Chromationary: Finding A Simple Graph-Based Approach to Generate Personalized Color Palettes

Jennifer Khang - 13524110
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: jenniferkhang07@gmail.com , 13524110@std.stei.itb.ac.id

***Abstract*— Color is a visual experience through the human eyes, often used for many purposes in various fields including art and design. This paper explains the fundamentals to understanding how a color palette generator algorithm may work with a graph-based approach, with nodes and weighted edges. Each node will represent a different color based on the sample used, meanwhile the weight of each edge will represent how the colors correspond to each other—based on principles such as contrast, similarity, or distance in a perceptual color space. The graph is then updated with inputs from the user's interactions, which influence the graph's weight through algorithms that iteratively select the most optimal combinations. This paper will also show a simple program, demonstrating how user preferences and graph structures can be combined to generate personalized color palettes.**

Keywords*— *Color Theory, Perceptual Color Space, Palette Generator, Weighted Graph, Louvain's Algorithm

I. Introduction

Throughout history, humans have sought to understand colors—initially through symbolism and later through scientific inquiry. In the earliest day, there was an attempt from Newton to systematically organize colors, arranging the visible spectrum into a circle, known as Newton Color Circle. This research then becomes the ground for many diverse findings later—although there are some debates regarding Goethe's theorem. One primary example that should be highlighted is the quantization of colors, in a mathematical way—syntax we nowadays known as color models and color spaces. The most known ones are the RGB model. With this, humans achieved a way to access colors through numeration.

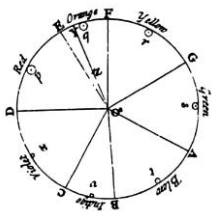


Fig. 1. Newton's Color Circle. *(Taken from programmingdesignsystems.com)*

Color has always played a pivotal role within the status quo —either for design, self-expression, and many more. For instance, an artist's tools to express their art are through the plethora of pigments from the strokes of their brush. Each color that they pick is unique and is reflected on every work they have done. They are a trademark for each piece of creation. However, choosing a color that harmonizes or matches with an illustration would be a challenging task for artists even for advanced professionals, as it requires many considerations. These considerations can range from hue, mood, and so on. Thus, the complexity of choice amplified the need for a tool to help artists, such as random color generators.

While traditional color palette generation tools often rely on predefined rules based on the static template of color theory, they may fall short in capturing an individual preference. It strikes monotony, causing many artists to ditch such tools. In an era where personalization is increasingly valued, developing a method or tool that reflects personal taste may be needed for artists. Although it sounds promising, there is barely anyone who discusses it due to its intricacy. It's even harder to create a precise algorithm to randomize colors due to the lack of distinction for computers to interpret the numerations of current coloring system. Therefore, this paper will only analyze how to implement simple graph theory for the tool (of what could theoretically work) and would not integrate more than that.

II. Theoritical Framework

A. Graph Theory

Graph is commonly used to represent multiple discrete objects and their relations to each other. A graph can be described as ordered pairs, which consist of vertices or nodes and edges. A graph G can be notated as $G = (V, E)$, where V is defined as a non-empty set of vertices $\{v_1, v_2, \dots, v_n\}$ (could also be called nodes) and E is a set of edges $\{e_1, e_2, \dots, e_n\}$ that connects a pair of vertices. The relation between an edge and a pair of vertices can be notated as $e = (v_i, v_j)$, where e is an edge connecting two vertices with the index i and index j.

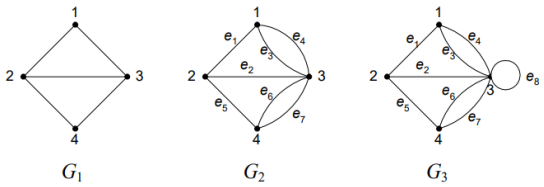


Fig. 2. Several types of graphs, where G1 is a graph with no multiple edges and loop, G2 is a graph with multiple edges, and G3 is a graph with multiple edges and loop. *(Taken from Rinaldi Munir's Powerpoint slides about graph)*

Graph can be classified into two types based on the absence of multiple edges (pair of edges) and loop. First is the simple graph, which has no multiple edges and no loop. Then, there is an unsimple graph, which is the opposite of what a simple graph is. It has at least either one multiple edge or loop (could be both too). This graph then can be divided further into 2 groups, which is multi-graph and pseudo-graph. A multi-graph has multiple edges while pseudo-graph has one or more loops on the vertexes. The following is more explanation regarding graphs.

