Analysis of A* Search on Minneapolis City Map

Silas Kati (kati0012)[1], Moyan Zhou (zhou0972)[1], and Sai Tarun Inaganti (inaga015)[2]

[1]Dept. of CSE, College of Science and Engineering, UMN
[2]Robotics Institute, College of Science and Engineering, UMN

December 14, 2022

Abstract

This project discusses the core idea of pathfinding in artificial intelligence, which has various applications. Several pathfinding methods are used to discover the shortest path between a source and a destination. The shortest path has been found using a variety of techniques. Dijkstra, A*, and ant colony optimization are the most popular methods. Despite significant progress in pathfinding over the previous few years, several problems still interest researchers. One such problem is the high-performance requirements that these algorithms must satisfy.

Pathfinding is a common search issue in computer strategy games and robotics that looks for the shortest path between two points on a map. This study uses the Minneapolis map and four different heuristics to determine the pathways with the lowest costs. This is later extended to a few other cities with varying topologies. This paper employs informed search methods focusing on the A* search. We contrast the four heuristics in terms of use cases, temporal complexity, and spatial complexity. As researchers are interested in finding diverse heuristics to satisfy different user needs, such as route planning based on landmarks, we aim to build a foundation for future exploration of more complex heuristics.

The material in this project report will help researchers choose the right heuristics for pathfinding algorithms and give suitable performance measures for shortest path planning estimation between two predetermined nodes in a graph grid layout. In addition, it explains how the A* algorithm's heuristics define the decision-making process and how to balance the weights of the time and cost aspects based on particular use scenarios.

1 Introduction

Nowadays, it has become increasingly important to reach a destination quickly, in the shortest path between all the options and possibilities. Different search engines like google maps address this need, for example, and it is common sense that the user is expecting the result in the least amount of time. There is also a requirement for high-performing algorithms in a short execution period since they frequently need to compute pathways for several components, and the resources allotted to these Algorithms are restricted. For better optimization, a specific algorithm might be employed.

Pathfinding algorithms can be divided into two categories, static and dynamic. The term "static" describes finding the route globally in a static environment. Dynamic pathfinding refers to locating the path locally in a dynamic environment. We can utilize pathfinding AI algorithms to discover the most practical route from a start node to a specific objective. Some pathfinding algorithms could consider different obstacles, while others might not. These pathfinding algorithms need a lot of memory and processing power to identify the best way to get from point A to point B while avoiding all obstacles.

The two main kinds of pathfinding subjects are multi-agent and single-agent. An extension of the single-agent pathfinding issue is the multi-agent pathfinding (MAPF) problem. In a multi-agent scenario, several agents look for their destinations at once. The root node is where pathfinding algorithms start their exploration of the network, and they keep going until they locate a solution.

A proper environment has to be set up before pathfinding algorithms can be used and their performance assessed. The ubiquitous undirected uniform-cost grid map is the most common and extensively used way of describing pathfinding settings. Additionally, we want to decrease the search area that single-agent pathfinding problems must comb through to identify the best answer.

The process of pathfinding has long been studied. It is arguably the most well-known but challenging Artificial Intelligence (AI) challenge. Commercial pathfinding issues must be resolved in real-time, typically with constrained memory and CPU resources. The computational effort is intended to use a search algorithm to identify a path. Because of this, pathfinding on big maps has the potential to cause serious performance bottlenecks.

"How to travel from a source to a destination?" might be answered using pathfinding. Most often, there are several ways to go from the source (current point) to the destination (next point), but if at all feasible, the solution should achieve the following objectives:

1. The route to take to travel from point A to point B.
2. The route to take to avoid obstacles.
3. The technique for determining the quickest route.
4. A simple technique to locate the path.

