State Based Models

Submitted By
Mikaela Nicole B. Amon
Thara Mae P. Corpuz
Zhean Robby L. Ganituen
Aaron Daniel L. Go
Emmanuel Gerald G. Punsalan

Submitted To
Dr. Norshuhani Zamin

October 23, 2024

I. Introduction

Students are subjected to an overwhelming juggling act, balancing academic performance, mental health, and social relationships. A critical but often overlooked aspect of this balancing act is the role of nutrition in shaping their development, success, and overall well-being.

Research shows that cognitive functions are heavily influenced by a person's quality of nutrition, extending beyond childhood and even into adulthood [1]. In low-income areas, students who lack access to regular meals may skip school to support household duties, sacrificing educational opportunities [2].

Furthermore, studies indicate that proper nutrition—in both type and quantity—influences cognitive function as well as emotional well-being, with poor eating habits linked to high levels of anxiety and depression [3].

On top of the students' juggling act is a balancing act they have to face every day—deciding where to eat. These decisions involve several factors, such as location, cost, and food satisfaction. Making an efficient choice can feel like a "minigame," where the player has to decide between various options with each having pros and cons… and just like a game, there is a way to simplify and optimize their decision-making process.

In this study, we model the campus as a graph, where points represent dining options and edges represent the travel time between them. Using the graph, we can apply search algorithms and identify the most optimal routes for students, trivializing their decision making process.

There are two types of searches used for this study: (1) Blind search and (2) Heuristic search. Blind search, also referred to as "uninformed search", is where a search algorithm has no other information about the vertices or states outside of what has been given in the problem definition [4]. The blind search algorithm used for this study is the Uniform-Cost Search (UCS) where it explores all possible options, then creates a path for the most efficient choice [5]. In our case, it creates the most efficient route students can take to get to a certain eatery, however this only factors one value – which is travel time.

Heuristic search, on the other hand, uses information specific to the problem beyond its definition, which helps in searching for a goal state more efficiently. This type of search utilizes a value called heuristic value, which is an estimate of how likely a given vertex will lead to a preferable result [4]. For this type of search, this study uses the A* algorithm (pronounced A-star), which utilizes both the heuristic value and the total cost to find an optimal path to a goal [4]. Not only does this algorithm construct the most optimal path, but it also does so in a faster and more efficient way.

To maintain realism in the problem, the weights of each edge in the graph represent the actual distances between establishments, which are sourced from Google Maps. Similarly, the heuristic values for each node are derived from the Google ratings of the corresponding establishments. Doing so will increase the accuracy of the algorithms in a realistic setting as this would allow for an effective route optimization based on realistic factors.

By comparing the two algorithms in terms of time and memory complexity as well as the optimality of their solutions, we aim to determine which approach is most suitable for students troubled with selecting a proper dining option.

II. Methodology

Each restaurant represented in the food map is modeled as a vertex within a graph structure. The connections between these vertices, or edges, are established based on the following criteria:

R1 An undirected edge $\mathrm{A} \leftrightarrow \mathrm{B}$ is established if and only if the restaurants corresponding to vertices$\mathrm{A}$ and B are not physically separated by a street. In this context, "not separated by a street" means that there is a direct, unobstructed pathway between the two vertices.

R2 An edge $\mathrm{A} \leftrightarrow \mathrm{B}$ is included in the graph if and only if the Euclidean distance between the two vertices falls within a predefined threshold considered walkable, the $\mathrm{d}(\mathrm{A} \leftrightarrow \mathrm{B}) \leq walk$. The parameter, $walk$, defining walkability is established based on the physical distance represented by the edge connecting vertices$\mathrm{U}$ and J, which serves as a reference unit for subsequent measurements.