1) *Weighted Graph*

A weighted graph is a graph where every edge has an assigned number. The assigned numbers hold some value which may represent cost, distance, and many other relative measuring units. In this paper, it will likely to be related to how each color might harmonize.

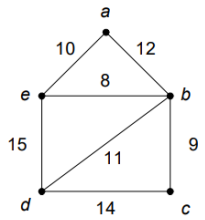


Fig. 3. Illustration of a weight graph. *(Taken from Rinaldi Munir's Powerpoint slides about graph)*

2) *Terminologies for Graph*

a) Adjacenct

Two vertexes are defined to be adjacent if both vertexes are directly connected through an edge. The following graph has 4 vertexes, whereas vertex 1 is adjacent to vertex 2 and 3 while vertex 1 is not adjacent to vertex 4.

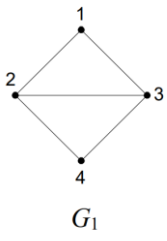


Fig. 4. Illustration for adjacent nodes. *(Taken from Rinaldi Munir's Powerpoint slides about graph)*

b) Incidency

An edge and a vertex that is connected directly are also called incidents. For any edge $e = (v_i, v_j)$, there are vertex I and vertex j which have an incidence relation with the edge e. On Fig. 4., graph G1, has shown that edge (2, 3) is incident to vertex 2 and vertex 3, edge (2, 4) is incident to vertex 2 and vertex 4. However, edge (1, 2) is not incident to vertex 4.

c) Empty/Null Graph

A graph with only vertices and has an empty set of edges is called a null org empty graph, which usually is denoted as N_n (where n is the number of vertices).

d) Degree

The degree of a vertex is a number that represents the amount of edges incident to it, usually notated as d(v). The figure below shows that vertex 1 has the same amount of degree as vertex 4, which both have 2 degrees. On the other hand, vertex 2 and vertex 3 each have 3 degrees.

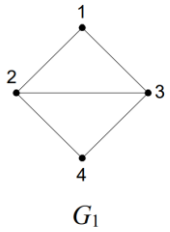


Fig. 5. Graph Illustration. *(Taken from Rinaldi Munir's Powerpoint slides about graph)*

e) Path

A sequence of edges which joins a sequence of vertices is called a path. The length of a path is defined to be the number of edges a path consists of.

f) Cycle/Circuit

A path that will start and end on the same node is called a cycle. The length of a circuit is the number of edges a cycle has.

g) Connected

Two vertexes, for example v_1 and v_2, are said to be connected if there is a path from v_1 to v_2. A graph G is called a connected graph if, for every pair of nodes (v_1, v_2) in the vertex set V, there exists a path from v_i to v_j. Otherwise, G is called a disconnected graph.

h) Subgraph

Say, there is a graph G that has a set of vertexes and edges (V, E). Then, there is graph $G_1 = (V_1, E_1)$. G1 is a subgraph of G if $V_1 \subseteq$ V (the vertices of G_1 are a subset of G's vertices) and $E_1 \subseteq$ E (the edges of G_1 are a subset of G's edges).

B. *Color Theory*

Throughout history, many scientists, art theorists and practitioners have tried to establish color harmony. A set of colors creates harmony when their combination produces and aesthetically pleasing effect when seen together. It is well known nowadays that harmonious colors can be represented a point in a color system that is distributed evenly. It may even be a ground rule to learn perfect fit combination. One of the most popular ones includes:

- Complementary harmony, which is colors lying opposite each other on the color wheel,
- Analogous harmony, which is colors with similar hues, near to each other on the color wheel,
- Triadic harmony, which is three colors that have separate hues, about 120 degrees on the color wheel,
- Split-complementary harmony includes three colors, with two being on either side of the complement of the third on the color wheel,
- Tetradic harmony, which is double complementary scheme, two complementary pairs which are opposite each other on the color wheel.

