# The Analysis of A* Search Algorithms and Heuristics Through Implementations of Rush Hour Search

Wing Har

Computer Science Department – CUNY College of Staten Island

CSC480 [23024] - Artificial Intelligence

**Abstract**

In this paper, we analyze A* search algorithms and heuristics through implementations of the Rush Hour game. Through the experiment, 40 comparisons were made, consisting of 40 typical Rush Hour puzzle boards, and the time it took for the machine to resolve to an answer as calculated and recorded. In this experiment, greedy best-first search was conducted, as well as various situations involving an A* heuristic algorithm. The general measurement to keep track of when it comes to comparing the algorithms is how many steps is taken to reach the eluded conclusion regarding time complexity and space complexity. The number of steps, n, is taken to result in a solution to each problem will allow us to measure the time complexity – how long it takes to come up with a solution, as well as the amount of space is needed to solve it. After the conduction of the experiments, it was shown that for various test experiments, the A* heuristics held far superior results with the greedy best-first-search taking the lead in some cases in relation to how fast a puzzle could be solved. The ideology was prevalent from the beginning due to the definitive properties of both types of algorithms and was further solidified with evidence by the data collected. It is also noted that the correlation between time and space complexity is direct, and throughout the experiments the primary form of measurements was the time elapsed alongside the number of steps taken by the program. This translates over to a comparison of space complexity.

**Keywords**

Time Complexity, Space Complexity, A* Algorithm, Heuristics, Uniform Cost Search, Rush Hour

**Introduction to Heuristics**

What exactly is a heuristic[1]? In computing, a heuristic is the process of attaining a solution through trial and error. This is widely used in the world of Machine Learning and Artificial Intelligence where the machine is given a task or a goal state, and proceeds to work towards a path to the solution through trial and error. It is a technique that was designed for solving problems quickly when the more classical methods are deemed too slow or fail to find a solution. Although algorithms that implement heuristics sacrifice optimality, accuracy, precision, or completeness for speed, it will arrive at an answer relatively quick compared to other methods which work to arrive at the same result. Incidentally, an article published in 1985 by Romanycia and Pelletier address the question of *what is a heuristic?* In the article, the history, and various viewpoints of what a heuristic is comes to fruition amongst the various members of the AI researchers committee, as well as previous workers of Artificial Intelligence. With varying opinions and digressions on what a heuristic is, this conclusion was eventually conceived:

*"We would like now to distinguish more carefully the different (but interrelated) "dimensions of meaning" that the concept embodies. We can distinguish four dimensions along which various researchers have judged whether a process if heuristic: uncertainty of outcome, basis in incomplete knowledge, improvement of performance, and guidance of decision making.*" Romanycia, Marc H.J; Pelletier, Francis J. "What is a heuristic?" *Computational Intelligence*. Issue (1985): p. 51. Print. Accessed on March 14, 2021.

This quote from the article by Romanycia and Pelletier indicate that a heuristic has four characteristics which various researchers were able to ascertain in distinguishing whether a program was heuristic or not. A program must have an uncertainty of outcome, a basis in incomplete knowledge, an improvement of performance over various trials, and a guidance of decision making which allows for it to come up with the fastest possible route to a solution.

[1] In computing, a heuristic is the process of attaining a solution through trial and error.

**Introduction to Algorithms**

What exactly is an algorithm[2]? According to Merriam Webster, an algorithm is *a procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation*. Algorithms are, in short, a methodology in computing for the machine to find a solution to a problem which features a finite number of steps. Some of the largest programs which utilize algorithms involve search engines such as Google and Bing. Algorithms may be described as algorithmic; guaranteed and complete. As stated by Romanycia and Pelletier,

*"'Algorithm' has many meanings, although it is doubtful that the ambiguity has caused any of the disagreements over definition. If we define algorithm as merely "a set of [formally defined and uniquely interpreted] rules which tell us, moment to moment, precisely how to behave" (Minsky 1968, p. 106), then any procedure for making decisions is algorithmic, and hence all heuristics implemented on computer or otherwise strictly formulated, are algorithmic… However, when 'heuristic' has been considered opposed to 'algorithm', 'algorithm' has always had a much stronger sense which included an element of guarantee about finding a solution."* Romanycia, Marc H.J; Pelletier, Francis J. "What is a heuristic?" *Computational Intelligence*. Issue (1985): p. 51. Print. Accessed on March 14, 2021.

