Shortest Path Problem with Common Search Algorithms

Cho, Chloe
cho00035@umn.edu

Kim, Hyunwoo
kim00186@umn.edu

Cho, Yongsuk
cho00085@umn.edu

May 2021

Abstract

This paper aims to find the shortest path on a map and compare it using various search algorithms to see how map sizes affect how the search algorithm performs. Our maps are constructed using coordinates of the Minneapolis intersection coordinates and are plotted via directed and undirected edges connected to make the graph. The algorithms used in this exploration are informed and uninformed search algorithms, the algorithms that will be used are Breadth-First Search, Depth-First Search, Uniform Cost Search, and A* Search. The different algorithms will be compared with respect to their path cost, successors, goal test, and states on different map size which is measured by the number of nodes on the map. Based on the results obtained from these searches, we expect to reveal the algorithms needed to construct the most optimized map navigation.

Keywords

**Search algorithm, *Heuristic, *Informed Search, *Uninformed Search, *Breadth-First Search, *Depth-First Search, *Uniform Cost Search, *A* Search, *Directed graph, *Undirected graph, *Shortest path problem, *Map navigation*

1 Introduction

There are many search algorithms to find the shortest path from the start to the end location. When using a map or navigation system, using any search algorithm is not always the most efficient in finding the shortest path. Therefore, we want to compare the most efficient and optimized algorithms to find the shortest path, and accordingly, we have an uninformed search and information search. Uninformed searches are known to be blind searches and this is more efficient when the path ahead is unknown. Informed searches are known to be heuristic searches and this is best used when knowing the distances of paths.

In this project, our team will propose a study that will be based on various search algorithms learned from the course CSCI4511W - Introduction to Artificial Intelligence by Maria Gini.

The project will have a strong impact on the shortest path problem for three reasons:

- The use of map navigation will be used to find the shortest path in a map with one-way and two-way streets.
- The use of map navigation provides a reliable model that even takes into account cost savings, providing economical functionality.
- Map navigation is based on a model that distinguishes the cost from the origin to the destination and one-way and two-way, so it can be extended indefinitely to other models by adding different considerations.

From this project, we expect that one of the best models from the informed and non-informed search algorithms used to find the shortest path problem can serve as map navigation by standardizing the shortest path. In other words, a directed graph describes a path that is bounded and is only able to move one-way to a given path, while undirected describes a path that is non-routine and can be moved to two-way.

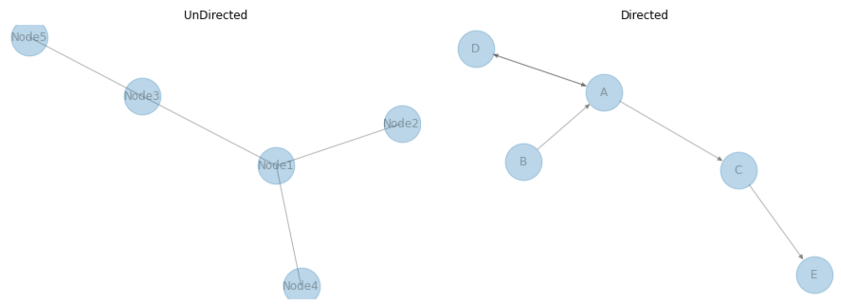


Figure 1: Undirected and Directed Graph

In Figure 1, it briefly shows the graphs separated by one-way and two-way edges which represent directed and undirected graphs respectively.

2 Literature Review

Searches are used to help find a path from a starting for to an end. “Before performing operations in a real-world environment, the agent simulates the order of ACTIONS in the model and searches until it finds the order of operations that reach the target. These sequences are called SOLUTION. The agent may need to simulate multiple sequences that have not reached its GOAL, but eventually, it finds a solution [9].” With search algorithms according to the search logic of Russell, we will compare ACTIONS up to GOAL for optimal SOLUTION.

The shortest path problem can be solved with the A* algorithm. The heuristic function should evaluate two costs, g and h. In the shortest path problem, g(n) represents the cost of selecting a path from the starting node to node n, and h(n) represents the optimal cost of node n for the target node. The cost of node n is given by f*(n) =g(n)+ h*(n). In most cases of shortest path problems, however, the value of h*(n) is unknown in most situations, so the value of f*(n) is unknown. However, the A* algorithm drives the best approximation for h*(n) [8].

2.1 Understanding Algorithms

Informed searches and uninformed searches are used to find a solution for the shortest path between points. Informed search is when heuristics are used to help estimate the path. While uninformed search is where there is no given estimate and the path is searched blindly. In addition, each search algorithm is optimized to select a progress path based on factors such as speed or cost, while other non-information retrieval algorithms may not perform a complete search or may not find a progress path. For full search performance and better understanding, prior to the experiment we will describe each algorithm in this part and explain what elements they are optimized for.

2.1.1 Uninformed search algorithms

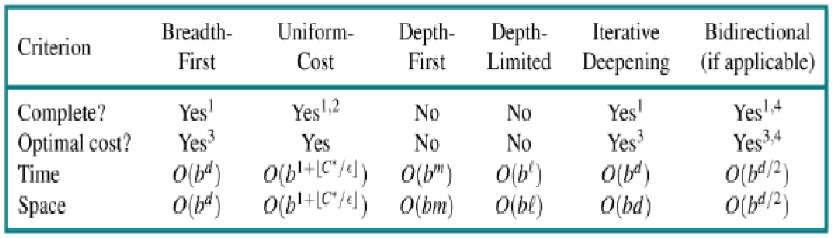
1. Breadth-First Search (BFS) - Breadth-first search starts searching from the start node of the graph then expands all successor nodes at the current level before moving to nodes in the next level [4].
2. Depth-First Search (DFS) - Depth-first search starts from the start node and goes through the path of the graph until the greatest depth node is reached before moving to the next path [4].
3. Depth-Limited Search - A depth-limited search is similar to DFS but it had a predetermined limit. Depth limit will go until there are no successor nodes left [4].