There were many arguments regarding color theory, as many believed that it is a subjective matter, this paper will still use color harmony as a ground set up for the algorithm that will be created.

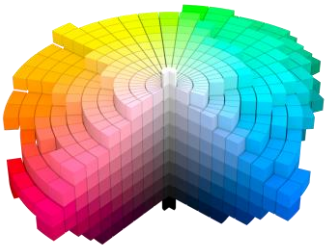


Fig. 6. Three-dimensional representation of the 1943 Munsell renotations, with portion cut away, marking as one of the earliest way to quantifies color. *(Taken from https://chromatone.center/theory/color/models/perceptual/)*

There is also a thing called perceptual color space. Perceptual color space is a way to discern or organize color in a way that reflects how humans perceive. Its main goal is to aim a uniform color difference towards changes. Color difference or distance here is a measure of how different colors will be perceived. It means changes in value correspond to roughly equal changes in perceived color. Perceptually similar color pairs usually have smaller color distances.

A color model is a system that uses numerical values to define colors. There are many popular models, such as RGB, CMYK, HSV, CIELAB, and many more. For the sake of this experiment, only RGB models and LAB models will be used for conveniency.

1) *RGB Model*

This model is one of the most widely known methods for color representation as numbers in computer graphics. It is based on three primary colors that we all know, which are red, green, and blue. The combination of these colors in many intensities forms the cube-shaped RGB color space, meaning that all colors can be created through their linear combination.

2) *LAB Model*

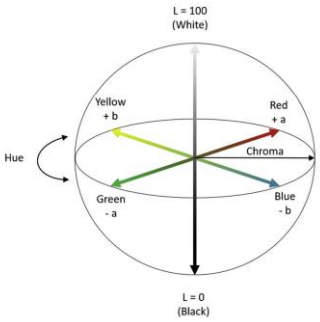


Fig. 7. LAB model color space diagram. *(Taken from http://dx.doi.org/10.1016/j.jid.2019.11.003)*

The LAB model, or known as CIE LAB, portrays color in three dimensions, which is L* that refers to lightness, a* that refers to the green-red component, and b* that refers to the blue-yellow component (see Fig. 5). The most common way to calculate the distance of color is by using CIELAB color difference, notated as ΔE^*_{ab}, as it is designed to be perceptually uniform. The calculation is done as the Euclidean distance between points representing the color stimuli in space. The following is an equation to determine the perceptive distance for the CIE 1976.

$$\Delta E^*_{ab} = \sqrt{(L^*_2 - L^*_1)^2 + (a^*_2 - a^*_1)^2 + (b_2 - b^*_1)^2} \quad (1)$$

The next equation of the formula for CIEDE2000 color difference is given by:

$$\Delta E_{00} = \sqrt{\left(\frac{\Delta L'}{k_L S_L}\right)^2 + \left(\frac{\Delta C'}{k_C S_C}\right)^2 + \left(\frac{\Delta H'}{k_H S_H}\right)^2 + R_T \left(\frac{\Delta H'}{k_H S_H}\right)\left(\frac{\Delta C'}{k_C S_C}\right)} \quad (2)$$

where the terms are defined as follows:

$$\Delta C' = (C'_2 - C'_1) \quad (3)$$

$$\Delta L' = (L^*_2 - L^*_1) \quad (4)$$

$$\Delta H' = 2\sqrt{C'_2 C'_1}\sin\left(\frac{\Delta h'}{2}\right) \quad (5)$$

$$R_T = -2\sqrt{\frac{C_7}{C_7 + 25^7}}\sin(2\Delta\theta) \quad (6)$$

$$S_L = 1 + \frac{0.015(L^*_1 - 50)^2}{\sqrt{20 + (L^*_1 - 50)^2}} \quad (7)$$

$$S_C = 1 + 0.045C_1' \quad (8)$$

$$S_H = 1 + 0.015C_1'(1 - 0.17\cos(\Delta h' - 30°) + 0.24\cos(2\Delta h') + 0.32\cos(3\Delta h' + 6°) - 0.20\cos(4\Delta h' - 63°)) \quad (9)$$

C. Louvain's Algorithm for Community Detection

Louvain's algorithm was proposed in a 2008 paper by Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. It took the name from Louvain-la-Neuve, a Belgium city where Lefebvre initially developed the algorithm during his Master thesis at Université catholique de Louvain in March 2007.