All heuristics can be counted as algorithms as both demand a level of procedural problem solving in a finite case. However, what heuristics lack that algorithms have is that algorithms have additional elements of guarantee. Algorithms guarantee finding a solution, whereas heuristics attempt for fastest path through sacrifice of other properties such as optimality, accuracy, precision, or completeness. Heuristics are utilized when other methods take too long whereas algorithms are utilized for find an accurate and guaranteed answer.

[2] An algorithm is a procedure for solving a mathematical problem in a finite number of steps which frequently involves the repetition of an operation.

**Time Complexity vs. Space Complexity**

Time complexity is a fundamental measurement of the "success" of a program in computing. The lower the time complexity, the better a program is. Time complexity measures how many times each statement in a program is executed. Time complexity is used to describe the amount of computer time is required for the execution of an algorithm. With time complexity, the unit of measurement is typically in **Big-O Notation**[3], such as *O(n), O(nlogn),* $O(n^{\alpha})$**,** $O(2^n)$ and so on. With time complexity, there are various indicators of success as well as how to measure such success. Time complexity generally features multiple cases, such as the worst-case scenario, which describes the maximum number of steps required for an algorithm to execute successfully, the average-case scenario which describes the average of a given input size for an algorithm, and best-case scenario which describes the least number of steps required for an algorithm to execute successfully. Time complexity is dependent on the input as well. As the size of the input increases, so too does the time complexity, as a larger program or algorithm would require more time to run.

Space complexity is the amount of working storage that an algorithm requires. Space refers to storage, or memory allocated for an algorithm to execute. The following is the definition by Chris Riesbeck of Northwestern University:

*"Space complexity is a measure of the amount of working storage an algorithm needs. That means how much memory, in the worst case, is needed at any point in the algorithm. As with time complexity, we're mostly concerned with how the space needs grow, in big-Oh terms, as the size N of the input problem grows."* Riesbeck, Chris. "Space Complexity." *EECS311*. Accessed on March 14, 2021.

Like time complexity, the amount of space required for an algorithm to progress will continuously expand based upon the size of the input. The larger an input, the larger the space required for the algorithm or program is needed. In short, the space complexity of an algorithm

[3] Big-Oh notation is frequently used in computing to determine the time complexity of an algorithm. It calculates and takes into consideration the total number of steps required for an algorithm to arrive at a solution or complete.

or a program is the total amount of space or computer storage taken up depending on its' input size. There is also the concept of auxiliary space, which refers to extra space created and used by an algorithm, or temporary space created during the execution of an algorithm.

**A* Search Algorithm**

The A* search algorithm is like Dijkstra's algorithm[4], in that it finds the shortest path between nodes[5] in a graph. How Dijkstra's algorithm works is that it visits vertices in the graph starting with the beginning node or starting point. It examines the closest node(s) not yet visited and adds its' vertices to the queue to be examined. It will then expand outwards until it reaches a goal. One of the reasons Dijkstra's Algorithm is so popular is that it is guaranteed to find the shortest path from the starting point to the end, or goal state. The other algorithm is the Greedy Best-First-Search, which is similar to Dijkstra's except for an additional element. This BFS search holds an estimate for how far away a goal is from any vertex on a graph or node field. Compared to Dijkstra's which begins by examining the vertex or nodes closest to the starting point, BFS begins by examining those closest to the goal. Although BFS does not guarantee a solution in the way that Dijkstra does, BFS is much quicker as it uses a heuristic function approach. The A* algorithm is like that of a Greedy Best-First-Search in that it uses a heuristic approach to guide itself towards a goal. The A* algorithm combines Dijkstra's algorithm and Greedy Best-First-Search, which results in an algorithm that favors vertices close to the goal and to the starting point. The A* algorithm combines information from both the Dijkstra and BFS algorithms and combines them to create pathing which guarantees a solution and is quick due to a heuristic function.