4. Iterative Deepening Depth-First Search - The iterative deepening algorithm finds the best depth limit and does it by gradually increasing the limit until a goal is found [4].

5. Uniform Cost Search - The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost. Uniform-cost search expands nodes according to their path costs from the root node [4].

6. Bidirectional Search – There are two simultaneous searches happening, one from the initial state which does a forward search, and the other from the goal node which calls a backward search. The search stops when these two graphs intersect each other [4].

As such, each uninformed search algorithm has its own advantages and disadvantages, each with its best case and its worst case. In the case of Depth Limited Search, if the solution is not above the depth-limit, this causes incompleteness and it is not optimal if the problem has more than one solution. In the case of Breadth-first-Search, it first scans of the width, the frontiers are implemented as queues that act as FIFO (First In First Out). This is a bad strategy if all solutions have access to long-length or some empirical information [2].



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^1	Yes^{1,2}	No	No	Yes^1	Yes^{1,4}
Optimal cost?	Yes^3	Yes	No	No	Yes^3	Yes^{3,4}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Figure 2: Comparing Uninformed Search Algorithms [9]

2.1.2 Informed search algorithms (Heuristic search)

While uninformed searches do not have knowledge of the paths ahead, informed searches are able to estimate the paths based on heuristics.

1. Best First Search (Greedy Search) - Greedy best-first search always selects the path that seems to be the best at the moment using the heuristic function and search [4]. It expands the node that is closest to the goal node and the closest cost is estimated by the heuristic function.

2. A* Search Algorithm - A* search ranks to the shortest path through the search space using the heuristic function [8]. This search algorithm expands fewer search tree and provides optimal results faster. A* algorithm is similar to uniform cost search except that it uses g(n) + h(n) which includes the heuristic. The A* algorithm combines the highlights of uniform cost and pure empirical approach searches to productively calculate the optimal solution [2]. An Admissible heuristic function (h(n) $\leq$ h*(n) returns a visited node from a closed list to an open list to obtain an optimal solution. Time complexity is O(bd) and space complexity is O(bd). where b is the branch factor and the depth of the solution [7].

Best First Search is the integration of Breadth-First Search and Depth-First Search implemented by the priority queue [8]. Therefore, the advantage of Depth First Search is that it provides a solution without searching all nodes. Depth First Search ensures that procedures are not performed while the solution is reached without a search. The best node selection generated at each step is easy to scale, can be at similar levels or different locations, and can switch between depth-first search and Depth-first search. It is also known as greedy search. Time complexity is O(bd) and space complexity is O(bd), where b is branching factor and d is solution depth [7].

Figure 3 describes the concept of the A* algorithm, which is a weighted graph that continues to merge with low h(n) values and is combined with g(n) values.

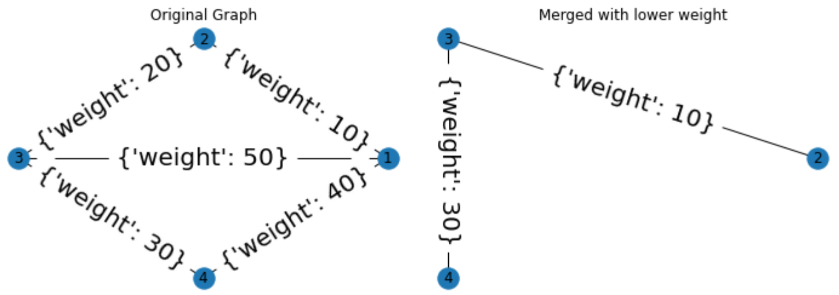


Figure 3: The Weighted Graph

2.2 Analysis of Algorithm Optimization

When comparing each algorithm, there are different methods of obtaining optimal values through each node. Each algorithm will be using its methods to find its own optimal or optimized method. From the research by Gary Michael Taylor, it was explained the differences between various search algorithms through analysis [**?**]. The research that was conducted, used limited maps that were supported by a 2D grid method to a variation of several tests to compare whether the given reference is accurate and reasonable with a clearer result table for the given data. For a given map there is information about the distances at each intersection and whether it is a two-way or one-way road, this article tries to obtain the optimal shape of the path by giving each direction to the problem nodes that can go wrong. By looking at this research, when working with algorithms to find the shortest and quickest path it is going to help decide how to handle the scale of each path on the map. Also, this can help solve problems like traffic since it is not as simple of a problem as finding the shortest road, but rather other factors need to be taken into account. Some factors would be a high-speed road like an intersection, a low scale like the straight road where the signal connects at the next signal, and a high road like left and right turns.

