Graph in Web Display Optimization: Leveraging Fitt's Law for Efficiency

Maulvi Ziadinda Maulana - 13522122[1]
Department of Informatics Engineering
School of Electrical Engineering and Informatics
Bandung Institute of Technology, Jl. Ganesha 10, Bandung 40132, Indonesia
[1]13522122@std.stei.itb.ac.id

***Abstract*—This research focuses on the integration of graph theory and Fitts' Law in web design development to enhance user experience. Graph theory visualizes relationships among web elements, while Fitts' Law analyzes the access time. The program evaluates UI changes, demonstrating its effectiveness in assessing design modifications. The results underscore the program's ability to calculate the Index of Difficulty and its potential for complex web structures. It is important to note that lower ID values don't universally mean better experiences, depending on the nature of UX itself.**

***Keywords*—Fitts' Law, Graph, UX**

I. Introduction

User experience used to be an extra part of a website but nowadays it is a vital component of any website as we live in a continuously changing digital era. Most online users today want interfaces that are not only attractive but also fast and easy to navigate.

User-friendliness has been added to visual attraction regarding creating effective web interface because of changes seen nowadays where one needs to be able to access the desired information on the site comfortably, navigate easily, or engage quickly with functions present. In conclusion, it is important to understand how a variety of elements interact on a web page and how people connect with them.

The point here should be noted that any element in a design – visual factors, navigation, or functionality could affect the entire user experience. Therefore, it was concluded that one of the ways to improve the website design development is using graph theory and Fits Law.

Deep inferences about the structure and connectivity of elements on a web page are possible using graph theory that requires only a visualized depiction of relations at discretion. However, from another perspective, Fitts' Law focuses on the relations between the size of an element, its distance, and the time the accessibility might demand during users' interaction.

This is not merely a theory outlined but an exploration of the concepts within contextual web-based user involvement and indulgence through a responsive, sensitively designed interface. Therefore, we endeavor to pinpoint practical design principles that our web designers can implement using thorough analysis.

Consequently, it is expected that this study will have a real impact on web design developing visually attractive yet efficient, easily accessible, and user-friendly.

II. Fundamental Theory

A. Graph

1. Definition

A graph is used to represent discrete objects and the relationships between these objects. According to the definition, a graph G = (V, E), where V is a non-empty set of vertices $\{v_1, v_2, ..., v_n\}$, and E is a set of edges connecting pairs of vertices $\{e_1, e_2, ..., e_n\}$. [1]

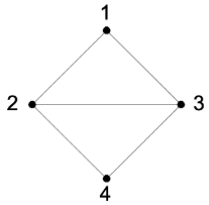
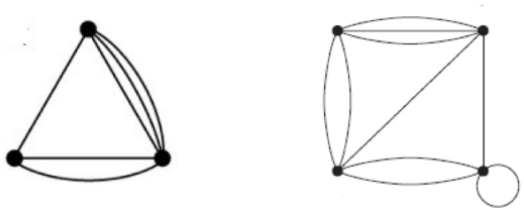


Figure 1. Simple Graph (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf)

Graphs are so named because they can be represented graphically, and it is this graphical representation that helps to understand many of their properties. Each vertex is indicated by a point, and each edge by a line joining the points that represent its ends. [2]

2. Types

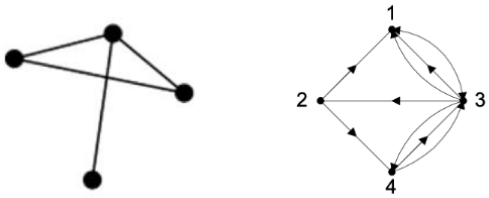
First, based on the presence or absence of loops or multiple edges in a graph, it is classified into two types: Simple Graph and Non-simple Graph. A Simple Graph is a graph that does not contain loops or multiple edges as shown in figure 1, while a non-simple graph is a graph that has multiple edges or loops. If a graph has multiple edges, it's called a multi-graph. If a graph has loops, it's called a pseudo-graph. [1]



a) b)