**Rush Hour A* Algorithm Experiment**

In order to demonstrate the effectiveness of the A* search algorithm, an experiment taking inspiration from the popular game *Rush Hour*[6] was conducted. Given a set of 40 typical

[4] Dijkstra's algorithm is an algorithm invented by Edsger Dijkstra. It computes and determines the shortest path to individual nodes within a graph.

[5] A vertex, or a node, is a fundamental concept in graph theory. Nodes represent destinations and plots on a graph that are able to be or have been visited and interconnect between other nodes.

[6] Rush Hour is a popular online puzzle game where the player must reach the end by formulating a solution to navigate through cars or other obstacles blocking the path.

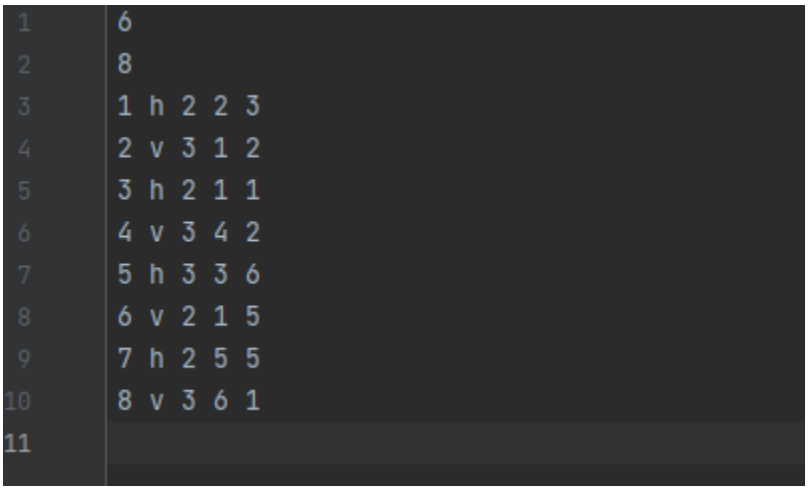
Rush Hour puzzle boards, an algorithm was created to test how quickly the algorithm could determine a solution to each level. The output returned was the running time and how long it took for the algorithm to reach a solution. However, since time complexity and space complexity are both directly correlated with one another, this also allows us to test which algorithm was the best for space complexity as well. Since the larger an input, the higher the time complexity as well as space complexity, we can say that the larger the time complexity, then the larger the input and space complexity. Vice-versa with the ratio of space complexity to time.

Beginning the experiment, 40 puzzle boards were created replicating the most general of the Rush Hour game. These 40 puzzle boards were all separated into their own individual text files and read into the algorithm one by one. Each file held a different time complexity, with each being measured with the time units of milliseconds (ms). In order to capture time complexity, we began each file with a new time, and we calculate elapsed time by subtracting the final time by the initial time. This resulted in the difference thus yielding to us how long it took for the algorithm to execute and return a result for that puzzle. There were 5 algorithms tested during this experiment. First was the Greedy Best First Search, which was the worst out of the 5 algorithms with the longest time complexity and thus the largest space complexity as well. Secondly was the normal A* Search Algorithm with Heuristics which yielded promising results. Next was another A* Search Algorithm with Heuristics, however this time there was an additional factor – the starting point was trapped between 2 vehicles. The results proved that due to this factor, this algorithm scenario took longer to compute a solution, however the difference was minimal. The fourth algorithm features empty places near the starting point which allowed for the manipulation of creating a path. This process also featured a heuristic A* search algorithm, and the results indicate that the time complexity as well as the space complexity were like those of the other A* heuristic algorithms. The difference was miniscule and nearly negligible[7], however the difference of time elapse in computing can make the difference between a good and an okay program. The final program utilized a similar scenario

[7] The difference between the results of the A* heuristic algorithms were so insignificantly small that it almost were identical for each puzzle.

to that akin of the prior, only that the condition involved a blockage of cars to the solution. The goal test of this program is how long it'd take to come up with a solution. As is the case with the prior A* heuristic algorithms, the result was near similar.