Some path-finding algorithms address just one of these issues, while others address them all. In some situations, none of these issues could be solved by any method. The fastest route in terms of time is not usually the one with the lowest distance. Going from source A to destination B will have different expenses than moving from source A to destination C. The challenge of determining the shortest route between a source and a destination while avoiding obstacles is addressed by pathfinding algorithms. Finding the shortest path was addressed by developing several search algorithms, such as the A*, Bread-First, and Depth-First search algorithms.

The rising relevance has made pathfinding a common and annoying issue in business. One of the biggest obstacles to creating realistic AI is agent mobility. Any AI movement system's core often uses pathfinding algorithms. The most frequent difficulty in pathfinding is how to avoid barriers carefully and locate the best route via various locations. The modern computer game business is growing much larger and more complicated yearly in terms of the map area and the number of units in the market.

2 Problem Statement

Given admissible heuristics, A* search is well known for ensuring optimal solutions. A heuristic function is said to be admissible if it is non-negative, never exceeds the actual cost from the current node to the goal, and ensures that the selected path will always have a cost that is lower than or equal to the path with the optimal cost, and gives the selected path priority when expanding. Therefore, choosing a good pathfinding heuristic is essential for improving the A* search algorithm.

Our work suggests three distinct admissible strategies while considering building foundations for certain user requirements. We identify 5 valid heuristics: basic (Euclidean, Manhattan, Chebyshev, and weighted Euclidean) and realistic (weighted Euclidean based on one-way or two-way). Also, in order to explore generalizability, we also apply the basic heuristics to maps for other locations. These 5 heuristics deploy different ways of calculating the distance between two points, and we aim to find the optimal heuristic with the lowest cost in time and memory complexity. We will research, test, and analyze them concerning A* search and their effects on the best course of action. To do this, we will create a comparison table of time complexity, space complexity, total cost, and best routes. Additionally, these 5 heuristics will also be covered in the method section.

This issue is not only important, but it also has a lot of intriguing features. The first is that the answer is founded on genuine issues that Minneapolis and St. Paul residents can use themselves. We carefully create the four heuristics to find the optimal way to calculate the distance between two points: on top of Google's consideration of time, this project focuses on the distance. And it provides users with another criterion when using a map in the Minneapolis area. Second, investigating various heuristics helps us comprehend how to develop effective admissible heuristics. A good, acceptable heuristic must be designed to solve a search problem that uses A*. We are confident that the lessons we learned will contribute to the research community by proposing how to design an admissible heuristic in the first place. Also, it serves as the foundational understanding of how distances should be considered and calculated, which allows future work on more complicated heuristics, such as heuristics that imitate real-life traffic and landmark tours.

3 Background

The study of finding a route from a source to a destination is known as pathfinding. One of the most extensively researched problems in computer science is the shortest path problem. The challenge is to find the shortest total weight path between any two nodes in a weighted graph. For different issue variations, several methods have been devised. Undirected versus directed edges are variations. In a labeled graph, each node has one or more descriptions that set it apart from other nodes in the graph. A graph is made up of nodes and the arcs that connect them. Blind search and heuristic search are the two categories of graph search.

Because it lacks topic knowledge, a blind search is also known as an uninformed search. Blind searches can only distinguish between states that are not goals and those that are goals. Heuristic search studies the methods and rules of discovery and development, whereas blind search has no preference for which state (node) may be enlarged next. Heuristics are general guidelines that may help solve an issue but do not always. Heuristics are facts about the subject matter that can direct search and inference in the subject area.

3.1 Depth-First Search Algorithm

The depth-first search algorithm employs a Last-In-First-Out stack and is recursive. If possible, this algorithm expands its search space at that moment. Although it is easy to build, this algorithm's main drawback is that it uses a lot of computer power for a relatively small increase in map size.

3.2 Depth-Limited Search Algorithm

A maximum restriction on the depth of the search allows the Depth-Limited search algorithm to overcome the shortcomings of the Depth-First search algorithm regarding completeness. This method will always discover a solution as long as it is under the depth limit, which ensures that all graphs are at least complete.

