# Optimizing A* Heuristics for GPS Useage

MJ Corey
Randy Tan

May 1, 2024

## Abstract

A* is an algorithm that takes the cost of an action in addition to a heuristic and produces a shortest path optimal solution to a graph. A* has been studied extensively about its efficiency as opposed to some other common search algorithms such as Breadth-First Search or Depth-First Search. However, the variability of a heuristic can allow A*'s efficiency to change based on the environment it is in. The environment this report aims to look at is that of an intersection graph, or rather a map. The objective of this report is to find a heuristic that reduces the amount of computing power the search algorithm has to do by reducing the number of intersection nodes in the search frontier and raising efficiency for the user by having routes that have fewer intersections and shorter distances to minimize fuel consumption. To find the best-suited heuristic, 50 trials of three varying lengths were conducted on all determined heuristics. The following report suggests that using a heuristic that minimizes the number of turns and also taking longer roads will minimize both of these things.

## Description

The problem at hand involves finding the shortest paths on a map, using pathfinding algorithms such as A* and Dijkstra's algorithm. This algorithm will incorporate five different heuristics. It will consolidate a heuristic which punishes left turns, since the left turns should take longer than other turns. Another punishment that follows this is the quantity of turns used in general, as that will slow down the traversal of the map. The map used to execute this, consists of a map of Minneapolis, Minnesota. This uses a CSV file containing the data necessary in order to create a map, including whether they are one-way or two-way streets, as well as the coordinates of their start and end points. The goal is to implement algorithms that can efficiently navigate this map to find the shortest routes between specified locations.

One of humanity's current goals is to emulate the human mind in an inorganic medium. The hardest part in this endeavor has been emulating human ingenuity; even picking something as simple as picking a path to get from a starting point to a destination can be difficult for non-humans without the nuance of understanding how ideas such as traffic, the danger of taking the path, and various other factors. The solution to modern attempts at such a feat is an algorithm called A*: an algorithm that combines the initial cost of an action with a user-defined heuristic to estimate the efficiency of taking such an action. This heuristic is the ‘ingenuity’ aspect of decision-making; a good, admissible heuristic will allow a non-human agent to find an optimal solution. This literature review will take a deep dive into the history of the A* algorithm specifically regarding path planning, some of its applications, and recent optimizations in the algorithm

In the technical aspects, the problem is especially interesting as it is relevant in the real world. For example, finding efficient solutions to optimize transportation routes in navigation systems. The trade-offs of the algorithm offers the scrutiny of complexity, optimality, and completeness. A* permits the application of heuristical elements in allowing unique situations such as penalizing left turns. Experimentation of these various heuristics allows for copious amounts of varying strategies to discover optimal solutions.

## Historical Context of A*

In 1966, a paper written by J. E. Doran and D. Michie proposed a new view of graph traversal problems. Doran and Michie claimed the field was heavily divided into two distinct approaches to graph traversal at the time of publishing. One approach was the evaluation of the states which was the overall productivity of each step to the end state. The other was the evaluation of operators which aimed to minimization of the cost of all actions taken. Doran and Michie proposed the Graph Traverser, the first documented algorithm akin to A*, as a potential algorithm that takes both ideologies of. The Graph Traverser aimed to find a solution that was, in Doran and Michie’s words, both ‘elegant’ and ‘economical’. An elegant solution has a short path, which aligns with the ideology of the second approach. An economical solution minimized the total search; this lowers the branching factor of the graph and thereby visited the least number of nodes possible which would align with the first approach of graph traversal. Doran and Michie’s work would go on to revolutionize what was known as graph traversal methodology. [DM66]

A* was born through multiple iterations beginning with A1. A1 was created by Nils Nilsson in 1964. The A1 algorithm was introduced as a heuristic-based approach applied to Dijkstra’s algorithm. The algorithm has since gone through many optimizations and iterations. More so the heuristics of the algorithm rather than the algorithm itself. This led to the algorithm being named, A*, as it refers to all the potential the algorithm has and more that may come. [Van11]