An example of the progression of the test would be featuring game puzzle 01.

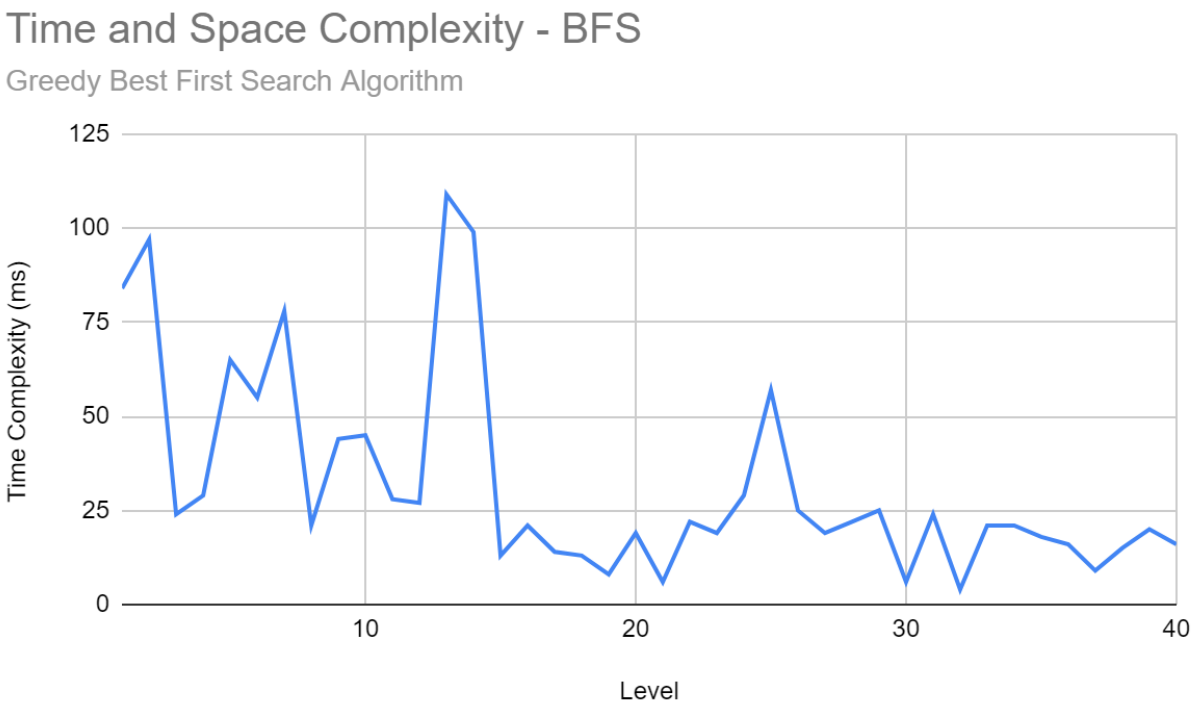



An image of the puzzle board for Game Puzzle 01

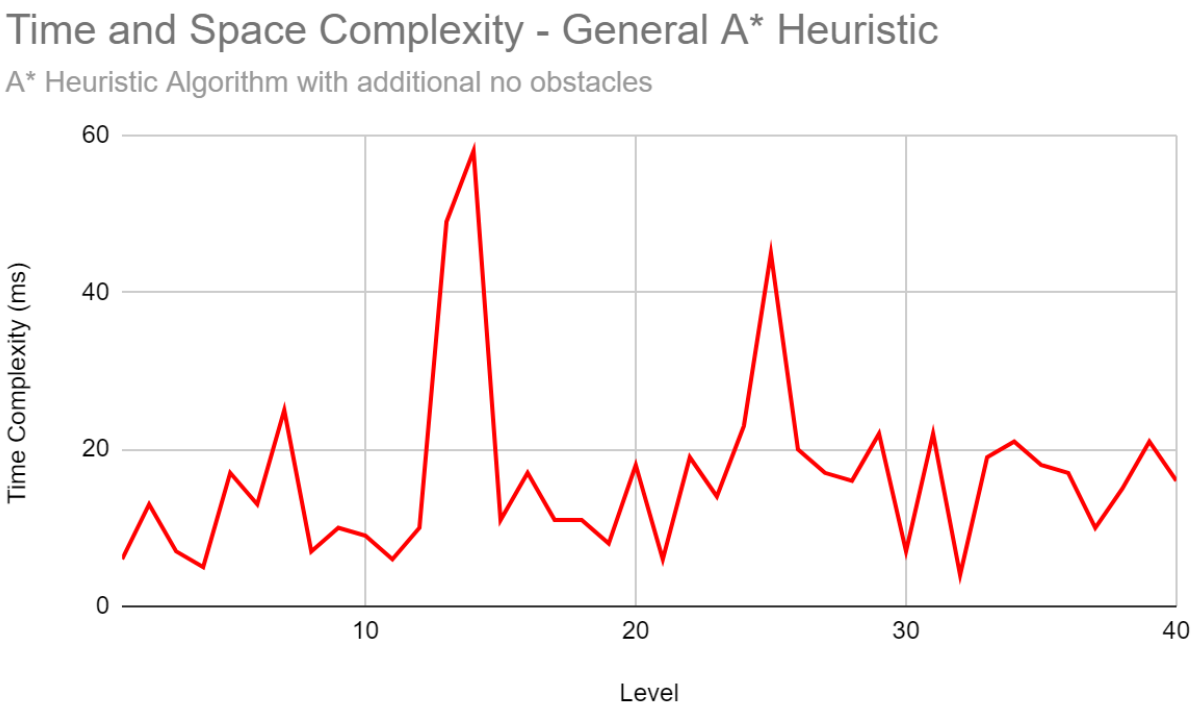
For this puzzle, the Greedy BFS search had to take a total of 8 steps to attain a solution. It had to begin by moving the car 3 rightwards by 1 case, then move car 2 upwards by 1 case. Following that, it would move car 6 upwards by 1 case and car 5 leftwards by 2 cases. Then it would move car 7 to the left by 3 cases and car 4 downwards by 2 cases. It would then move car 8 downwards by 3 cases, and finally car 1 rightwards by 1 case. This would allow for us to get to the end and reach a solution, taking a grand total of 8 steps and 2ms for our elapsed time.

