Far Eastern University

A COMPARATIVE ANALYSIS OF DIFFERENT HEURISTIC METHODS AND DIJKSTRA'S ALGORITHM IN FINDING OPTIMAL ROUTES FOR FEU-MANILA BUILDINGS

By

Ralph Niclaus S. Lobos

Artaire C. Abulencia

Mr. Jeffrey Alvarina

Adviser

An undergraduate thesis submitted to the

Department of Mathematics

Institute of Arts and Sciences

Far Eastern University

Sampaloc, Manila

In partial fulfillment of the requirements for the degree of

Bachelor of Science in Applied Mathematics

(Information Technology Track)

May 2024

Declaration of Authorship

We, Ralph Niclaus S. Lobos, Artaire C. Abulencia, and , declare that this thesis titled, A Comparative Analysis of Different Heuristic Methods and Dijkstra's Algorithm in Finding Optimal Routes for FEU-Manila Buildings, and the work present in it are our own. We confirm that:

- This work was done wholly or mainly while in candidature for an undergraduate degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where we have consulted the published work of others, this is always clearly attributed.
- Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely our work.
- We have acknowledged all main sources of help.
- Where the thesis is based on work done by ourselves jointly with others, we have made clear exactly what was done by others and what we have contributed ourselves.

Signed: _____

Date: _____

Endorsement

This is to certify that this undergraduate thesis titled, A Comparative Analysis of Different Heuristic Methods and Dijkstra's Algorithm in Finding Optimal Routes for FEU-Manila Buildings, prepared and submitted by Ralph Niclaus S. Lobos, Artaire C. Abulencia, and , in partial fulfillment of the requirements for the degree of Bachelor of Science in Applied Mathematics (Information Technology Track) is hereby accepted.

Mr. Jeffrey Alvarina
Thesis Adviser

The Department of Mathematics endorses the acceptance of this undergraduate thesis in fulfillment of the requirements for the degree of Bachelor of Science in Applied Mathematics (Information Technology Track).

Al O. Advincula, Ph.D.
Chair, Department of Mathematics

APPROVAL SHEET

This is to certify that this undergraduate thesis entitled **"A Comparative Analysis of Different Heuristic Methods and Dijkstra's Algorithm in Finding Optimal Routes for FEU-Manila Buildings"**, prepared and submitted by **Ralph Niclaus S. Lobos**, **Ralph Niclaus S. Lobos**, and **Ralph Niclaus S. Lobos** in partial fulfillment of the requirements for the degree of **Bachelor of Science in Applied Mathematics (Information Technology Track)** has been examined and was successfully defended on **May 2024**.

Mr. Jeffrey Alvarina
Thesis Adviser

May Anne C. Tirado, Ph.D.
Research Course Instructor

Mr. Teodolfo Bonitez
Thesis Examiner

Mrs. Floredeliza Francisco, Ph.D.
Thesis Examiner

Mrs. Joanna Eugenio
Thesis Examiner

Approved in partial fulfillment of the requirements for the degree of **Bachelor of Science in Applied Mathematics (Information Technology Track)**

Al O. Advincula, Ph.D.
Chair, Department of Mathematics

Diego Jose R. Abad
Dean, Institute of Arts and Sciences

Far Eastern University
Institute of Arts and Sciences
Department of Mathematics

Bachelor of Science in Applied Mathematics
(Information Technology Track)

Abstract

A Comparative Analysis of Different Heuristic Methods and Dijkstra's Algorithm in Finding Optimal Routes for FEU-Manila Buildings

by
Ralph Niclaus S. Lobos
Artaire C. Abulencia

This study explores the comparative analysis of *Dijkstra's* algorithm and other heuristics methods with respect to their time and distance which were applied in finding the optimal route for Far Eastern University-Manila buildings. The researchers chose any of the starting positions of the buildings with one ending position to find the optimal route. The researchers produced the codes in *Python* programming language for the algorithms namely the *Dijkstra's algorithm* and the three heuristics which are being used to solve the shortest path problem and compare the speed and accuracy of the given algorithms. In terms of the speed runtime, the researchers used statistical analysis in each of the runtimes after the execution of the produced code to compare the speed of the algorithm. From the speed and accuracy of Dijkstra's Algorithm compared to other heuristic methods, the researchers find out that Dijkstra's Algorithm has a significant difference from other heuristic method in terms of their speed and that the Bidirectional Search with chosen node has more accuracy and speed compared to other heuristic algorithms and Dijkstra's Algorithm.

Acknowledgments

We would like to acknowledge our supervisor and professor Dr. May Anne Caspe Tirado, and our adviser Mr. Jeffrey Alvarina, for giving guidance and support of the research process. We also like to thank our department head, Dr. Al Advincula for assisting in our curriculum for Thesis 1 and 2.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background of the Study

In a fast-changing world, time is indeed gold, particularly for those residing in highly urbanized areas where delays can be a significant adversary. As a survival instinct, people are inclined to devise ways to enhance their efficiency and convenience without wasting any second. This shows how good people are at coming up with new ideas, which leads to the development of new technology. Technological advancements have empowered individuals to work more swiftly and efficiently. Utilizing and innovating these technologies in the 21st century put the exchange of goods and services at the tip of your fingers, especially during the COVID-19 pandemic wherein online shopping and delivery services became the dominant platform of trading, from buying your daily essentials to tracking places of your delivery areas while using a real-time map.

Hence, the creation of various delivery and transportation mobile applications such as Shopee, Lazada, Angkas, Grab, and J&T Express became one of the backbones of surviving economy during the pandemic. The network routing system, which is also essential for navigation services like Waze and Google Maps to deliver precise directions and traffic information, is what increases the convenience, dependability, and efficiency of these applications.

Graphs visually depict a collection of objects, known as nodes or nodes, interconnected by edges or links. These connections can be either directed (one-way) or undirected (two-way), representing the relationships between the nodes.

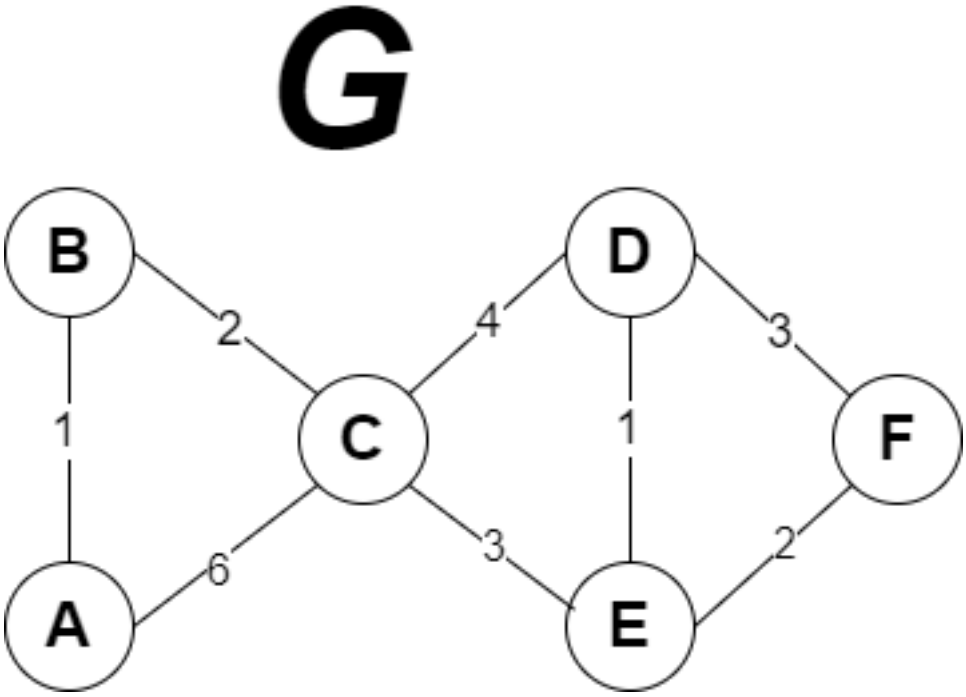

Figure 1.1: Example of Graph with nodes and edges

Figure 1.1 above is an example of a graph G with node set $V(G) = \{A, B, C, D, E, F\}$ and with a collection of edges denoted $E(G) = \{AB, AC, BC, CD, CE, DE, DF, EF\}$. Here, $V(G)$ is the collection of nodes or "location" in some context and $E(G)$ is the collection of connections between the nodes of G which can be interpreted as "pathway" between two location. We are able to generate a dictionary of weights that indicate the distances between nodes based on the image, which seems to be a weighted graph. Here is the dictionary of weights based on the figure 1.1: $weights = \{('A','B') : 1, ('A','C') : 5, ('B','C') : 2, ('C','D') : 4, ('C','E') : 3, ('D','F') : 1, ('E','F') : 2\}$. An optimization problem with the goal of minimizing the overall time or distance over the network can be formulated using this dictionary.

In this study, the buildings and special location in Far Eastern University-Manila were considered as the nodes and the path or road that connects them as our edges.

Graphs are data structures used extensively in various fields, such as computer networks, transportation systems, and social networks. One of the problems is to find the shortest path between two nodes. One example which considering the shortest route among several options for commuting from home to school. Cormen et al. (2009) stated that Dijkstra's Algorithm, proposed by Dutch computer scientist Edsger W. Dijkstra in 1956, is a widely used algorithm for solving the single-source shortest path problem in graphs with non-negative edge weights. Dijkstra's Algorithm is essential in graph theory because it efficiently finds the shortest paths within these structures. It accomplishes this by tracking nodes while iteratively exploring the graph's nodes. The algorithm initiates from the source node and progressively investigates neighboring nodes, updating their tentative distances from the source until reaching the target node. In each iteration, it selects the node with the minimum estimated distance and checks its neighbors, updating their distances if a shorter path is found.

Within the specific context of identifying an efficient route from one location to another, individual decision-making is influenced by personal strategies and biases. A heuristic is a practical, experience-based method for problem-solving or decision-making, employing trial-and-error and practical insights. While prioritizing efficiency through shortcuts, heuristics may not ensure the optimal solution in every case. Although based on human experience and can be applied through actual situations, heuristics can be programmable. Heuristic methods, in problem-solving, are often based on experience, intuition, and rules of thumb rather than difficult algorithms. Making a heuristic programmable involves translating these informal decision-making rules into a formalized, algorithmic structure that can be implemented in computer programs. For example, if a human expert uses certain rules to quickly identify a good solution in a specific scenario, those rules can be encoded into a computer program, allowing it to make similar decisions based on the heuristic. These programmable heuristics also play a crucial role in the process of determining the most optimal route, considering the current situation and the resources at hand. However, it is also important to recognize that the path chosen by

individuals may not be the actual optimal route identified by some algorithm, Dijkstra's Algorithm in particular, and to mean that there's an optimal route based on the graph theory aside from relying upon the heuristics.

Comparing the Heuristics algorithms to Dijkstra's Algorithm, Dijkstra's Algorithm would make a process of finding the shortest path. However, there are different ways and strategies to find the shortest path that can affect its accuracy and speed. Using Figure 1.1, this illustration explains Dijkstra's Algorithm ang three heuristic algorithms used in this study:

Dijkstra's Algorithm: Suppose we have a source of A and the destination of E. Step 1, We mark the source with a weight of zero. Designate this source as current. Step 2 was to find all nodes leading to the current node. In this case A has AB and AC. Next step was to calculate their weight to the end. Therefore, AB has the distance of 1 and AC has the distance of 6. Since we already know the weight, the current node is from the end which means that B or C is the end from A. This will just require adding the weight in the most recent edge. Don't record the weight if it is longer than a previously recorded weight. For instance, you have the path A →B with the weight 1 and A →C with the weight 6, do not record the A →C with the weight 6 because it may have temporary use. Step 3, mark the current node as visited. We will never look at this node again and then last step is mark the nodes with the smallest weight as current and go back to step 2 again.

Greedy Path Algorithm: Greedy Path: Suppose you have a source node A and the destination node which is E, A is your current node which has the edge of AB and AC, Greedy Path will choose the shortest edge from the current node, so from the weight in AB and AC, you the shortest edge would be AB which the current node will be "B" and the path would be A →B. "B" has an edge between BA and BC. BA would be excluded because it has been visited. Since BC is the only one edge left then the current "B" would be "C" and the would be A →B →C. "C" has an edge over CB, CA, CD, and CE. CB and CA are already visited and what is left is CD and CE the shortest edge from CD

and CE is CE so the path would be A →B →C →E. Repeating the process would result in the path of A →B →C →E →D→F.

Breadth-First Search: Breadth-First Search: Suppose you have a source node A and a destination node E. This algorithm scans every node step-by-step. Starting at A, it scans each path from each neighbor. For example, from A, we have paths AB and AC, so the first queue is A →B and A →C. Next, it finds the neighbors of B and C. Since B's neighbor is C, and C's neighbors are D and E, the second queue becomes A →B →C, A →C →D, and A →C →E. Repeating this process, we get paths A →B →C →D, A →B →C →E, A →C →D →E, A →C →D →F, A →C →E →D, and A →C →E →F. The search ends when all paths are scanned. Since the study focuses on the shortest path in terms of the number of nodes, The final result must meet two conditions: it needs the shortest number of nodes and must include both the source and destination nodes.