Modularity is one of the most suitable metrics for a cost function to decide on convergence criterion—it's a measure to evaluate how well a network can be divided. Louvain's algorithm is based on optimization of this Modularity. Partitions (quality of communities) are measured by Modularity of the partition. Modularity Q may be defined as the following.

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j),$$

where

Aij is the weight of the edge between i and j.
ki is the sum of weights of the vertex attached to the vertex I, also called as degree of the node
ci is the community to which vertex i is assigned
δ(x,y) is 1 if x = y and 0 otherwise
m = (1/2)∑ij Aij i.e number of links

Fig. 8. Definition of Modularity. *(Taken from https://medium.com/walmartglobaltech/demystifying-louvains-algorithm-and-its-implementation-in-gpu-9a07cdd3b010)*

The algorithm works with 2 major steps which are repeated iteratively.

1) Let there be N nodes in a graph network. To start, each node will be assigned to different community. For each neighbor a of node b, it is checked if the overall modurality increases by moving node a from its community to partition b. This will be repeated iteratively in a sequential matter till no improvemment in modularity can be achieved.
2) Next, it involves rebuilding a new network by clustering the nodes labelled in the same community. The weights between partitions are determined by adding up weights of edges from every node in a community.

Two of these steps will be done repeatedly until there is no more changes in community label (a maximum modularity is reached).

III. Proposed Method

One method to reinvent the color palette recommender is by using weighted graph and reinforcement learning. There are several actions needed to make this simple model.

1. Graph Initialization

The first step that needs to be taken is to build an undirected weight graph as the ground for the system; this graph resembles a similar purpose to a database. Please note that this paper does not focus on generating random colors. Instead, it aims to provide recommendations based on similar color palette inputs by the User. Thus, this experiment is under the assumption that the database of the system has already been loaded with colors, depending on the designer of the graph.

In this model, a vertex represents different colors. Each vertex will have a connection with the others through edges. These edges represent the color difference or distance that is stated in theoretical framework at II.B.. Each edge's numerical value will be weighted by ΔE_{00}, whereas the smaller the weight means the similar colors are.

Given Fig. 9., say we have three colors, but the graph has not been given any weight. We firstly need to calculate the similarity between the pairs of colors by Eq. 10, where delta E is just the color distance between the pairs.

$$weight = \frac{1}{1+\Delta E_{00}} \quad (10)$$

Then, the program will simply do the calculation based on lightness, chroma, and hues with the following equation for each pair.

$$\Delta E_{00} = \sqrt{\left(\frac{\Delta L'}{k_L S_L}\right)^2 + \left(\frac{\Delta C'}{k_C S_C}\right)^2 + \left(\frac{\Delta H'}{k_H S_H}\right)^2 + R_T\left(\frac{\Delta H'}{k_H S_H}\right)\left(\frac{\Delta C'}{k_C S_C}\right)}$$

After that, we updated each edges with the corresponding color differences.

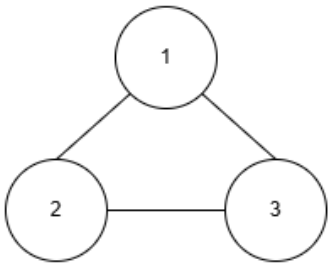


Fig. 9. A representation graph for the system. *(Taken from author's creation)*

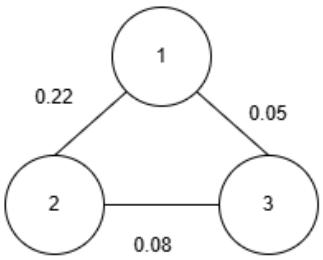


Fig. 10. Updated version of the previous unweighted graph. *(Taken from author's creation)*

2. Update Edges' Weights Based on User's Inputs

For the system to work, it needs data from user's input to search for similar colors towards the likings. The more input the system receives, the better it can recommend unique colors. Once given, the program will automatically add new nodes from the inputs and calculate the weights of each edge to pair with every node.