**Results of the Rush Hour A* Algorithm Experiment**

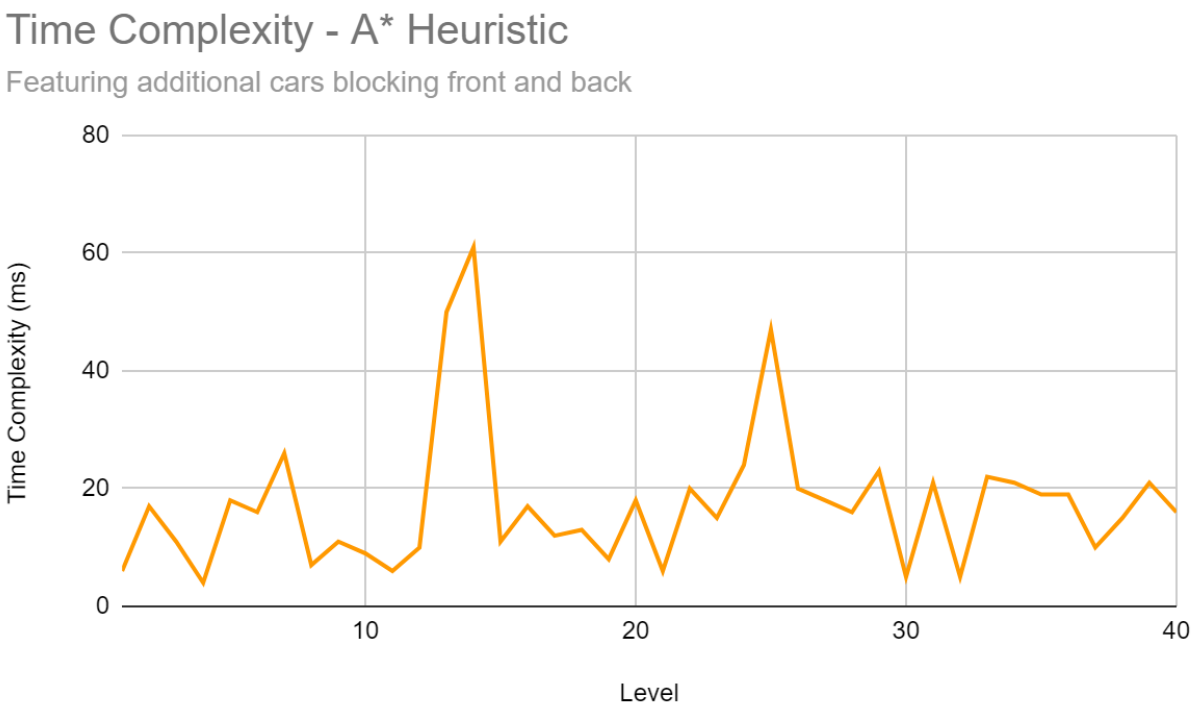
Depicted in the following is the result of each individual test as well as a comparison between all of them.



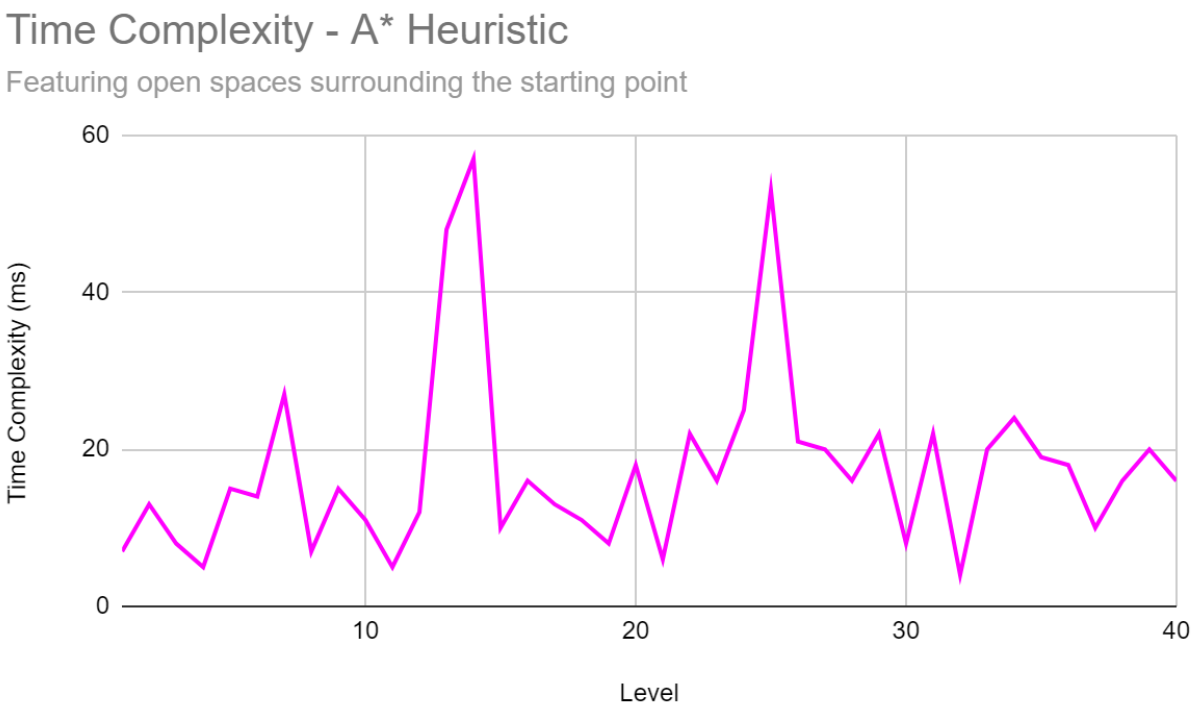

This depicts the results of the Greedy Best First Search Algorithm, which took the longest on cases 1 to 33, and acted better than the A* heuristic searches from puzzle 33 to 40.