Bidirectional Search with chosen node: Suppose you have a source node A and a destination node E. Apply Dijkstra's Algorithm from A to E and simultaneously from E to A. The algorithm stops when the queues from both directions intersect. The main goal of Bidirectional Search Algorithm is to find the intersection point of the queues from A to E and E to A. However, in the study also add a variant of choosing the node for the midpoint which becomes Bidirectional Search with chosen node.

By addressing these practical challenges and improving the algorithm's functionality, this research seeks to contribute to the ongoing evolution of Dijkstra's Algorithm and its relevance in solving real-world problems.

1.2 Statement of the Problem

The primary goal of this study is to write codes that would execute Dijkstra's Algorithm and the three heuristics-based algorithms namely; (1) The Greedy Algorithm, (2) Breadth- First Search Algorithm, and (3) Bidirectional Search with chosen node with chosen nodes, in solving the shortest path problem. This research aims to compare the above algorithms in their accuracy and speed in finding the shortest paths of these objectives, the researchers have identified the following specific problems to be addressed in this paper:

1. What paths do the algorithms suggest, and are the three heuristic algorithms accurate compared to Dijkstra's Algorithm for the following routes:

   - Science Building to FEUture Center
   - Arts Building to FEUture Center
   - Nicanor Reyes Hall to FEUture Center
   - Education Building to FEUture Center
   - Nursing Building to FEUture Center

   to its destination which is suggested to be in the FEUture Center Building based on the edges which students get through from each node excluding the shortcut of Admission Building to FEUture Center?

2. What are the paths of these four algorithms will suggests from one building to another?

3. Is there a significant difference to the speed in finding the shortest path between two buildings in Far Eastern University in terms of time or distance between Dijkstra's Algorithm and the three heuristic approaches?

1.3 Scope and Limitation

This study aims to analyze and compare the three heuristic methods and Dijkstra's Algorithm for navigating from one specific location to another in Far Eastern University-Manila buildings. This study investigated and compared Dijkstra's Algorithm and the three heuristic methods through the collected data supervised by the researchers. The code works for any graph with adjacency matrix of size n. The researchers produced matrices, see Appendix B, with respect to distance and time from one building to the other.

However, this research is narrowly focused on a specific theory (Dijkstra's algorithm), methodology (Heuristic Methods), and location (Far Eastern University-Manila buildings). It is limited to the Far Eastern University-Manila buildings as the location. Other institutions, places, streets, and such are not included in this study, since it is limited to the buildings of Far Eastern University in Manila. The scope does not extend to exploring alternative algorithms for identifying the shortest paths or variables outside of those specified. The process of finding the time value was limited to the researchers' collected data which are used for time computations. Also, the researchers physically measured the amount of time across five trials using the same path before averaging the result. Distance data was acquired through the utilization of mapdevelopers.com software to compute the approximate distances between nodes. The implementation of the methodology was facilitated through the utilization of the Python programming language. Hence, in this study, the focus is on comparing the Dijkstra's Algorithm with the heuristics and is limited to the results of the collected data. The other participants' preferences, the use of relevant applications, programming language or websites, the use of different pathways, and the conduct of more specific time computations were recommended to the future researchers.

1.4 Significance of the Study

This study would have a contribution in many fields, especially in figuring out the geographic directions, as this can provide information to aid organizations, institutions, and groups by providing solutions to optimize routes and resource allocation, thereby improving overall operations. This study can be also utilized in information technology to enhance efficiency and decision-making processes.

Moreover, this study will be beneficial to students as it enable them to assess the correctness and efficiency of their chosen paths, leading to better decision-making and time-management skills. Likewise, teachers can utilize the insights gained from this study to enhance their teaching methodologies, incorporating real-world applications and problems.

1.5 Definition of terms

- Algorithm Variants: Modified versions or adaptations of a base algorithm (e.g., Dijkstra's Algorithm) with specific alterations or enhancements to achieve different objectives or improve performance.
- Dijkstra's Algorithm: As stated by AbuSalim et al. (2020), Dijkstra's Algorithm—originated by the Dutch computer scientist Edsger W. Dijkstra in 1956—is a widely used algorithm for solving the single-source shortest path problem in graphs with non-negative edge weights.
- Nodes: Nodes, or nodes, are interconnected by edges, representing entities or objects, the nature of which varies based on the specific application or problem being modeled.
- Edge: A connection between two nodes in a graph, often associated with a weight or cost in the context of finding the shortest path.
- Efficiency: The ability of an algorithm to achieve its goal (e.g., finding the shortest path) with minimal time and resource consumption.
- Graph: A mathematical representation of a set of objects where some pairs of the objects are connected by edges.
- Heuristics: Techniques or methods that approximate a solution more quickly than standard algorithms, often sacrificing accuracy for speed.
- Shortest Path: The path between two nodes in a graph with the minimum total weight, often defined by the sum of edge weights.
- Single-Source Shortest Path Problem: A problem in graph theory where the goal is to find the shortest paths from a designated source node to all other nodes in a graph.

- Visited nodes: nodes in a graph that have been explored or traversed by the algorithm during its execution.
- Pseudocode: A high-level algorithm description using natural language and informal programming conventions, serving as a precursor to actual code implementation.
- Directed: A graph in which the edges have a direction. This usually represents a one-way relationship between the nodes connected by the edge.
- Pathway: In graph theory and Dijkstra's Algorithm, "pathways" denote the edges between nodes in a graph
- Computational Runtime: In the context of a computer algorithm, computational runtime refers to the time it takes for the algorithm to execute and accomplish a specific task.

Chapter 2

Review of Related Literature

In this chapter, the researchers reviewed different literature to have a better understanding of the heuristic algorithm, Dijkstra's algorithm and their relationships towards it. This helped provide wider perspectives in comprehending the complexities of these two concepts and it will help researchers hypothetically spot the difference in accuracy, efficiency, complexity, and speed between Dijkstra's algorithm and other heuristics. The researchers mainly focused on the pseudo code or codes, and the process of the algorithm in each of the different heuristics and Dijkstra's algorithm.

While the research relates to finding the shortest path to the given problem, there are some of the real-life problems that can involve finding the shortest path. N. Ahmad & Aminuddin (2023) have an example of a university that is widespread, and the location of every office is different and far away from each other which is a factor that needs to solve the shortest path towards it. Xie & Chen (2016) have another problem which is the course-scheduling system which is about the arrangement of courses in college that implements the teaching plans. The rationale for finding the shortest path in the problem was to solve the lack of consideration of students toward walking distance in the classrooms. Hassan et al. (2019) have a factor that is related to the activity of the students which is the transportation around the campus that involves finding the shortest path to reduce vehicle emission, promote a healthy lifestyle of transportation, and sustain the awareness of the students towards the campus.

In terms of the real-life problems, we provide studies that focuses the application of Dijkstra's algorithm. Mirzaeinia et al. (2019) has a main goal of reducing the power consumption by optimizing drone navigation in areas where GPS is unavailable, like tunnels and subways. The Dijkstra's algorithm and router systems equipped with global

network data are used to determine the safest and quickest routes between source and destination pairs in each tunnel cross-section.

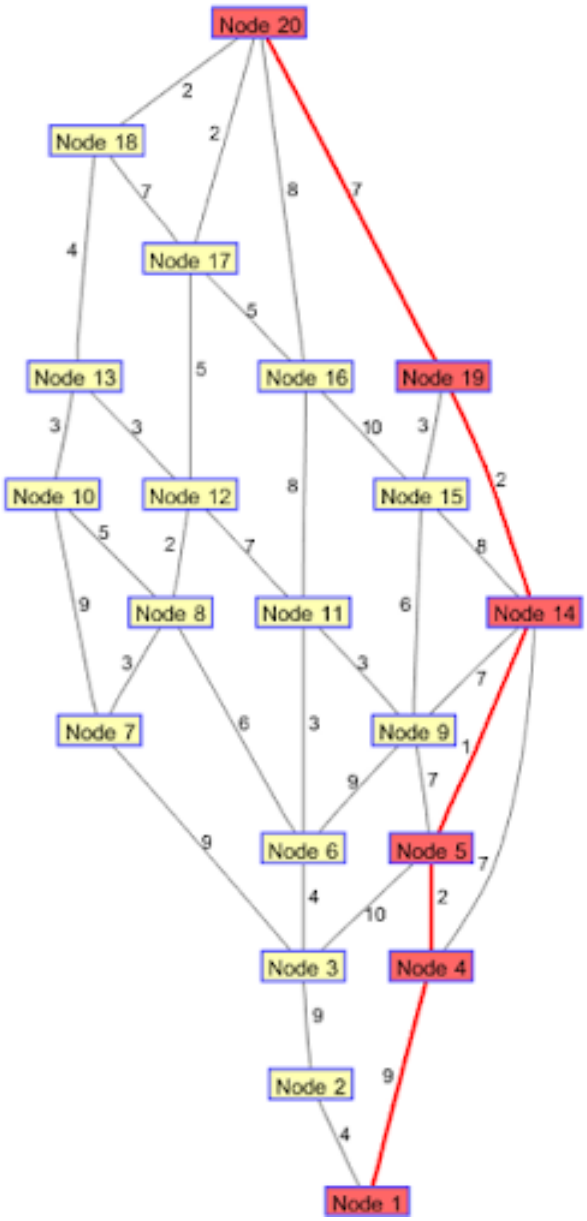

Figure 2.1: View of Modeled Graph of Subway and the Shortest Path from Node 1 to Node 20

The study has its subway map and has been modeled as a graph in Figure 2.1. From

the subway map, the researchers conclude that it was possible to find the shortest path from each node to another. Figure 2.1 shows the routing from node 1 to node 20 which results in a path of 1,4,5,14,19,20 with a total weight that is equal to 9+2+1+2+7=21. Flying drones that had passed and detected from the shortest of path node 1 to node 20 will reduce the required power to travel through a subway network.

When it comes of the application of Dijkstra's algorithm, we have found studies that use programming as a tool for implementing path-finding algorithms but also have real-life problems towards it. Deepa et al. (2018) built Dijkstra's algorithm in C Programming. The aim was to find the shortest path between buildings. The researchers manage to take the input of a starting node as it calculates the shortest distance between the start nodes to all the nodes present in their created the graph with the use of C programming. Alam & Faruq (2009) encountered diverse applications and products centered on the creation of various shortest path algorithms, particularly relying on the utilization of Dijkstra's algorithm. The primary goal of this algorithm is to reduce implementation costs and effectively address the single-source shortest path problem within a static graph. The wider context of the shortest path problem has undergone comprehensive examination, given its relevance in graph theory, artificial intelligence, computer networks, and the design of transportation systems.

```
Procedure CalculateArrow(p1, p2, node, weight, dir_x, dir_y, startp, endp, arrow):
// Calculate direction vector between p1 and p2
dx = node[p2].x - node[p1].x
dy = node[p2].y - node[p1].y

// Calculate distance between p1 and p2
l = sqrt(dx^2 + dy^2)

// Normalize direction vectors
dir_x[p1][p2] = dx / l
dir_y[p1][p2] = dy / l

// Calculate start and endpoints of the arrow
// Adjust startpoints if there is an arrow from p2 to p1
if weight[p2][p1] > 0:
    startp[p1][p2] = Point(node[p1].x - 5 * dir_x[p1][p2], node[p1].y + 5 * dir_x[p1][p2])
    endp[p1][p2] = Point(node[p2].x - 5 * dir_y[p1][p2], node[p2].y + 5 * dir_x[p1][p2])
else:
    startp[p1][p2] = Point(node[p1].x, node[p1].y)
    endp[p1][p2] = Point(node[p2].x, node[p2].y)

// Calculate range for arrowhead (not all the way to the start/endpoints)
diff_x = abs(20 * dir_x[p1][p2])
diff_y = abs(20 * dir_y[p1][p2])

// Calculate new x-position of the arrowhead
if startp[p1][p2].x > endp[p1][p2].x:
    arrow[p1][p2] = Point(endp[p1][p2].x + diff_x + (abs(endp[p1][p2].x - startp[p1][p2].x) - 2 * diff_x) * (100 - w) / 100, 0)
else:
    arrow[p1][p2] = Point(startp[p1][p2].x + diff_x + (abs(endp[p1][p2].x - startp[p1][p2].x) - 2 * diff_x) * w / 100, 0)

// Calculate new y-position of the arrowhead
if startp[p1][p2].y > endp[p1][p2].y:
    arrow[p1][p2].y = endp[p1][p2].y + diff_y + (abs(endp[p1][p2].y - startp[p1][p2].y) - 2 * diff_y) * (100 - w) / 100
else:
    arrow[p1][p2].y = startp[p1][p2].y + diff_y + (abs(endp[p1][p2].y - startp[p1][p2].y) - 2 * diff_y) * w / 100
```

Figure 2.2: Pseudo code for shortest path of Dijkstra's algorithm

The given pseudo code is forming a segment of a more extensive algorithm designed to draw arrows connecting two points (p_1 and p_2) on a graph. It establishes the orientation between these points and adapts the positions of the arrow's starting and ending points accordingly. The algorithm also addresses scenarios where there is an arrow from p2 to p1, making necessary adjustments to the starting points. Hence, the calculation of the arrowhead takes into account the direction and distance between the start and end points.