Figure 2. (a) Multi-graph and (b) Pseudo-Graph (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf)

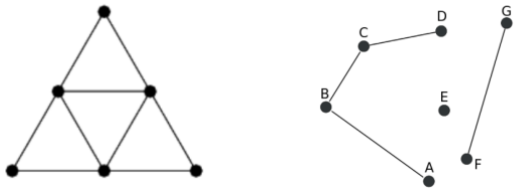
Based on the directional orientation of the edges in a graph, the graph is classified into two types: Undirected Graph and Directed Graph. An Undirected Graph is a graph whose edges do not have a directional orientation, while a Directed Graph is a graph in which each edge is given a specific direction.



a) b)

Figure 3. (a) Undirected-graph and (b) Directed-Graph (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf)

A graph is connected if, for every partition of its vertex set into two nonempty sets X and Y, there is an edge with one end in X and one end in Y, otherwise the graph is disconnected. In other words, a graph is disconnected if its vertex set can be partitioned into two nonempty subsets X and Y so that no edge has one end in X and one end in Y.



a) b)

Figure 4. (a) Connected-graph and (b) Disconnected-Graph (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf)

3. Terminologies

The paper is underpinned by several graph terminologies. Firstly, two nodes are considered adjacent if they are directly connected. Additionally, $e(v_j, v_k)$ is said to be incident with node v_j or incident with node v_k.

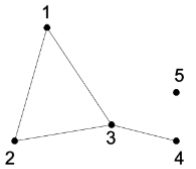


Figure 5. Example of Neighboring and Incidence Terminology (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf)

In this figure, observe node 1. Node 1 is adjacent to nodes 2 and 3 but is not adjacent to node 4. Furthermore, edge (2,3) is incident with nodes 2 and 3 but is not incident with node 1.

The next terminology to discuss is paths. A path of length n from the initial node v_0 to the destination node v_n in graph G is a sequence alternating between nodes and edges in the form v_0, e_1, v_1, e_2, v_2, ..., v_{n-1}, e_n, v_n, such that $e_1 = (v_0, v_1)$, $e_2 = (v_1, v_2)$, ..., $e_n = (v_{n-1}, v_n)$ are the edges of the graph G. In Figure 4, the path 1, 2, 3, 4 is a path with overlapping edges (1,2), (2,3), and (3,4). The length of this path is 3 because there are 3 overlapping edges. [1]

The last important terminology of the graph is weighted graph. A weighted graph is a graph in which each edge is assigned a value (weight). [1]

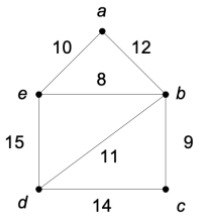


Figure 6. Weighted Graph (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf)

B. Dijkstra's Shortest Path Algorithm

The shortest path refers to the most direct or economical route between two points in a network, typically measured by the sum of weights or costs associated with traversing the edges or links between those points. In graph theory and network analysis, the "path" is a sequence of edges that connect nodes (or vertices), and the "shortest" path is the one with the minimum cumulative cost or weight.

One algorithm for finding the shortest path from a starting node to a target node in a weighted graph is Dijkstra's algorithm. The algorithm creates a tree of shortest paths from the starting vertex, the source, to all other points in the graph. [5] The algorithm maintains a set of vertices whose shortest distance from the source node is known. It iteratively selects the vertex with the smallest known distance, explores its neighbors, and updates their distances if a shorter path is found. This process continues until the algorithm has determined the shortest path to all vertices from the source. The algorithm proceeds as follows:

1. While *Q* is not empty, pop the node *v*, which is not already in *S*, from *Q* with the smallest distance (*v*). In the first run, source node *s* will be chosen because distance (*s*) was initialized to 0. In the next run, the next node with the smallest distance value is chosen.
2. Add node *v* to *S*, to indicate that *v* has been visited.
3. Update dist values of adjacent nodes of the current node *v* as follows: for each new adjacent node *u*, if dist (*v*) + *weight*(*u*,*v*) < dist (*u*), there is a new minimal distance found for *u*, so update dist (*u*) to the new minimal distance value; otherwise, no updates are made to dist(*u*).