R3 If and only if an outlier vertex C exists—defined as a vertex that either lacks a direct edge to any other vertices or effectively partitions the graph into two disjoint subgraphs—then C will be connected to the nearest vertex , regardless of whether this connection violates the established if it violates the rule regarding obstructions (R1) or Euclidean distance threshold (R2). This approach aims to preserve the integrity of the graph structure and ensure the inclusion of otherwise isolated vertices, thereby enhancing the overall connectivity and cohesion of the graph.

These rules collectively facilitate the construction of a graph that accurately represents the spatial relationships between restaurants within the defined food map.

The edge and vertices that are used as the defining value for the walk parameter are shown in the following Figure 1. On the other hand, Figure 2 shows the complete resultant graph with all vertices and edges which were created using the above-mentioned rules.

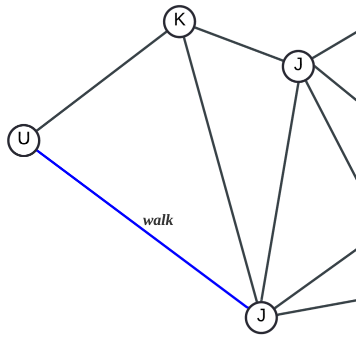

Figure 1. *Edge* U ↔ J *Which Represents the walk Parameter*

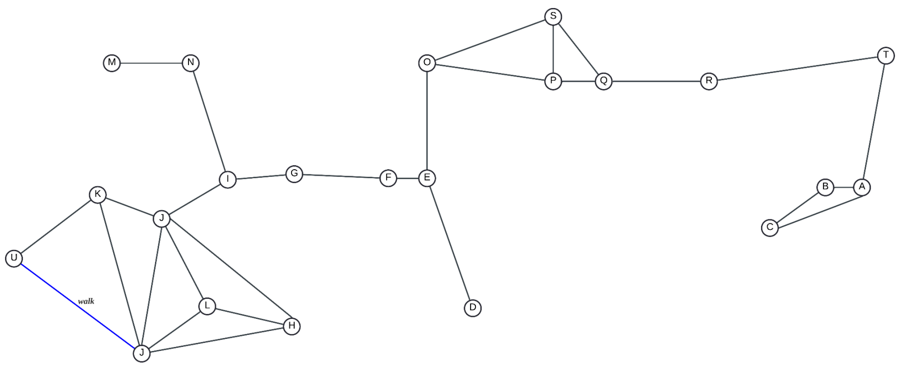

Figure 2. *Resultant Graph of The Food Map Using The Rules Established*

New restaurants can also open up in the food map, consequently, existing restaurants can close business entirely. Following this train of thought, nodes and edges made to open or close obey the rules mentioned below:

One of the algorithms chosen to solve the problem provided is Uniform Cost Search (UCS). Similar to Dijkstra's, this algorithm searches for the path to a goal with the least cumulative cost. It exclusively considers the cost of edges during traversal, overlooking any and all heuristics being a kind of Blind Search. In the context of the problem at hand, UCS considers getting to the goal restaurant based solely on how far students have to travel from the restaurant they're currently at.