An effective algorithm has been created to address the shortest path problem within transportation networks. This algorithm optimizes computational efficiency by conducting focused searches around specific nodes. Remote learning opportunities are facilitated through applets, enabling students to utilize the tool from any location that supports Java Virtual Machine capabilities.

In terms of Dijkstra's algorithm, we therefore found some studies that give emphasis about the relationship to any heuristics algorithm. Sun et al. (2017) focused on the network topology structures and proposed an enhanced Dijkstra heuristic pathfinding algorithm. Recognizing the efficiency limitations of the core Dijkstra's algorithm due to its inherent greedy strategy, the paper incorporates the heuristic-based path selection and investigating a multi-intermediate points pathfinding algorithm where it achieves a streamlined and effective approach to finding the shortest path rapidly, with path accuracy reaching 100%. A recent study by Ab Wahab et al. (2020) examined the two approaches in the context of mobile robot path planning: the traditional and meta-heuristic methods. They use fancy names for the methods they study, like Genetic algorithm (GA) or Particle Swarm Optimization (PSO), but they're like different ways to solve puzzles for robots. Some ways are better for certain situations, just like some puzzle-solving methods are better for different types of puzzles. The experiment focuses on a maze layout, and results indicated that PSO outperformed other algorithms in execution time and battery consumption. It also showcased a clear difference between classical and meta-heuristic methods in traveled distance performance factor, with meta-heuristic algorithms demonstrating better performance.

In the realm of disaster management, the efficiency and effectiveness of escape route planning during fire emergencies are the most important factors to ensure safety. In relevance to this, the study of Rahayuda & Santiari (2020) delves into the application of advanced techniques, namely Dijkstra and Bidirectional Search with chosen nodes, as tools to optimize these crucial escape routes.

This study employed Dijkstra and Bidirectional Search with chosen nodes techniques to optimize escape routes during fire disasters. The analysis revealed Bidirectional Search with chosen nodes superiority in 40% of cases, surpassing the standard Dijkstra's 10%, with equal performance in 50% of scenarios.

This exploration not only underscores the significance of these algorithms in disaster preparedness but also sheds light on their comparative effectiveness in safeguarding lives

during fire disasters. Efficient route determination is crucial across various contexts, akin to finding the fastest path from home to school amidst traffic variables. Therefore, Bidirectional Search with chosen nodes consistently proves advantageous, suggesting its potential for further enhancement. Future research should explore its efficacy in diverse contexts, emphasizing enhanced safety protocols for fire evacuations.

In Prasad's research Prasad et al. (2021), the Genetic algorithm emerges as a valuable tool proficient at discerning optimal routes between two points, much like a seasoned guide revealing the swiftest and most cost-efficient paths on a map. The primary objective is to explore the application of the Genetic algorithm in uncovering optimal routes within a given map, underscoring its pivotal role in route planning and optimization.

Hence Self-driving vehicles, seen as the future of transportation, are undergoing continuous enhancements led by tech giants like Google and Audi. Despite current model limitations, ongoing research aims to improve self-driving features, promising safer and more effortless driving experiences. The project highlights the enduring value of Dijkstra's algorithm in path detection and stresses the potential for cost-effective, efficient solutions in self-driving vehicle development, exemplified by the integration of components like ultrasonic sensors.

Chapter 3

Methodology

This chapter describes the methods and research design, theoretical analysis, the instrument used, data gathering procedures, and statistical treatment of data obtained.

3.1 Research Methodology

The researchers arbitrarily assigned nodes and considered the distance between buildings to determine which building was closer to the next node. They utilized a smartphone's stopwatch application to record the time, with measurements recorded in seconds. The researchers assumed and pinpointed the starting and ending points for each location, designating them near elevators, entrances, or high-traffic areas. Although there were many possible paths, the researchers only considered data collected from walking where there were many people passing by. This means that if there is no direct connection, one needs to pass through intermediate nodes or other nodes before reaching the desired nodes. The researchers also averaged the time over five trials using the same path and conducted the measurements at the same time, following the same route. The distance data was collected using the mapdevelopers.com software to calculate the approximate distance between nodes, with measurements recorded in meters. The structured representation in Figure 3.1 below offers a clear and organized visualization of the spatial relationships between various locations within Far Eastern University-Manila.

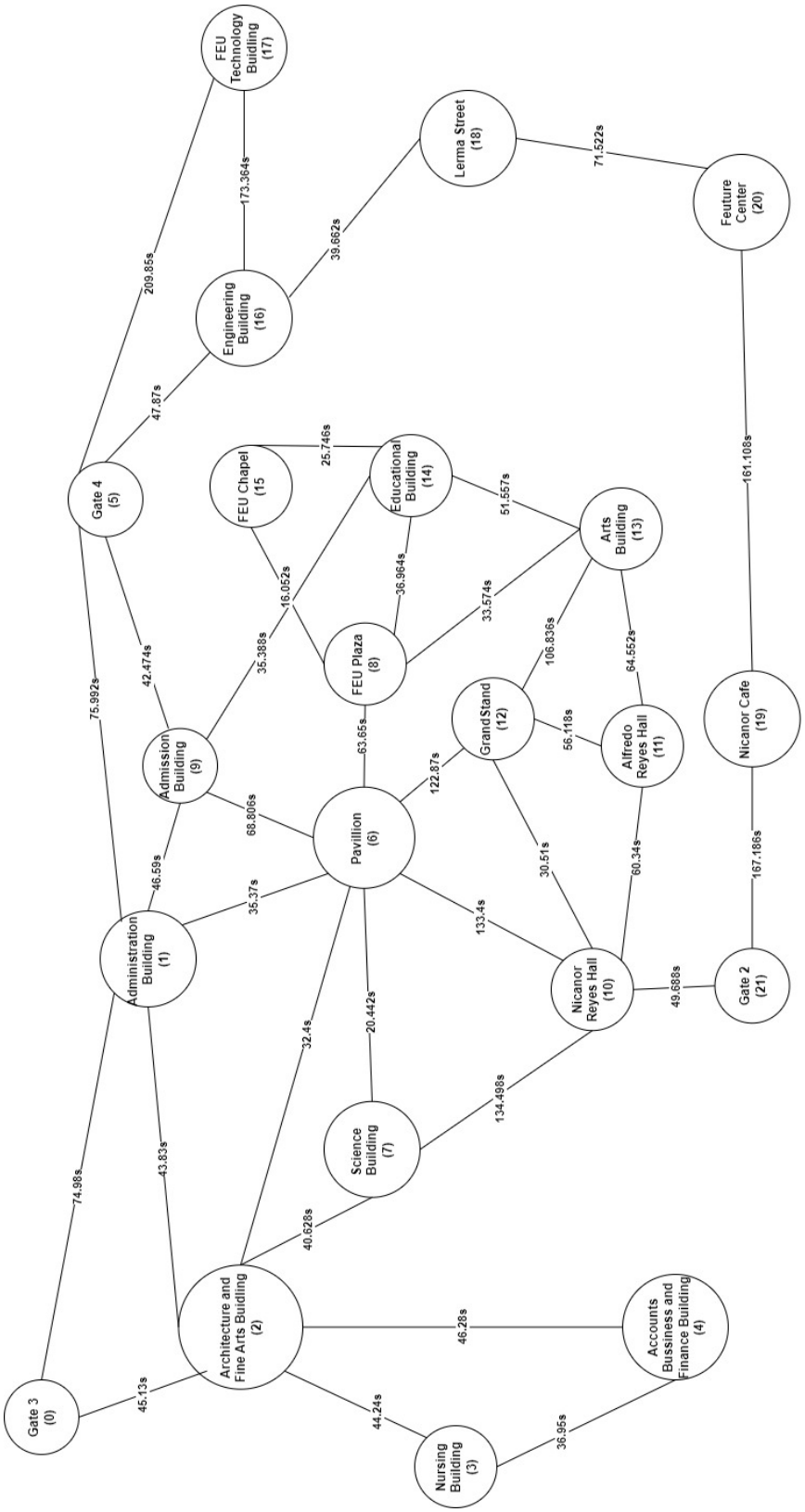

Figure 3.1: Time Graph

The figure above 3.1 represents a weighted graph from which an adjacency matrix can be formed, illustrating the distances as edges, which can be found in Appendix B. Each row and column of the matrix corresponds to a location in the graph, and the values represent the time (weights) between respective locations. In a weighted graph, the absence of a value indicates that the weight or distance is not defined for the corresponding pair of nodes. This can occur if there is no direct path between the locations. Although there are more alternative routes or pathways, this study only focused on one pathway predetermined by the researchers. Particularly, this is useful in running loops in Python, which is why we started at 0. Additionally, the abbreviation is used for concise notation for convenience to shorten the discussion. The following table assigns the nodes and indices used in the graph for FEU-Manila buildings:

Building	Nodes	Abbreviation
Gate 3	0	G3
Adminstration Building	1	AB
Architecture & Fine Arts	2	IARFA
Nursing Building	3	NB
Accounts, Business, & Finance	4	IABF
Gate 4	5	G4
Pavilion	6	Pav
Science Building	7	SB
FEU Plaza	8	FPark
Admission Building	9	ADB
Nicanor Reyes Hall	10	NRH
Alfredo Reyes Hall	11	ARH
Grandstand	12	GS
Arts Building	13	ArtsB
Educational Building	14	EB
FEU Chapel	15	Chapel
Enigneering Building	16	ENB
FEU Technology Building	17	FEUTECH
Lerma	18	Lerma
Nicanor Cafe	19	NC
FEUture Center	20	FC
Gate 2	21	G2

Figure 3.2: Index and Abbreviation for Far Eastern University-Manila buildings

This study use quantitative methods, involving the use of mathematical, computational, and statistical methods to establish relationships between variables, as defined by S. Ahmad et al. (2019). Data gathering involved physically walking from one building to another on the Far Eastern University campus. The researchers conducted five trials for each path, averaged the time and used these averages as the weights for the edges in our graph. This location was selected as it provided the necessary resources and environment to develop code for Dijkstra's Algorithm and heuristic approaches. In analyzing the data, experimental design was used. Yan et al. (2020), conducted simulated experiments and

gathered results to validate hypotheses about optimal pathways between campus buildings. Three heuristics were implemented and evaluated: (1) The Greedy Algorithm, (2) Breadth-First Search Algorithm, and (3) Optimal route with chosen node. Comparative analysis across conditions quantified the extent to which each heuristic impacted the identification of efficient routes across the campus.

3.2 Research Design

This research aims to analyze and compare the results of the three heuristic methods with Dijkstra's Algorithm with the following objectives: (1) The researchers intended to write the code and compare the computational runtime of the four mentioned algorithms; (2) Identifying the paths suggested by the four mentioned algorithms from one building to another; and (3) The comparison of the efficiency and accuracy in finding the shortest path from one building to another in Far Eastern University in terms of time or distance between Dijkstra's Algorithm and the three heuristic approaches. Hence, the following are the general procedures conducted to outline the essential steps in coding an algorithm, providing a concise overview of the implementation process:

1. Algorithm Design

    - Read data from an Excel file into a pandas DataFrame, representing a matrix of weights.
    - Create an undirected graph using NetworkX, with nodes representing DataFrame indices and weighted edges representing non-zero matrix values.
    - Use Dijkstra's Algorithm to calculate the shortest paths and their lengths between user-specified source and destination nodes.
    - Visualize the graph using Matplotlib, highlighting critical path nodes and edge weights.

- Identify the critical path as the shortest path from the source to the last node based on distance.
- Highlight the critical path edges in the graph visualization.
- Output the results, including shortest paths, total critical path distance, and specific nodes in the critical path.

2. **Pseudocode Development:** The researchers produced an algorithm in pseudocode to provide a high-level representation of the steps without focusing on specific programming syntax.

3. **Programming Language Selection:** The algorithm was implemented in Python and executed in a Jupyter Notebook. Python's readability and libraries like NetworkX and pandas enabled efficient coding.

4. **Parameterization:** The code was designed with adjustable parameters and constants to ensure flexibility and adaptability across multiple possibilities.

5. **Testing and Debugging:** The algorithm underwent rigorous testing to ensure proper functionality, with any issues addressed through systematic debugging.

6. **Validation:** The algorithm's effectiveness and reliability were validated through comparisons with benchmarks, testing on known datasets, and verification against theoretical expectations.

7. **Documentation:**

   (a) **Inline Comments:** Detailed comments were strategically placed within the code to explain complex logic, highlight key decision points, and provide insights into variable purposes.

   (b) **Function and Variable Naming:** This allows readers to follow the code's flow and understand the rationale behind specific design choices.