2.3 Heuristics in Navigation Systems

After looking at the different search algorithms, it can be seen that A* will be the most optimal to finding the shortest path at the most optimal time. For calculating the shortest path, A* search algorithm which uses heuristics to help find an optimal solution. Heuristics are estimated distances from the current node to the end node[1]. Using A* search is very much ideal for navigation systems, maps, and any path-finding problems because when looking for a path it is important to find the shortest path. A* search allows for this because of heuristics. Being able to use heuristics allow for path-finding be more optimal since it is an estimated cost that allows for the search to be more informed about the shortest path [9]. Heuristics are important for navigation systems because there is a lot of different factors that need to be accounted for that heuristics help with estimating the path. It is beneficial because for large maps with many nodes, the uninformed searches will have to explore more paths to get to the end goal compared to an informed search since searches like A* prune paths are not optimal path cost. Also with A*, it is cost-optimal when the heuristic is admissible which means it never overestimates the cost to reach the end node [9].

An example of how heuristics are used is when driving, traffic is a big factor that needs to be accounted for when calculating the time that it takes to get to a certain destination. For calculating the shortest path, A* search algorithm which uses heuristics to help find an optimal solution. Heuristics are estimated distances from the current node to the end node[1]. For navigation systems, it is not as simple as just getting from start to end judging only by the distance, there is traffic that needs to be accounted for. That is why Halaoui suggested a variation of A* search called A* traffic, which accounts for the delay of traffic. The new heuristic function becomes F = TH + TG + TD. TH is the average time needed to get from one place to another. TG is the time of the distance that has been covered. TD is the traffic delay which is calculated by adding all the delays of the streets that form the path[3]. From the results of testing the algorithm, it was found that the solution was optimal for shorter distances [3].

After looking at different literature references to help understand the research at hand, there was a lot learned about different search algorithms as well as how heuristics can be used in navigating through the streets. For the search algorithms, the advantages and disadvantages of the different uninformed and informed search algorithms can be shown through the different sources. While researching, it was found that the algorithm that will help with path-finding for a map is A* search. This is because in a map the path distance is known and can be easily estimated since paths that are not the best can be eliminated. This allows for the path to be found faster. However, navigating to a destination in the real world is not as simple as getting from point A to point B. Also, from researching and reading through different sources, traffic is a huge component that needs to be accounted for when calculating the time. That is why a variation of A* was suggested to help account for the delays caused[3]. From reading these different literature pieces and writing this review, it can help our research to use heuristics to find the shortest path as well as use the knowledge of different search algorithms to prove that A* search is the most optimal. With these algorithms, there can be other scenarios besides using maps that can be explored with using search algorithms to find the shortest path. While our research does not touch on the effects of traffic and stop lights with A* algorithms, it can be seen how heuristics take a role in finding the shortest path as well as the performance when using a map. This is seen through how uninformed searches differ from informed searches.

3 Experiment and Results

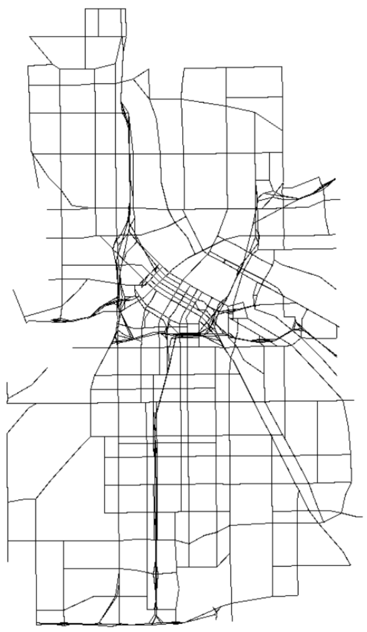


Figure 4: The result of drawing the Minnesota Map.csv

3.1 Data Logistics and Pre-processing

To find the shortest path, we used the Minneapolis map shown in the above figure provided in the CSCI 4511W course which is drawn out to be shown in Figure 4. The provided file has a Way representing one-way or multi-directional in the first column, and the second to fifth columns represent the latitude and longitude of each node consisting of x and y coordinates. The data file with 1358 rows can consist of a total of 2716 nodes. Each x and y were named as one node, and the distance was calculated to connect each node in a dictionary format. A map of different sizes consists of a map with 120 nodes and a map with 948 nodes, limiting the number of nodes.

Figure 5 can be seen connecting nodes based on the calculated distance. The program that was used did not have a way of modifying the spacing of the nodes so it is all clumped up but a better representation of the map can be seen in Figure 4.