After that, Louvain's algorithm will create partitions of colors to effectively help the program search similar colors based on the several inputs that were given. Automatically, the system will search for top-k color, whereas k is the number of colors that we want the program to generate. The following figure is just an example of the clustering after the User's input.

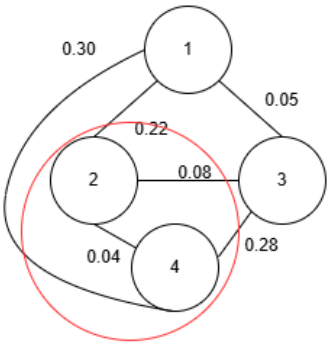


Fig. 11. Updated graph after adding User's input (node 4). *(Taken from author's creation)*

After being given a new color (represented by node 4), the program will calculate and automatically add weights to other nodes. Then, the program tries to cluster it to find a similar taste based on the inputs were given. The red circle marks the partitions created by the algorithm where the given color is similar towards what is currently in the database.

Although the simple theoretical implementation, there are still many flaws with the current system. For instance, it has not implemented the color harmony needed to guide beginner artists. At best, this system creates a fun way for artists to create palettes based on what colors they have right now. The system will only find the closest color to all the user's input. After further research, the author found that there were indeed ways to implement the theory, which is by adding more rules towards the weight system of the graph. However, due to limited time, the author is unable to continue the experimentation for this paper. This still didn't beat the purpose of the paper, which was about finding simple ways to implement graph, not perfecting the current tools that exist.

IV. Chromationary: Simple Program Implementation

After discovering the theory, there are ways to implement the system with simple python programs. There are also results for the implementation too.

A. *Program Implementation*

This simple program will show how the color generator works. To be able to implement this program, the author uses some libraries from Python, which are networkx, matplotlib, community, and colormath.

```python
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
from colormath.color_objects import LabColor, sRGBColor
from colormath.color_conversions import convert_color
from colormath.color_diff import delta_e_cie2000
import community  # python-louvain package
```

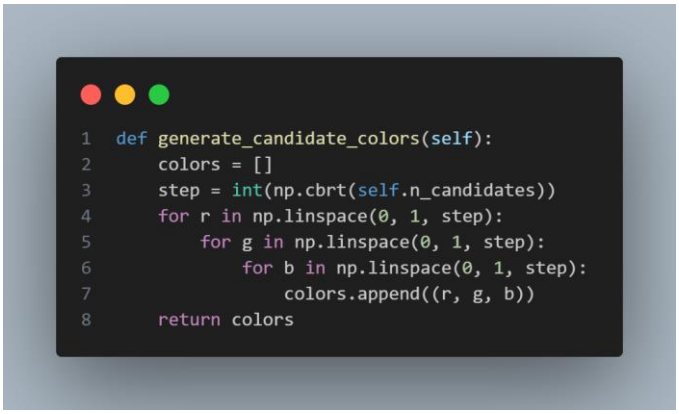
Fig. 12. Libraries used. *(Taken from author's archive)*

Then, there were several functions made for the simulatiom of program. Firstly, the system needs to initialize several candidates of color (this program specifically use 512 colors) and the threshold for the color distance where the program consider it as 'close'. This is also where the graph got initialize.

```python
def __init__(self, n_candidates=512, similarity_threshold=10):
    self.n_candidates = n_candidates
    self.similarity_threshold = similarity_threshold
    self.candidate_colors = self.generate_candidate_colors()
    self.graph = self.build_similarity_graph()
```

Fig. 13. Init function. *(Taken from author's archive)*

After initializing, the colors then got generated by the next function with the RGB model. It then got stored in a list.



```python
def generate_candidate_colors(self):
    colors = []
    step = int(np.cbrt(self.n_candidates))
    for r in np.linspace(0, 1, step):
        for g in np.linspace(0, 1, step):
            for b in np.linspace(0, 1, step):
                colors.append((r, g, b))
    return colors
```