The A* algorithm would go on to revolutionize problem-solving. Many problems could be represented through states and actions which qualified it to be solved via A*. A state is any point in time within a problem that could branch into other states by taking an action, and an action is any way a problem could proceed with an associated cost for taking said action. One of the most practical problems for A* to solve in recent times has been that of vehicle traversal; in 2023 Huixia Zhang, Yadong Tao, and Wenliang Zhu set out to use A* to plan the path of an unmanned surface vehicle (USV), an autonomously navigated intelligent naval ship. A* is a great algorithm for vehicle traversal due to its adaptability to changes in the graph environment. In Zhang, Tao, and Zhu’s work, the environment the USV was in would contain obstacles and boundaries unknown to the USV; A* allowed the USV to traverse these obstacles efficiently since the heuristic used by the program allowed the algorithm to deem pathways near the obstacles to be inefficient. Vehicle path planning has been innovated by A* and will continue to be one of the best solutions to the problem. [ZTZ23]

Although Dijkstra's algorithm was one of the most studied and efficient of path-finding algorithms in 2007, A* was found to be under-researched with the potential to beat Dijkstra's. The fundamental problem of finding the shortest path on road networks and proposition of improvements to the A* algorithm for this purpose. The authors aim to compare A* with efficient implementations of Dijkstra's algorithm, particularly in the context of geographic information systems (GIS). Previous research on A* is reviewed, noting its potential advantages for certain applications but also its under-representation in comparison studies. The authors conduct tests on real road networks and conclude that A* can outperform Dijkstra's algorithm when implemented efficiently, especially in GIS applications. Overall, the article presents evidence supporting the effectiveness of A* for finding shortest paths on road networks, particularly when integrated with GIS data. [Zen09] This is later proven to be true by an article done by Kevin Chen. The algorithm used in this article has gone under 15 years of evolution, and now with this research focusing on the k-step look ahead heuristic. It improves the computational aspect of the algorithm by estimating possible future paths and taking the path that seems the most promising. [Che22]

A* as an algorithm has also undergone plenty of innovations. One of the largest advantages of the A* algorithm is the adaptability of the heuristic portion to the environment the algorithm is working. One such example was a stationary robot to identify and pick up a basketball and then return it to a goal destination was researched by Taicheng Yang in 2023. To help optimize this process, Yang utilized a technique called Colony Algorithm, which is a subset of A*. The Colony Algorithm gets its name from ant colonies because of the behavior of ants leaving behind pheromones upon finding important items for the rest of the colony. Similarly, upon the success of A* reaching a goal state, it will leave behind 'pheromones' which further influence the heuristic of the path the previous successor to travel a similar path. This ideology works for Yang's experiment since the robot is stationary and the obstacles from previous trials will persist. This makes future trials more likely to take the optimal path with less searching. Yang utilizing the Colony Algorithm showed the power of A* and how it can be further optimized for use. [Yan23]

Another practical usage of A* and its optimization is the D* algorithm: an algorithm for dynamic or changing environments derived from A*. D* is a version of A* that will use A* to find a path to a goal state initially, and upon finding that the path the algorithm was intending on taking is blocked off or no longer seen as efficient will reroute the current route around the obstacle. One instance where this may be useful would be for any environment with moving obstacles within it as shown by Firas A. Raheem and Umniah I. Hameed. D* proved to be an integral part of making efficient pathways within an environment with obstacles that were non-stationary. This experiment done by Raheem and Hameed proved that A* can also be improved upon so that its heuristic is also adaptable as well in the form of D*. [RH18]

The incorporation of turning costs in A* also offers a more optimized heuristic when it comes to geometric graphs. The research done by Karlijn Fransen and Joost van Eekelen presents the A* algorithm on geometric graphs with turning costs. It highlights the limitations of existing heuristics. More specifically, the turning costs, relate to the deceleration before and acceleration after the turn. The authors propose a new monotone and admissible A* heuristic incorporating turning costs, applicable to any geometric graph structure. They validate its effectiveness by demonstrating a decrease in the number of iterations needed to find the lowest-cost path compared to other heuristics. The decrease in the number of iterations and the closer the heuristics are to the actual costs allow the heuristic to become more efficient. The article suggests further improvements such as considering graph structure and outgoing edges at the current vertex. Despite increasing computational complexity, the new heuristic shows promising results, given that the scenarios contain non-zero turning costs. [Fra21]

Modern society has quickly adopted the Global Positioning System (GPS) as the default method of determining an optimal path from an origin location to a destination, but how is that path determined? While given enough time and actions any search algorithm would eventually find a solution from start to final destination, good algorithms will minimize either the total distance traveled or the time spent to get to that location. Most GPS systems use an A* algorithm to find the best solution: a search algorithm that uses a heuristic to determine how productive a move is. The problem this project aims to solve is finding good heuristics for a GPS to use for an A* algorithm.