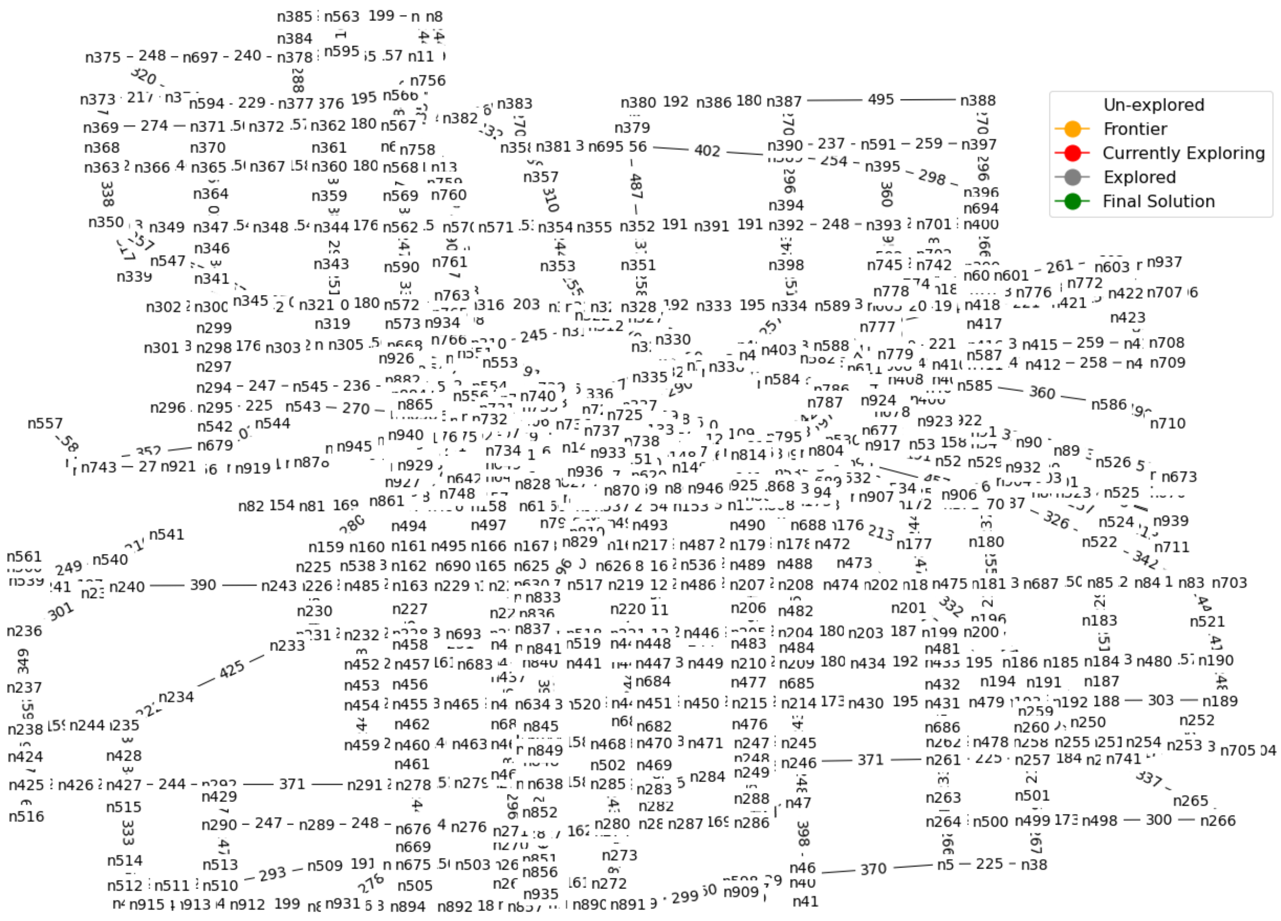
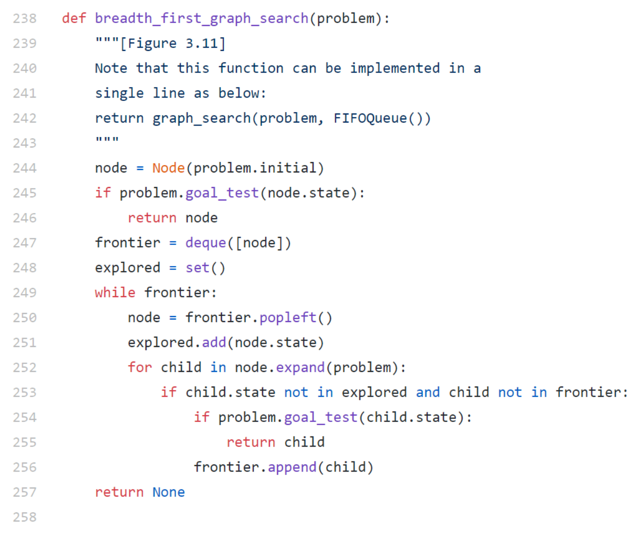