The algorithm has visited all nodes in the graph and found the smallest distance to each node. Dist now contains the shortest path tree from source *S*. [5]

C. Fitts's Law

Fitts's law, a one-dimensional model of human movement, is commonly applied to two-dimensional target acquisition tasks on interactive computing systems. [3] Fitts's law gives us the relationship between the time it takes a pointer (such as a mouse cursor, a human finger, or a hand) to move to a particular target (e.g., a physical or digital button) to interact with it in some way (e.g., by clicking or tapping it, grasping it, etc.). [4]

According to Fitts' law, the time (MT) to move to and select a target of width W which lies at distance (or amplitude) A is

$$MT = a + b \log_2(\frac{2A}{W}) \quad (1)$$

where a and b are constants determined through linear regression and vary depending on the type of pointer (e.g., mouse, finger, etc.). W corresponds to "accuracy" – the required region where an action terminates. The quantity

$$ID = \log_2(\frac{2A}{W}) \quad (2)$$

is sometimes called an index of difficulty and is measured in bits.

Technically, Fitts's law equation uses the width of the target in the direction of the movement, but for most rectangular targets that are common in user interfaces, it can be replaced with the smallest of the target dimensions, whether it's height or width (as shown by Scott MacKenzie and Bill Buxton in 1992). [4]

The main idea of this equation is captured by the following two statements. The bigger the distance to the target, the longer it will take for the pointer to move to it. In other words, closer targets are faster to acquire. The larger the target, the shorter the movement time to it. In other words, bigger targets are better. [4]

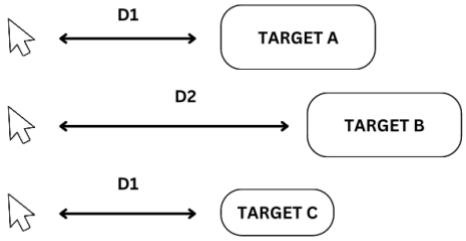


Figure 7. Fitts' Law Illustrations (Source: Author's Documentation)

Fitts's law says that the time to reach Target A is shorter than the time to reach any of the other targets. Although Targets A and B have the same size, the distance from the cursor to A (D1) is shorter than the distance to B (D2), so movement to A will be faster. Target C is placed at the same distance (D1) from the cursor as Target A, but it's smaller, so it will take longer to move the cursor to it than to A. [4]

This paper mainly focuses on using the Index of Difficulty (ID) from Fitts' law, represented by equation (2). This decision comes from the fact that the constants 'a' and 'b' in Fitts's law depend on the type of pointer used, like a mouse or a finger. Because these constants can vary based on the pointer, the decision is taken to simplify things by concentrating on the Index of Difficulty. This not only makes the analysis more straightforward but also aligns with practical considerations in designing user interfaces, especially when dealing with common rectangular targets.

III. METHODOLOGY

A. Graph Representation of a Webpage

A graph representation of a webpage describes different parts of the webpage using graphs. This method presents a pictorial and structural depiction of the relation of different aspects on the web page. Two primary examples illustrate this concept: the layout of this interface menu button, and the user interaction flow.