## Approach

To find the optimal heuristic for a map, both the cost of an action in the environment map and the common problems of driving need to be considered. Since the A* algorithm uses both the cost of an action and associated heuristic values, both need to be clearly defined to determine how efficient the algorithm is. In addition, there needs to be metrics to determine how efficient each different heuristic is. The approach section will clearly define each of these values and why the values were chosen to represent their respective values in the A* algorithm.

First, a consistent cost of the action of traversing two points needs to be determined. The intuitive value for the cost of traversing from a starting node to a destination node would be to calculate the distance between the two in 2-dimensional space using the distance formula as shown:

$$Diagram1 : D(x1, y1, x2, y2) = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

This is a good metric to determine the cost of moving from one point to the other; however, it is not perfect. Roads are atypically straight, especially in rural areas where roads are formulated around the natural environment. Due to the limited scope of this report and the fact that a metropolitan area was used for the data collection in this report, the distance between the start and end points of the road was sufficient to draw data that yielded conclusions. Another reason why the distance formula is a good shortcut for the cost of traversal is that to properly emulate windy roads additional nodes need to be placed along the road to show the slight turn or the length of the road needs to be contained in the algorithm; both of these solutions tax the memory requirements of the algorithm and therefore directly oppose the objectives of the report.

In addition to the cost of traversal, measurable metrics need to be defined to determine how productive a heuristic is for the algorithm. Since the objectives of the report are to minimize the memory requirements of the algorithm, minimize time spent traveling, and minimize the fuel consumption of the trip, a numerical metric will be defined for each objective to analyze heuristic performance. For memory requirement, the number of nodes added to the frontier will determine how memory-intensive a heuristic can be. To minimize the time spent traveling, the number of nodes in the final path from the start node to the finish node will be looked at. The thought process is that since each node represents an intersection, by minimizing the number of intersections the final path goes through, the smaller the chance the user has to wait at a red light or wait for a vehicle in front of the user to turn. Finally, to determine how much fuel is used throughout the trip, the sum of all the distances from each node traversal will be analyzed.

Finally, the heuristics that will be tested are designed to try and address each objective and finish by combining heuristics to see if they are more efficient in tandem. The control heuristic that will be used is that of Dijkstra's Algorithm or rather no heuristic at all. As mentioned in the context of A* section, most GPS algorithms use Dijkstra's as the pathing algorithm of choice. Dijkstra's algorithm will simply add the shortest edge of all neighbors of the frontier to the frontier until the desired destination is in the frontier. This is a good control heuristic since none of the objectives have an advantage from the lack of a heuristic. Another common heuristic for free movement in a 2D graph space is the Euclidian distance: the distance between the two points. Since it is the most common heuristic for a problem within the scope of the report, it will be tested too.

The other heuristics are made with the objectives of the report in mind. Heuristic3 is a heuristic that takes the angle between the vectors created from the start to the destination and the intersection that introduces the start node to the frontier and the start node. Based on predetermined angle values for a left or right turn, a predetermined ratio based on what kind of turn is taken is applied to the distance between the start and destination to keep the heuristic admissible. Since continuing through an intersection is faster than turning to the right, and turning right is faster than waiting to turn left (since turning left relies on traffic on the other side of the street), heuristic3 aims to minimize the total time spent turning. Heuristic4 aims to minimize the number of nodes added to the frontier by preferring to add the higher cost transaction, the higher valued transactions also inherently cover more distance towards the goal if the goal is far away, saving time due to the lower intersection interaction. Heuristic4 prefers longer distance transactions by making a ratio of the longest node cost in the entire graph, subtracting the distance of the transaction, dividing it by the longest cost value, and finally multiplying it by the distance of the transaction to keep the heuristic admissible. Heuristic5 is a combination of heuristic3 and heuristic4. Each previous heuristic can contribute up to 1/2 to the final ratio heuristic5 multiplies to the distance of the transaction. The pseudocode for the heuristic calculation is as follows:

```
Diagram 2:
get_heuristic(startX, startY, destX, destY, prevX, prevY):
    if heuristic = Dijkstra
        return 0

    if heuristic = Euclidian:
        return distance(startX, startY, destX, destY)
```

```
if heuristic = heuristic3:
    line1 = vector(prevX, prevY, startX, startY)
    line2 = vector(startX, startY, destX, destY)
    angle = getAngle(line1, line2)
    if angle < ltAngle: //Right turn, 1 > c > a > b > 0
        return a*distance(startX, startY, destX, destY)
    if angle > rtAngle: //Left turn
        return c*distance(startX, startY, destX, destY)
    else: //Straight
        return b*distance(startX, startY, destX, destY)

if heuristic = heuristic4:
    //maxRoad is the largest distance in the entire graph
    roadSize = (maxRoad - distance(startX, startY, destX, destY))
    ratio = roadSize / maxRoad
    return ratio*distance(startX, startY, destX, destY)

if heuristic = heuristic5:
    heur3value = getHeuristic(.., heuristic = heuristic3)
    heur4value = getHeuristic(.., heuristic = heuristic4)
    //1 > heur3ratio + heur4ratio
    return heur3ratio * heur3value + heur4ratio * heur4value
```