(c) **Pseudocode Integration:** The pseudocode developed during the algorithm design phase was seamlessly integrated into the comments.

(d) **Algorithmic Steps Explanation:** Each major algorithmic step was explained using inline comments to provide an overview of the purpose and functionality.

(e) **Graph Visualization Annotations:** Edge labels were incorporated to display the weights of each edge, aiding in the interpretation of the graph structure.

(f) **Results Output Comments:** Comments were added to the results output section, explaining the significance of each printed statement, such as the total distance and the critical path nodes.

Chapter 4

Results

This chapter presents the analysis and interpretation of the study conducted for the comparison of Dijkstra's Algorithm (DA) and the three heuristics in finding the Bidirectional Search between buildings in Far Eastern University to one another. The data obtained from the experiments is presented in tabulated form and has undergone statistical analysis. The corresponding analysis and interpretation of the data are incorporated in this section. The researchers compared the performance of Dijkstra's Algorithm, a well-known algorithm for finding the shortest path in a weighted graph, with three heuristic methods developed by the researchers: (1) The Greedy Algorithm, (2) the Breadth-First Search Algorithm, and (3) Bidirectional Search with chosen node.

The comparison was conducted through diagnostic tests using four algorithms: Dijkstra's Algorithm (DA), the Greedy Algorithm (H1), Depth-First Search (H2), and a Bidirectional Search with chosen node (H3). The tests utilized three different starting points: the Science Building, the Arts Building, Nicanor Reyes Hall, the Education Building, and the Nursing Building, to determine the optimal path to the FEUture Center on the university campus. This study aimed to evaluate whether there is a significant difference in the speed of Dijkstra's Algorithm compared to the three heuristic approaches. The Greedy Algorithm (H1) selects the smallest adjacent edge at each step until it reaches the destination. Depth-First Search (H2) explores the entire graph, searching for the path with the fewest nodes. The Bidirectional Search with chosen node with chosen nodes (H3) runs simultaneously from the start and end points, allowing users to select specific nodes to pass through.

4.1 Tables of any Given Routes with respect to time and Distance

Route	**Algorithms**	**Chosen Node**	**Shortest Path Time**	**Shortest Path Distance**	**Shortest Time (s) and Distance (m)**
Route SB to FC	Dijkstra's Algorithm	None	SB →Pav →ADB →G4 →ENB →Lerma →FC	SB →Pav →AB →G4 →ENB →Lerma →FC	**Time:** 290.776s **Distance:** 318.2m
	Greedy Path Algorithm	None	SB →Pav →IARFA →AB →ADB →EB →Chapel →FPark →ArtsB →ARH →GS →NRH →G2 →NC →FC	SB →Pav →AB →ADB →G4 →ENB →Lerma →FC	**Time:** 782.828s **Distance:** 354.5m
	Breadth First Search	None	SB →NRH →G2 →NC →FC	SB →NRH →G2 →NC →FC	**Time:** 512.48s **Distance:** 394.5m
	Bidirectional Search with chosen node	G4	SB →Pav →ADB →G4 →ENB →Lerma →FC	SB →Pav →AB →G4 →ENB →Lerma →FC	**Time:** 290.776s **Distance:** 318.2m

Table 4.2: Table for Science Building to FEUture Center Building

In terms of path finding and total distances, the Bidirectional Search with chosen node and Dijkstra's Algorithm generated identical shortest time (290.776s) and shortest distance (318.2m). Greedy Path Algorithm (Heuristics 1) took longer (728.828s) than Breadth-First Search (Heuristics 2) (512.48s), but Breadth-First Search covered a larger distance (394.5m) compared to Greedy Path Algorithm (354.5m).

Route	**Algorithms**	**Chosen Node**	**Shortest Path Time**	**Shortest Path Distance**	**Shortest Time (s) and Distance (m)**
Route AB to FC	Dijkstra's Algorithm	None	ArtsB →EB →ADB →G4 →ENB →Lerma →FC	ArtsB →EB →ADB →G4 →ENB →Lerma →FC	**Time:** 288.473s **Distance:** 327.9m
	Greedy Path Algorithm	None	ArtsB →FPark →Chapel →EB →ADB →G4 →ENB →Lerma →FC	ArtsB →GS →NRH →G2 →NC →FC	**Time:** 312.288s **Distance:** 364.7m
	Breadth First Search	None	ArtsB →ARH →NRH →G2 →NC →FC	ArtsB →ARH →NRH →G2 →NC →FC	**Time:** 502.874s **Distance:** 392.7m
	Bidirectional Search with chosen node	FPark	ArtsB →FPark →EB →ADB →G4 →ENB →Lerma →FC	ArtsB →FPark →Pav →IARFA →AB →G4 →ENB →Lerma →FC	**Time:** 307.454s **Distance:** 422.6m

Table 4.4: Table for Arts Building to FEUture Center Building

The table in 4.4, the performance of Dijkstra's Algorithm and three different heuristics is compared based on the paths found and their total distances. The shortest time of Bidirectional Search with chosen node (307.454s) is closer to Dijkstra's Algorithm which produced the identical shortest time (288.473s) while the shortest distance of the Bidirectional Search with chosen node (422.6m) has the largest distance among the three algorithms. Breadth-First Search (Heuristics 2) showed longer times and distances (502.874s and 392.7m) compared to Greedy Path Algorithm (Heuristics 1) with (312.288s) and (364.7m).

Route	**Algorithms**	**Chosen Node**	**Shortest Path Time**	**Shortest Path Distance**	**Shortest Time (s) and Distance (m)**
Route NRH to FC	Dijkstra's Algorithm	None	NRH - G2 - NC - FC	NRH - G2 - NC - FC	**Time:** 377.98s **Distance:** 283.5m
	Greedy Path Algorithm	None	NRH →GS →ARH →ADB →FPark →Chapel →EB →ArtsB →G4 →ENB →Lerma →FC	NRH →G2 →NC →FC	**Time: 463.47s Distance:** 283.5m
	Breadth First Search	None	NRH →G2 →NC →FC	NRH →G2 →NC →FC	**Time:** 377.98s **Distance:** 283.5m
	Bidirectional Search with chosen node	Lerma	NRH →Pav →ADB →G4 →ENB →Lerma →FC	NRH →Pav →ADB →G4 →ENB →Lerma →FC	**Time:** 377.98s **Distance:** 283.5m

Table 4.6: Table for Nicanor Reyes Hall Building to FEUture Center Building

The table in 4.6 has the same shortest distance and path distance. According to the of Greedy Path Algorithm, it results in having only one difference among others when it comes to time and path time. The Greedy Path Algorithm only chooses the shortest edge towards their local path or to their current neighbor which results in having different time and path time. The results of the same distance and path distance is that the source node which is Nicanor Reyes Hall has only a few paths and this can conclude that the shorter the neighbor, the similar the result of the path. However, when it comes to the execution time, the Bidirectional Search with chosen node is the fastest while Dijkstra's Algorithm is the slowest. Greedy Path Algorithm is faster than the Breadth-First Search which still results the Greedy Path Algorithm and the Breadth-First Search being the middle when it comes to the execution time.

Route	**Algorithms**	**Chosen Node**	**Shortest Path Time**	**Shortest Path Distance**	**Shortest Time (s) and Distance (m)**
Route EB to FC	Dijkstra's Algorithm	None	EB →ADB →G4 →ENB →Lerma →FC	EB →ADB →G4 →ENB →Lerma →FC	**Time:** 236.916s **Distance:** 285m
	Greedy Path Algorithm	None	EB →Chapel →FPark →ArtsB →ARH →GS →NRH →G2 →NC →FC	EB →Chapel →FPark →ArtsB →GS →NRH →G2 →NC →FC	**Time:** 604.534s **Distance:** 438.9m
	Breadth First Search	None	EB →ADB →G4 →ENB →Lerma →FC	EB →ADB →G4 →ENB →Lerma →FC	**Time:** 236.916s **Distance:** 285m
	Bidirectional Search with chosen node	NRH	EB →ArtsB →ARH →NRH →G2 →NC →FC	EB →ArtsB →GS →NRH →G2 →NC →FC	**Time:** 554.431s **Distance:** 407.6m

Table 4.8: Table for Education Building to FEUture Center Building

The table in 4.8 has two difference which is The Greedy Path Algorithm and the Bidirectional Search with chosen node. All of the algorithms have the same time and path time as well as the distance and path distance except for the Greedy Path Algorithm and the Bidirectional Search with chosen node. The Algorithm of the Greedy Path and the Bidirectional Search with chosen node affects the difference between the shortest path, time, and distance and the number of neighbors implementing Dijkstra's Algorithm and Breadth-First Search affects the similarities of the path, time, and distance.

Route	**Algorithms**	**Chosen Node**	**Shortest Path Time**	**Shortest Path Distance**	**Shortest Time (s) and Distance (m)**
Route NB to FC	Dijkstra's Algorithm	None	NB →IARFA →AB →G4 →ENB →Lerma →FC	NB →IARFA →AB →G4 →ENB →Lerma →FC	**Time:** 323.114s **Distance:** 328.6m
	Greedy Path Algorithm	None	NB →IABF →IARFA →Pav →SB →NRH →GS →ARH →ArtsB →FPark →Chapel →EB →ADB →G4 →ENB →Lerma →FC	NB →IARFA →AB →G4 →ENB →Lerma →FC	**Time:** 733.676s **Distance:** 328.6m
	Breadth First Search	None	NB →IARFA →AB →G4 →ENB →Lerma →FC	NB →IARFA →AB →G4 →ENB →Lerma →FC	**Time:** 323.114s **Distance:** 328.6m
	Bidirectional Search with chosen node	IARFA	NB →IARFA →AB →G4 →ENB →Lerma →FC	NB →IARFA →AB →G4 →ENB →Lerma →FC	**Time:** 323.114s **Distance:** 328.6m

Table 4.10: Table for Nursing Building to FEUture Center Building

The table in 4.10 has only one difference which is The Greedy Path Algorithm in terms of the shortest time. All of the algorithms have the same shortest time and shortest path time except for the Greedy Path Algorithm while all the shortest path distances and shortest distances are the same. The Algorithm of the Greedy Path affects the difference between the shortest path time and the shortest time because it only chooses the shortest edge in the local path.

4.2 Statistical Analysis of the Computational Runtime

Descriptive Statistics for Science Building to FEUture Center

Descriptive Statistics

	SB to FC - DA	SB to FC - H1	SB to FC - H2	SB to FC - H3
Valid	15	15	15	15
Missing	0	0	0	0
Mean	0.002806	0.001636	0.001874	0.001161
Std. Deviation	0.000693	0.000599	0.000649	0.000527
Minimum	0.001000	0.001002	0.001023	0.000182
Maximum	0.003576	0.002518	0.003315	0.002555

Figure 4.1: Descriptive Statistics of the Route of Science Building to FEUture Center

This table 4.1 presents statistics for four variables related to Science Building to FEUture Center. Each column represents a variable: Dijkstra's Algorithm (DA), Greedy Algorithm (H1), Breadth-First Search (H2), and Bidirectional Search with chosen node with chosen nodes (H3). Descriptive statistics were calculated using JASP Version 18.3. There are 15 valid data points for all variables. It shows the mean, standard deviation, minimum, and maximum values. The means range from approximately 0.001 to 0.003, suggesting small average values across the variables. Dijkstra's Algorithm (DA) has the highest mean (M = 0.002806), indicating the slowest average runtime, while Bidirectional Search with chosen node with chosen nodes (H3) has the lowest mean (M = 0.001161), suggesting the fastest average runtime. The standard deviations indicate how spread out the data is from the mean. Dijkstra's Algorithm (DA) has the highest standard deviation (SD = 0.000693), showing a wider spread of runtime compared to other variables.

Descriptive Statistics for Arts Building to FEUture Center

Descriptive Statistics

	AB to FC - DA	AB to FC- H1	AB to FC - H2	AB to FC - H3
Valid	15	15	15	15
Missing	0	0	0	0
Mean	0.002833	0.001209	0.001907	0.001412
Std. Deviation	0.000757	0.000291	0.000571	0.000547
Minimum	0.001526	0.000998	0.000994	0.000998
Maximum	0.004333	0.002024	0.003010	0.002553

Figure 4.2: Descriptive Statistics of the Route of Arts Building to FEUture Center

This table 4.2 displays statistics for four variables related to Arts Building to FEUture Center: Dijkstra's Algorithm (DA), Greedy Algorithm (H1), Breadth-First Search (H2), and Bidirectional Search with chosen node with chosen nodes (H3). Descriptive statistics were calculated using JASP Version 18.3. There are 15 valid data points for all variables. It includes mean, standard deviation, minimum, and maximum values. The means range from about 0.001 to 0.003, indicating small average values across the variables. Dijkstra's Algorithm (DA) has the highest mean (M = 0.002833), suggesting the slowest average runtime, while Greedy Algorithm (H1) has the lowest mean (M = 0.001209), implying the fastest average runtime. Standard deviations show how data spreads from the mean. Dijkstra's Algorithm (DA) has the highest standard deviation (SD = 0.000757), indicating wider runtime variations compared to other variables.

