Use of Graph and Probability Theory in Spotify's Smart Shuffle Recommendations

Angelina Efrina Prahastaputri - 13523060[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]efrinaprahastaputri@gmail.com, [2]13523060@std.stei.itb.ac.id

***Abstract*—This paper explores the use of graph theory and probability theory in Spotify's Smart Shuffle Recommendations on generating song shuffle and recommendations through different approaches. We will explore how CBF and IBCF works. The result of this research aims to get a better understanding about Spotify's Smart Shuffle Recommendations to improve song shuffle and recommendations, further enhancing the user experience on listening and enjoying music.**

***Keywords*—Graph Theory, Probability Theory, Spotify's Smart Shuffle Recommendations, User Experience.**

I. INTRODUCTION

In this modern era of entertainment media, music became one of the most popular media known in society. One of the most popular music-streaming platforms, Spotify, has become one of the first of its kind to transform the way people can listen and enjoy music. Even though Spotify has provided unlimited access to an unlimited number of songs across various genres, artists, etc., there are still challenges and difficulties on enhancing the user experience especially about managing and understanding the complexity of user's preferences. To face these challenges and difficulties, Spotify keeps on trying to improve its features to enhance the user experience.

One of Spotify's features that has helped users explore their music taste is called Smart Shuffle Recommendations. This feature combines shuffling and recommendations between songs to help create continuous experience for users. This shuffling and recommendations algorithm plays an important role in enhancing the user experience by creating satisfaction but at the same time preventing boredom. That's the reason why we need a deeper understanding of how songs are correlated to each other, either through the same genres, artists, vibes, etc.

The author chose the title "Use of Graph and Probability Theory in Spotify's Smart Shuffle Recommendations" not only because it is now widely used by Spotify's users, but also because it's important to know how songs inside the Spotify's environment correlated to each other more importantly by using graph theory to know each song's attributes and probability theory to know how songs are spread throughout a playlist. This paper discusses how Spotify's Smart Shuffle Recommendations works by using graph and probability theory to get a better understanding behind the shuffling and recommendations algorithm. With better understanding, the author hopes that this paper can help enhance Spotify users' experience on listening and enjoying music

II. THEORETICAL BASIS

A. Graph Theory

A.1. Definition

Graph is generally used to represent discrete objects and their relationships. A graph G = (V, E) consists of V, a nonempty set of vertices (or nodes) and E, a set of edges. Each edge has either one or two vertices associated with it, called its endpoints. An edge is said to connect its endpoints [2].

A.2. Terminologies

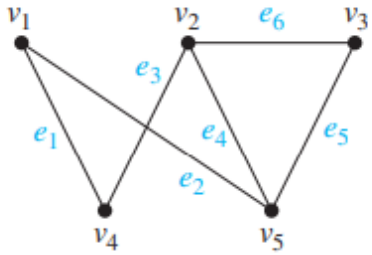


Figure 2.1 Simple Graph
Source: *Discrete Mathematics and Application to Computer Science 8th Edition*

a. Vertices/Nodes

Vertices or nodes in a graph represent objects or entities. It is typically denoted by letters. Vertices can store attributes or other information corresponding to their objects or entities.

Based on Fig. 2.1, v_1, v_2, v_3, v_4, and v_5 are the graph's vertices. On any other application, these vertices can represent information such as locations, objects, names, etc.

b. Edges

Edges in a graph are the connections or relationships between pair of vertices. It is typically denoted as (u, v) where u and v are vertices.