3.3 Breadth-First Search Algorithm

A first-In-First-Out queue is used by the breadth-first search algorithm. This method visits each node individually. This breadth-first search method visits nodes according to how far they are from the source node, where distance is determined by the number of edges that have been traveled.

3.4 Best-First Search Algorithm

Like the Breadth-First search algorithm, the Best-First search algorithm limits states using two lists: an open list and a closed list. Best-First search sorts the queue in accordance with a heuristic function at each stage. This approach chooses the unvisited node with the best heuristic value as the next node to visit.

3.5 Hill-Climbing Search Algorithm

The Hill-Climbing search algorithm enlarges the search's present condition and assesses its offspring. It is an iterative method that attempts to discover a better solution by modifying a single element of the current solution after starting with an arbitrary solution to the issue. If the modification results in a superior answer, the new one is incrementally improved, and the process is repeated until no more room for improvement remains. This algorithm's strategy failures cannot be fixed.

3.6 A* Search Algorithm

The A* algorithm is the standard for artificial intelligence search optimization. It uses a best-first search approach to find the most viable solution while eliminating those that could turn out to be inadequate. For it to be effective, H, the heuristic estimate of the separation between the current state and the goal state, must be exact. If the heuristic is valid (does not exaggerate). When D is the distance to a target state, A* may be used to explore a space proportional to D2 on a grid.

3.7 IDA* Search Algorithm

Iterative-deepening A* (IDA*) is an A* that utilizes less memory. It eliminates the open and closed lists by achieving a balance between time and space. Applications frequently have space constraints; hence IDA* is preferred. However, since IDA* iterates and constantly explores paths, this may result in a wasteful search that is asymptotically optimal (e.g., DNA sequence alignment).

4 Literature Review

There has been a significant interest in the computer science field of search algorithms for a long time. There is a limited amount of time or processing power in many applications where search algorithms are used. Searches need to be quick, precise, and effective; deviance from these requirements is considered a major failure. The primary two sorts of searches, informed and uninformed, both have their places but informed searches are by far the more prevalent type. They make use of a heuristic function, an addition to the program that measures the distance from the objective, in order to make better judgments. Although there are numerous sorts of heuristic functions for various search algorithms, we will center our attention on A* search, how well it performs in contrast to other pathfinding methods, its developments, and possible future.

Single-agent best-first search algorithms have been extensively employed to resolve numerous issues. One of the most significant algorithms in this field is A*. The pathfinding research community has long employed the A* search algorithm. Its efficiency, simplicity, and versatility are often mentioned as its virtues compared to other tools. Due to its universality and extensive usage, A* has become a frequent alternative for academics looking to address pathfinding difficulties. A* has been utilized to tackle issues in diverse areas, such as the alignment of numerous DNA sequences in biology, path planning in robotics and digital games, and traditional artificial intelligence problems like the fifteen-puzzle. Despite being extensively utilized, A* may offer complications in particular instances. Firstly, depending on the features of the problem and the heuristics applied, its cost might be exorbitant. Also, some settings, such as dynamic surroundings or real-time searches, may demand adjustments in the original method. As a result, various algorithmic enhancements have been implemented in recent years.

Now, what also has been observed is that A* has not been able to keep up with the demands of modern pathfinding issues. Other methods can keep the same speed while also needing less overhead, and this problem worsens as the grid size rises. However, when working with big maps, A* may reach very quick timings with good accuracy while only having somewhat higher overhead costs because of the application of creative modifications like various heuristic types or supplementary components to the algorithm. Even though it is getting older, updated algorithms built on the traditional A* algorithms are more than capable of keeping up with current pathfinding requirements. To get around A*'s constraints, some studies have employed derivative search algorithms like HPA*. HPA* may compete with and even outperform its opponents depending on the challenge.

