rush hour game.

"""

from typing import List, Tuple

CARS = ('X', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K')
TRUCKS = ('O', 'P', 'Q', 'R')


class Vehicle:
  """A class used to represent a Vehicle"""

  def __init__(self, symbol: str, x: int, y: int, orientation: str) -> None:
    """Vehicle initializer

    Args:
        symbol (str): Vehicle symbol. should be in ('X', 'A', 'B', 'C', 'D',
            'E', 'F', 'G', 'H', 'I', 'J', 'K') for cars and in ('O', 'P', 'Q',
            'R') for trucks.
        x (int): Vehicle x coordinate. should be between 0-5.
```

```python
            y (int): Vehicle y coordinate. should be between 0-5.
            orientation (str): Vehicle orientation. should be between "V" or "H".

        """
        # Validate symbol
        if (symbol not in CARS) and (symbol not in TRUCKS):
            raise ValueError(
                f"symbol most be in {CARS} for cars and in {TRUCKS} for trucks. "
            )

        # Validate orientation
        if orientation != 'H' and orientation != 'V':
            raise ValueError("orientation most be V or H")

        # Assign properties
        self._symbol = symbol
        self._x = x
        self._y = y
        self._orientation = orientation
        self._length = 2 if symbol in CARS else 3
        self._x_end = (
            self._x if self._orientation == "V" else self._x + (self._length - 1)
        )
        self._y_end = (
            self._y if self._orientation == "H" else self._y + (self._length - 1)
        )

    def __eq__(self, __o: object) -> bool:
        # Test if object is an instance of Vehicle
        if not isinstance(__o, Vehicle):
            return False
        # Check if object is equal
        return (
            self.symbol == __o.symbol and self.x == __o.x and self.y == __o.y and
            self.orientation == __o.orientation
        )

    def __ne__(self, __o: object) -> bool:
        # Check if object is not equal
        return not self.__eq__(__o)

    def __repr__(self) -> str:
        return f"Vehicle({self.symbol}, {self.x}, {self.y}, {self.orientation})"

    def __hash__(self) -> int:
        return hash(self.__repr__())

    @property
```

```python
def symbol(self) -> int:
  """Vehicle symbol. should be in ('X', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
      'H', 'I', 'J', 'K') for cars and in ('O', 'P', 'Q', 'R') for trucks."""
  return self._symbol

@property
def x(self) -> int:
  """Vehicle x coordinate. should be between 0-5."""
  return self._x

@property
def x_end(self) -> int:
  """Vehicle x end coordinate. should be between 0-5."""
  return self._x_end

@property
def y(self) -> int:
  """Vehicle y coordinate. should be between 0-5"""
  return self._y

@property
def y_end(self) -> int:
  """Vehicle y end coordinate. should be between 0-5."""
  return self._y_end

@property
def orientation(self) -> str:
  """Vehicle orientation. should be between "V" or "H\""""
  return self._orientation

@property
def length(self) -> str:
  """Vehicle length. should be between 2 for car and 3 for truck"""
  return self._length

@symbol.setter
def symbol(self, value) -> None:
  raise AttributeError("Can set attribute only on creation.")

@x.setter
def x(self, value) -> None:
  # Get x end value
  x_end = value if self._orientation == "V" else value + (self._length - 1)
  # Validate x and x_end
  if (value < 0) or (value > 5):
    raise ValueError("x most be between 0-5")
  if (x_end < 0) or (x_end > 5):
    raise ValueError("x_end most be between 0-5")
```

```python
        # Assign property
        self._x = value
        self._x_end = x_end

    @x_end.setter
    def x_end(self, value) -> None:
        raise AttributeError("x_end is decided by the value of y")

    @y.setter
    def y(self, value) -> None:
        # Get y end value
        y_end = value if self._orientation == "H" else value + (self._length - 1)
        # Validate y and y_end
        if (value < 0) or (value > 5):
          raise ValueError("y most be between 0-5")
        if (y_end < 0) or (y_end > 5):
          raise ValueError("y_end most be between 0-5")
        # Assign property
        self._y = value
        self._y_end = y_end

    @y_end.setter
    def y_end(self, value) -> None:
        raise AttributeError("y_end is decided by the value of y")

    @orientation.setter
    def orientation(self, value) -> None:
        raise AttributeError("Can set attribute only on creation.")

    @length.setter
    def length(self, value) -> None:
        raise AttributeError("length is decided by the value of symbol")

    @symbol.deleter
    def symbol(self) -> None:
        del self._symbol

    @x.deleter
    def x(self) -> None:
        del self._x

    @x_end.deleter
    def x_end(self) -> None:
        del self._x_end

    @y.deleter
    def y(self) -> None:
        del self._y
```

```python
@y_end.deleter
def y_end(self) -> None:
  del self._y_end

@orientation.deleter
def orientation(self) -> None:
  del self._orientation

@length.deleter
def length(self) -> None:
  del self._length

def move_vehicle(self, direction: str) -> None:
  """Method to move vehicle.

  Args:
      direction (str): Direction to move. should be left, right, down or up.

  Raises:
      ValueError: If direction is incorrect or invalid direction.