Based on Fig. 2.1, e_1, e_2, e_3, e_4, e_5, and e_6 are the graph's edges. A vertice can have more than one edge connected to it. An edge

can also connect a vertice with the vertice itself, this connection is called loop.

c. Adjacent

Two vertices u and v in an undirected graph G are called adjacent (or neighbors) in G if u and v are endpoints of an edge e of G [2]. Based on Fig. 2.1, v_1 and v_4 are adjacent (connected by e_1), and v_1 and v_5 are adjacent (connected by e_2).

d. Incident

An edge e with vertices u and v as its endpoints is called incident with vertices u and v and e is said to connect u and v [3]. Based on Fig. 2.1, e_1 is incident with v_1 and v_4, e_2 is incident with v_1 and v_5.

e. Degree

The degree of a vertex in an undirected graph is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. The degree of the vertex v is denoted by deg(v) [2]. Based on Fig. 2.1, deg(v_1) is 2 and deg(v_2) is 3.

f. Path

A path is a sequence of edges that begins at a vertex of a graph and travels from vertex to vertex along edges of the graph. As the path travels along its edges, it visits the vertices along this path, that is, the endpoints of these edges [2]. A path with the length of n starts from v_0 and ends on v_n in a graph consists of alternating row of vertices and edges [2]. Based on Fig. 2.1, one of the paths from v_1 to v_3 is v_1-e_1-v_4-e_3-v_2-e_4-v_5-e_5-v_3.

g. Cycle/Circuit

A cycle or a circuit is a path that starts and ends on the same vertice [2]. Based on Fig. 2.1, one of the circuits is v_1-e_1-v_4-e_3-v_2-e_4-v_5-e_2-v_1, which starts and ends on v_1.

h. Connectedness

An undirected graph is called connected if there is a path between every pair of distinct vertices of the graph. An undirected graph that is not connected is called disconnected. We say that we disconnect a graph when we remove vertices or edges, or both, to produce a disconnected subgraph. A directed graph is strongly connected if there is a path from a to b and from b to a whenever a and b are vertices in the graph. A directed graph is weakly connected if there is a path between every two vertices in the underlying undirected graph [2].

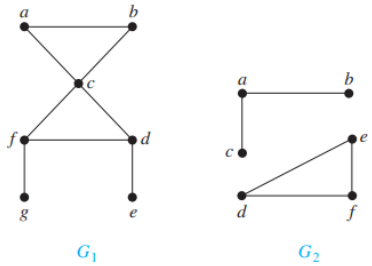


Figure 2.2 Connected Graph (G_1) and Disconnected Graph (G_2)
Source: *Discrete Mathematics and Application to Computer Science 8th Edition*

i. Subgraph

When edges and vertices are removed from a graph, without removing endpoints of any remaining edges, a smaller graph is obtained. Such a graph is called a subgraph of the original graph. A subgraph of a graph G = (V, E) is a graph H = (W, F), where W ⊆ V and F ⊆ E. A subgraph H of G is a proper subgraph of G if H = G. Let G = (V, E) be a simple graph. The subgraph induced by a subset W of the vertex set V is the graph (W, F), where the edge set F contains an edge in E if and only if both endpoints of this edge are in W [2].

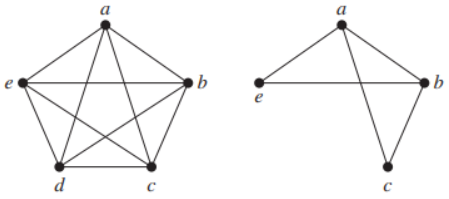
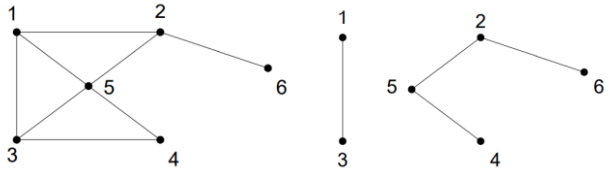


Figure 2.3 Subgraph of K_5
Source: *Discrete Mathematics and Application to Computer Science 8th Edition*

j. Cut-Set

Cut-set of a connected graph G is a set that contains edges that if removed from graph G makes graph G a disconnected graph. Cut-set always creates two separate graph components [3].



(a) (b)

Figure 2.4 Before (a) and After (b) Cut-Set of a Graph
Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

Based on Fig. 2.4, graph (a) is a connected graph and graph (b) are components after cut-set of graph (a).

A.3. Simple Graph

Simple graph is a graph that does not contain any loop or multiple edges between the same pair of vertices as shown in Fig. 2.1.