Descriptive Statistics for Nicanor Reyes Hall to FEUture Center

Descriptive Statistics

	NRH to FC - DA	NRH to FC- H1	NRH to FC - H2	NRH to FC - H3
Valid	15	15	15	15
Missing	0	0	0	0
Mean	0.002580	0.001436	0.001343	0.001271
Std. Deviation	0.000454	0.000515	0.000445	0.000422
Minimum	0.001997	0.000995	0.000911	0.000996
Maximum	0.003352	0.002534	0.002092	0.002289

Figure 4.3: Descriptive Statistics of the Route of Nicanor Reyes Hall Building to FEUture Center

Descriptive statistics were calculated using JASP Version 18.3.This table 4.3 shows stats for four variables related to Nicanor Reyes Hall Building to FEUture Center: Dijkstra's Algorithm (DA), Greedy Algorithm (H1), Breadth-First Search (H2), and Bidirectional Search with chosen node with chosen nodes (H3). Each column displays the count of valid (15) data points, mean, standard deviation, minimum, and maximum values. The means represent the average runtime, with Dijkstra's Algorithm (DA) having the highest mean (M = 0.002580), suggesting it's the slowest among these variables. Conversely, Bidirectional Search with chosen node with nodes (H3) has the lowest mean (M = 0.001271), indicating the fastest average runtime. Standard deviations measure the spread of data from the mean. Greedy H1 has the highest standard deviation (SD = 0.000515), implying wider runtime variations compared to other variables.

Descriptive Statistics for Education Building to FEUture Center

Descriptive Statistics

	EB to FC - DA	EB to FC - H1	EB to FC - H2	EB to FC - H3
Valid	15	15	15	15
Missing	0	0	0	0
Mean	0.002490	0.001333	0.001717	0.000857
Std. Deviation	0.000414	0.000432	0.000617	0.000461
Minimum	0.001843	0.000703	0.000621	0.000181
Maximum	0.003327	0.002288	0.002873	0.002128

Figure 4.4: Descriptive Statistics of the Route of Education Building to FEUture Center

Descriptive statistics were calculated using JASP Version 18.3. This table 4.4 displays stats for four variables related to computational runtime from the Education Building to the FEUture Center : Dijkstra's Algorithm (DA), Greedy Algorithm (H1), Breadth-First Search (H2), and Bidirectional Search with chosen node with chosen nodes (H3). It includes counts of valid (15) data points, mean, standard deviation, minimum, and maximum values for each variable. Dijkstra's Algorithm (DA) has the highest mean (M = 0.002490), suggesting the slowest average runtime, while Bidirectional Search with chosen node with nodes (H3) has the lowest mean (M = 0.000857), indicating the fastest average runtime. Standard deviations show how spread out the data is from the mean. Breadth-first Search has a higher standard deviation (SD = 0.000617), implying wider runtime variations compared to other variables.

Descriptive Statistics for Nursing Building to FEUture Center

Descriptive Statistics

	NB to FC - DA	NB to FC - H1	NB to FC - H2	NB to FC - H3
Valid	15	15	15	15
Missing	0	0	0	0
Mean	0.002643	0.001311	0.001696	0.001148
Std. Deviation	0.000345	0.000447	0.000813	0.000657
Minimum	0.002259	0.000508	0.000679	0.000276
Maximum	0.003496	0.002176	0.003355	0.002690

Figure 4.5: Descriptive Statistics of the Route of Nursing Building to FEUture Center

The table 4.5 presents descriptive statistics for the computational runtime of different algorithms used in the Nursing Building to FEUture Center. Descriptive statistics were calculated using JASP Version 18.3. There are 15 valid data points for each algorithm. The mean represents the average runtime, and the standard deviation shows the spread of the data from the mean. The algorithm with the highest mean runtime (M = 0.002643) is Dijkstra's Algorithm (DA), indicating it has the slowest runtime on average. Conversely, the algorithm with the lowest mean runtime (M = 0.001148) is Bidirectional Search with chosen node with nodes (H3), suggesting it has the fastest runtime on average.

Overall Conclusion, across all five buildings, Dijkstra's Algorithm exhibited slower computational runtimes compared to the other algorithms, while the Greedy Algorithm and Bidirectional Search with chosen node with nodes generally had the fastest runtimes. This information could be valuable in optimizing the algorithms used for these buildings and improving their computational efficiency.

4.2.1 Analysis of Variance of the given routes

In this Statistical Method, we implement the Analysis of Variance (ANOVA) towards the seconds in the execution runtime. In this case, we had constructed the Hypothesis in this statement wherein we have the means towards Dijkstra's Algorithm denoting μ_{DA}, Greedy Path Algorithm denoting μ_{H1}, Breadth-First Search denoting μ_{H2} and Bidirectional Search with chosen node denoting μ_{H3}

$$H_0 : \mu_{DA} = \mu_{H1} = \mu_{H2} = \mu_{H3} \tag{4.1}$$

This is the null hypothesis which means that in the means of the execution runtime of the algorithm, there is no significant difference towards each other

$$H_A : at \quad least \quad one \quad mean \quad has \quad a \quad significant \quad differences \tag{4.2}$$

This is the alternative hypothesis which means that in the means of the execution runtime of the algorithm, there exist one or more means that has significant difference

Analysis of Variance for Science Building to FEUture Center

ANOVA - Time - SB to FC

Cases	Sum of Squares	df	Mean Square	F	p	η^2
Algorithm	2.152×10^{-5}	3	7.172×10^{-6}	18.650	< .001	0.500
Residuals	2.154×10^{-5}	56	3.846×10^{-7}			

Note. Type III Sum of Squares

Post Hoc Tests

Standard

Post Hoc Comparisons - Algorithm

		Mean Difference	SE	t	p_{tukey}
Bidirectional	Dijkstra	−0.002	2.264×10^{-4}	−7.267	< .001***
	Greedy	-4.752×10^{-4}	2.264×10^{-4}	−2.098	0.166
	(Breadth-First Search)	-7.131×10^{-4}	2.264×10^{-4}	−3.149	0.014*
Dijkstra	Greedy	0.001	2.264×10^{-4}	5.168	< .001***
	(Breadth-First Search)	9.324×10^{-4}	2.264×10^{-4}	4.118	< .001***
Greedy	(Breadth-First Search)	-2.380×10^{-4}	2.264×10^{-4}	−1.051	0.720

* p < .05, ** p < .01, *** p < .001

Note. P-value adjusted for comparing a family of 4

Figure 4.6: Analysis of Variance for Science Building to FEUture Center

This analysis of the route of the execution time of the Science Building to the FEUture Center Building in figure 4.6 implements the use of One-Way ANOVA. One-way

ANOVA was utilized to determine if significant differences exist between the conditions. The results show a significant difference in runtime across the four algorithms, $F(3, 56) = 18.650, p < 0.001$ indicates that there are statistically significant differences among the mean performance of the four algorithms. Tukey's HSD post hoc test was used for pairwise comparison. It indicates that Dijkstra's Algorithm is significantly higher than the Greedy Algorithm (p= 0.000019), Bidirectional Search with chosen node (p = 7.480528×10^{-9}), and Breadth-First Search (p = 0.000717). These findings show that the choice of algorithm significantly affects the computational runtime for the path-finding problem. Dijkstra's Algorithm has substantially higher runtimes compared to the Greedy Algorithm, Breadth-First Search Algorithm, and Bidirectional Search with chosen node. Because of the accuracy of Dijkstra's Algorithm, the longer runtime satisfies the goal of this study which is to compare Dijkstra's Algorithm to all heuristics. Hence, in this analysis, the Bidirectional Search with chosen node with a chosen node has the fastest computational runtime.

Analysis of Variance for Arts Building to FEUture Center

ANOVA - Time - AB to FC

Cases	Sum of Squares	df	Mean Square	F	p	η^2
Algorithm_36	2.358×10^{-5}	3	7.861×10^{-6}	24.513	< .001	0.568
Residuals	1.796×10^{-5}	56	3.207×10^{-7}			

Note. Type III Sum of Squares

Post Hoc Tests

Standard

Post Hoc Comparisons - Algorithm_36

		Mean Difference	SE	t	p_{tukey}
Bidirectional	Dijkstra	−0.001	2.068×10^{-4}	−6.872	< .001***
	Greedy	2.033×10^{-4}	2.068×10^{-4}	0.983	0.760
	(Breadth-First Search)	-4.948×10^{-4}	2.068×10^{-4}	−2.393	0.090
Dijkstra	Greedy	0.002	2.068×10^{-4}	7.855	< .001***
	(Breadth-First Search)	9.261×10^{-4}	2.068×10^{-4}	4.479	< .001***
Greedy	(Breadth-First Search)	-6.981×10^{-4}	2.068×10^{-4}	−3.376	0.007**

* p < .05, ** p < .01, *** p < .001

Note. P-value adjusted for comparing a family of 4

Figure 4.7: Analysis of Variance for Arts Building to FEUture Center

This analysis of the route of the execution time of the Arts Building to the FEUture Center Building in figure 4.7 implements the use of One-Way ANOVA. One-way ANOVA was utilized to determine if significant differences exist between the conditions. The results show a significant difference in runtime across the four algorithms, $F(3, 56) = 24.513, p < 0.001$ indicates that there are statistically significant differences among the mean performance of the four algorithms. Tukey's HSD post-hoc test was used for pairwise comparison. It indicates that Dijkstra's Algorithm is significantly higher than the Greedy Algorithm ($p = 8.076173 \times 10^{-10}$), Bidirectional Search with chosen node ($p = 3.354886 \times 10^{-8}$), and Breadth-First Search ($p = 0.000215$). The Dijkstra's Algorithm and Greedy Path Algorithm performs significantly differently from the other algorithms, according to the post hoc tests and the ANOVA results, which show significant differences between the methods. The fastest computational runtime is achieved by the Greedy Algorithm.

Analysis of Variance for Nicanor Reyes Hall to FEUture Center ▾

ANOVA - Time - NRH to FC

Cases	Sum of Squares	df	Mean Square	F	p	η^2
Algorithm_39	1.721×10^{-5}	3	5.737×10^{-6}	27.094	< .001	0.592
Residuals	1.186×10^{-5}	56	2.118×10^{-7}			

Note. Type III Sum of Squares

Post Hoc Tests

Standard

Post Hoc Comparisons - Algorithm_39

		Mean Difference	SE	t	P_{tukey}
Bidirectional	Dijkstra	−0.001	1.680×10^{-4}	−7.787	< .001***
	Greedy	-1.649×10^{-4}	1.680×10^{-4}	−0.981	0.761
	(Breadth-First Search)	-7.215×10^{-5}	1.680×10^{-4}	−0.429	0.973
Dijkstra	Greedy	0.001	1.680×10^{-4}	6.806	< .001***
	(Breadth-First Search)	0.001	1.680×10^{-4}	7.358	< .001***
Greedy	(Breadth-First Search)	9.273×10^{-5}	1.680×10^{-4}	0.552	0.946

*** p < .001

Note. P-value adjusted for comparing a family of 4

Figure 4.8: Analysis of Variance for Nicanor Reyes Building to FEUture Center

This analysis of the route of the execution time of the Nicanor Reyes Hall Building

to the FEUture Center Building in figure 4.8 implements the use of One-Way ANOVA. One-way ANOVA was utilized to determine if significant differences exist between the conditions. The results show a significant difference in runtime across the four algorithms, $F(3, 56) = 27.094, p < 0.001$ indicates that there are statistically significant differences among the mean performance of the four algorithms. Tukey's HSD post-hoc test was used for pairwise comparison. It indicates that Dijkstra's Algorithm is significantly higher than the Greedy Algorithm ($p = 4.303771x10^{-8}$), Bidirectional Search with chosen node ($p = 1.041331 \times 10^{-9}$), and Breadth-First Search ($p = 5.291062 \times 10^{-9}$). The post hoc tests show that the Dijkstra's algorithm differs significantly from the other algorithms (Bidirectional, Breadth-First Search, and Greedy), and the ANOVA findings show significant differences among the algorithms. The fastest com- computational runtime is achieved by the Bidirectional Search with chosen node with a selected node for this reason.