Figure 5: Visualisation of the Minnesota Map.csv

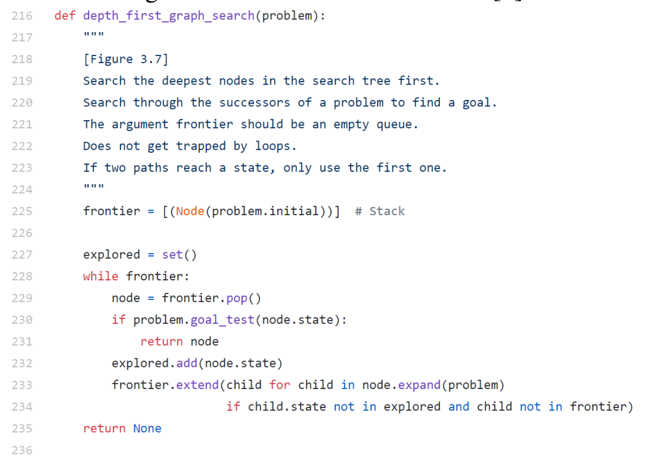
3.2 Experiment

To go off the literature review, there are informed and uninformed search algorithms so this experiment will further explore the idea that A* is the most optimal for finding the shortest path as well as comparing how map size affects the outcome of the algorithms. For the algorithms, the AIMA-python GitHub code, which can be found here [6], was modified to take in the data of the CSV file to help create a dictionary of all the points in the map. The code was then run on Google Colab Notebooks. The map with all the data will be represented as a graph with each intersection being a node. When the program was run, 946 nodes were created which will represent the city of Minneapolis. To compare the different data of the search algorithms on the map, there is a compare searcher function that prints information about the path cost, successor, goal test, and state. The path cost is the total cost of each action along the path which is the total distance of the path found which will be used to see what the shortest path is. The successor, goal test, and state are used to help measure the performance of the algorithm. The goal test is based on the number of times the algorithm checks if the current node is a goal node. States are the number of states that are generated. The reason why the successor was the main comparison measurement is that it showed significantly different results than other performance measurements. It also computes all the nodes that have been explored, so it can show the results based on the characteristics of each algorithm.



```python
def breadth_first_graph_search(problem):
    """[Figure 3.11]
    Note that this function can be implemented in a
    single line as below:
    return graph_search(problem, FIFOQueue())
    """
    node = Node(problem.initial)
    if problem.goal_test(node.state):
        return node
    frontier = deque([node])
    explored = set()
    while frontier:
        node = frontier.popleft()
        explored.add(node.state)
        for child in node.expand(problem):
            if child.state not in explored and child not in frontier:
                if problem.goal_test(child.state):
                    return child
                frontier.append(child)
    return None
```

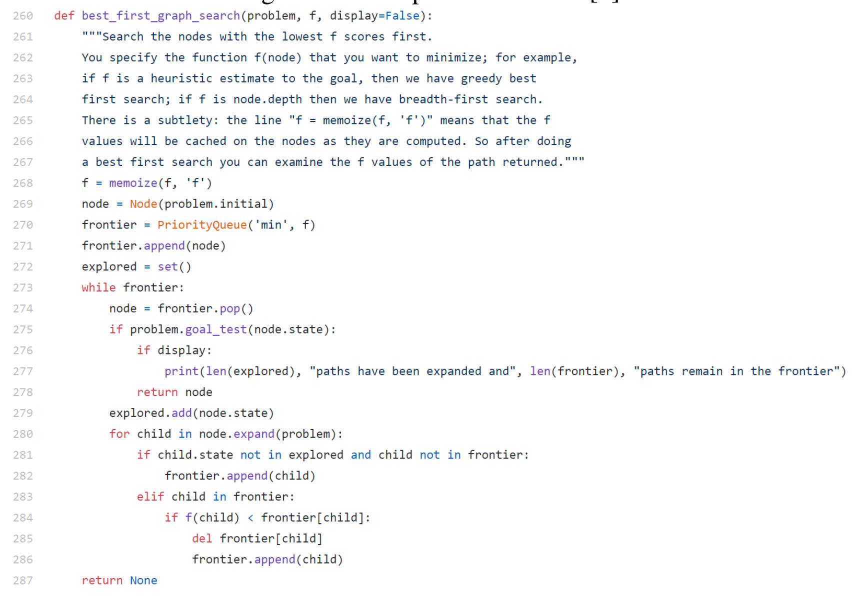
Algorithm 1: Breadth-First Search [6]



```python
def depth_first_graph_search(problem):
    """
    [Figure 3.7]
    Search the deepest nodes in the search tree first.
    Search through the successors of a problem to find a goal.
    The argument frontier should be an empty queue.
    Does not get trapped by loops.
    If two paths reach a state, only use the first one.
    """
    frontier = [(Node(problem.initial))]  # Stack

    explored = set()
    while frontier:
        node = frontier.pop()
        if problem.goal_test(node.state):
            return node
        explored.add(node.state)
        frontier.extend(child for child in node.expand(problem)
                        if child.state not in explored and child not in frontier)
    return None
```

Algorithm 2: Depth-First Search [6]



```python
def best_first_graph_search(problem, f, display=False):
    """Search the nodes with the lowest f scores first.
    You specify the function f(node) that you want to minimize; for example,
    if f is a heuristic estimate to the goal, then we have greedy best
    first search; if f is node.depth then we have breadth-first search.
    There is a subtlety: the line "f = memoize(f, 'f')" means that the f
    values will be cached on the nodes as they are computed. So after doing
    a best first search you can examine the f values of the path returned."""
    f = memoize(f, 'f')
    node = Node(problem.initial)
    frontier = PriorityQueue('min', f)
    frontier.append(node)
    explored = set()
    while frontier:
        node = frontier.pop()
        if problem.goal_test(node.state):
            if display:
                print(len(explored), "paths have been expanded and", len(frontier), "paths remain in the frontier")
            return node
        explored.add(node.state)
        for child in node.expand(problem):
            if child.state not in explored and child not in frontier:
                frontier.append(child)
            elif child in frontier:
                if f(child) < frontier[child]:
                    del frontier[child]
                    frontier.append(child)
    return None
```

Algorithm 3: Best-First Search that is used for A* and Uniform-Cost Searches [6]

For this research, the searches that were used are breadth-first search (Algorithm 1), depth-first search (Algorithm 2), uniform-cost search, and A* search. The reason for these particular algorithms is because they are a mix of uninformed and informed searches which is what our paper focuses on. For uniform-cost search and A* search, the algorithm in the AIMA code [6] uses best-first search (Algorithm 3) where uniform-cost uses weights and A* uses heuristics. There are several ideas that will be explored with these different search algorithms. The data sets that were used to help conduct this research were chosen so that the results give a better understanding of the differences in performance and path cost for each algorithm and how map size also makes a difference.

The first idea that was explored is how the search algorithms are affected by the size of the map. The small size map was created with about 120 nodes which are significantly less than the number of nodes present in the entire map of Minneapolis. For the tests, there were 3 start and end data points that were tested on both the small and big maps where the results for the data captured are shown in Table 1. The data shown are the same nodes being compared, but there is a clear difference between path cost and how much each node passes. It can be seen that the path cost for each algorithm ran is the same values and even the performance of the successor, goal test, and states all do not have too much of a difference in the values.