## Experiment

The approach used in order to find an optimal heuristic was by setting up a repeatable environment to test on. For the data collection of this report, a comma-separated variable (CSV) file representing the metro area of Minneapolis, Minnesota in the United States was used to form the environment graph. The testing environment used in this report was in the form of a 5-column CSV file that would use the first column to determine if the street information is a one-way street or not. The second and third columns would be the starting x and y position respectively, and the fourth and fifth columns are the x and y position of the destination. This could be interpreted as the beginning and end coordinates of a street. The graph will be created by making a connection from the start position to the destination position and a cyclical connection from the destination to the start if the CSV file indicates it is a two-way street. This process is repeated for each line within the CSV file.

Once the graph has been formulated, the startX, startY, destinationX, and destinationY are selected randomly from the previously created map. This is repeated, taking a sample of ten thousand pairs of start and ending points in order to find the average, shortest, and longest distance between randomly selected start and destination points on the map. This is done in order to calculate the lower bound and upper bound. The lower and upper bounds are used to define the acceptable range of distances for the randomly selected start and destination points. The bounds are dependent on the first argument given by the user via argument[1], which consist of 'short', 'medium', and 'long'. If 'short' is given, the lower bound is 0 and upper bound is 0.5. If 'medium' is given as the user input then the lower bound becomes 0.5, and upper bound is adjusted to 1.5. If 'long' is provided then the lower bound is set to 1.5 and upper bound is set to 2. These bounds are then multiplied into the previously defined average distance variable. The purpose of these bounds is to ensure that the distances selected fall under certain criteria for testing heuristic algorithms. 'short': distances less than half the average distance. 'medium': distances greater than half the average distance but less than one and a half times the average distance. 'long': distances greater than one and a half times the average distance but less than double the average distance. Argument[2] contains the number of iterations the program will run.

Once all of the necessary bounds are collected, the next step is selecting random startX, startY, destinationX, and destinationY points. All while making sure these start and destination points cannot start as the same point as that would result in a distance of 0. In an A* algorithm, the heuristic function is used to guide the search towards the goal by estimating the cost from the current state to the goal. The program runs through each heuristic which ranges from one through five. Heuristic1 is essentially Dijkstra's algorithm, using the value 0. When the heuristic function returns 0, it essentially reduces A* to Dijkstra's algorithm. Heuristic2 is an algorithm that incorporates the euclidean distance heuristic, which calculates the straight line distance between 2 points. The formula used for calculating the euclidean distance is stated in [Diagram 1]. The third heuristic adds a fraction of the distance based on if the turn is a left turn, a straight, or right turn. The type of turn is determined by calculating the degrees of the angle. After the angle is calculated, the heuristic value returned is correlated with [Diagram 3] below. In [Diagram 3], the distance is the euclidean distance between the start and neighboring nodes to reach the destination. If it is a left turn, the angle calculated is below -30 and the multiplier is set to 1.5. If the angle is ¡120, a right turn, the multiplier becomes 2. Otherwise, it is a straight turn and the multiplier is set to 2. Heuristic4 is a heuristic that prefers to take longer roads to avoid intersections. In [Diagram 2] the maxRoad value was determined to be 691 as the largest distance in the entire graph. Heuristic5 is a mixture of heuristic3 and heuristic4; it prefers longer roads while taking note of the turns it takes. Using the same values depending on the angle of the turns and the same value for the maxRoad determined in last two heuristics mentioned.