Analysis of Variance for Eductional Building to FEUture Center

ANOVA - Time - EB to FC

Cases	Sum of Squares	df	Mean Square	F	p	η²
Algorithm_42	2.145×10^{-5}	3	7.151×10^{-6}	30.061	< .001	0.617
Residuals	1.332×10^{-5}	56	2.379×10^{-7}			

Note. Type III Sum of Squares

Post Hoc Tests

Standard

Post Hoc Comparisons - Algorithm_42

		Mean Difference	SE	t	P_{tukey}
Bidirectional	Dijkstra	−0.002	1.781×10^{-4}	−9.174	< .001***
	Greedy	-4.766×10^{-4}	1.781×10^{-4}	−2.676	0.047*
	(Breadth-First Search)	-8.602×10^{-4}	1.781×10^{-4}	−4.830	< .001***
Dijkstra	Greedy	0.001	1.781×10^{-4}	6.498	< .001***
	(Breadth-First Search)	7.735×10^{-4}	1.781×10^{-4}	4.343	< .001***
Greedy	(Breadth-First Search)	-3.836×10^{-4}	1.781×10^{-4}	−2.154	0.149

* p < .05, *** p < .001

Note. P-value adjusted for comparing a family of 4

Figure 4.9: Analysis of Variance for Education Building to FEUture Center

This analysis of the route of the execution time of the Education Building to the

FEUture Center Building in Figure 4.9 implements the use of One-Way ANOVA. One-way ANOVA was utilized to determine if significant differences exist between the conditions. The results show a significant difference in runtime across the four algorithms, $F(3, 56) = 30.061, p < 0.001$ indicates that there are statistically significant differences among the mean performance of the four algorithms. Tukey's HSD post hoc test was used for pairwise comparison. It indicates that Dijkstra's Algorithm is significantly higher than the Greedy Algorithm ($p = 1.385111 \times 10^{-7}$), Bidirectional Search with chosen node ($p = 1.301093 \times 10^{-11}$), and Breadth-First Search ($p = 0.000339$). Also, the Breadth-First Search and Bidirectional Search with chosen node ($p = 0.000064$) is highly significant. The post-hoc tests show that the Dijkstra, Bidirectional, and Breadth-First Search algorithms differ significantly from one another in terms of the performance metric under study. The ANOVA findings also show significant differences across the methods. Furthermore, there are differences between the Greedy algorithm and Dijkstra's Algorithm and Breadth-First Search, but not with the Bidirectional Search with chosen node.

Analysis of Variance for Nursing Building to FEUture Center

ANOVA - Time - NB to FC

Cases	Sum of Squares	df	Mean Square	F	p	η^2
Algorithm_45	2.017×10^{-5}	3	6.724×10^{-6}	19.066	< .001	0.505
Residuals	1.975×10^{-5}	56	3.527×10^{-7}			

Note. Type III Sum of Squares

Post Hoc Tests

Standard

Post Hoc Comparisons - Algorithm_45

		Mean Difference	SE	t	p_{tukey}
Bidirectional	Dijkstra	−0.001	2.169×10^{-4}	−6.893	< .001***
	Greedy	-1.628×10^{-4}	2.169×10^{-4}	−0.751	0.876
	(Breadth-First Search)	-5.481×10^{-4}	2.169×10^{-4}	−2.528	0.066
Dijkstra	Greedy	0.001	2.169×10^{-4}	6.142	< .001***
	(Breadth-First Search)	9.466×10^{-4}	2.169×10^{-4}	4.365	< .001***
Greedy	(Breadth-First Search)	-3.853×10^{-4}	2.169×10^{-4}	−1.777	0.295

* p < .05, *** p < .001

Note. P-value adjusted for comparing a family of 4

Figure 4.10: Analysis of Variance for Nursing Building to FEUture Center

In this analysis of the route of the execution time of the Nursing Building to the FEUture Center Building in Figure 4.10, One-way ANOVA was utilized to determine if significant differences exist between the conditions. The results show a significant difference in runtime across the four algorithms, $F(3, 56) = 19.066, p < 0.001$ this indicates that there are statistically significant differences among the mean performance of the four algorithms. Tukey's HSD post-hoc test was used for pairwise comparison. It indicates that Dijkstra's Algorithm is significantly higher than the Greedy Algorithm ($p = 0.000019$), Bidirectional Search with chosen node ($p = 7.480528 \times 10^{-9}$), and Breadth-First Search ($p = 0.000717$). These findings show that the choice of algorithm significantly affects the computational runtime for the path-finding problem. Dijkstra's Algorithm has substantially higher runtimes compared to the Greedy Algorithm, Breadth-First Search Algorithm, and Bidirectional Search with chosen node. Because of the accuracy of Dijkstra's Algorithm, the longer runtime satisfies the goal of this study which is to compare Dijkstra's algorithm to all heuristics. Hence, in this analysis, the Bidirectional Searc with a chosen node has the fastest computational runtime.

Chapter 5

Conclusion

This chapter presents the summary of finding, conclusion and recommendation for further studies

5.1 Summary of Findings

From a thorough analysis of the data collect and result obtained, the following significant finding are summarized:

1. **Compare the computational runtime:**

Path	Dijkstra's Algorithm	Greedy Path	Breadth-First Search	Bidirectional Search with chosen node
Science Building to FEUture Center	0.002806	0.001636	0.001874	0.001161
Arts Building to FEUture Center	0.002833	0.001209	0.001907	0.0014112
Nicanor Reyes Hall Building to FEUture Center	0.00258	0.001436	0.001343	0.001271
Education Building to FEUture Center	0.00249	0.001333	0.001717	0.000857
Nursing Building to FEUture Center	0.002643	0.001311	0.001696	0.001148

Table 5.1: Average Execution Runtime of each Algorithm

The overall conclusion reached is that the Bidirectional Search with chosen node outperforms the other three algorithms, exhibiting the shortest computational runtime. This conclusion is drawn from the accuracy of the Dijkstra's Algorithm, which results in longer runtimes.

2. **Analyze the speed and accuracy differences between Dijkstra's algorithm and three heuristic approaches in graph pathfinding**

   (a) Based on the results of Table 4.2 (Science Building to FEUture Center), this shows that Bidirectional Search with chosen node has more accuracy than the

rest of the algorithm which has the same answer of the shortest time and shortest distance to Dijkstra's algorithm. However, Greedy Path Algorithm and Breadth-First Search Algorithm has less accuracy towards the Dijkstra's Algorithm which has larger shortest distance and shortest time. In terms of speed, comparing in the table 5.1, Bidirectional Search with chosen node is the fastest among other algorithm having the the least second in terms of computational runtime while the Greedy Path Algorithm and Breadth-First Search were in the middle level of speed which is in the middle in terms of the computational runtime.

In terms of the significant difference of the speed, in the analysis of variance in figure 4.6, Greedy Path has no significant difference towards Bidirectional Search with chosen nodend Breadth-First Search Algorithm while the rest of the comparison has significant difference. As much as Dijkstra's Algorithm, comparing the means of Dijkstra's Algorithm results having a significant difference to all heuristic algorithms

(b) When it comes to the accuracy in Table 4.4 (Arts Building to FEUture Center), the Bidirectional Search has a chosen node that is not part of the optimal route in Dijkstra's Algorithm which would lessen the accuracy. However, Greedy Path Algorithm and Breadth-First Search Algorithm has less accuracy towards the Dijkstra's Algorithm which has larger shortest distance and shortest time. In terms of speed, comparing in the table 5.1 Greedy Path has more speed towards other algorithms having the least second in terms of computational runtime while Bidirectional Search with chosen nodend Breadth-First Search Algorithm were in the middle level of speed which is in the middle in terms of the computational runtime.

In terms of the significant difference of the speed, in the analysis of variance in figure 4.7, Bidirectional Search with chosen node has no significant difference towards Greedy Path Algorithm and Breadth-First Search Algorithm

while the rest of the comparison has significant difference. As much as Dijkstra's Algorithm, comparing the means of Dijkstra's Algorithm results having a significant difference to all heuristic algorithms

(c) Table 4.6 (Nicanor Reyes Hall Building to FEUture Center) concludes that the Bidirectional Search with chosen node has more accuracy because it has the same results as Dijkstra's Algorithm in terms of the shortest distance and shortest time. The path in comparison to Dijkstra's Algorithm has a different path but the shortest time and distance is the same because Bidirectional Search with chosen node can also scan for the same shortest time and distance but different path. Breadth-First Search has the same answer of shortest time and shortest distance to Dijkstra's Algorithm which results in more accuracy but Breadth-First Search has more accuracy if the minimum number of nodes and the value of the shortest time and shortest distance in Breadth-First Search is equal to the number of nodes and the value of the shortest time and shortest distance of Dijkstra's Algorithm. Lastly in the Greedy Path Algorithm, since the algorithm is concerned with the shortest local edges, this has less accuracy which is the only difference among the other algorithms but looking at the shortest distance of the Greedy Path Algorithm, it was also the same among other Algorithms. In this case, when it comes to the neighbors, if the source node has less number of neighbors, it resulted of having the same shortest distance. In terms of speed, comparing in the table 5.1, Bidirectional Search with chosen node is the fastest among other algorithm having the the least second in terms of computational runtime while the Greedy Path Algorithm and Breadth-First Search were in the middle level of speed which is in the middle in terms of the computational runtime.

In terms of the significant difference of the speed, in the analysis of variance in figure 4.8, Bidirectional Search with chosen node has no significant difference towards Greedy Path Algorithm and Breadth-First Search Algorithm and

Breadth-First Search has no significant difference towards the Greedy Path Algorithm while the rest of the comparison has significant difference. As much as Dijkstra's Algorithm, comparing the means of Dijkstra's Algorithm results having a significant difference to all heuristic algorithms

(d) For Table 4.8 (Education Building to FEUture Center), concludes that Greedy Path Algorithm has the least speed which has a result of a different shortest time and shortest distance among other algorithms. Bidirectional Search with chosen node is the second to the slowest which has a different answer to Dijkstra's Algorithm but shorter time and distance to Greedy Path Algorithm and also affects the chosen node which is not in the optimal route in Dijkstra's Algorithm. Breadth-First Search has more accuracy which has the same shortest time and shortest distance towards Dijkstra's Algorithm. Breadth-First Search has the same answer of shortest time and distance to Dijkstra's Algorithm which results in more accuracy but Breadth-First Search has more accuracy if the number of nodes and the value of the shortest time and shortest distance in Breadth-First Search is equal to the number of nodes and the value of the shortest time and shortest distance path in Dijkstra's Algorithm. In terms of speed, comparing in the table 5.1 Bidirectional Search with chosen node is the fastest among other algorithm having the the least second in terms of computational runtime while the Greedy Path Algorithm and Breadth-First Search were in the middle level of speed which is in the middle in terms of the computational runtime.

In terms of the significant difference of the speed, in the analysis of variance in figure 4.9, only the Greedy Path Algorithm and Breadth-First Search has no significant difference while the rest of the comparison has significant difference. As much as Dijkstra's Algorithm, comparing the means of Dijkstra's Algorithm results having a significant difference to all heuristic algorithms

(e) For Table 4.10 (Nursing Building to FEUture Center), this concludes that

the Bidirectional Search with chosen node has more accuracy because it has the same results as Dijkstra's Algorithm. Breadth-First Search has the same answer of shortest time and distance to Dijkstra's Algorithm which results in more accuracy but Breadth-First Search has more accuracy if the number of nodes and the value of the shortest time and shortest distance in Breadth-First Search is equal to the number of nodes and the value of the shortest time and shortest distance in Dijkstra's Algorithm. Lastly in the Greedy Path Algorithm, since the algorithm is concerned with the shortest local edges, this has less accuracy which is the only difference among the other algorithms but looking at the shortest distance of the Greedy Path Algorithm, it was also the same among other Algorithms. In this case, from the neighbor, if the source node has less number of neighbors, it results in having the same shortest distance. In terms of speed, comparing in the table 5.1 Bidirectional Search with chosen node is the fastest among other algorithm having the the least second in terms of computational runtime while the Greedy Path Algorithm and Breadth-First Search were in the middle level of speed which is in the middle in terms of the computational runtime.

In terms of the significant difference of the speed, in the analysis of variance in figure 4.10, only the Bidirectional Search with chosen node and Greedy Path Algorithm has no significant difference while the rest of the comparison has significant difference. As much as Dijkstra's Algorithm, comparing the means of Dijkstra's Algorithm results having a significant difference to all heuristic algorithms

5.2 Conclusions

The Bidirectional Search with chosen node demonstrated superior performance in terms of speed and accuracy, making it the most suitable choice for path-finding problems here

at Far Eastern University. However, if the chosen node does not exist in the optimal path of the Dijkstra's Algorithm, The Bidirectional Search with chosen node would be less accurate. Conversely, Dijkstra's Algorithm exhibits exceptional accuracy compared to other algorithms, despite the slowest runtime. This makes Dijkstra's Algorithm the slowest in terms of computational runtime, yet it excels in finding the shortest path. However, the choice of algorithm may depend on specific requirements, such as whether the entire graph needs to be scanned or if the starting and destination nodes are predetermined.

5.3 Recommendations

Based on the results and conclusions of the study the following recommendations are offered.