As seen in Table 1, the resulting values for the small map data do not show an impact difference between each search algorithm. Since there is not much of a difference in the results, each algorithm was tested on the Minneapolis map with the greater amount of nodes with greater with the same data as the smaller map. From the result can be seen that there is a slight increase in the performance of BFS, uniform-cost, and A* search but there is a huge increase in the path cost as well as the performance for DFS.

This then leads to the last data set that is being compared, which is how breadth-first search and depth-first search are affected by where the start and end nodes are. Based on the x and y coordinates the performance of the breadth-first search and the depth-first search will differ. The start and end nodes chosen to be tested can be seen in Table 2 and 3. For BFS, it can be seen that in Figure 7 it had greater values than DFS. While in Figure 8, it is the opposite and DFS has greater values than BFS. Also, another interesting thing about the results that can be seen is that the uniform-cost search is always similar to the performance of BFS is because uniform-cost search is similar to BFS but instead uniform-cost will search path with lowest path cost instead of just going level by level.

4 Analysis Result

After running the experiment we can now analyze the results deeper to see the effects of search path cost and performance with map size as well as the use of the different search algorithms.

To make this experiment fair, we set different size of node with search algorithms. The accuracy we compared is as shown in below table:

Map Size Small vs Big Test Data (`succs/goal_tests/states/path_cost`)						
Node -> Node Start (x,y) End (x,y)	5 -> 47 (2351, 5288) (1976, 5670)		5 -> 39 (2351, 5288) (1901, 5194)		5 -> 41 (2351, 5288) (1995, 5062)	
Map Size	Small	Big	Small	Big	Small	Big
`breadth_first`	6/ 8/ 12/ 816	7/ 14/ 19/ 816	2/ 4/ 6/ 461	2/ 5/ 7/ 461	5/ 7/ 10/ 532	6/ 13/ 17/ 532
`depth_first`	4/ 5/ 9/ 816	648/ 649/ 1808/ 36016	7/ 8/ 13/ 461	643/ 644/ 1794/ 36635	6/ 7/ 12/ 532	945/ 946/ 2624/ 532
`uniform_cost`	7/ 8/ 13/ 816	20/ 21/ 49/ 816	5/ 6/ 11/ 461	8/ 9/ 20/ 461	6/ 7/ 12/ 532	10/ 11/ 27/ 532
`astar`	3/ 4/ 8/ 816	5/ 6/ 14/ 816	2/ 3/ 6/ 461	2/ 3/ 7/ 461	3/ 4/ 8/ 532	3/ 4/ 9/ 532

Table 1: Map Size Small vs Big Test Data.

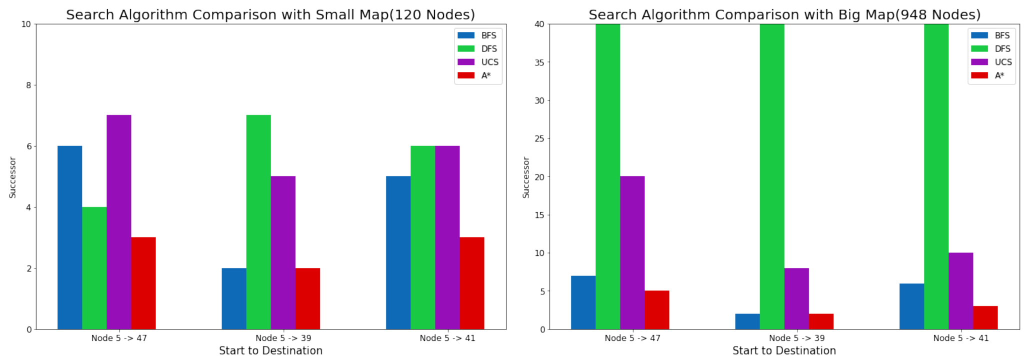


Figure 6: Visualisation of the Small Map and Big Map Search Test from Table 1

The map size table shows that many of the algorithms differ when searching for the same node. In the case of the data in Table 1, where the map is small, the number of nodes is significantly smaller, and in the case of large maps, the number of passes on most nodes is significantly increased. This is a process that nodes check and pass according to the characteristics of each algorithm, so the more complex the map becomes, the more things nodes need to consider differently than before, showing the data that differs in the size of the map.

Difference in Y Coordinate Test Data (`succs/goal_tests/states/path_cost`)			
Node->Node Start (x, y) End (x, y)	373 -> 915 (188, 9994) (315, 5044)	375 -> 266 (202, 10252) (3045, 5561)	516 -> 385 (7, 5588) (690, 10500)
`breadth_first`	913/ 930/ 2544/ 810	941/ 945/ 2614/ 1755	841/ 875/ 2362/ 1323
`depth_first`	515/ 516/ 1465/ 14177	565/ 566/ 1601/ 6435	416/ 417/ 1202/ 2028
`uniform_cost`	920/ 921/ 2567/ 810	905/ 906/ 2527/ 1573	943/ 944/ 2618/ 1323
`astar`	255 /226 /625/ 810	158/ 159/ 474/ 1573	173/ 174/ 487/ 1323

Table 2: Difference in Y Coordinate Test Data

Comparing Y Coordinate Test tables shows unusual results in typical situations. These results show that, unlike other test data, the depth-first search algorithm passes significantly fewer nodes and is derived at the destination. The above data is that the deep-first search algorithm searches down the path depth before going to the next path with is different from X Coordinate test table. The data shown in Table 2, shows how having the start and end nodes with a similar x coordinate and further y coordinate allow for better performance when looking at successor, goal test, and state even if it does not find the shortest path. As a result, these results are shown in directions for two nodes that are not entangled with complex nodes.