A.3. Unsimple Graph

Unsimple graph is a graph that contains any loop or multiple edges between the same pair of vertices.

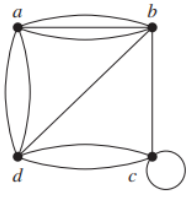


Figure 2.5 Unsimple Graph
Source: *Discrete Mathematics and Application to Computer Science 8th Edition*

Unsimple graph is divided into two categories. First, multigraph, a graph that contains multiple edges between the

same pair of vertices but no loop. Second, pseudograph, a graph that contains both loop and multiple edges between the same pair of vertices. Fig. 2.5 is a pseudograph.

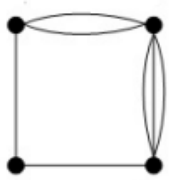


Figure 2.6 Multigraph
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

A.4. Undirected Graph

Undirected graph is a graph where each edge does not have any certain directions. The connections between vertices are reciprocal. Based on Fig. 2.7, the connection between a and b vertices is reciprocal. The edge only means that there is connection between two vertices [3].

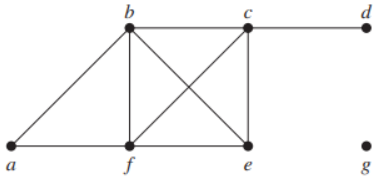


Figure 2.7 Undirected Graph
Source: *Discrete Mathematics and Application to Computer Science 8th Edition*

A.5. Directed Graph

Directed graph is a graph where each edge certain any certain directions. The connections between vertices are not reciprocal. When (u, v) is an edge of the graph G with directed edges, u is said to be adjacent to v and v is said to be adjacent from u. The vertex u is called the initial vertex of (u, v), and v is called the terminal or end vertex of (u, v). The initial vertex and terminal vertex of a loop are the same. In a graph with directed edges the in-degree of a vertex v, denoted by $\deg^-(v)$, is the number of edges with v as their terminal vertex. The out-degree of v, denoted by $\deg^+(v)$, is the number of edges with v as their initial vertex. Based on Fig. 2.7, the connection between a and b vertices is not reciprocal. [2].

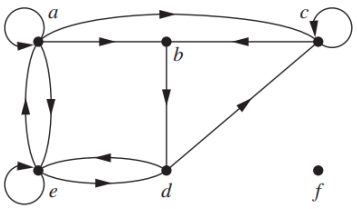


Figure 2.8 Directed Graph
Source: *Discrete Mathematics and Application to Computer Science 8th Edition*

A.6. Complete Graph

A complete graph on n vertices, denoted by Kn, is a simple graph that contains exactly one edge between each pair of distinct vertices. A simple graph for which there is at least one pair of distinct vertices not connected by an edge is called noncomplete [ ]. A complete graph's unique property is that every vertex has the same number of degree.

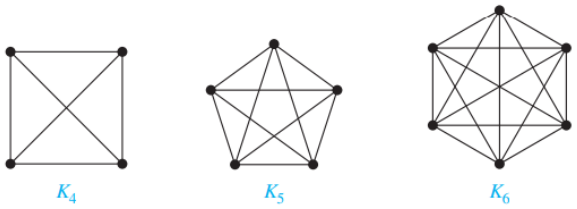


Figure 2.8 Complete Graphs
Source: *Discrete Mathematics and Application to Computer Science 8th Edition*

A.7. Bipartite Graph

A simple graph G is called bipartite if its vertex set V can be partitioned into two disjoint sets V1 and V2 such that every edge in the graph connects a vertex in V1 and a vertex in V2 (so that no edge in G connects either two vertices in V1 or two vertices in V2). When this condition holds, we call the pair (V1, V2) a bipartition of the vertex set V of G [2].