The researchers produced the code without using libraries or packages, manually coding the algorithms to work with any matrix size, as long as it's a square of $n \times n$. This paper serves as a valuable resource for future researchers aiming to delve deeper into the topic. They can explore additional heuristics and contextualize common factors to further optimize algorithms and solutions for route optimization, advancing the field

Appendix A

Pseudocode

This contains the explanation of the how we create the code for each of the algorithm. We created the pseudocode of each of the algorithm but note that the pseudocode does not contain all of our codes but only the function of the algorithm

A.1 Dijkstra's Algorithm Pseudocode

```
1: procedure Input and Output of Dijkstra's Algorithm
2:     Input: graph[sourcenode, edge, weight], start
3:     Output: distance, shortest_path
4: end procedure
5: function Dijkstra(graph, start)
6:     distances ← {node: ∞ for node in graph}
7:     distances[start] ← 0
8:     previous_node ← {node: None for node in graph}
9:     priority_queue ← [(0, start)]
10:    while priority_queue is not empty do
11:        current_distance, current_node ← HeapPop(priority_queue)
12:        if current_distance > distances[current_node] then
13:            continue
14:        end if
15:        for neighbor, weight in graph[current_node].items() do
16:            distance ← current_distance + weight['weight']
17:            if distance < distances[neighbor] then
```

```
                distances[neighbor] ← distance
                previous_node[neighbor] ← current_node
                HeapPush(priority_queue, (distance, neighbor))
            end if
        end for
    end while
    shortest_paths ← {node: [] for node in graph}
    for node in graph do
        current_node ← node
        while previous_node[current_node] is not None do
            Insert current_node at the beginning of shortest_paths[node]
            current_node ← previous_node[current_node]
        end while
        if current_node = start then
            Insert start at the beginning of shortest_paths[node]
        end if
    end for
    return distances, shortest_paths
end function
```

A.2 Greedy Path Algorithm Pseudocode

```
procedure Input and Output of Greedy Path Algorithm
    Input: graph[sourcenode, edge, weight], start, destination
    Output: distance, shortest_path
end procedure
function GreedyTravel(graph, start, destination)
    current_node ← start
```

```
7:      visited_nodes ← [current_node]
8:      new_visited_nodes ← [start]
9:      total_distance ← 0
10:     duplicate_value ← 0
11:     while current_node ≠ destination do
12:         neighbors ← graph[current_node]
13:         unvisited_neighbors ← {node: weight for node, weight in enumerate(neighbors)
    if node not in visited_nodes and weight ≠ 0}
14:         count_dict ← COUNTER(unvisited_neighbors.values())
15:         result ← [key for key, value in unvisited_neighbors.items() if count_dict[value]
    > 1]
16:         duplicate_value ← LENGTH(result)
17:         if duplicate_value > 0 then
18:             break                          ▷ If same edge, the algorithm stops
19:         else
20:             if not unvisited_neighbors then          ▷ When path does not reach
    destination, reset the algorithm
21:                 current_node ← start
22:                 neighbors ← graph[current_node]
23:                 total_distance ← 0
24:                 unvisited_neighbors ← {node: weight for node, weight in enumer-
    ate(neighbors) if node not in visited_nodes and weight ≠ 0}
25:                 new_visited_nodes ← [start]
26:             end if
27:             if unvisited_neighbors then              ▷ Choosing the shortest edge
28:                 next_node ← MIN(unvisited_neighbors, key=unvisited_neighbors.get)
29:                 total_distance += unvisited_neighbors[next_node]
30:                 visited_nodes.APPEND(next_node)
```

```
                new_visited_nodes.APPEND(next_node)
                current_node ← next_node
            end if
        end if
    end while
    if duplicate_value > 0 then
        return [], 0, "Not Applicable"                    ▷ No path found
    else
        if current_node == destination then
            path ← ' → '.JOIN([rednode for node in new_visited_nodes])
            return new_visited_nodes, total_distance, path
        else
            return [], 0, ""                              ▷ No path found
        end if
    end if
end function
```

A.3 Breadth-First Search Pseudocode

```
procedure INPUT AND OUTPUT OF BREADTH-FIRST SEARCH ALGORITHM
    Input: graph[sourcenode, edge, weight], start, end
    Output: distance, shortest_path
end procedure
function GREEDYBFS(graph, start, end)
    frontier ← deque([(start, [start], 0)])          ▷ Queue of (node, path, cost)
    explored ← {}                                    ▷ Set of explored nodes
    while frontier is not empty do
        node, path, cost ← frontier.POPLEFT
```

```
10:         explored.ADD(node)
11:         if node == end then
12:             return path, cost
13:         end if
14:         for neighbor in graph[node] do
15:             if neighbor not in explored then
16:                 new_path ← path + [neighbor]
17:                 new_cost ← cost + graph[node][neighbor]['weight']
18:                 frontier.APPEND((neighbor, new_path, new_cost))
19:             end if
20:         end for
21:         Sort the frontier based on the heuristic function
22:         frontier ← deque(sorted(frontier, key=lambda x: heuristic(x[0], end)))
23:     end while
24:     return [], 0                                          ▷ No path found
25: end function

26: function HEURISTIC(node, end)
27:     # Implement your heuristic function here
28:     # For this example, we'll use the number of characters in common as
    the heuristic
29:     return −max(0, int(node == end))
30: end function
```

A.4 Bidirectional Search Algorithm

```
1: procedure INPUT AND OUTPUT OF BIDIRECTIONAL SEARCH ALGORITHM
2:     Input: graph[sourcenode, edge, weight], source, target, input_intersection
```

```
    Output: distance, shortest_path
end procedure
function BIDIRECTIONAL SEARCH WITH CHOSEN NODE(graph, source, target, input_intersection)
    # Run Dijkstra's algorithm from both directions
    path_forward_length, path_forward = DIJKSTRA(graph, source)
    path_backward_length, path_backward = DIJKSTRA(graph, target)
    intersection = SET(path_forward.keys()) ∩ SET(path_backward.keys())
    desired_intersection = input_intersection
    if intersection then
        shortest_distance = ∞
        shortest_path = None
        shortest_path_with_desired_intersection = None
        shortest_distance_with_desired_intersection = None
        for node in intersection do
            distance = path_forward_length[node] + path_backward_length[node]
            if distance < shortest_distance then
                if node == source or node == target then
                    continue
                else if node == desired_intersection then
                    shortest_distance = distance
                    path1 = path_forward[node][0:]
                    path2 = path_backward[node][::-1]
                    shortest_path = path1 + [n for n in path2 if n not in path1]
                    if desired_intersection in shortest_path then
                        shortest_path_with_desired_intersection = shortest_path
                        shortest_distance_with_desired_intersection = shortest_distance
                    else
```

```
30:                     continue
31:                 end if
32:             end if
33:         end if
34:     end for
35:     # Repeating Dijkstra's algorithm for the same distance
36:     other_distance = 0
37:     other_shortest_path = None
38:     new_shortest_path = []
39:     for node in intersection do
40:         other_distance = path_forward_length[node] + path_backward_length[node]
41:         if other_distance != shortest_distance_with_desired_intersection then
42:             continue
43:         else
44:             if node == source or node == target then
45:                 continue
46:             else if node == desired_intersection then
47:                 path1 = path_forward[node][0:]
48:                 path2 = path_backward[node][::-1]
49:                 other_shortest_path = path1 + [n for n in path2 if n not in
    path1]
50:                 if other_shortest_path == shortest_path_with_desired_intersection
    then
51:                     continue
52:                 else if desired_intersection in other_shortest_path then
53:                     new_shortest_path.append(other_shortest_path)
54:                 end if
55:             end if
```

```
            end if
        end for
        return shortest_path_with_desired_intersection, shortest_distance, new_shortest_path
    elsereturn None, ∞ # No path found
    end if
end function
```

Appendix B

Process of Distance and Time

This was the other specific process for how we made the distance and time for the given algorithm. In this case, we used the Excel in terms of creating the format of it. In the result in Chapter 4 we only use the computational runtime with respect to time and the Methodology in Chapter 3 the given was only the graph of the time. In Appendix B we will show the overall process of the distance and time.

B.1 Distance Graph

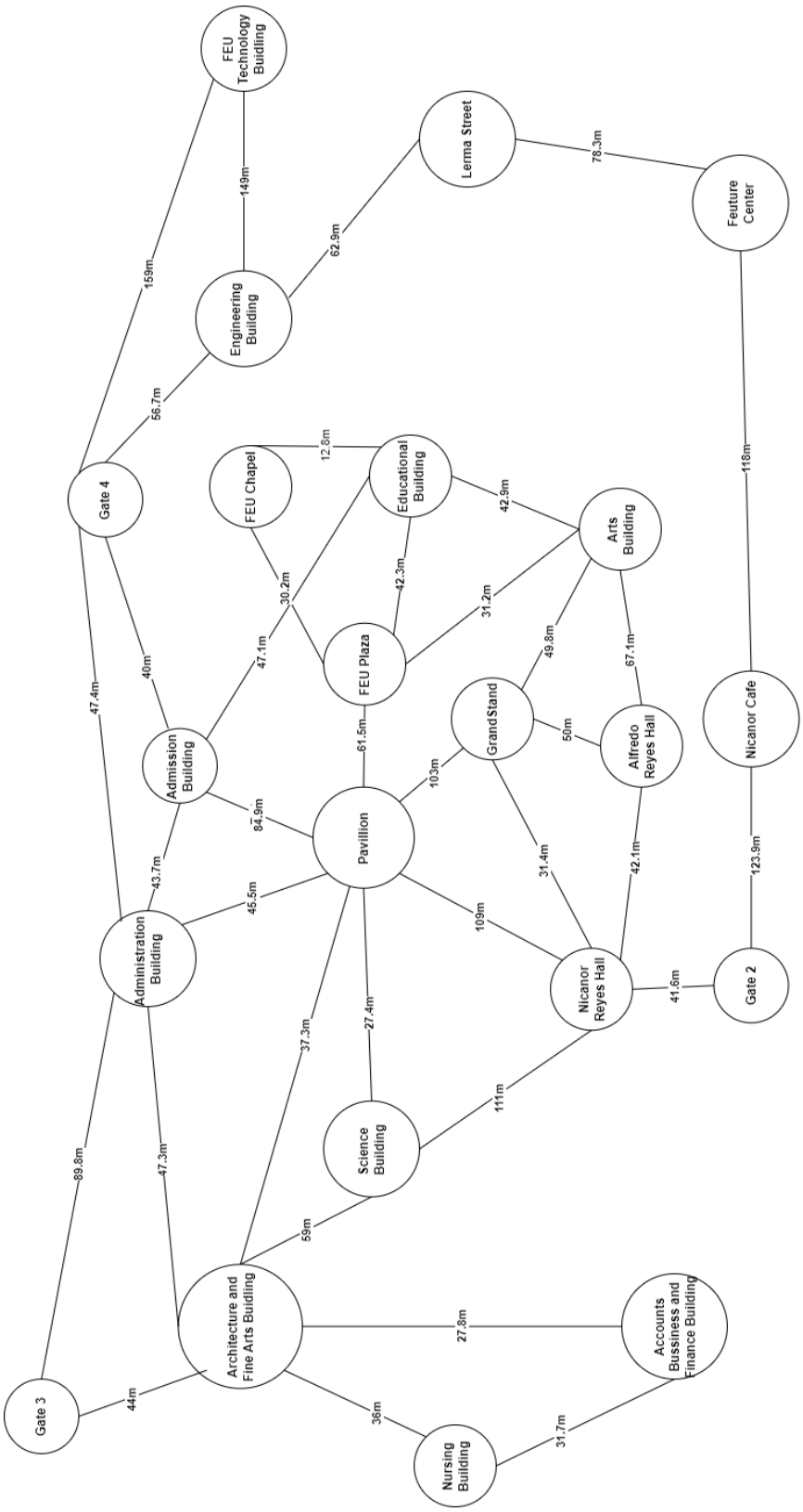

Figure B.1: Distance Graph

B.2 Matrices

This section shows the overall process of making the Excel file for the time and distance.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	74.976	45.132	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	74.976	0	43.83	0	0	75.992	35.374	0	0	46.59	0	0	0	0	0	0	0	0	0	0	0	0
2	45.132	43.83	0	44.238	46.276	0	32.044	40.628	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	44.238	0	36.948	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	46.276	36.948	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	75.992	0	0	0	0	0	0	0	42.474	0	0	0	0	0	0	47.87	209.854	0	0	0	0
6	0	35.374	32.044	0	0	0	0	20.442	63.648	68.806	133.422	0	122.87	0	0	0	0	0	0	0	0	0
7	0	0	40.628	0	0	0	20.442	0	0	0	134.498	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	63.648	0	0	0	0	0	0	33.574	36.964	16.052	0	0	0	0	0	0
9	0	46.59	0	0	0	42.474	68.806	0	0	0	0	0	0	0	35.388	0	0	0	0	0	0	0
10	0	0	0	0	0	0	133.422	134.498	0	0	0	60.34	30.51	0	0	0	0	0	0	0	0	49.688
11	0	0	0	0	0	0	0	0	0	0	60.34	0	56.118	64.552	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	122.866	0	0	0	30.51	56.118	0	106.838	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	33.574	0	0	64.552	106.838	0	51.557	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	36.964	35.388	0	0	0	51.557	0	25.746	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	16.052	0	0	0	0	0	25.746	0	0	0	0	0	0	0
16	0	0	0	0	0	47.87	0	0	0	0	0	0	0	0	0	0	0	173.364	39.662	0	0	0
17	0	0	A	0	0	209.854	0	0	0	0	0	0	0	0	0	0	173.364	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	39.662	0	0	0	71.522	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	161.108	167.186
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	71.522	161.108	0	0
21	0	0	0	0	0	0	0	0	0	0	49.688	0	0	0	0	0	0	0	0	167.186	0	0