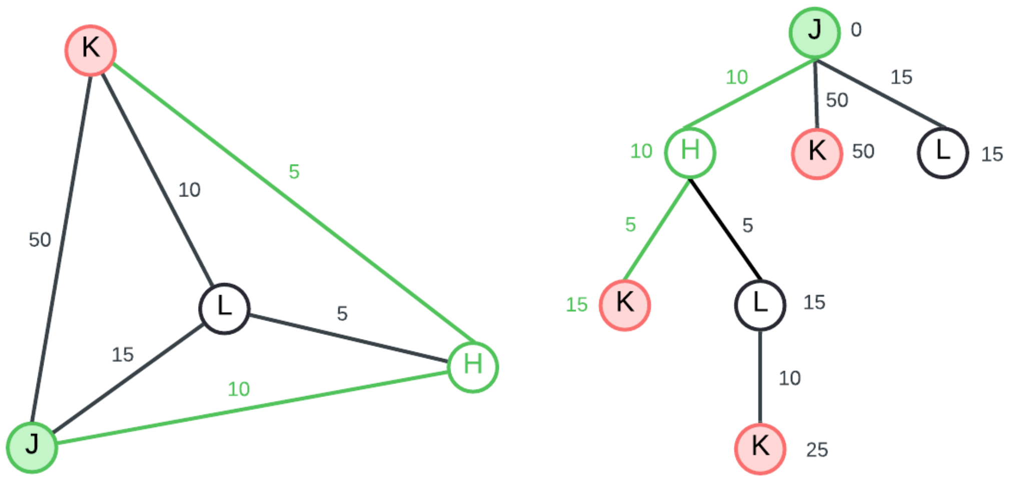

Figure 3. Sample graph (Left) Sample UCS Traversal (Right)

To further illustrate the process of UCS, an example will be given. Let vertex J (filled in green) be the start and vertex K (filled in red) be the goal from the sample graph. To start UCS traversal, vertex J as well as its neighboring vertices are opened and the edges connecting them are labeled with their respective costs. Vertex J is marked as "visited" meaning other vertices cannot open it up as a neighbor. Next, the neighboring vertices K, L, and H are considered which would cost 50, 15, and 10 to traverse respectively. The neighbors of vertex H are opened up, vertices L, and K, because it costs the least compared to the other two vertices at 10, and mark it as "visited". After that, vertex L is opened up since it costs only 15 cumulatively (adding 10 from $\mathrm{J} \rightarrow \mathrm{H}$ to 5 from $\mathrm{H} \rightarrow \mathrm{L}$) to traverse and we go alphabetically even if vertex K costs the same. Next, vertex K connected to vertex H is opened up since it costs 15 cumulatively also and is marked as "visited". Even if we open up goals, we don't stop the traversal because there may be a more optimal path in terms of cost. This process is repeated until all vertices are marked as "visited", and we consider the path to a goal with the least cost. This would be the path of $\mathrm{J} \rightarrow \mathrm{H} \rightarrow \mathrm{K}$ with a cost of 15 (edges and vertices outlined in Green)

The second algorithm chosen for the problem is the A* Algorithm. This algorithm is essentially an improved version of UCS, where not only does it consider the cost of edges, but it also takes note of the heuristic estimate of the vertices. It is the informed version of UCS wherein the nodes are visited based on their f value, the sum of the actual cumulative cost needed to traverse from the starting vertex to the given vertex, and the given vertex's defined heuristic value. Adding the heuristic value makes this algorithm potentially run faster than UCS, as it could visit fewer nodes and still return the same path. In the problem, A* searches for the goal by taking note of both the distance from one restaurant to another and the ratings of said restaurants in Google.