$$Diagram3 : (\frac{1}{multiplier}) * distance$$

Next, the method used in this implementation A* algorithm integrates the use of each heuristic with it in each iteration. This is done through using 3 lists, visitedNodes, frontier, and finalpath. VisitedNodes is used to keep track of visited nodes. the frontier is a list of nodes that are currently being considered for exploration to find the shortest path from the starting node to the destination node. finalpath contains the final route that the algorithm took to get from the start to the destination points. It does this all while making there are no one way dead end streets or starting points with no neighboring nodes. If the heuristic is greater than the distance from start to destination then this is when it is considered inadmissible, in which the program will exit. Otherwise, a fitness value is calculated by adding the distance with the heuristic; the distance and heuristic are both calculated with the current [x,y] and the the neighboring point [neighborX, neighborY]. Then, it will append the neighboring point to the frontier and visitedNodes if it is admissible.

Once all of that is complete, the program will create the finalpath given the nodes from the frontier. In this process the program will also calculate the totaldistance by calculated the distance between each node in the finalpath along with counting the number of nodes necessary to get from the start x and y to the destination x and y. With this information, the program will write the necessary info out to a .txt file in order for the creation of graphs. The text file contains 6 columns, of which, first column contains the number of iterations. The second column holds the argument[1], which is the distance provided, 'short', 'medium', or 'long'. The third column consists of the path length. The fourth column is made up of the total distance traversed from the finalpath. Finally the fifth column is composed of the distance between the starting point and destination point.

Below are the recorded results for the path length. This data contains 50 iterations of each heuristic. In each iteration the start and destination points are the same across the heuristics so the differences between each heuristic is observable. [Diagram 4] contains the data from of the path length. [Diagram 5] containing the frontier size statistics. [Diagram 6] accommodates the information about distance traveled.

Diagram 4:

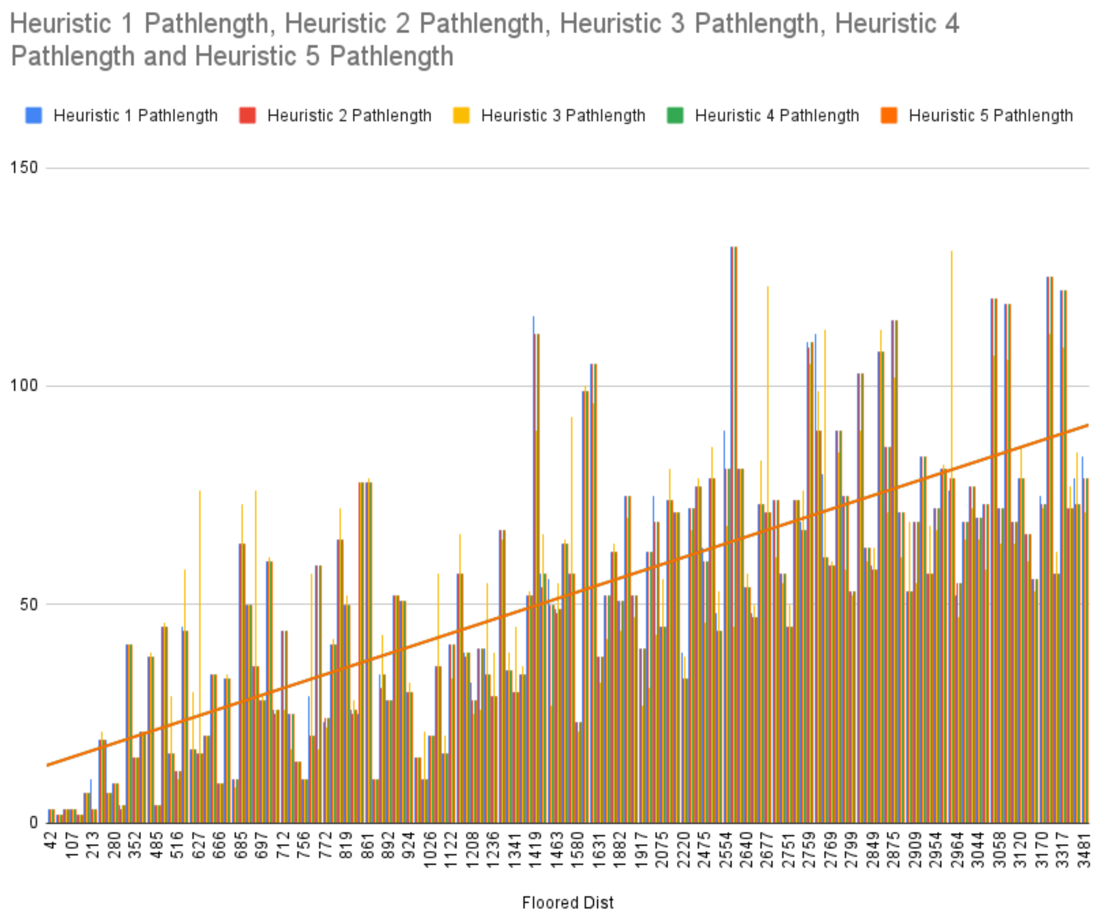


Diagram 5:

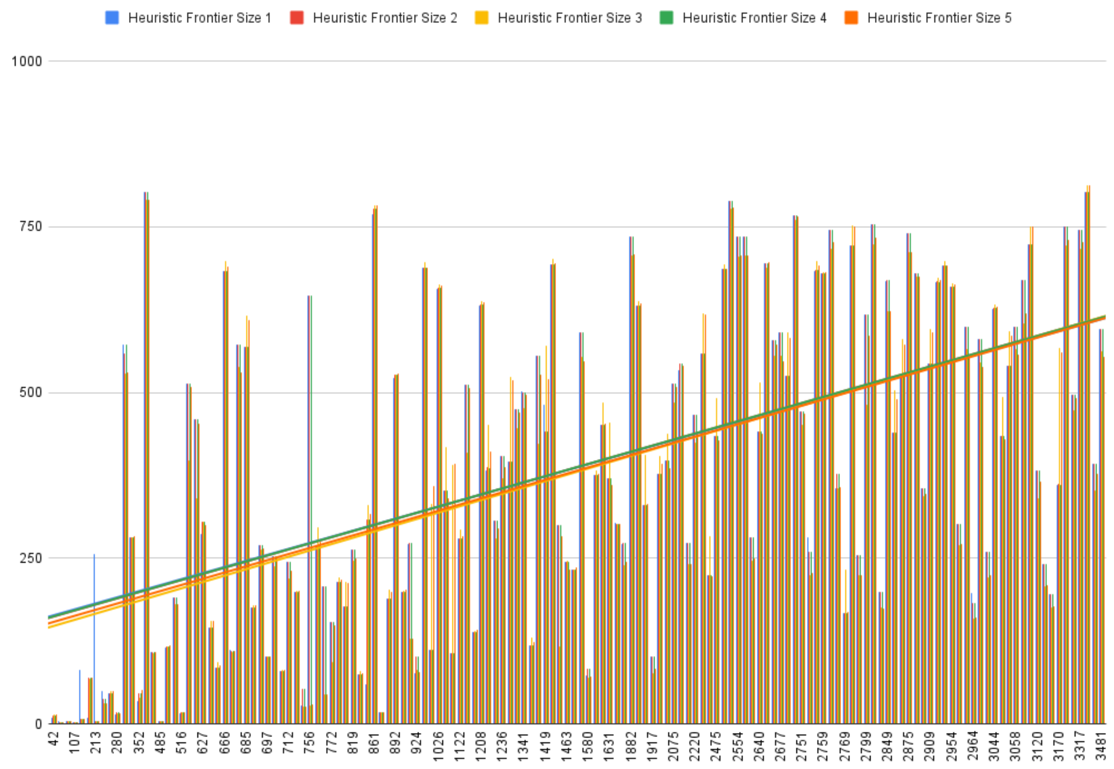
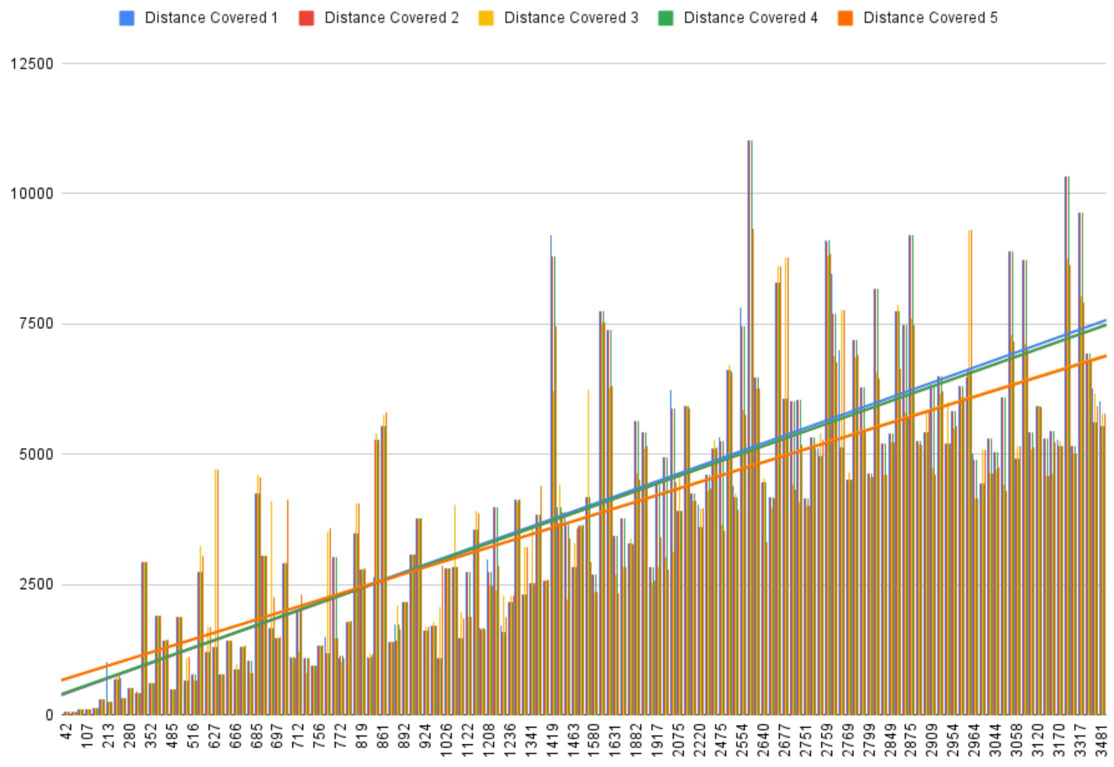