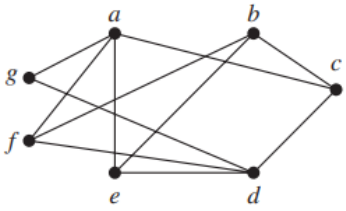


Figure 2.9 Bipartite Graph
Source: *Discrete Mathematics and Application to Computer Science 8th Edition*

Fig. 2.9 is a bipartite graph because its vertex set is the union of two disjoint sets, {a, b, d} and {c, e, f, g}, and each edge connects a vertex in one of these subsets to a vertex in the other subset [2].

A.8. Weighted Graph

Weighted graph is a graph with weight assigned to each edge. The weight of the edge is typically represented by number as shown in Fig. 2.10.

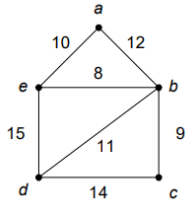


Figure 2.10 Weighted Graph
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

B. Probability Theory

B.1. Finite Probability

An experiment is a procedure that yields one of a given set of possible outcomes. The sample space of the experiment is the set of possible outcomes. An event is a subset of the sample space. Laplace's definition of the probability of an event with finitely many possible outcomes will now be stated.

If S is a finite nonempty sample space of equally likely outcomes, and E is an event, that is, a subset of S, then the *probability* of E is $p(E) = \frac{|E|}{|S|}$.

Figure 2.11 Finite Probability
Source: *Discrete Mathematics and Application to Computer Science 8th Edition*

According to Laplace's definition, the probability of an event is between 0 and 1. To see this, note that if E is an event from a finite sample space S, then $0 \leq |E| \leq |S|$, because $E \subseteq S$. Thus, $0 \leq p(E) = |E|/|S| \leq 1$. Let E be an event in a sample space S. The probability of the event $E^o = S - E$, the complementary event of E, is given by $p(E^o) = 1 - p(E)$. [2].

B.2. Bayes' Theorem

we can find the conditional probability that an event F occurs, given that an event E has occurred, when we know $p(E \mid F)$, $p(E \mid F)$, and $p(F)$. The result we can obtain is called Bayes' theorem [2].

BAYES' THEOREM Suppose that E and F are events from a sample space S such that $p(E) \neq 0$ and $p(F) \neq 0$. Then

$$p(F \mid E) = \frac{p(E \mid F)p(F)}{p(E \mid F)p(F) + p(E \mid \overline{F})p(\overline{F})}.$$

Figure 2.12 Bayes' Theorem
Source: *Discrete Mathematics and Application to Computer Science 8th Edition*

B.3. Permutations and Combinations

A permutation of a set of distinct objects is an ordered arrangement of these objects. We also are interested in ordered arrangements of some of the elements of a set. An ordered arrangement of r elements of a set is called an r-permutation. Let S = {1, 2, 3}. The ordered arrangement 3, 1, 2 is a permutation of S. The ordered arrangement 3, 2 is a 2-permutation of S. The number of r-permutations of a set with n elements is denoted by P (n, r). We can find P (n, r) using the product rule.

If n and r are integers with $0 \leq r \leq n$, then $P(n, r) = \frac{n!}{(n-r)!}$.

Figure 2.13 Permutation
Source: *Discrete Mathematics and Application to Computer Science 8th Edition*

An r-combination of elements of a set is an unordered selection of r elements from the set. Thus, an r-combination is simply a subset of the set with r elements. Let S be the set {1, 2, 3, 4}. Then {1, 3, 4} is a 3-combination from S. The number of r-combinations of a set with n distinct elements is denoted by C(n, r) [2].

The number of r-combinations of a set with n elements, where n is a nonnegative integer and r is an integer with $0 \leq r \leq n$, equals

$$C(n, r) = \frac{n!}{r!\,(n-r)!}.$$

Figure 2.14 Combination
Source: *Discrete Mathematics and Application to Computer Science 8th Edition*

C. Smart Shuffle Recommendations

There are two ways to shuffle your playlist in Spotify. First is the regular Shuffle. It is able to shuffle any playlist, album, or artist profile to mix up what song plays next. Second, is the one we will focus on this paper, Smart Shuffle. It is the default play mode that keeps listening sessions fresh not only by shuffling your playlist but also by mixing in recommendations that match the vibe. Any recommendation will have a spark symbol next to the artist name [4].