Many recent studies in the area also compared the same algorithm without the authors' change to one particular variable to the same algorithm with the enhancement. However, there are additional publications that compare several methods that resolve a certain existing problem. Algorithms that determine the best way to travel to building sites were compared in a study where the visibility graph is the quickest way when there are less than 15 obstacles. It takes quite an acceptable time compared to other approaches for up to 30 obstacles on the site. The Euclidean distance was found to be more accurate than the grid-based method. And in cases when the visibility graph is slow, the Euclidean distance should be used instead.

Dijkstra's algorithm finds an ideal path but is inefficient for large-scale issues. Instead, A* produces optimal and nearly optimal solutions more quickly and effectively owing to its heuristics. A probabilistic optimization strategy based on a genetic algorithm (GA) yields a collection of viable, optimum, and close-to-optimal solutions that captures globally optimal solutions in their test scenarios. In less time, genetic algorithms eventually form the optimum or a solution close to the optimum. Good parts of the search space obtain exponentially more copies and become integrated by the activity of GA operators. In such a scenario, offline learning would be advantageous because it takes a long time to fine-tune the GA to its final state.

There is also a research study where several pathfinding techniques with navigation learning in an electric wheelchair are evaluated. The results claim the evolutionary algorithms applied to a continuous workspace enable acquiring an acceptable path extremely near to the globally optimal path without being dependent on the requisite resolution to adjust the trajectory. In a similar problem to determine the shortest routes between various nodes, several pathfinding methods for unmanned aerial vehicles are compared by examining

many search algorithms and determining that A* is the approach that is most appropriate for their particular situation.

In the domain of what we are trying to solve, much early research work has been done to find the shortest path with various search methods. The basic idea is that the path-finding problem can be extended to any location where the map and coordinate data are available. One interesting way of imagining is the roads in nations where people drive on the right versus countries where people drive on the left. Previous works have inspired researchers to consider user needs for developing heuristics based on many observations and other information.