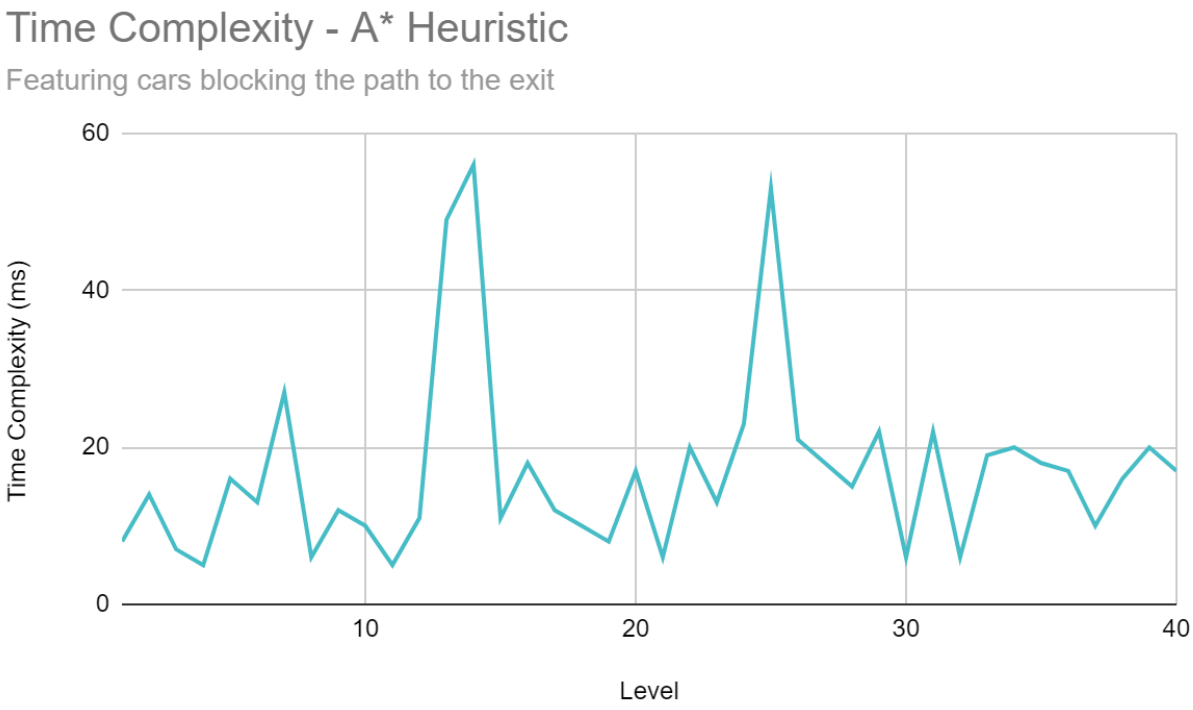

This depicts the time complexity of the general A* heuristic search with no restrictions from puzzles 1 to 40.