Contrary to the Y Coordinate Test, the X Coordinate Test shown in Table 3 again shows that depth-first-search has to pass many nodes. This is because with BFS there it searches level by level so if the end goal has a similar y coordinate and further x coordinate then the search for BFS will find it quickly since it is closer in level while DFS needs to go through the path by path before getting to the end goal. The other data shows the smallest path cost possible and better performances for algorithms such as A* search. For all the data sets, the reason for uniform-cost having the same path cost but worse performance is because the search is uninformed so it has no heuristics to estimate where the next best path. That is why it will have to search more nodes to see where the smaller path cost is.

The above test data show that the A* search algorithm invariably passes the fewest nodes with the fewest path costs [5]. Contrary to the initial conceived expected results, other search algorithms also performed similarly, resulting in another comparison value comparing the speed of each search algorithm can search with the number of nodes first. This is to establish paths and compare the memory or time spent in searches that show fast and direct paths, even if each device does not perform

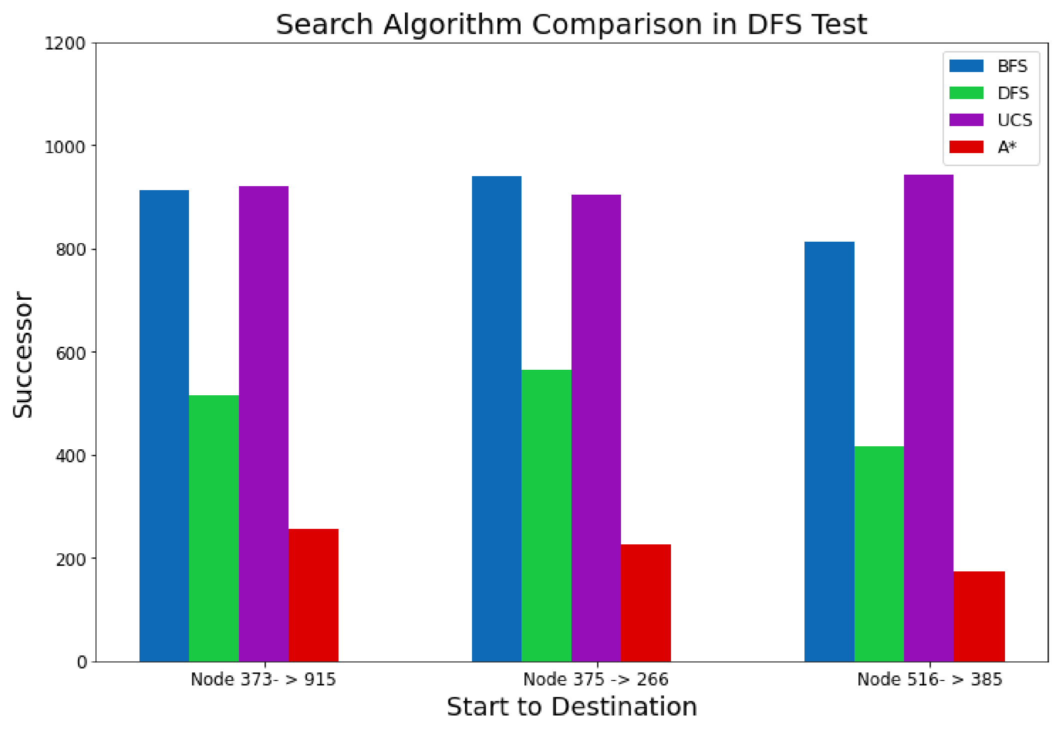


Figure 7: Visualisation of the Difference in Y Coordinate Test Data from Table 2

Difference in X Coordinate Test Data (`succs/goal_tests/states/path_cost`)			
Node->Node Start (x, y) End (x, y)	375 -> 11 (202, 10252) (1012, 10256)	232 -> 481 (862, 6701) (2344, 6619)	450 -> 189 (1729, 6285) (3052, 6296)
`breadth_first`	13 /20/ 35/ 6300	290/ 340/ 812/ 7071	133/ 172/ 390/ 6101
`depth_first`	864/ 865/ 2413/ 14177	890/ 891/ 2480/ 6435	928/ 929/ 2575/ 2028
`uniform_cost`	18/ 19/ 48/ 810	299/ 300/ 843/ 1573	195/ 196/ 567/ 1323
`astar`	5 /6 /13 810	26/ 27/ 82 1573	9/ 10/ 29 1323

Table 3: Difference in X Coordinate Test Data

well. As a result, we were able to get the photos and data in the test section. This allowed us to cross-represent the superiority and expected results of the initially conceived A* algorithm.

5 Conclusion and Future Works

Through the project, we learned about the detailed differences in search between each algorithm. Additional projects often required additional proof or inspection in a different way because there were various unpredictable variables that we could limit or control when used in class. The initial prediction was that heuristic would have a more efficient path cost, but non-heuristic algorithms also had reasonable path costs, which then required more testing. This allowed each algorithm to search for and use more nodes to test which algorithm was a more effective and reasonable choice. This test allows us to choose how many nodes are passed between each algorithm and how many are needed. This test also shows that the uniform algorithm has a higher number of node uses, so we know the best choice algorithm, the Search algorithm of A*.