This includes components such as menu buttons, clicking on which leads to the reveal of sub-components or child elements. The hierarchical relations may easily be represented as a graph by linking such menu points and connected nodes. For instance, a menu button:

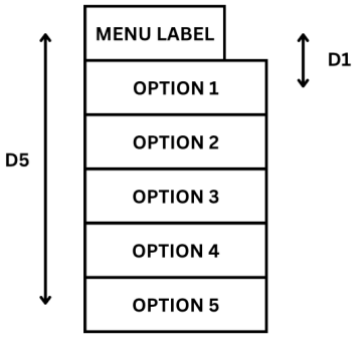


Figure 8. Menu Example (Source: Author's Documentation)

can be expressed as a weighted graph as follows:

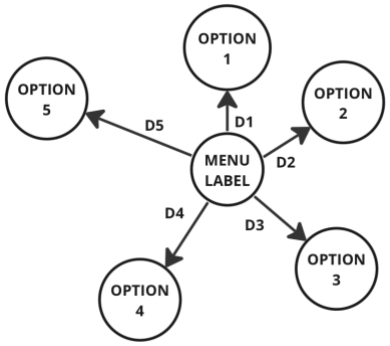


Figure 9. Graph Representation of Menu (Source: Author's Documentation)

This way, the hierarchy of the menus and their different components are presented visually as graphs. Thus, it is possible to make conclusions about the ease to navigate and to organize the content on the page while analyzing it by Fitts' law.

The flow of the users' interaction with the web page may also be depicted using graphs. For instance, take a scenario where users want to log out. First, if a person wants to log out, he or she clicks on the profile button. Next, they click on "logout." The user interaction flow can be drawn as a graph with nodes representing clickable elements (buttons) and edges denoting how far one element is from another. For example, the user flow:

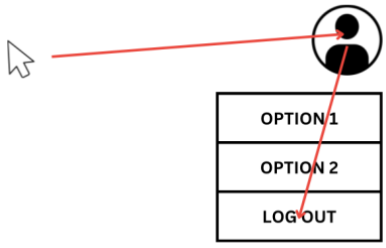


Figure 10. The user flow when a user wants to log out (Source: Author's Documentation)

can be expressed as a weighted graph as follows:

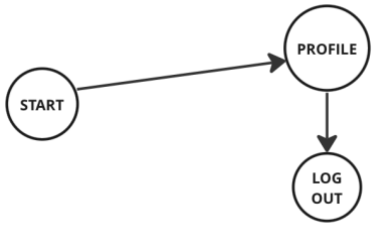


Figure 11. Graph Representation of the User Flow (Source: Author's Documentation)

In this case, the plot pictures the path traveled by the user beginning with a 'start' label, moving on to the profile button, and ending at log out. It is also referred to as the "Start" node where either the cursor or users touch down on the site for the first instance. This usually takes place on the right thumb side for mobile devices and on the middle section between the top and of the screen for desktop users. In this, there is mention of how the entrance point of the user has been acknowledged, which prepares the way for following the route that the user may have taken while performing various options on the interface.

It has a high degree of flexibility with adjustability for displaying many website structures. The diagram can be used for illustration purposes depicting anything from simple items such as the menu buttons, to more elaborate components and user flows.

The spaces between various components of the graph should be calculated by hand. For instance, there are some extensions or apps that allow users to measure the distance between two points on the screen, an example is Measure Rope on macOS.

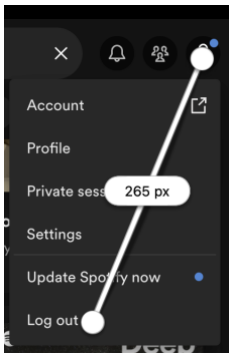


Figure 12. Measuring Distance between Components(Source: Author's Documentation)

B. *Fitt's Law Implementation*

The Author chose to implement Fitts' Law using Python because it is easy to use. There is also a library called NetworkX that is available for handling graphs which will make the process easier when implementing Fitts' Law. Moreover, this library is an easy way of doing complicated tasks like making graphs or using shortest-path algorithms like Dijkstra.