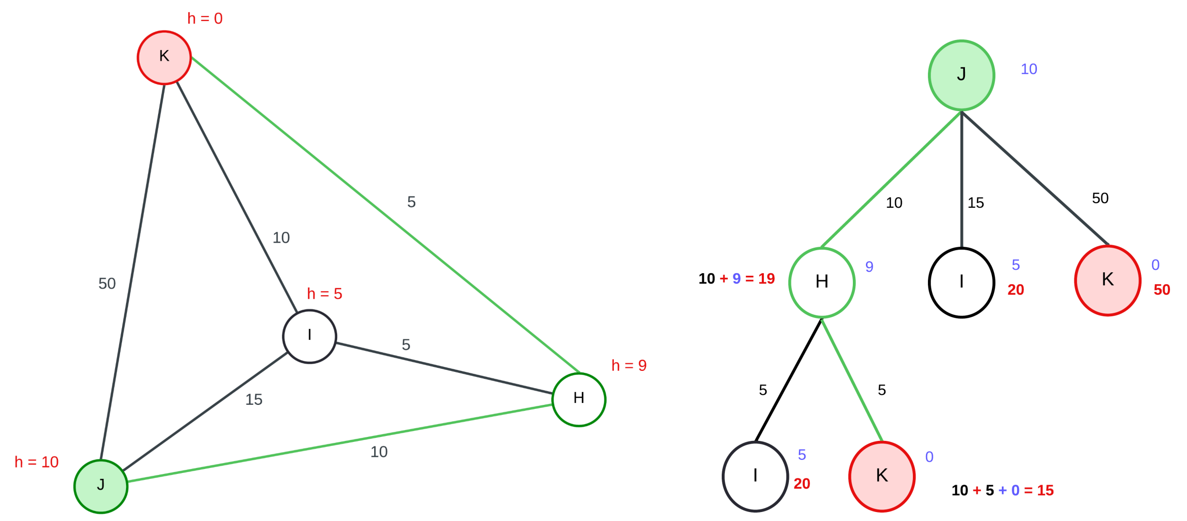

Figure 4. Sample graph (Left) Sample A* Traversal (Right)

Figure 4 illustrates the traversals performed by the A* algorithm. From the graph, let J be the start node and K be the end node. The heuristics value $(h(n))$ for each node is highlighted in blue, the actual cost in black $(g(n))$, and $f(n)$ = $\sum g(n) + g(n)$ in red.

Using the A* algorithm to traverse the graph from node J to node K, we start by initializing the algorithm at node J. The initial cost, $g(J)$ is set to zero, and the heuristic value, $h(J)$ is 10, giving an $f(j)$ of 10 for node J. From node J, we explore its neighbors, nodes H, I, and K. The cost to reach H from J is 19, from J to I is 20, and the cost to reach from J to K is 50. After calculating their $f(n)$ values, we determine that node H has the lowest $f(n)$ value, so we expand node H.

At node H, we evaluate its neighbors I and K. The $f(n)$ cost to reach I from H is 20, while the cost to reach K from I is 15. Since K is the goal node the A* algorithm has reached its destination. The shortest path constructed by the algorithm is $J \rightarrow H \rightarrow K$, with a total cost of 15.

III. Results and Analysis

In order to compare and contrast the performances of UCS and A* in terms of time and memory complexity as well as optimality and cost, various tests were performed and measured a posteriori in order to deduce how both fared.

For the comparison of time complexity, both algorithms were executed 10 times per varying and 15 slowly increasing input sizes (number of vertices and edges) then averaged. The start and goal vertices were randomized, along with the weight and heuristics to avoid bias or swaying of data. Multiple test cases were done with a separate program exclusively using the implemented algorithms. The timeit module was used to keep track of the runtime of each algorithm.