Diagram 6:



## Analysis

The analysis of the data will be looked at through how each heuristic performed for each objective that was defined in the approach section compared to the other heuristics observed in this report. The first objective defined in the report was how many nodes are in the final path from the start to the destination, or rather how many intersections does the user have to go through to get from the start to the destination. From the data generated, heuristic3 was the most optimal heuristic for minimizing the path length by a decently large margin when the trendlines are viewed. One possible reason for this trend is related to the fact that heuristic3 prefers to go straight and only turn when it needs to. By doing this, the path from the start point to the endpoint will likely be similar to two straight paths, one traveling straight until its x or y is close to the destination x or y respectively, and then a turn followed by another line that matches the one that was not matched before. On the contrary, taking the path provided by just taking the shortest possible road at an intersection (Dijkstra's) will make the path closer to the hypotenuse of the two aforementioned lines. Traveling on the hypotenuse will likely involve multiple intersections to stimulate traveling in a diagonal motion. On the contrary, heuristc3 can struggle for the same reasons for very short distances because it may extend to too many nodes when fewer nodes along the 'hypotenuse' may be more optimal. However; the trendline proves that this tradeoff is worth it since it is not very long distances before heuristic3 performs better than the other heuristics. Because of all of these reasons, heuristic3 is the best heuristic to use if time consumption for the user is the primary concern.

For memory optimization, no conclusive best heuristic was found. Even though heuristic4 had a slightly better performance than the other heuristics, the advantages of using heuristic4 for the slight memory improvements is negligible if the other objectives of the report are also considered for heuristic choice. The reason that heuristic4 has a slightly better performance than the other heuristics is likely due to the fact that heuristic4 will expand across the entirety of the environment with fewer additions to the frontier because of the heuristic's preference for larger costed actions. The more interesting question from this part of the analysis is why are none of the heuristics particularly more effecient than Dijkstra's algorithm? The intuitive answer to that question is that none of the heuristics are actually adding nodes to the frontier based on how close the new node is to the destination. The heuristics used are more based on the performance of how the new addition will perform in the final path. This report suggests that further research should be put into an additional heuristic that is an admissable ratio based on how close the new addition is to the final destination.