  """
  if direction == "left":
    if self.orientation != "H":
      ValueError("Cannot move vertical vehicle right")
    self.x -= 1
  elif direction == "right":
    if self.orientation != "H":
      ValueError("Cannot move vertical vehicle right")
    self.x += 1
  elif direction == "down":
    if self.orientation != "V":
      ValueError("Cannot move vertical vehicle right")
    self.y += 1
  elif direction == "up":
    if self.orientation != "V":
      ValueError("Cannot move vertical vehicle right")
    self.y -= 1
  else:
    raise ValueError("Invalid direction")

def can_move_vehicle(self, direction: str) -> bool:
  opposite_direction = {
      "left": "right",
      "right": "left",
      "down": "up",
      "up": "down"
```

```
    }
    opposite_direction = opposite_direction[direction]
    try:
      self.move_vehicle(direction)
      self.move_vehicle(opposite_direction)
      return True
    except ValueError:
      return False

  def get_location_indexes(self) -> Tuple[List[int], List[int]]:
    """Method to get vehicle indexes on grid

    Returns:
        Tuple[List[int], List[int]]: y indexes, x indexes
    """
    if self.orientation == "H":
      return [self.y] * self._length, list(range(self.x, self.x_end + 1))
    return list(range(self.y, self.y_end + 1)), [self.x] * self._length
```

A.2.3 Rush hour problems (heuristics)

```
""" This module contains rush hour problems. """

from py_search.base import Problem, Node
from typing import Generator


class ZeroHeuristic(Problem):
  """A class used to represent a zero heuristic problem"""

  def successors(self, node: Node) -> Generator[Node, None, None]:
    """Method to Computes successors.

    Args:
        node (Node): Node to computes successors.

    Yields:
         Generator[Node, None, None]: Successors.
    """
    for action, new_node in node.state.get_next_possible_states():
      path_cost = node.cost() + 1
      yield Node(new_node, node, action, path_cost)

  def heuristic(self, node: Node) -> int:
    """Method to get heuristic.

    Args:
        node (Node):  Node to compute heuristic.

    Returns:
```

```python
            int: Heuristic value.
        """
        return 0

    def node_value(self, node: Node):
        """Method used to compute the value of a node.

        Args:
            node (Node): Node to compute value.

        Returns:
            _type_: Value of a node.
        """
        return node.cost() + self.heuristic(node)

    def goal_test(self, state_node: Node, goal_node: Node = None) -> bool:
        """Method to test of whether a complete assignment has been reached.

        Args:
            state_node (Node): Current state node.
            goal_node (Node, optional): Goal node. Defaults to None.

        Returns:
            bool: Complete assignment has been reached or not.
        """
        return state_node.state.is_solved()


class ManhattanDistanceHeuristic(ZeroHeuristic):
    """A class used to represent a manhattan distance heuristic problem"""

    def heuristic(self, node: Node):
        """Method to get heuristic.

        Args:
            node (Node):  Node to compute heuristic.

        Returns:
            int: Heuristic value.
        """
        return node.state.get_distance_to_exit()


class BlockingVehiclesHeuristic(ZeroHeuristic):
    """A class used to represent a blocking vehicles heuristic problem"""

    def heuristic(self, node: Node):
        """Method to get heuristic.
```

```
        Args:
            node (Node):  Node to compute heuristic.

        Returns:
            int: Heuristic value.
        """
        return node.state.get_num_blocking_vehicles()


class ImprovedBlockingVehiclesHeuristic(ZeroHeuristic):
    """A class used to represent a improved blocking vehicles heuristic problem"""

    def heuristic(self, node: Node):
        """Method to get heuristic.

        Args:
            node (Node):  Node to compute heuristic.

        Returns:
            int: Heuristic value.
        """
        return node.state.get_improved_num_blocking_vehicles()


class DistanceImprovedBlockingVehiclesHeuristic(ZeroHeuristic):
    """A class used to represent a manhattan distance with improved blocking
        vehicles heuristic problem"""

    def heuristic(self, node: Node):
        """Method to get heuristic.

        Args:
            node (Node):  Node to compute heuristic.

        Returns:
            int: Heuristic value.
        """
        return (
            node.state.get_improved_num_blocking_vehicles() +
            node.state.get_distance_to_exit()
        )
```

A.2.4 Main code to run and get results

```
""" This module contains the main code to run.

"""

import os
```

```python
from rush_hour.vehicle import Vehicle
from rush_hour.board import RushHourBoard
from rush_hour.problems import (
    ZeroHeuristic, ManhattanDistanceHeuristic, BlockingVehiclesHeuristic,
    ImprovedBlockingVehiclesHeuristic, DistanceImprovedBlockingVehiclesHeuristic
)
from py_search.uninformed import (breadth_first_search, depth_first_search)
from py_search.informed import (
    best_first_search, iterative_deepening_best_first_search
)
from py_search.utils import compare_searches

# Get boards path
DIR_PATH = os.path.abspath(os.path.dirname(__file__))
BOARDS_PATH = os.path.join(DIR_PATH, "boards")

if __name__ == "__main__":
  # Loop over each board
  for board_number in os.listdir(BOARDS_PATH):
    # Open board file
    with open(os.path.join(BOARDS_PATH, board_number), 'r') as f:
      # Get each vehicle in board
      vehicles = [
          Vehicle(symbol, int(x), int(y), orientation)
          for symbol, x, y, orientation in f.read().splitlines()
      ]
      # Create board
      board = RushHourBoard(vehicles)

      # print board number
      print(f"\n\n{board_number}")

      # Run BFS and DFS
      compare_searches(
          problems=[
              ZeroHeuristic(initial=board),
          ], searches=[breadth_first_search, depth_first_search]
      )

      # Run A* & IDA*
      compare_searches(
          problems=[
              ZeroHeuristic(initial=board),
              ManhattanDistanceHeuristic(initial=board),
              BlockingVehiclesHeuristic(initial=board),
              ImprovedBlockingVehiclesHeuristic(initial=board),
              DistanceImprovedBlockingVehiclesHeuristic(initial=board),
          ],
```

```
    searches=[best_first_search, iterative_deepening_best_first_search]
)
```