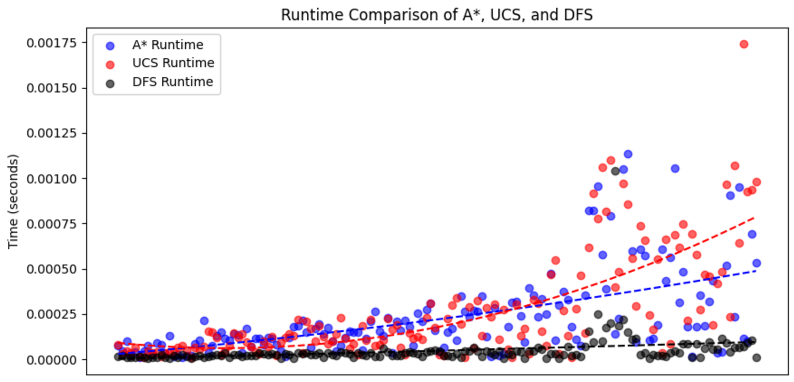

Figure 5. Test Case Plot of Average Time Complexities of Large Input Sizes

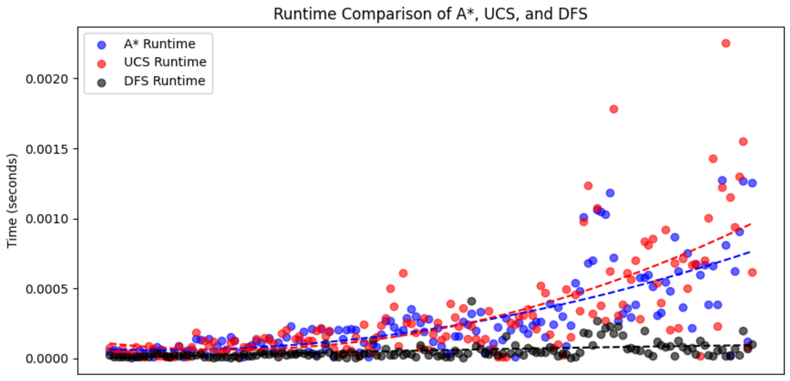

Figure 6. Different Test Case of Plot of Average Time Complexities of Large Input Sizes

As seen in the figures shown above, UCS tends to consume more time as the input size gets larger as compared to A* which consumes less time all throughout the different input sizes.

Although in the context of the provided problem, students are limited to a small amount of restaurants to an area, meaning input sizes would be smaller when it comes to choosing where to eat. A test provided with the vertices, weights, and heuristics interpreted from the given food map was used given the same treatment as the test above.

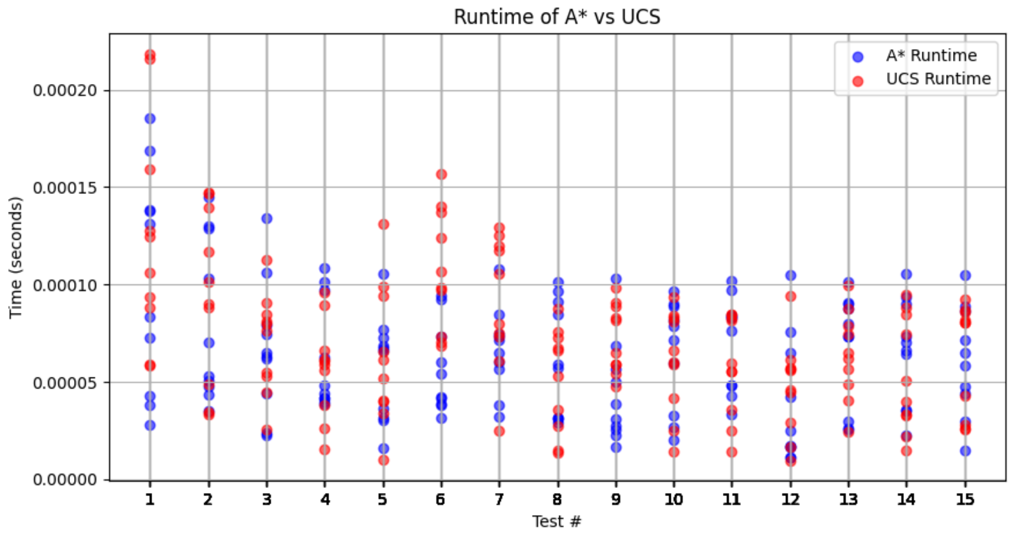

Figure 7. Test Case of Plot of Average Time Complexities of the Food Map's Input Size