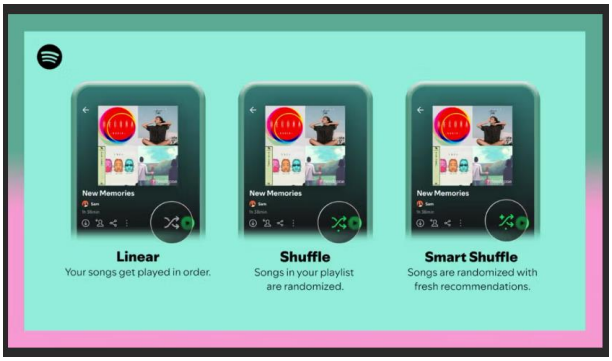
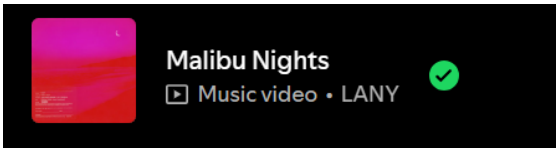
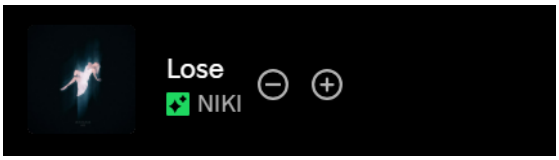


Figure 2.15 Spotify's Shuffle Feature
Source: https://support.spotify.com/id-en/article/shuffle-play/



(a)



(b)

Figure 2.16 Spotify's Smart Shuffle Recommendations

Based on Fig. 2.14, (a) shows a song that is already in the playlist, indicated by the checkmark symbol on the far right. Meanwhile, (b) shows a song that isn't in the playlist, indicated by the nonexistent checkmark symbol and the plus symbol (for adding song to the playlist). There is also a spark symbol next to the artist name that indicates the song is a recommendation from the Smart Shuffle. Song recommendation from Smart Shuffle will appear every three songs.

III. IMPLEMENTATION

Spotify initially used the Fisher-Yates shuffle algorithm to randomize playlists, which produces a perfectly random order for songs. However, users still perceived this randomness as "not random enough" because songs by the same artist could play consecutively creating clusters that felt unshuffled. To address this, Spotify developed an algorithm that spaces out songs by the same artist throughout the playlist, reducing noticeable clustering. This approach creates sequence that feels "more random" to listeners even though it's technically less random than the original method. This adjustment acknowledged the "gambler's fallacy" where people expect outcomes to balance out in short term. By spacing out songs from the same artist, Spotify's shuffle aligns better with user expectations of randomness, enhancing the listening experience [4].

In this paper, we will focus on the implementation of graph theory and probability theory via two different approaches, content-based filtering and item-based collaborative filtering.

A. Content-Based Filtering

Content-Based Filtering algorithm for Spotify's Smart Shuffle recommends songs based on the features of the song itself. It includes extracting features from songs that have been listened to by the user. Such features for example, genre, artist, tempo, or even mood/vibe. We will represent songs as nodes and song features as edge weights between the nodes, creating a weighted graph model. Next, we will calculate the similarity of two features using cosine similarity to get the shortest path/distance between two songs based on feature similarity to get recommendations. To create randomness, we will add random walks over the feature graph to probabilistically explore connected nodes and recommend based on visitation probability.