To implement Fitts' Law on a graph in Python, the following steps will be taken:

1. Firstly, a graph is constructed to represent the components to be evaluated for their Fitts' Law ID values. Each edge in the graph is assigned a weight, representing both the width and distance between the connected components.

```
import networkx as nx


G = nx.Graph()
G.add_nodes_from(['A','B','C'])
G.add_edge('A','B', weight={'width': 3, 'distance': 5})
G.add_edge('B','C', weight={'width': 2, 'distance': 7})
```

2. The user is prompted to specify the start and end nodes for which the ID will be calculated. If a path from node A to C requires passing through B first, the ID for AC is calculated as the sum of ID for AB and BC.

```
def calculate_id(graph, path):
    id_value = math.log2(2*distance / width)
    return id_value
```

3. To manage complex graphs where nodes can be accessed through multiple paths, Dijkstra's algorithm is used to find the shortest path. The final ID is determined as the total ID calculated in step 2, considering the components along the resulting Dijkstra's path.

```
shortest_path = nx.shortest_path(G, source='A', target='C', weight='weight')
```

These steps ensure the implementation of Fitts' Law on a graph, considering user-defined start and end nodes and handling the complexities of graphs with multiple access paths. Below is the final program.

```
import networkx as nx
import math

class WebGraph:
    def __init__(self):
        self.graph = nx.Graph()

    def add_node(self, node, width):
        self.graph.add_node(node, width=width)

    def add_edge(self, node1, node2, weight):
```

```
        self.graph.add_edge(node1, node2,
weight=weight)

    def id_difficulty(self, src, dest):
        shortest_path = nx.shortest_path
(self.graph, source=src, target=dest,
weight='weight')

        total_id = 0

        for i in range(len(shortest_path) -
1):
            current_node, next_node =
shortest_path[i], shortest_path[i + 1]
            distance =
nx.shortest_path_length(self.graph,
source=current_node, target=next_node,
weight='weight')
            width =
self.graph.nodes[next_node]['width']

            id_value = math.log2(2*distance /
width)
            total_id += id_value

        return total_id
```

To test whether the program is working or not, a sample web graph has been created that includes five nodes labeled A, B, C, D, and Z with the corresponding width value of 100 for each one. Edges illustrate the distances between them that are indicated within the respective weight values. Thus, this edge goes from A to B, and weight 300 implies that there is 300 pixels spacing for that node pair. Similarly, edges connect B to C (weight: 200), C to D scale (weight: 100), and a D to Z scale (weight: 50).

```
web_graph = WebGraph()

web_graph.add_node("A", width=100)
web_graph.add_node("B", width=100)
web_graph.add_node("C", width=100)
web_graph.add_node("D", width=100)
web_graph.add_node("Z", width=100)

web_graph.add_edge("A", "B", weight=300)
web_graph.add_edge("B", "C", weight=200)
web_graph.add_edge("C", "D", weight=100)
web_graph.add_edge("D", "Z", weight=50)
src_node, dest_node = "A", "Z"
total_id = web_graph.id_difficulty(src_node,
dest_node)
```

And below is the result of the test:

```
maulvizm@Maulvis-MacBook-Air Graph_Fitts-Law % /opt/homebrew
/bin/python3 /Users/maulvizm/Documents/GitHub/Graph_Fitts-La
w/main.py
Total ID from node A to node Z is 5.584962500721156
```

Figure 12. Program result (Source: Author's Documentation)

C. Program Testing

In the first program testing phase, the program is going to be used to compare the Index of Difficulty (ID) between two types of menu buttons: linear menu and pie menu. A linear menu has option buttons that are vertically organized. The second type called the pie menu, utilizes a circular arrangement in which elements spiral outward from the center point. This explains about whether the two menu architectures are better or worse concerning Fitts' Law.

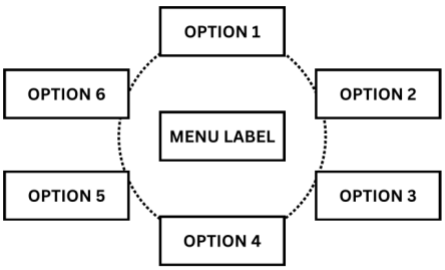


Figure 13. Pie Menu Example (Source: Author's Documentation)

For the linear menu, each menu option is defined as a square with dimensions of 100 px x 100 px. These options are arranged in a straightforward, horizontal alignment, positioned closely to one another for easy access.