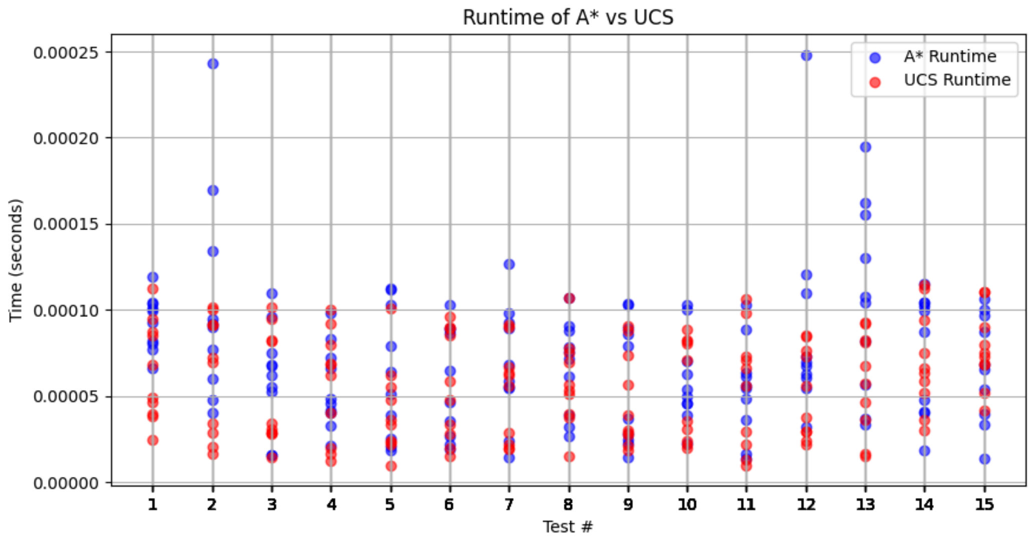

Figure 8. Different Test Case of Plot of Average Time Complexities of the Food Map's Input Size

Based on the figures above, A* consumes as much time as UCS given the Food Map's number of vertices, weights, and heuristics.

For the testing of memory complexity, the tracemalloc module was used in order to keep track of memory allocations during the respective algorithms' runtime. Randomly generated start vertices and goal vertices were generated, the number of visited vertices were listed during traversal in order to compare UCS and A*. path is represented as an array where the element at index 0 is the start vertex and the last element is the goal vertex.

Trial	**UCS**			**A***		
	Path	**Number of Visited Vertices**	**Memory Used**	**Path**	**Number of Visited Vertices**	**Memory Used**
1	[1, 20, 18, 17, 19, 15, 5, 6, 7, 9, 10]	16	312.0 MiB	[1, 20, 18, 17, 19, 15, 5, 6, 7, 9, 10]	16	296.0 MiB
2	[4, 5]	2	264.0 MiB	[4, 5]	2	240.0 MiB
3	[17, 19, 15, 5, 6, 7, 9, 10, 8]	20	192.0 MiB	[17, 19, 15, 5, 6, 7, 9, 10, 8]	20	208.0 MiB
4	[1, 20, 18, 17, 19, 15, 5, 6, 7, 9]	14	224.0 MiB	[1, 20, 18, 17, 19, 15, 5, 6, 7, 9]	14	208.0 MiB
5	[12, 8]	3	192.0 MiB	[12, 8]	3	162.0 MiB

Table 1. Comparison the Memory Complexities between UCS and A*

As we can observe from the table, UCS allocates slightly more memory when compared to A*, though is a bit negligible. UCS may be more memory inefficient either due to it exhausting all of the vertices available in the graph which would be a lot worse in larger graphs or because UCS doesn't have any guidance from heuristic values which help A* a lot in lessening the memory allocated.

For measuring cost and optimality, a random start and goal node was generated then traversed by the respective algorithms. Consequently, the cost of traversal between A* and UCS was compared to look for any differences. The output of the path is represented as an array where the element at index 0 is the start vertex and the last element is the goal vertex.

Trial	**UCS**		**A***	
	Path	**Cost**	**Path**	**Cost**
1	[19, 15, 5, 6, 7, 9, 10, 12]	763	[19, 15, 5, 6, 7, 9, 10, 12]	763
2	[9, 14]	93	[9, 14]	93
3	[1, 20, 18, 17, 19, 15, 5, 6, 7, 9, 10, 11]	1670	[1, 20, 18, 17, 19, 15, 5, 6, 7, 9, 10, 11]	1670
4	[2, 1, 20, 18, 17, 19, 15, 5, 6, 7]	1429	[2, 1, 20, 18, 17, 19, 15, 5, 6, 7]	1429
5	[9, 14, 13]	353	[9, 14, 13]	353

Table 2. Comparison the Cost and Optimality between UCS and A*

The table above illustrates the optimal path and the cost of said path when applying UCS and A* to the interpreted graph of the food map and its values. It can be seen that both algorithms return the exact same optimal path and cost, meaning the optimality of both algorithms within the food map is the same.