The final metric analyzed in this report, and potentially the most relevant, is the total distance of the final path from the start position to the destination position. The data supports that heuristic5, the heuristic that is calculated by both what kind of turn the addition would be in addition to how long the road is that is being added. This heuristic is the most optimal heuristic of the tested heuristics by a large margin. The reason this heuristic is the best at minimizing the distance of the final path is in part because of the reason mentioned above about heuristic3 being an efficient solution. In addition to that however, heuristic5 does not put nearly as much weight on the type of turn as it only makes up half of the heuristic. In addition to the efficient logic of heuristic3, heuristic5 also will take a left turn when the road is long enough such that it travels a sufficient amount of the environment because of the shared logic of heuristic4. These two heuristics in tandem allows heuristic5 to make very sensible and efficient maneuvers to minimize the total distance traveled.

There are some other interesting observations from the experiments done as well. First, for final path intersections traversed, frontier size, and total distance traveled, the Euclidian Distance heuristic mirrored the data exactly for Dijkstra's nonexistent heuristic. This intuitively makes sense however once it is observed that Euclidian Distance and the cost of traversal are the same. This fact opens the derivation that the cost of A* for every possible transaction is the distance from the start node to the neighboring node in addition to the same distance. If cost is equal to the heuristic value the following derivation can be made:

$$\begin{aligned}
A^* &= \text{cost} + (\text{cost}) && \text{(where cost = heuristic)} && (1)\\
A^* &= 2\text{cost} && && (2)\\
A^* &= \frac{2\text{cost}}{2} && \text{(simplify, as all transactions are } 2 \times \text{cost)} && (3)\\
A^* &= \text{cost} + 0 && \text{(equal to Dijkstra's algorithm)} && (4)
\end{aligned}$$

Another interesting observation is the fact that when looking at the lengths of the paths that heuristics take to get from the start to finish, heuristic4 and heuristic5 performs exactly the same. This means that when determining what nodes should be added to the final path, the logic of heuritic4 within heuristic5 takes president over the logic that heuristic3 adds to heuristic5. This shows that there may be further optimization to be had within heuristic5 specifically by changing the weights of how heuristic3 and heuristic4 add to the final value of heuristic5 such that heuristic3 has a higher weight when determining what to add to the frontier.

## Conclusion

The A* algorithm has played a significant role in revolutionizing problem-solving, particularly in path-finding problems. Its adaptability to various environments and its ability to efficiently find optimal paths make it a valuable tool in fields such as GIS, also known as geographic information systems. Through historical developments and ongoing optimizations, A* has demonstrated its versatility and effectiveness. Beginning from the 1960s to the recent advancements incorporating turning costs in geometric graphs and looking ahead heuristics, A* continues to evolve and provide solutions to complex path-finding problems. In conclusion, the study highlights the pivotal role of heuristic selection in optimizing A* algorithm performance for GPS applications. While different heuristics present distinct advantages and trade-offs, heuristic5 emerges as the most efficient in minimizing total travel distance. The analysis reveals heuristic5's superiority due to its balanced approach, which considers both directional preferences and road lengths. By combining elements from heuristic3's preference for straight paths and heuristic4's inclination toward longer but faster roads, heuristic5 achieves optimal route optimization.

## References

[Che22] Kevin Chen. An improved a* search algorithm for road networks using new heuristic estimation. *An Improved A* Search Algorithm for Road Networks Using New Heuristic Estimation*, 2022.

[DM66] J. E. Doran and D. Michie. Experiments with the graph traverser program. *Proceedings of the Royal Society*, 1966.

[Fra21] Karlijn Fransen. Efficient path planning for automated guided vehicles using a* (astar) algorithm incorporating turning costs in search heuristic. *International Journal of Production Research*, 2021.

[RH18] Firas A. Raheem and Umniha I. Hameed. Path planning algorithm using d* heuristic method based on pso in dynamic environment. *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)*, 2018.

[Van11] Phaneendhar Reddy Vanam. Shortest path using a algorithm. *Indiana State University*, 2011.

[Yan23] Taicheng Yang. Colony algorithm and A* based optimized ball picking robot. *AIP Conference Proceedings*, 2023.

[Zen09] Wen Zeng. Finding shortest paths on real road networks: the case for a*. *International Journal of Geographical Information Science*, 2009.

[ZTZ23] Huixia Zhang, Yadong Tao, and Wenliang Zhu. Global path planning of unmanned surface vehicle based on improved a-star algorithm. *MDPI Sensors*, 2023.

## Contributions

Abstract - MJ
Description - Randy
Historical Context of A* - MJ & Randy
Approach - MJ
Experiment - Randy
Analysis - MJ
Conclusion - Randy