On the other hand, in the pie menu, where each option is also considered a square with dimensions of 100 px x 100 px, a circular layout is employed. In this arrangement, each menu option is deliberately positioned at an equal distance of 100 px from the central menu label. Below is the result of the test that is computed by the program.

Table 1. Linear Menu Index of Difficulty

Option	Distance from menu label (px)	ID
1	50	1,0
2	150	1,58
3	250	2,32
4	350	2,80
5	450	3,16
6	550	3,45
Average		2,39

Table 2. Pie menu Index of Difficulty

Option	Distance from menu label (px)	ID
1	100	1,0
2	100	1,0
3	100	1,0
4	100	1,0
5	100	1,0
6	100	1,0
Average		1,0

Based on the result, in the pie menu, the ID remains consistently low at 1.0 for each option, resulting in an average ID of 1.0. On the other hand, the linear menu shows an increase in ID as the distance from the menu label expands. The calculated average ID for the linear menu is 2.39. This comparison underscores the efficiency of a pie menu layout, where options maintain a constant level of difficulty.

The second program testing phase, the program designed for testing UI changes on a website aims to evaluate whether these alterations result in improvements in terms of user interaction and experience. Specifically considering Fitts's Law principles, the evaluation focuses on how well the new UI design accommodates ergonomic principles, particularly regarding the size and distance of interactive elements.

Firstly, the program measures and compares the Index of Difficulty (ID) of interactive elements before and after the UI changes. The ID is calculated based on Fitts's Law formula, incorporating the size of elements and the distance between them. By comparing IDs, the program can determine whether the UI changes have led to an increase or decrease in usability. Below is an example of UI changes.

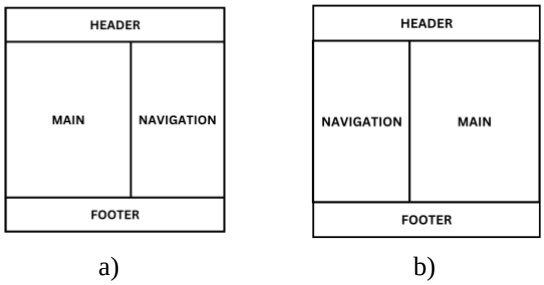


Figure 14. (a) Before and (b) after UI change (Source: Author's Documentation)

The navigation section which has a size of 400x800 px will guide users to a new page that has a Call-to-Action (CTA) button located at the top left corner of the screen, with dimensions set at 100 px x 100 px. By measuring the Index of Difficulty (ID) using Fitts' Law, the program will provide insights, into whether the changes are good or bad. Below is the graph representation of the changes.

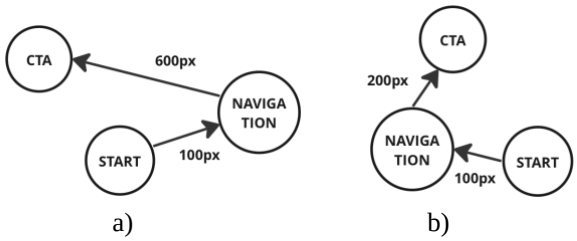


Figure 15. (a) Before and (b) after UI change graph representation (Source: Author's Documentation)

Here is the ID for each condition that is computed by the program:

Table 3. Index of Difficulty for each condition

Condition	ID
Before	1,99
After	0,41

In the final testing phase, the program will be used to evaluate the Index of Difficulty (ID) within a more complex user flow, using *tokopedia.com* as a test case. In this phase, the graph is representing the process a user takes when attempting to make a purchase. Below is the picture of the webpage and its corresponding graph representation.