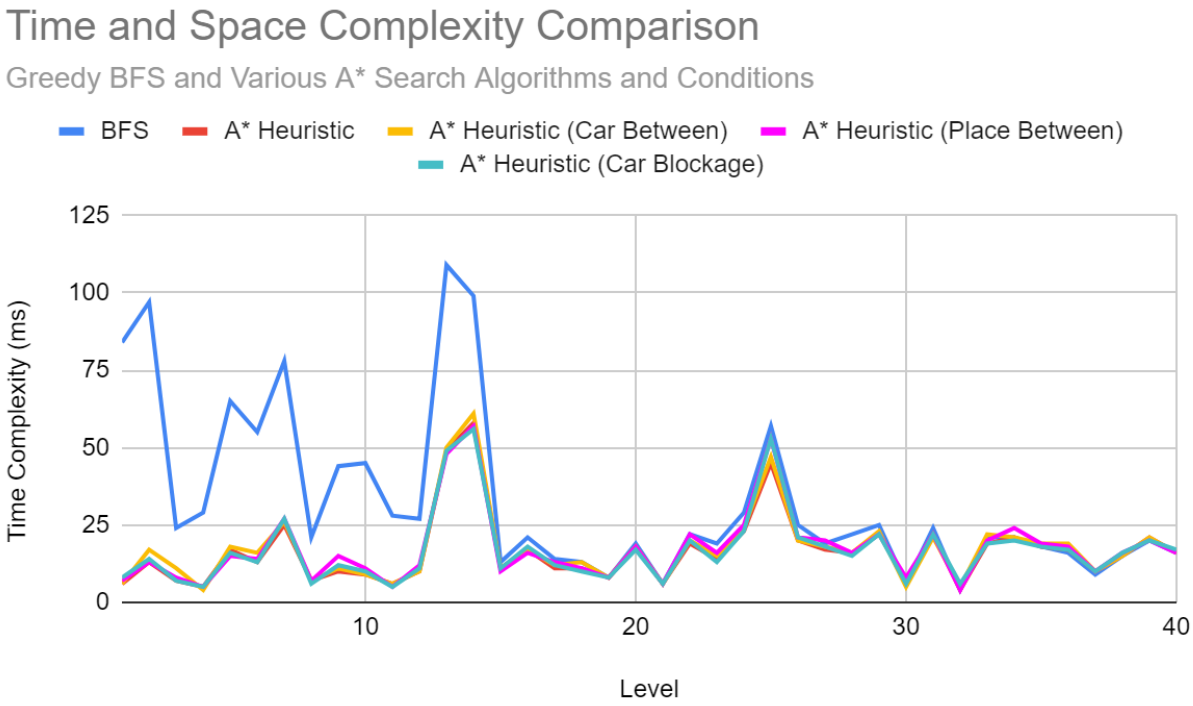

This depicts the time complexity of the A* heuristic algorithm with cars blocking the front and the back, and how long it took to solve under these scenarios.




This depicts the time complexity of the A* heuristic algorithm with open spaces surrounding the starting point and how long it took to solve under these scenarios.




This depicts the time complexity of the A* heuristic algorithm with cars blocking the path to the solution, and how long it took to conclude a solution.




**Conclusion**

As told by the diagrams and the data presented, the BFS (Greedy Best First Search) was the worst of the 5 algorithms tested as it had taken the most time to complete each of the individual test puzzles aside from the last few. In some cases, the BFS worked better than the A* heuristic due to the specific puzzle used, since the BFS was able to find the solution quickly rather than scour the entire puzzle to figure out an answer. Based upon each puzzle, we are able to determine the factors that led to the BFS having such a longer time complexity when compared to the other searches. This is all relative to the specific puzzle that was presented, and although it acts upon a case-by-case basis, in general the A* search algorithm performs better than the greedy BFS algorithm, as it yielded better results in more cases.

References

Romanycia H.J; Pelletier, Francis J. “What is heuristics?” *Computational Intelligence*. 1985.

https://www.researchgate.net/publication/227733871_What_is_a_heuristic

Riesbeck, Chris. “Space Complexity.” *Northwestern University*.

https://courses.cs.northwestern.edu/311/html/space-complexity.html#:~:text=Space%20complexity%20is%20a%20measure,of%20the%20input%20problem%20grows.

Amit, P. “Introduction to A*.” *Stanford University*.

http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

https://www.redblobgames.com/pathfinding/a-star/introduction.html

Cormen, Thomas H; Leiserson, Charles E; Rivest, Ronald L; Stein Clifford. “Introduction to Algorithms.” MIT Press, 1990.

https://mitpress.mit.edu/books/introduction-algorithms