Not only that, but we will also integrate Bayes' theorem to build a Bayesian network where nodes are features and songs, and edges represent probabilistic dependencies. Then, we will assign probabilities to edges based on similarity scores, using these to rank recommendations.

```python
import matplotlib.pyplot as plt
import networkx as nx
import networkx as nx
import random

def build_feature_graph_with_probabilities(song_features):
    G = nx.Graph()
    for song1, features1 in song_features.items():
        for song2, features2 in song_features.items():
            if song1 != song2:
                similarity = calculate_similarity(features1, features2)
                G.add_edge(song1, song2, weight=similarity)

    # Normalize edge weights to probabilities
    for node in G.nodes:
        total_weight = sum(G[node][neighbor]['weight'] for neighbor in G.neighbors(node))
        for neighbor in G.neighbors(node):
            G[node][neighbor]['probability'] = G[node][neighbor]['weight'] / total_weight
    return G
```

Figure 3.1 Building Feature Graph

```python
def random_walk_with_probabilities(G, current_song, num_recommendations):
    walks = [current_song]
    for _ in range(100):  # Random walk iterations
        current = walks[-1]
        neighbors = list(G.neighbors(current))
        probabilities = [G[current][neighbor]['probability'] for neighbor in neighbors]
        next_song = random.choices(neighbors, weights=probabilities)[0]
        walks.append(next_song)

    # Aggregate and recommend based on frequency
    recommendations = {}
    for song in walks:
        recommendations[song] = recommendations.get(song, 0) + 1

    sorted_recommendations = sorted(recommendations.items(), key=lambda x: x[1], reverse=True)
    return [song for song, _ in sorted_recommendations[:num_recommendations]]
```

Figure 3.2 Random Walk Over the Feature Graph

```python
def calculate_similarity(features1, features2):
    # Example: Cosine similarity
    dot_product = sum(f1 * f2 for f1, f2 in zip(features1, features2))
    norm1 = sum(f1**2 for f1 in features1)**0.5
    norm2 = sum(f2**2 for f2 in features2)**0.5
    return dot_product / (norm1 * norm2)
```

Figure 3.3 Calculating Similarity Between Features

```python
def recommend_from_feature_graph(G, current_song, num_recommendations):
    # Get neighbors for the current song and their associated probabilities
    neighbors = list(G.neighbors(current_song))
    recommendations = {}

    for neighbor in neighbors:
        recommendations[neighbor] = G[current_song][neighbor]['probability']

    # Sort the recommendations based on the probabilities and get the top N
    sorted_recommendations = sorted(recommendations.items(), key=lambda x: x[1], reverse=True)
    return [song for song, _ in sorted_recommendations[:num_recommendations]]
```

Figure 3.4 Getting Recommendations from Feature Graph

B. Item-Based Collaborative Filtering

Item-Based Collaborative Filtering algorithm for Spotify's Smart Shuffle recommends songs based on their relationship to other songs in the dataset. We will represent users and songs as two sets of nodes, and edges as user-song interactions, creating a bipartite graph model. Then, we will apply projection to create a song-song similarity graph.

Not only that, but we will also use probabilistic models to calculate the likelihood of two songs being listened to together and recommend songs based on the highest probabilities.

```python
import matplotlib.pyplot as plt
import networkx as nx
import networkx as nx
import random

def build_bipartite_graph(user_song_matrix):
    B = nx.Graph()
    for user, songs in user_song_matrix.items():
        for song in songs:
            B.add_node(user, bipartite=0)
            B.add_node(song, bipartite=1)
            B.add_edge(user, song)
    return B
```

Figure 3.5 Building Bipartite Graph

```python
def project_to_song_graph_with_probabilities(B):
    # Project bipartite graph to song graph
    song_nodes = [n for n in B.nodes if B.nodes[n]['bipartite'] == 1]
    song_graph = nx.Graph()
    for song1 in song_nodes:
        for song2 in song_nodes:
            if song1 != song2:
                shared_users = sum(1 for user in B.neighbors(song1) if user in B.neighbors(song2))
                if shared_users > 0:
                    song_graph.add_edge(song1, song2, weight=shared_users)

    # Normalize edge weights to probabilities
    for node in song_graph.nodes:
        total_weight = sum(song_graph[node][neighbor]['weight'] for neighbor in song_graph.neighbors(node))
        for neighbor in song_graph.neighbors(node):
            song_graph[node][neighbor]['probability'] = song_graph[node][neighbor]['weight'] / total_weight
    return song_graph
```