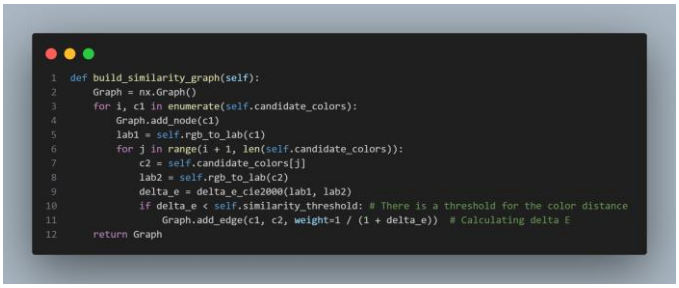
Fig. 14. Function to generate the colors. *(Taken from author's archive)*

The next function is a tool to convert RGB values to CIELAB color space.

```python
def rgb_to_lab(self, rgb):
    rgb_color = sRGBColor(rgb[0], rgb[1], rgb[2], is_upscaled=False)
    return convert_color(rgb_color, LabColor)
```

Fig. 15. RGB to LAB converter. *(Taken from author's archive)*

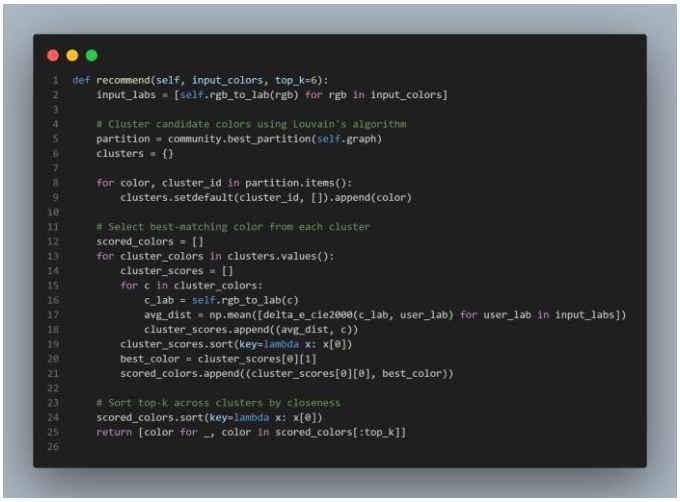
There will also be a need for generating the graph itself. This function specifically create edges that connects the current existing 'database' colors.



```python
def build_similarity_graph(self):
    Graph = nx.Graph()
    for i, c1 in enumerate(self.candidate_colors):
        Graph.add_node(c1)
        lab1 = self.rgb_to_lab(c1)
        for j in range(i + 1, len(self.candidate_colors)):
            c2 = self.candidate_colors[j]
            lab2 = self.rgb_to_lab(c2)
            delta_e = delta_e_cie2000(lab1, lab2)
            if delta_e < self.similarity_threshold: # There is a threshold for the color distance
                Graph.add_edge(c1, c2, weight=1 / (1 + delta_e))  # Calculating delta E
    return Graph
```