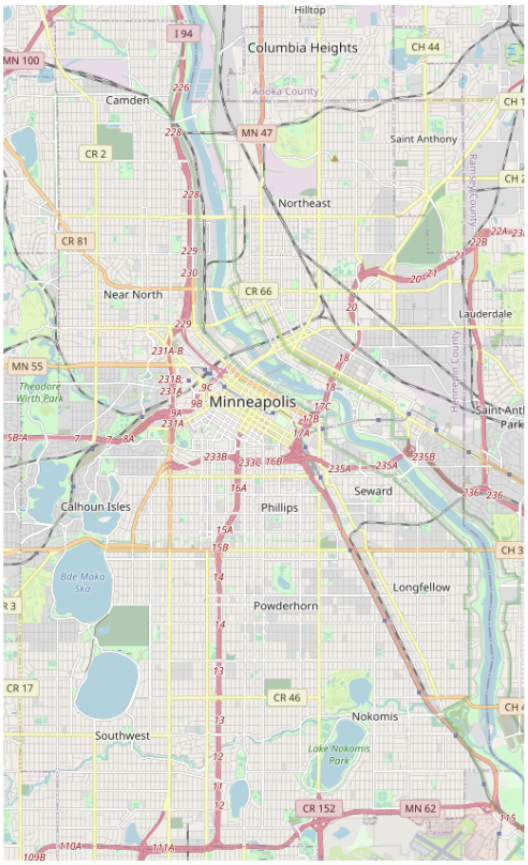

Figure 1: Open Street Map of Minneapolis

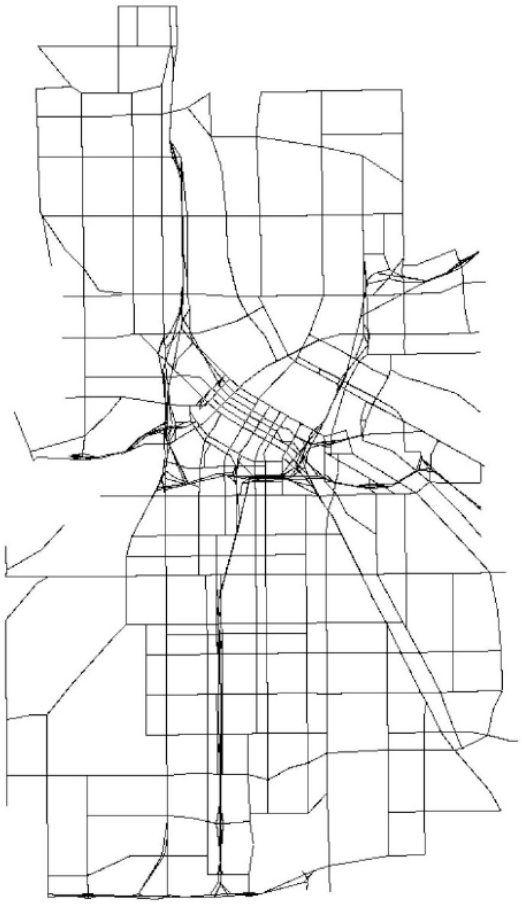

Figure 2: Search Space of Minneapolis

5 Methods

5.1 Dataset

An X and Y coordinate system map of Minneapolis with point coordinates for each road section serves as our dataset. In addition, we have details about the type of road (One-way or Two-way). The Minneapolis map's boundaries are depicted graphically in Figure 1, and its search area is shown in Figure 5.

5.2 Experiment Setup

Our platform is set up on Google Colab with a 2-core Intel(R) Xeon(R) CPU @ 2.20GHz and 13GB of RAM, and we are using Python as our programming language. We utilize the aima-python module in our programming assignments to create the reference code. We make changes to the code to incorporate the numerous heuristics we intended to test.

5.3 Evaluation Metrics

We intend to assess the performance of our variations of A* algorithms using goal tests, branching factor, state count, total path cost, and run-time metrics. The total number of nodes that are enlarged throughout the search will be the branching factor. The total number of states represented by the search space will be known as the state count. The number of times the algorithm determines whether or not the current state is the goal state or whether or not the search has found the goal node would be the number of goal tests. Branching factor, state count, and goal tests are essential indicators for assessing the memory efficiency of the algorithms with different heuristics.

6 Basic Heuristics

Our study will start as the basis with the commonly used heuristics and build upon more specific heuristics that apply to our problem. The commonly used heuristics are the Euclidean distance, Manhattan distance, and Chebyshev distance. We will also compare these commonly used heuristic performances with a weighted heuristic to Euclidean distance ($f = g + 2 \times h$). The heuristics are defined below:

$$\text{Euclidean Distance(x, y)} = \sqrt{\sum_{i=0}^{n} \left(x_i - y_i\right)^2} \quad (1)$$

$$\text{Manhattan Distance(x, y)} = \sum_{i=0}^{n} \left|x_i - y_i\right| \quad (2)$$

$$\text{Chebyshev Distance(x, y)} = max\left\{\left|x_i - y_i\right|\right\} \quad (3)$$

6.1 Result

Since all variants of A* search with commonly known heuristics are optimal, all had the same total path cost of 1.46 * 10^{10}. Coming to the runtimes, the A* algorithm with weighted Euclidean heuristic ran the fastest with a runtime of 34 seconds. Table 9 shows the different runtime comparisons. The goal tests, branching factor, and state count are tabulated in Table 10.