Figure 3.6 Projecting the Bipartite Graph

```python
def recommend_from_projection(song_graph, target_song, num_recommendations):
    neighbors = list(song_graph.neighbors(target_song))
    recommendations = sorted(neighbors, key=lambda song: song_graph[target_song][song]['weight'], reverse=True)
    return recommendations[:num_recommendations]
```

Figure 3.7 Getting Recommendations from Projection

IV. ANALYSIS

A. Content-Based Filtering

Here's the dataset for analysis:

```
song_features = {
    "Song1": [0.9, 0.2, 0.5],
    "Song2": [0.7, 0.4, 0.3],
    "Song3": [0.5, 0.8, 0.6],
    "Song4": [0.6, 0.6, 0.4],
    "Song5": [0.3, 0.9, 0.7],
    "Song6": [0.8, 0.1, 0.4],
    "Song7": [0.2, 0.7, 0.5],
    "Song8": [0.6, 0.4, 0.5],
    "Song9": [0.4, 0.8, 0.6],
    "Song10": [0.9, 0.3, 0.5],
    "Song11": [0.7, 0.5, 0.4],
    "Song12": [0.5, 0.7, 0.2],
}

user_song_matrix = {
    "User1": ["Song1", "Song2", "Song4"],
    "User2": ["Song2", "Song3", "Song5"],
    "User3": ["Song1", "Song3", "Song6"],
    "User4": ["Song4", "Song7", "Song8"],
    "User5": ["Song5", "Song6", "Song9"],
    "User6": ["Song1", "Song7", "Song10"],
    "User7": ["Song2", "Song8", "Song11"],
    "User8": ["Song3", "Song5", "Song12"],
    "User9": ["Song6", "Song9", "Song12"],
    "User10": ["Song4", "Song8", "Song10"],
    "User11": ["Song1", "Song11", "Song12"],
    "User12": ["Song7", "Song9", "Song10"]
}
```

Figure 4.1 Dataset for CBF

Here's the result for the dataset:

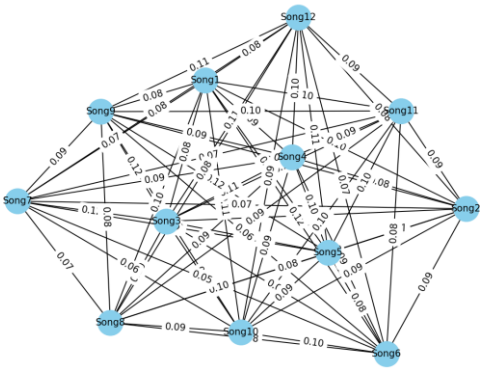


Figure 4.2 Feature Graph for Dataset

B. Item-Based Collaborative Filtering

Here's the dataset for analysis:

```
song_features = {
    "Song1": [0.9, 0.2, 0.5],
    "Song2": [0.7, 0.4, 0.3],
    "Song3": [0.5, 0.8, 0.6],
    "Song4": [0.6, 0.6, 0.4],
    "Song5": [0.3, 0.9, 0.7],
    "Song6": [0.8, 0.1, 0.4],
    "Song7": [0.2, 0.7, 0.5],
    "Song8": [0.6, 0.4, 0.5],
    "Song9": [0.4, 0.8, 0.6],
    "Song10": [0.9, 0.3, 0.5],
    "Song11": [0.7, 0.5, 0.4],
    "Song12": [0.5, 0.7, 0.2],
}

user_song_matrix = {
    "User1": ["Song1", "Song2", "Song4"],
    "User2": ["Song2", "Song3", "Song5"],
    "User3": ["Song1", "Song3", "Song6"],
    "User4": ["Song4", "Song7", "Song8"],
    "User5": ["Song5", "Song6", "Song9"],
    "User6": ["Song1", "Song7", "Song10"],
    "User7": ["Song2", "Song8", "Song11"],
    "User8": ["Song3", "Song5", "Song12"],
    "User9": ["Song6", "Song9", "Song12"],
    "User10": ["Song4", "Song8", "Song10"],
    "User11": ["Song1", "Song11", "Song12"],
    "User12": ["Song7", "Song9", "Song10"]
}
```