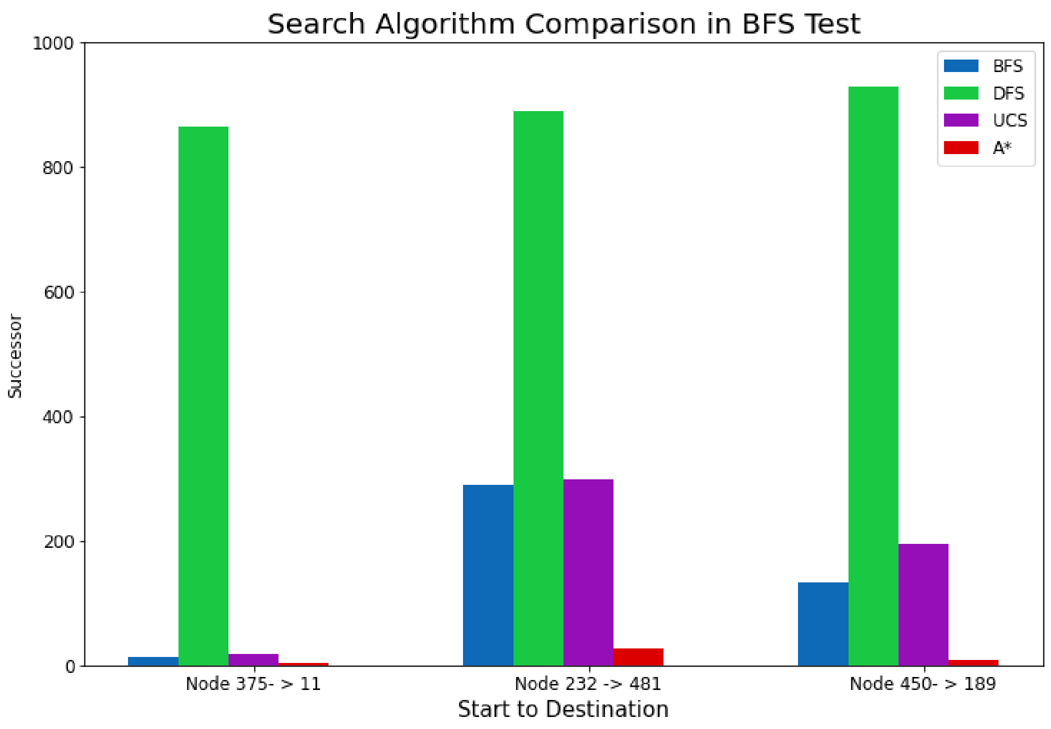


Figure 8: Visualisation of the Difference in X Coordinate Test Data from Table 3

This project once again proved that various artificial intelligence-based search algorithms that were talked about are useful in reality. Our project has shown that informed searches are more effective and acceptable than uninformed searches. Uninformed search techniques lack knowledge of the problem and thus show less effective results. The strength of an informed search is to develop solutions not only in optimal solutions but also incompleteness using well-formed heuristic functions. Therefore, we select a model of Map navigation through the A* algorithm.

We selected weight by considering each distance, and we want to select more weights and select a better model. In other words, we want to derive as many considerations from the origin to the destination as possible and compare them as parameters or variables in the model. Of course, the A* search algorithm is best at finding the shortest path, so we expect that there will be little difference in algorithm comparison, but we expect that more considerations will be added to the selection of Map navigation, which will be much more reliable and accurate model. In addition, we would like to introduce various maps, such as the representative cities of the United States and the capital of all countries, as well as Minnesota maps, to provide navigation to walking, driving, and aviation.

Bibliography

[1] A. Durdu E. Dere. Usage of the A* algorithm to find the shortest path in transportation systems. In *International Conference on Advanced Technologies, Computer Engineering and Science (ICATCES'18)*, 2018.

[2] Deepika. G. Comparative study of various searching algorithms. In *National Conference On Innovative Trends In Computer Science Engineering*, 2015.

[3] H. F. Halaoui. Smart traffic systems: Dynamic A* traffic in GIS driving paths applications. In *2009 WRI World Congress on Computer Science and Information Engineering*, volume 4, pages 626–630, 2009.

[4] Maharshi J., Ronit L., and Sonal P. Comparative analysis of search algorithms. *International Journal of Computer Applications*, 179(50):40–43, 2018.

[5] HA Hasanvand M Nosrati, R Karimi. Investigation of the * (Star) Search Algorithms: Characteristics, methods and approaches. *World Applied Programming(WAP)*, pages 251–256, 2012.

[6] Norvig. aima-python. `https://github.com/aimacode/aima-python`. (accessed: 03.10.2021).

[7] A. Panghal. A comparative study of searching and optimization techniques in artificial intelligence. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pages 72–80, 2020.

[8] Girish P Potdar and R C Thool. Comparison of various heuristic search techniques for finding shortest path. *International Journal of Artificial Intelligence amp; Applications*, 5(4):63–74, 2014.

[9] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4 edition, 2020.

Each member's contribution to the project

- Cho, Chloe: Abstract, Literature Review - Understanding algorithms(Modify and add materials), Data Tables, Experiment and results
- Kim, Hyunwoo: Keywords, Introduction, Literature Review - Understanding Algorithms, Data logistics and pre-processing, Visualization part, and Conclusion and Future works
- Cho, Yongsuk: Literature Review - Understanding algorithms (Modify and add materials), Data Tables, Conclusion (Modify and add materials), Analysis result