Algorithm	Runtime (in Seconds)
A* with Euclidean Heuristic	38
A* with Manhattan Heuristic	36
A* with Chebyshev Heuristic	41
A* with weighted Euclidean Heuristic	34

Table 1: Runtime comparison among the 4 heuristics

Algorithm	No. of Goal Tests	Branching Factor	State Count
A* with Euclidean Heuristic	447888	446942	1313767
A* with Manhattan Heuristic	447877	446931	1313736
A* with Chebyshev Heuristic	447928	446982	1313886
A* with weighted Euclidean Heuristic	447850	446904	1313664

Table 2: Comparison of memory efficiency metrics of A* with 4 heuristics

6.2 Discussion

Table 1 shows that A* with weighted Euclidean Heuristic had the fastest runtime in seconds. This result demonstrates that with the given Minneapolis map, A* with weighted Euclidean Heuristic has the best

performance among Euclidean Heuristic, Manhattan, Chebyshev, and weighted Euclidean. This result also gives us insights about choosing between these 4 heuristics: when the user wants to prioritize the runtime, weighted Euclidean should be selected. On top of that, the rank of runtime efficiency is: weighted Euclidean, Manhattan, Euclidean, and Chebyshev. This result makes sense intuitively, as Chebyshev has the more sophisticated calculations among the 4 heuristics.

When it comes to additional analysis in terms of memory efficiency, the results from Table 2 appear to be consistent with the results from Table 1. A* with a weighted Euclidean heuristic has the lowest number of goal tests, branching factors, and state counts. This result implies several things. First, it means that the weighted Euclidean heuristic enlarged the least number of nodes throughout the search with the Minneapolis map. Second, weighted Euclidean uses the least number of representation states in the search space. Last. Weighted Euclidean checks the least number of current states to determine the goal states. We also note that the rank of the 4 heuristics is consistent with results in Table 1: weighted Euclidean, Manhattan, Euclidean, and Chebyshev across all measures in terms of the number of goal tests, branching factors, and state counts. This confirms that weighted Euclidean should be selected when the user wants to prioritize memory efficiency, which leads to the conclusion that weighted Euclidean is preferred in both runtime and memory complexity.

7 Realistic Heuristic

From the Basic Heuristic section, we are able to conclude that the weighted Euclidean heuristic produced the optimal solution in terms of both runtime and memory efficacy for the Minneapolis map. Our second heuristic, therefore, extends from weighted Euclidean and considers the road as one-way or two-way. This heuristic will specifically satisfy users who need to account for road conditions in real life. We reasonably assume that two-way roads take more time than one-way roads because two-way roads could potentially make drivers wait longer than one-way roads, especially for left turns. Therefore, the realistic heuristic rewards one-way roads on the map by multiplying the weighted heuristic by 0.9, while it penalizes two-way roads on the map by multiplying the weighted heuristic by 1.1. The formula is demonstrated below:

$$f = \begin{cases} 0.9 * (g + 2 * h) \text{ if the road is one-way} \\ 1.1 * (g + 2 * h) \text{ if the road is two-way} \end{cases} \quad (4)$$

7.1 Result

Same as the basic heuristic, the realistic heuristic also had the same total path cost of $1.46 * 10^{10}$, which proves that it is an optimal heuristic. For the purpose of this heuristic, we will only compare it with the original weighted Euclidean heuristic. Table 3 shows the different runtime comparisons. The goal tests, branching factor, and state count are tabulated in Table 4.

Algorithm	Runtime (in Seconds)
A* with weighted Euclidean Heuristic	25.52
A* with realistic weighted Euclidean heuristic	25.52

Table 3: Runtime comparison between original weighted Euclidean and realistic heuristics

Algorithm	No. of Goal Tests	Branching Factor	State Count
A* with weighted Euclidean Heuristic	447850	446904	1313664
A* with realistic weighted Euclidean heuristic	447850	446904	1313664

Table 4: Comparison of memory efficiency metrics of A* with 2 heuristics

7.2 Discussion

Both result tables show that the updated realistic heuristic has the same runtime and memory as the original weighted Euclidean heuristic. Both heuristics report runtime of 25.52 seconds and 447850 goal tests, 446904 branching factors, and 1313664 state counts. This is desirable, as we successfully satisfy a different user need than the basic heuristics without hurting the program's efficiency. This is important, as we illustrate that it is possible to maintain efficiency and user satisfaction without any additional cost. However, some limitations could be addressed in future work, and we will discuss them in the next section.