Figure 4.3 Dataset for IBCF

Here's the result for the dataset:

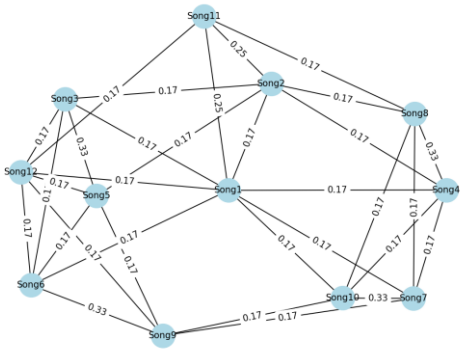


Figure 4.4 Bipartite Graph Projection for Dataset

In this paper, we explored the implementation of recommendation systems using Graph Theory and Probability Theory, specifically applied to song recommendations as in Spotify's Smart Shuffle. We focused on Content-Based Filtering (CBF) and Item-Based Collaborative Filtering (IBCF), both of which leverage graph structures to model relationships between songs, users, and song features.

VI. CONCLUSION

This project demonstrated how graph theory and probability theory can be effectively used in building recommendation systems. Both Content-Based Filtering and Item-Based Collaborative Filtering benefit from these methods:

1. Graph Theory helps visualize and model relationships between entities (songs, users, or song features) in a meaningful way, enabling the construction of projection graphs that uncover similarities.
2. Probability Theory enables the ranking of potential recommendations, making sure that highly connected or similar items are more likely to be recommended based on their probability.

Both methods provide complementary approaches for generating music recommendations: while IBCF focuses on shared user preferences, CBF leverages song characteristics directly. These techniques are foundational in music recommendation systems like Spotify's Smart Shuffle feature, which combines user data, content features, and probabilistic models to offer dynamic, personalized recommendations.

Further research could involve incorporating hybrid filtering approaches, combining both CBF and IBCF, or exploring more advanced machine learning models like matrix factorization, which have shown great promise in large-scale recommendation systems.

VII. ACKNOWLEDGEMENT

The author would like to express their gratitude to several parties that helped the making of this paper. First and foremost, sincere thanks to God for guiding the author through the entire process of making this paper from learning, researching, and writing, until eventually this paper is complete. The author also acknowledges the immense support and guidance from the lecturer of IF1220 Discrete Mathematics, Mr. Rila Mandala and

Mr. Rinaldi Munir, that has significantly helped the author enrich their knowledge. Special thanks also to the author's family, friends, and all the ITB Informatics students for the unwavering support throughout the entire semester.

Through this paper, the author hopes it can bring more knowledge for the author and for the readers on better understanding about the use of graph and probability theory in one of the mostly used music-streaming platform features, Spotify Smart Shuffle Recommendations.

REFERENCES

[1] Fernando, Jason. (2023). "Pemodelan Hubungan Antar Lagu dalam Spotify Menggunakan Teori Graf dan Wrapper untuk Pengalaman Pengguna yang Lebih Personal". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/Makalah2023/Makalah-Matdis-2023%20(156).pdf accessed on January 6, 2025.

[2] K. H. Rosen. (2018). *Discrete Mathematics and Application to Computer Science 8th Edition*. Mc Graw-Hill, Inc.

[3] Munir, Rinaldi. (2024). "IF2120 Matematika Diskrit – Semester I Tahun 2024/2025". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/matdis24-25.htm accessed on January 6, 2025.

[4] Spotify Engineering. (2014). "How to Shuffle Songs?" https://engineering.atspotify.com/2014/02/how-to-shuffle-songs/ accessed on January 6, 2025.

[5] Spotify. (2024). "Shuffle Play". https://support.spotify.com/id-en/article/shuffle-play/ accessed on January 8, 2025.

[6] Varun. (2020). "Cosine similarity: How does it measures the similarity, Maths behind and usage in Python". https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db accessed on January 8, 2025.

[7]

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, 6 Januari 2025

Angelina Efrina Prahastaputri 13523060