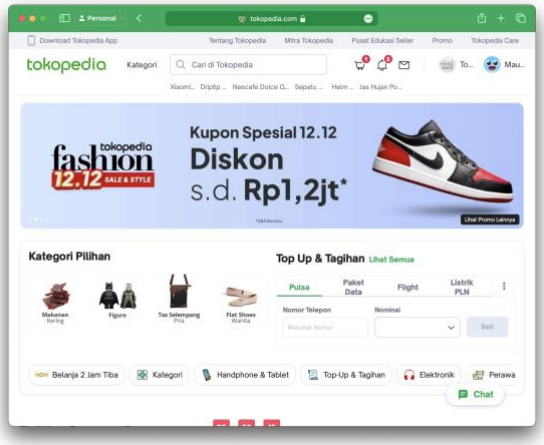


Figure 16. *Tokopedia.com* Web Interface (Source: *Tokopedia.com*)

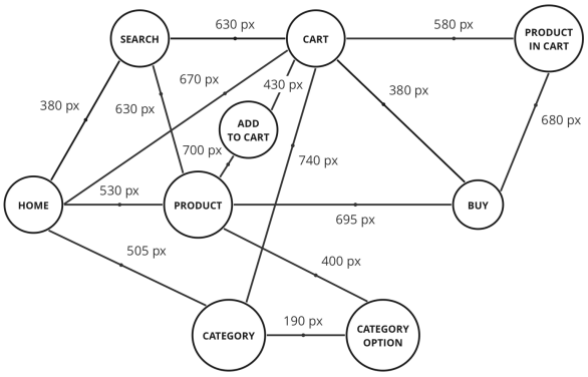


Figure 17. Graph representation of purchasing a product process in *tokopedia.com* (Source: Author's Documentation)

This graph shows distinct processes a buyer might navigate, exemplified by varied paths, such as those marked in red or blue in the figure below.

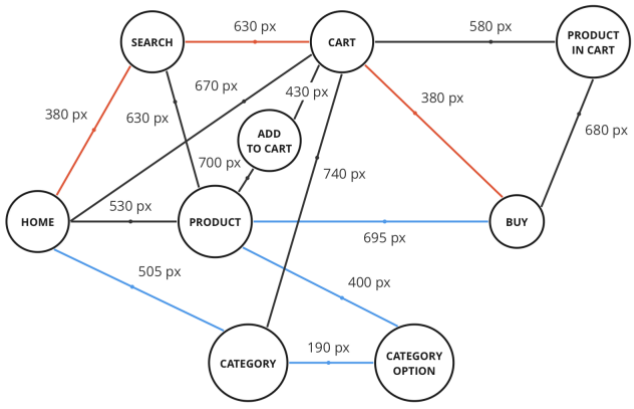


Figure 18. Different purchase a product (Source: Author's Documentation)

The program will examine and find shortest path using Dijkstra's Algorithm for this user flow complexities since they are many in number. Then, the program will calculate their ID based on the shortest path. Below is the preparation to run the program. All measurements listed (width and distance) have been adjusted directly from the *tokopedia.com*.

```
web_graph = WebGraph()

web_graph.add_node("HOME", width=100)
web_graph.add_node("SEARCH", width=550)
web_graph.add_node("CART", width=25)
web_graph.add_node("PRODUCT", width=175)
web_graph.add_node("ADD TO CART", width=215)
web_graph.add_node("CATEGORY", width=64)
web_graph.add_node("CATEGORY OPTION",width =
180)
web_graph.add_node("BUY", width = 215)
web_graph.add_node("PRODUCT IN CART", width =
670)

web_graph.add_edge("HOME","SEARCH", weight =
380)
web_graph.add_edge("HOME", "CART", weight =
670)
web_graph.add_edge("HOME", "PRODUCT", weight
= 530)
web_graph.add_edge("HOME", "CATEGORY", weight
= 505)
web_graph.add_edge("PRODUCT", "ADD TO CART",
weight = 700)
web_graph.add_edge("PRODUCT", "BUY", weight =
695)
web_graph.add_edge("PRODUCT",       "CATEGORY
OPTION", weight = 400)
web_graph.add_edge("PRODUCT",       "SEARCH",
weight=630)
web_graph.add_edge("SEARCH",          "CART",
weight=630)
web_graph.add_edge("ADD  TO  CART",  "CART",
weight=430)
web_graph.add_edge("PRODUCT IN CART", "CART",
weight=580)
web_graph.add_edge("PRODUCT IN CART", "BUY",
weight=680)
web_graph.add_edge("BUY", "CART", weight=380)
web_graph.add_edge("CATEGORY",        "CART",
weight=740)
web_graph.add_edge("CATEGORY",      "CATEGORY
OPTION", weight=190)
```