8 Results on Other Maps

We would like to show the performance of the A* algorithm with the mentioned heuristics on other locations. For this purpose, Saint Paul, which is the sister city of Minneapolis, has been considered along with a couple of other big cities in the USA. The data was extracted using Open Street Map's overpass API[8]. A highway level similar to that of the map of Minneapolis was chosen.

8.1 Saint Paul, MN, USA

Saint Paul has an infrastructure similar to that of Minneapolis, which means that the edge and node densities are comparable, though the exact number of nodes differ significantly due since the data was collected using a slightly different method. The city also has latitude boundaries that are close to that of Minneapolis.

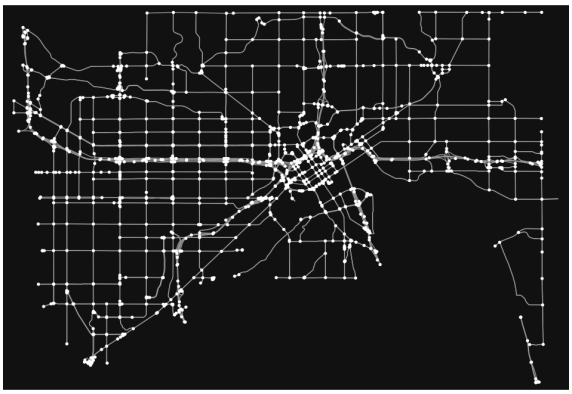

Figure 3: Graph of Saint Paul

Heuristic	Runtime (in Seconds)
Euclidean	56.6
Manhattan	51.3
Chebyshev	58.5
Weighted Euclidean	49.7

Table 5: Runtime comparison among the 4 heuristics

Heuristic	No. of Goal Tests	Branching Factor	State Count
Euclidean	561722	561637	1742126
Manhattan	561697	561643	1742088
Chebyshev	561779	561841	1742192
Weighted Euclidean	561682	561585	1742010

Table 6: Comparison of memory efficiency metrics of A* with 4 heuristics

8.2 New York City

New York City differs from the twin cities in that it the node density is significantly higher while the edge density is comparable. Also, the graph can almost be divided into sub-graphs without severing too many edges due to the city being situated on a couple of islands for the most part. Due to this, the memory usage of the A* algorithm was slightly lower than what it could have been, given the amount of nodes.

Figure 4: Graph of New York City

Heuristic	Runtime (in Seconds)
Euclidean	187.7
Manhattan	175.3
Chebyshev	201.1
Weighted Euclidean	171.0

Table 7: Runtime comparison among the 4 heuristics

Heuristic	No. of Goal Tests	Branching Factor	State Count
Euclidean	2027816	2027542	6289074
Manhattan	2027712	2027566	6288933
Chebyshev	2028042	2028254	6289310
Weighted Euclidean	2027651	2027337	6288656

Table 8: Comparison of memory efficiency metrics of A* with 4 heuristics

8.3 San Francisco

San Francisco had a graph whose features were more of a blend between New York City and the Twin Cities. Only the paths utilized for driving are being considered since the city's public transport system was making the graph too complex for the study.

Figure 5: Graph of San Francisco

Heuristic	Runtime (in Seconds)
Euclidean	90.3
Manhattan	82.8
Chebyshev	95.8
Weighted Euclidean	77.6

Table 9: Runtime comparison among the 4 heuristics

Heuristic	No. of Goal Tests	Branching Factor	State Count
Euclidean	1247022	1246834	3867519
Manhattan	1246967	1246847	3867435
Chebyshev	1247149	1247287	3867666
Weighted Euclidean	1246934	1246718	3867262

Table 10: Comparison of memory efficiency metrics of A* with 4 heuristics

9 Conclusion

Pathfinding has been a core topic in search algorithms, and its applications are utilized in many different areas. Several search algorithms include depth-first search, depth-limited search, breadth-first search, best-first search, hill-climbing search, and A* search. This project focuses on A* search, applies and compares four different heuristics on the Minneapolis map, and finds the optimal heuristic regarding runtime and memory usage. The four different heuristics are respectively Euclidean, Manhattan, Chebyshev, and weighted Euclidean. After running a modified A* search, we found that weighted Euclidean is the optimal heuristic in both runtime and memory. The rest three heuristics rank in the order of Manhattan, Euclidean and Chebyshev. Future work is discussed in the next section.