Figure B.2: Time Matrix in Excel

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	89.8	44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	89.8	0	47.3	0	0	47.4	45.5	0	0	43.7	0	0	0	0	0	0	0	0	0	0	0	0
2	44	47.3	0	36	27.8	0	37.3	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	36	0	31.7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	27.8	31.7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	47.4	0	0	0	0	0	0	0	40	0	0	0	0	0	0	56.7	159	0	0	0	0
6	0	45.5	37.3	0	0	0	0	27.4	61.5	84.9	109	0	103	0	0	0	0	0	0	0	0	0
7	0	0	59	0	0	0	27.4	0	0	0	111	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	61.5	0	0	0	0	0	0	31.2	42.3	30.2	0	0	0	0	0	0
9	0	43.7	0	0	0	40	84.9	0	0	0	0	0	0	0	47.1	0	0	0	0	0	0	0
10	0	0	0	0	0	0	109	111	0	0	0	42.1	31.4	0	0	0	0	0	0	0	0	41.6
11	0	0	0	0	0	0	0	0	0	0	42.1	0	50	67.1	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	103	0	0	0	31.4	50	0	49.8	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	31.2	0	0	67.1	49.8	0	42.9	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	42.3	47.1	0	0	0	42.9	0	12.8	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	30.2	0	0	0	0	0	12.8	0	0	0	0	0	0	0
16	0	0	0	0	0	56.7	0	0	0	0	0	0	0	0	0	0	0	149	62.9	0	0	0
17	0	0	0	0	0	159	0	0	0	0	0	0	0	0	0	0	149	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	62.9	0	0	0	78.3	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	118	123.9
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	78.3	118	0	0
21	0	0	0	0	0	0	0	0	0	0	41.6	0	0	0	0	0	0	0	0	123.9	0	0

Figure B.3: Distance Matrix in Excel

The matrices in B.2 and B.3 has been inputted in the Excel. Note that the information that is not a number has been omitted in the Excel because the code will read the Excel not by file but by matrix. The Matrix is an adjacency matrix which has been taught in Graph Theory. Eventually, the numbers indicated are the weight of the graph. The row number and column number are the nodes while the location of the row number and its intersection towards the column number are the edges. For instance, Row 1 and Column 3 means the edge between the Row 1 and Column 3

Bibliography

AbuSalim, S. W. G., Ibrahim, R., Saringat, M. Z., Jamel, S., & Abdul Wahab, J. (2020). *Comparative analysis between dijkstra and bellman-ford algorithms in shortest path optimization.* Retrieved from `https://iopscience.iop.org/article/10.1088/1757-899X/917/1/012077/meta`

Ab Wahab, M. N., Nefti-Meziani, S., & Atyabi, A. (2020). A comparative review on mobile robot path planning: Classical or meta-heuristic methods? *Annual Reviews in Control*, *50*, 233-252.

Ahmad, N., & Aminuddin, W. M. W. M. (2023). Application of shortest path problem in the university campus using dijkstra's algorithm. In *Aip conference proceedings* (Vol. 2880).

Ahmad, S., Wasim, S., & Irfan, S. e. a. (2019). Qualitative v/s quantitative research- a summarized review. *J. Evid. Based Med. Health*.

Alam, M. A., & Faruq, M. O. (2009). Finding shortest path for road network using dijkstra's algorithm.
(Retrieved from `https://www.cribfb.com/journal/index.php/BJMSR/article/view/366`)

Aranski, A. W. (2022). Depth first search algorithm in solving the shortest route using the concept of generate and test. *IJISTECH (International Journal of Information System and Technology)*, *6*(3), 353–360.

Ata, K. M., Soh, A. C., Ishak, A. J., Jaafar, H., & Khairuddin, N. A. (2019). Smart indoor parking system based on dijkstra's algorithm. *Int. J. Integr. Eng*, *2*(1), 13-20.

Bolund, E. (2020). The challenge of measuring trade-offs in human life history research. *Evolution and Human Behavior*, *41*(6), 502-512.

Borowska-Stefańska, M., Kowalski, M., Turoboś, F., & Wiśniewski, S. (2021a). On determining the weight of edges in map-representing graphs-applications of heuristic methods in planning escape routes.

(Retrieved from `https://www.sciencedirect.com/science/article/pii/S2095756422001039`)

Borowska-Stefańska, M., Kowalski, M., Turoboś, F., & Wiśniewski, S. (2021b). On determining the weight of edges in map-representing graphs-applications of heuristic methods in planning escape routes.

(Retrieved from `https://www.sciencedirect.com/science/article/pii/S2095756422001039`)

Candra, A., Budiman, M. A., & Hartanto, K. (2020, July). Dijkstra's and a-star in finding the shortest path: a tutorial. *IEEE Xplore*. Retrieved from `https://ieeexplore.ieee.org/abstract/document/9190342` doi: 10.1109/ICITEE50287.2020.9190342

Chartrand, G., & Zhang, P. (2019). *Chromatic graph theory.* CRC Press.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press. (Retrieved from `https://books.google.com/books?id=2EpD_LGlU0C`)

Deepa, G., Kumar, P., Manimaran, A., Rajakumar, K., & Krishnamoorthy, V. (2018). Dijkstra algorithm application: Shortest distance between buildings. *International Journal of Engineering & Technology*, *7*(4.10), 974. doi: 10.14419/ijet.v7i4.10.26638

de la Hoz, J., Tirado, D., & Garzón, A. (2021). Habitual route choice and the adoption of new transportation infrastructure: A case study of bicycle lanes in bogota, colombia. *Transportation Research Part D: Transport and Environment*, *101*, 103252.

De Souza, C. P., de Miranda, S. R. C., & de Mello, O. C. L. (2019). Habitual route choice behavior: A study of factors influencing commuters' route preferences. *Transportation Research Procedia*, *40*, 877-884.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269-271. (Retrieved from `https://link.springer.com/article/10.1007/%2FBF01386390`)

Drake, J. H., Kheiri, A., Özcan, E., & Burke, E. K. (2019). Recent advances in selection hyper-heuristics. *European Journal of Operational Research*. doi: 10.1016/j.ejor.2019.07.073

Fitriansyah, A., Parwati, N. W., Wardhani, D. R., & Kustian, N. (2019, October). Dijkstra's algorithm to find shortest path of tourist destination in bali. In *Journal of physics: Conference series* (Vol. 1338, p. 012044). IOP Publishing.

Gross, J. L., & Yellen, J. (Eds.). (2003). *Handbook of graph theory* (1st ed.). CRC Press. Retrieved from `https://www.taylorfrancis.com/books/mono/10.1201/9780203490204/handbook-graph-theory-jonathan-gross-jay-yellen`

Gunawan, R. D., Napianto, R., Borman, R. I., & Hanifah, I. (2019). Implementation of dijkstra's algorithm in determining the shortest path (case study: Specialist doctor search in bandar lampung). *Int. J. Inf. Syst. Comput. Sci*, *3*(3), 98-106.

Gupta, A., Mishra, P., Pandey, C., Singh, U., Sahu, C., & Keshri, A. (2019). Descriptive statistics and normality tests for statistical data. *Annals of Cardiac Anaesthesia*, *22*(1), 67.

Hassan, M., Amerudin, S., & Yusof, Z. M. (2019). Assessment of the car-free day implementation on utm campus using a shortest path method. *Journal of Transport System Engineering*.

Horner, M. W., Autorino, A. L., & Brown, G. J. (2021). Route choice behavior under the influence of perceived risk and travel time: An empirical analysis of route preference and risk compensation. *Transportation Research Part C: Emerging Technologies*, *129*, 103074.

Huxley, K. (2020). Content analysis, quantitative. In P. Atkinson, S. Delamont, A. Cernat, J. W. Sakshaug, & R. A. Williams (Eds.), *The sage encyclopedia of research methods* (Online ed.). SAGE Publications, Inc. Retrieved from `https://orca.cardiff.ac.uk/id/eprint/134705/`

Liu, J., Haklay, E., & Goodchild, M. F. (2020). Urban route selection: A multi-modal perspective on the role of familiarity, perceived safety, and accessibility. *Journal of Transport Geography*, *87*, 100741.

Mirzaeinia, A., Shahmoradi, J., Roghanchi, P., & Hassanalian, M. (2019). Autonomous routing and power management of drones in gps-denied environments through dijkstra algorithm. In *Aiaa propulsion and energy 2019 forum.* doi: 10.2514/6.2019-4462

Mohammad Ata, K. I., Che Soh, A., Ishak, A. J., Jaafar, H., & Khairuddin, N. A. (2019). Smart indoor parking system based on dijkstra's algorithm.
(Retrieved from `https://core.ac.uk/download/pdf/229280683.pdf`)

Nilsson, N. J. (1982). *Principles of artificial intelligence.* Palo Alto, CA: Tioga Publishing Co.

Prasad, D., Manikanta, S. P., & Shubhaker, B. (2021). Design and implementation of self-driving vehicle using dijkstra's algorithm.
(Retrieved from `https://www.smec.ac.in/assets/images/research/ece/20-21/30.IJSRED-V4I2P174.pdf`)

Rahayuda, I. G. S., & Santiari, N. P. L. (2020). Dijkstra and bidirectional dijkstra on determining evacuation routes.
(Retrieved from `https://iopscience.iop.org/article/10.1088/1742-6596/1803/1/012018/meta`)

Rahman, M. S. (2017). *Basic graph theory.* Springer. Retrieved from `https://link.springer.com/book/10.1007/978-3-319-49475-3`

Siedlecki, S. L. (2020). Understanding descriptive research designs and methods. *Clinical Nurse Specialist*, *34*(1), 8–12.

Smith, J., & Johnson, A. (2023). Robot motion planning: Classical vs. meta-heuristic based methods.
(Retrieved from `https://www.sciencedirect.com/science/article/abs/pii/S1367578820300675`)

Strout, M. M., Carter, L., & Ferrante, J. (2003, June). Compile-time composition of run-time data and iteration reorderings. *ScienceDirect*. Retrieved from `https://dl.acm.org/doi/abs/10.1145/781131.781142` doi: 10.1145/781131.781142

Sun, Y.-J., Ding, X.-Q., & Jiang, L.-N. (2017). Heuristic pathfinding algorithm based on dijkstra.
(Retrieved from `https://www.atlantis-press.com/article/25884610.pdf`)

Wachs, M., McNally, M. G., & Schimek, J. (2019). Familiarity and risk perception in route choice: A meta-analysis and new empirical evidence. *Journal of Transport Geography*, *75*, 120-132.

Wahab, M. N. A., Nefti-Meziani, S., & Atyabi, A. (2020a). A comparative review on mobile robot path planning: Classical or meta-heuristic methods? *Annual Reviews in Control*, *50*, 233-252. doi: 10.1016/j.arcontrol.2020.10.001

Wahab, M. N. A., Nefti-Meziani, S., & Atyabi, A. (2020b). Robot motion planning: Classical vs. meta-heuristic based methods. *Journal of Robotics Research*. Retrieved from `https://www.sciencedirect.com/science/article/abs/pii/S1367578820300675` doi: https://doi.org/10.1016/j.robot.2020.05.007

Wang, H., Goodchild, M. F., & Kwan, M. (2022). The impact of perceived distance on urban route choice behavior. *Annals of the American Association of Geographers*, *112*(3), 720-734.

Wayahdi, M. R., Ginting, S. H. N., & Syahputra, D. (2021). Greedy, a-star, and dijkstra's algorithms in finding shortest path. *International Journal of Advances in Data and Information Systems*, *2*(1), 45–52.

Wilson, M., Stromswold, R., Wudarski, F., Hadfield, S., Tubman, N. M., & Rieffel, E. G. (2021). Optimizing quantum heuristics with meta-learning. *Quantum Machine Intelligence*, *3*, 1-14.

Xie, F., & Chen, L. (2016). Average shortest path optimization method of the course-scheduling system in universities based on geographic information system information. *Journal of Computational and Theoretical Nanoscience*, *13*(12), 10486–10491.

Yan, S.-R., Moria, H., Pourhedayat, S., Hashemian, M., Asaadi, S., Dizaji, H. S., & Jermsittiparsert, K. (2020, October). A critique of the effectiveness concept for heat exchangers; theoretical-experimental study. *Applied Thermal Engineering*. Retrieved from `https://www.sciencedirect.com/science/article/abs/pii/S0017931020330969` (Advance online publication) doi: https://doi.org/10.1016/j.applthermaleng.2020.115905

Zhou, P., Xie, Z., Zhou, W., & Tan, Z. (2023). A heuristic integrated scheduling algorithm based on improved dijkstra algorithm. *Electronics*, *12*(20), 4189. doi: 10.3390/electronics12204189