Fig. 16. Function to generate graph's edges. *(Taken from author's archive)*

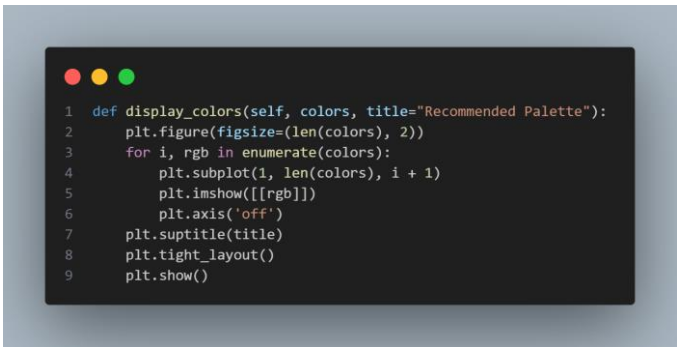
The recommend function is basically an implementation of Louvain's Algorithm. It will firstly do the iterative step as previously mentioned in theoritical framework on II.C.. It will firstly find the most efficient modularity and then rebuild the new network. It repeatedly does this until its goal to find the maximum of modularity achieved. The program will then continue find the top-k color from each cluster. In this case, the program will use k = 6.



```python
def recommend(self, input_colors, top_k=6):
    input_labs = [self.rgb_to_lab(rgb) for rgb in input_colors]

    # Cluster candidate colors using Louvain's algorithm
    partition = community.best_partition(self.graph)
    clusters = {}

    for color, cluster_id in partition.items():
        clusters.setdefault(cluster_id, []).append(color)

    # Select best-matching color from each cluster
    scored_colors = []
    for cluster_colors in clusters.values():
        cluster_scores = []
        for c in cluster_colors:
            c_lab = self.rgb_to_lab(c)
            avg_dist = np.mean([delta_e_cie2000(c_lab, user_lab) for user_lab in input_labs])
            cluster_scores.append((avg_dist, c))
        cluster_scores.sort(key=lambda x: x[0])
        best_color = cluster_scores[0][1]
        scored_colors.append((cluster_scores[0][0], best_color))

    # Sort top-k across clusters by closeness
    scored_colors.sort(key=lambda x: x[0])
    return [color for _, color in scored_colors[:top_k]]
```

Fig. 17. Louvain's algorithm implementation. *(Taken from author's archive)*

The following piece of code is just a function to display color in the UI interface.



```python
def display_colors(self, colors, title="Recommended Palette"):
    plt.figure(figsize=(len(colors), 2))
    for i, rgb in enumerate(colors):
        plt.subplot(1, len(colors), i + 1)
        plt.imshow([[rgb]])
        plt.axis('off')
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()
```

Fig. 18. Function to display colors. *(Taken from author's archive)*

Lastly, the main function is used to run the program as an overall.

```python
# Chromationary's Simulation
if __name__ == "__main__":
    system = ChromationarySystem()

    print("Enter your color(s) as RGB (0-255) values. Example: 255 0 0")
    print("Type 'finish' to stop entering colors.")

    user_colors = []

    while True:
        line = input("Enter a color (or type 'finish' to stop): ")
        if line.strip().lower() == "finish":
            break
        try:
            r, g, b = map(int, line.strip().split())
            user_colors.append((r / 255, g / 255, b / 255))
        except ValueError:
            print("Invalid format. Enter three numbers separated by spaces.")

    if not user_colors:
        print("No colors entered.")
    else:
        recommendations = system.recommend(user_colors, top_k=6)
        system.display_colors(user_colors, title="Your Input Colors")
        system.display_colors(recommendations, title="Recommended Colors")
```

Fig. 19. Main function. *(Taken from author's archive)*

B. Result

The following is the result of implementing the system. Firstly, the program will ask you for several input until the User wanted to end it by typing "finish".

```
Enter your color(s) as RGB (0-255) values. Example: 255 0 0
Type 'finish' to stop entering colors.
Enter a color (or type 'finish' to stop): 23 0 100
Enter a color (or type 'finish' to stop): 36 80 234
Enter a color (or type 'finish' to stop): finish
```

Fig. 20. Program's console. *(Taken from author's archive)*

After that, the program will give an interface of the color you chose and then recommend other colors as a recommendation.