10 Future work

There are mainly two directions for future work. On the one hand, the examination of generalization is appreciated. Our project is limited because we only used several regional maps to run the experiment, which is a small portion of the world map. Therefore, to conclude that the result from our project is generalizable, future work could apply the algorithms to maps of larger regions like counties or even whole states. However, the study could limit to motorways alone by discarding the smaller highways, for example. If the result is the same, we will be able to conclude that the results are generalizable, which is an essential consideration in research.

On the other hand, our research serves as a foundation for the optimal heuristic for distance calculation; future work could develop more sophisticated heuristics involving distance. For example, a realistic heuristic that considers right turn and left turn can develop the heuristic based on the weighted Euclidean heuristic. In this way, the innovative heuristic would be more likely to be optimal.

11 Author Contributions

1. Silas Kati - Worked on all of the Abstract, Introduction, Background, Literature Review, and the experimentations, results, analysis of the common heuristics of the A* algorithm (Euclidean, Manhattan, Chebyshev, and weighted Euclidean), and references. Fixed formatting and ordering in LateX.

2. Moyan Zhou - Worked on Problem Statement, Method, Discussion for basic heuristic, all sections in realistic heuristic and Conclusion, and Future Work. Also moved the project from Google Docs to Overleaf for Latex.

3. Sai Tarun Inaganti - Extracted data for the cities of Saint Paul, New York City, and San Francisco and compared the path graphs with that of Minneapolis, adjusted edge densities, added map images, ran the common heuristics on each map, made minor modifications to Future Work and some other places, fixed figure ordering issues in Latex.

References

[1] Sharmad Rajnish Lawande, Graceline Jasmine, Jani Anbarasi, and Lila Iznita Izhar. A systematic review and analysis of intelligence-based pathfinding algorithms in the field of video games. *Applied Sciences*, 12(11):5499, May 2022.

[2] Estela Pogaçe and Dimitrios A. Karras. On a* graph search algorithm heuristics implementation towards efficient path planning in the presence of obstacles. *International Journal of Innovative Technology and Interdisciplinary Sciences*, Vol. 5 No. 4:1033–1051 Pages, Nov 2022.

[3] Daniel Foead, Alifio Ghifari, Marchel Budi Kusuma, Novita Hanafiah, and Eric Gunawan. A systematic literature review of a* pathfinding. *Procedia Computer Science*, 179:507–514, 2021.

[4] Safa Belhaous, Sohaib Baroud, Soumia Chokri, Zineb Hidila, Abdelwahab Naji, and Mohammed Mestari. Parallel implementation of a search algorithm for road network. In *2019 Third International Conference on Intelligent Computing in Data Sciences (ICDS)*, page 1–7, Marrakech, Morocco, Oct 2019. IEEE.

[5] Victor Martell and Aron Sandberg. *Performance Evaluation of A* Algorithms.* 2016.

[6] Ayoub Bagheri, Mohammad-R Akbarzadeh-T, and Mohamad Saraee. Finding shortest path with learning algorithms. *International Journal of Artificial Intelligence [electronic only]*, 1, 01 2008.

[7] Shrawan Kumar Sharma and B. L. Pal. Shortest path searching for road network using a* algorithm. *International Journal of Computer Science and Mobile Computing*, 4(7):513–522, Jul 2015.

[8] Md Kaisar Ahmed. Converting openstreetmap data to road networks for downstream applications, 2022.