UCS and A* as search algorithms perform almost the same in terms of time, memory complexity, and optimality. Regarding time complexity, there is a discrepancy depending on the number of vertices and edges within a graph. In the context of the food map, UCS will be faster than A* most of the time. This is because A* has to compare and compute heuristic values even if there's only a small number of vertices and edges, while UCS doesn't have to worry about heuristics. But it's a different story talking about generally larger graphs with more vertices and edges. Results from testing concerned with larger input sizes (number of vertices and edges) showed that A* is more time-efficient compared to UCS.

When it came to memory complexity, UCS a bit more more memory compared to A* which can be attributed to UCS having to exhaust the vertices to visit. A* also has heuristics to base off of traversal. This memory inefficiency of UCS may be a detriment when it comes to larger graphs seeing as it already has a big gap for the food map. But the memory consumption is negligible at best due to the small input size.

Both UCS and A* return the exact same optimal paths and costs but this is assuming good conditions. Many conditions/limitations could hinder the time complexity, memory complexity, and optimality of both UCS and A* when searching for the most optimal path to a restaurant in the Food Map. UCS can take up so much time and memory depending on the size of a generally large graph. While A* could only perform as well in the Food Map graph interpretation because the heuristic values were admissible. Otherwise, A* would be non-optimal given inadmissible heuristics for vertices to traverse through.

In conclusion, UCS and A* have their own respective strengths and weaknesses when it comes to searching for the optimal path to an eatery in the DLSU Food Map. The main fault with both algorithms is that it's a heavy generalization of going to restaurants, they leave out so many factors a simple heuristic or weight cannot consider. There should be a balance of human influence and computer computations in order to have a more holistic and satisfactory experience, just like a well-balanced diet.

IV. References

[1] M. F. Royer, N. Guerithault, B. B. Braden, M. N. Laska, and M. Bruening, “Food Insecurity Is Associated with Cognitive Function: A Systematic Review of Findings across the Life Course,” International Journal of Translational Medicine, vol. 1, no. 3, pp. 205–222, Nov. 2021, doi: https://doi.org/10.3390/ijtm1030015.

[2] World Food Program USA. “The Effects of Child Nutrition on Academic Performance: How School Meals Can Break the Cycle of Poverty - World Food Program USA,”, Sep. 21, 2023. https://www.wfpusa.org/articles/effects-child-nutrition-academic-performance-how-school-meals-can-break-cycle-poverty/ (accessed Oct. 20, 2024).

[3] P. Pandey and A. Mishra, “Impact of Daily Life Factors on Physical and Mental Health,” Mar. 2024, doi: https://doi.org/10.1109/icdt61202.2024.10489692.

[4] S. Russell and P. Norvig, “Artificial Intelligence: A Modern Approach Third Edition,” 2010. Available: https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf

[5] GeeksforGeeks, "Uniform Cost Search (UCS) in AI," Aug. 23, 2024.: www.geeksforgeeks.org/uniform-cost-search-ucs-in-ai/.

V. Contributions of Each Member

Group Member	Contributions
Mikaela Nicole B. Amon	**Code:** A* algorithm **Paper:** A* algorithm, Introduction
Thara Mae P. Corpuz	**Code:** A* algorithm **Paper:** A* algorithm
Zhean Robby L. Ganituen	Constructing *Food Map* and construction rules **Code:** GUI **Code:** Graph implementation **Code:** a posteriori time complexity **Paper:** Results and Analysis
Aaron Daniel L. Go	**Code:** UCS algorithm **Paper:** UCS algorithm, Introduction
Emmanuel Gerald G. Punsalan	**Paper:** UCS algorithm; Results and Analysis, Introduction **Code:** a posteriori space and time complexity