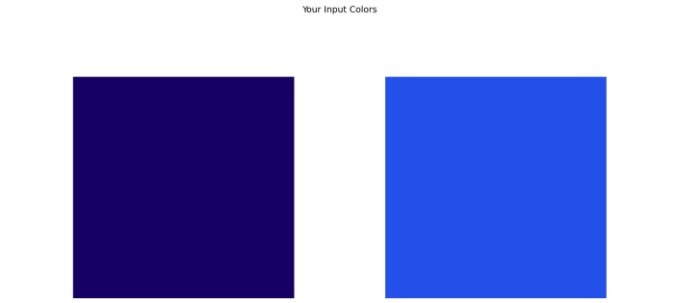


Fig. 21. Colors that we picked. *(Taken from author's archive)*

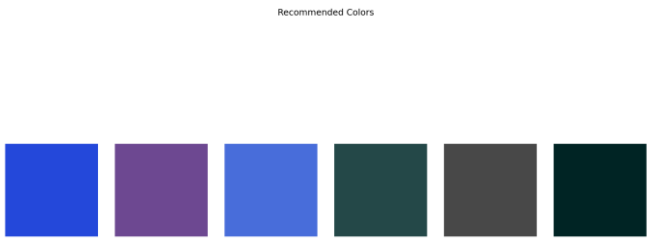


Fig. 22. Recommendation results. *(Taken from author's archive)*

The author also tried to visualize how the graph works with its weight. However, since the initial program consists of 512 seed colors, a simpler program that consists of only a few colors is needed (from graph_representation.py). The graph that got generated should look like the following figure.

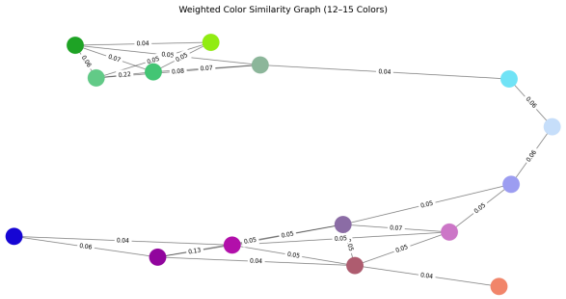


Fig. 23. A simple graph representation from the program. *(Taken from author's archive)*

V. Conclusion

With this research, it can be concluded that graph theory has shown many applications to many sectors. It has been discussed that weight-based graphs can be utilized as a base system for a color palette generator system. This paper specifically explored a graph-based approach to color palette generation, where nodes represent colors and weighted edges define their relations based on principles like contrast, similarity, or perceptual distance. By incorporating user interactions, we may create a system that gives artists a much more fluid way to express themselves and not to be stuck without the personalization of color inputs. This is plausible due to the iterative algorithm, Louvain's algorithm. Further improvements, such as implementing additional rules into the weight system, could enhance the algorithm's main purpose to help novice's artist in understanding color theorem regarding harmony.

By applying this approach, especially for fields such as the art industry, the author hopes it can serve as a tool to enhance artists' creative freedom and self-expression. The author also hopes for further research on this topic to perfect the algorithm better when given the time to or for anyone who wants to hold it.

APPENDIX

Source code: https://github.com/jenka-h/itb/tree/e39dd8cc084cb90c2e7988795f59b3a06ac4e642/semester-2/matdis

VIDEO LINK AT YOUTUBE

https://youtu.be/mX_6aH8Dj5w

ACKNOWLEDGMENT

The author would like to express gratitude towards God for the guidance and the opportunity to learn and move forward through the process of hardship during the semester. The author would also like to thank the lecturers of Discrete Mathematics IF1220, Mr. Arrival Dwi Santosa and Mr. Rinaldi Munir for the vast knowledge they have shared throughout the course. There were so many insightful ideas that were given to us. Lastly, the writer will always be in debt to her parents. They are her muse and reasons to continue to live and roam the world, to be curious. Without their support, the author would have not advanced this far in life.

REFERENCES

[1] R. Munir, (2024). "IF2120 Matematika Diskrit." Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/matdis.htm

[2] P. Weingerl, Theory of Colour Harmony and Its Application, vol. 25. Tehnicki vjesnik - Technical Gazette, 2018.

[3] A. Burambekova and P. Shamoi, (2024). "Comparative Analysis Of Color Models For Human Perception And Visual Color Difference." Available: https://arxiv.org/abs/2406.19520

[4] Anon, (2018). "A short history of color theory - Programming Design Systems." Accessed: June 20, 2025. [Online]. Available: https://programmingdesignsystems.com/color/a-short-history-of-color-theory/index.html

[5] A. Mishra, (2019). "Demystifying Louvain's Algorithm and Its implementation in GPU." Accessed: June 20, 2025. [Online]. Available: https://medium.com/walmartglobaltech/demystifying-louvains-algorithm-and-its-implementation-in-gpu-9a07cdd3b010

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Juni 2025

Jennifer Khang NIM 13524110