Here is the result from the program:

```
maulvizm@Maulvis-MacBook-Air Graph_Fitts-Law % /opt/homebrew
/bin/python3 /Users/maulvizm/Documents/GitHub/Graph_Fitts-La
w/main.py
Shortest path from HOME to BUY:  ['HOME', 'CART', 'BUY']
The ID is 7.565823854311898
```

Figure 19. Different purchase a product (Source: Author's Documentation)

IV. Result & Analysis

Based on the first phase of testing, the program is giving a great result, which the pie menu will have a lower ID. But while a lower ID, suggests uniform ease of interaction, it's essential to recognize that a lower difficulty level may not always defined as an optimal design. Certain functionalities, like a logout or exit button, may intentionally require a higher ID, making it harder for users to access. For example, implementing a higher ID for the logout button is a good design choice to prevent accidental logouts. This perspective emphasizes that the ideal user interface design depends on the specific goals and user experience considerations of the application, where intentional increases in difficulty may serve specific functions. In the second phase of testing, the program also gives a correct result. The result of the test itself is positive, indicating that the implemented changes have proven beneficial because the CTA button is easier to reach. In the final testing phase, the program demonstrated excellent proficiency in assessing user flow ID within a sufficiently complex graph.

V. Conclusion

In conclusion, the developed program effectively computes the Index of Difficulty (ID) based on Fitts's Law for a given path in a graph. The program demonstrates its functionality in calculating the ID considering the effective distance and width of targets. While the program currently operates well for relatively simple graphs, its design suggests its capability to handle more complex structures that sadly has not been explored in this paper. Additionally, it is important to recognize that lower ID values do not always same as better experience. The interpretation of ID's effectiveness depends on specific user needs and the context of the interaction.

VII. Acknowledgment

Praise and gratitude are only to Allah Swt., for it is through His blessings and abundant grace that the author has been able to complete this paper successfully. Special thanks are also extended to Mr. Dr. Ir. Rinaldi Munir, M.T., and Mr. Monterico Adrian, S.T., M.T., as the lecturer for the IF2120 Discrete Mathematics course, Class K-03, for the knowledge imparted to the author, enabling the successful completion of this paper. Additionally, heartfelt thanks are conveyed to the parents for their constant support and motivation provided to the author.

References

[1] R. Munir. *Matematika Diskrit edisi ketiga*. Bandung, 2009.
[2] Bondy, J. A., & Murty, U. S. R. (2008). *Graph Theory*. Springer.
[3] Mackenzie, I. S., & Buxton, W. (Year). Extending Fitts' Law to Two-Dimensional Tasks. *Conference on Human Factors in Computing Systems*, Volume (92), 219-226.
[4] Budiu, R. 2022. *Fitts's Law and Its Applications in UX*. Nielsen Norman Group. Retrieved from https://www.nngroup.com/articles/fitts-law/ on 8 December 2023.
[5] Abiy, T., Pang, H., & Williams, C. *Dijkstra's Shortest Path Algorithm*. Brilliant.org. Retrieved from https://brilliant.org/wiki/dijkstras-short-path-finder/ on 9 December 2023.

Statement

I hereby declare that this paper I have written is my work, not a translation or reproduction of someone else's paper, and it is not plagiarized.

Bandung, December 3rd, 2023

Maulvi Ziadinda Maulana
13522122