Pemodelan Hubungan Antar Lagu dalam Spotify Menggunakan Teori Graf dan Wrapper untuk Pengalaman Pengguna yang Lebih Personal

Jason Fernando - 13522156
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13522156@std.stei.itb.ac.id

***Abstract*—This paper explores the modeling of relationships between songs in the context of Spotify using graph theory and a wrapper. By leveraging graph algorithms to analyze patterns of relationships between songs, we developed a wrapper to gather music attribute information from the Spotify interface. The results of this modeling are expected to enhance song recommendations, create a more personalized user experience, and strengthen Spotify's competitiveness in the music streaming service market.**

***Keywords*—Spotify, Graph Theory, Wrapper, Modeling Song Relationships, Song Recommendations, User Experience.**

I. PENDAHULUAN

Dalam era transformasi media menuju digitalisasi, layanan streaming musik, terutama Spotify, menjadi pionir dalam mengubah cara konsumen mengakses dan menikmati musik. Walaupun akses mudah ke jutaan lagu telah meningkatkan kebebasan musikal, tantangan baru muncul terkait pengelolaan dan pemahaman atas kompleksitas pilihan yang tersedia. Fenomena ini membuka peluang untuk mengeksplorasi lebih lanjut cara meningkatkan kualitas pengalaman pengguna dalam menjelajahi ranah musik digital.

Salah satu masalah signifikan yang dihadapi pengguna Spotify saat ini adalah kurangnya personalisasi dalam rekomendasi lagu. Meskipun algoritma rekomendasi telah memainkan peran penting dalam membimbing pengguna menuju lagu-lagu baru berdasarkan histori mendengarkan mereka, sering kali rekomendasi tersebut dirasakan sebagai terlalu umum dan kurang mendalam. Pengalaman mendengarkan yang lebih memuaskan membutuhkan pemahaman yang lebih mendalam mengenai hubungan antar lagu, seperti bagaimana lagu-lagu tertentu berkaitan dalam genre, suasana, atau elemen musikal lainnya.

Oleh karena itu, pemilihan judul "Pemodelan Hubungan Antar Lagu dalam Spotify Menggunakan Teori Graf dan Wrapper untuk Pengalaman Pengguna yang Lebih Personal" menjadi sangat relevan di era ini. Penelitian ini dipicu oleh kebutuhan untuk menjelajahi lebih dalam dinamika dan keterkaitan antar lagu dalam ekosistem Spotify. Dengan menerapkan teori graf dan menggunakan wrapper untuk menggali informasi tambahan tentang atribut musik, penelitian ini bertujuan untuk memberikan kontribusi positif terhadap kemampuan Spotify dalam memberikan rekomendasi lagu yang lebih cerdas, kontekstual, dan personal. Oleh karena itu, diharapkan bahwa penelitian ini dapat menjadi solusi inovatif untuk meningkatkan kepuasan pengguna, menciptakan pengalaman mendengarkan yang lebih mendalam, dan mengatasi tantangan masa kini dalam konsumsi musik digital.

II. LANDASAN TEORI

A. Teori Graf

A.1. Definisi Graf

Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Menurut definisi, Graf G = (V, E), yang dalam hal ini, V adalah = himpunan tidak-kosong dari simpul-simpul {v1,v2,…,vn}, sedangkan E adalah himpunan sisi yang menghubungkan sepasang simpul {e1,e2,…,en}.

Bedasarkan orientasi arah pada sisi, graf dibedakan menjadi 2 jenis, yaitu :

a. Graf tak berarah (undirected graph)

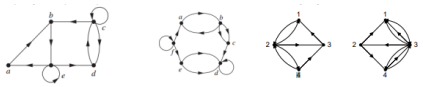
Graf tak berarah merupakan jenis graf di mana tiap edge (sisi atau busur) tidak memiliki arah yang ditentukan. Dengan kata lain, koneksi antar simpul dalam graf tak berarah bersifat timbal balik. Jika ada edge yang menghubungkan simpul A dan simpul B, maka hubungan tersebut dianggap berjalan ke dua arah: dari A ke B dan sebaliknya dari B ke A. Dalam graf tak berarah, tidak ada indikasi arah pada edge, dan edge hanya mencerminkan keterkaitan antar simpul.

Gambar 1. Ilustrasi Graf tak berarah
Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf, diakses pada 02/12/2023

b. Graf berarah (directed graph atau digraph)

Graf berarah ialah suatu jenis graf di mana setiap edge memiliki arah yang jelas. Artinya, relasi antara simpul-simpul dalam graf berarah memiliki orientasi tertentu, yakni dari simpul awal menuju simpul tujuan. Pada graf

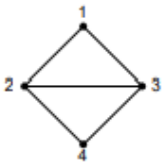
berarah, edge ditandai dengan panah yang menunjukkan arah hubungan.



Gambar 2. Ilustrasi Graf berarah
Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf, diakses pada 02/12/2023

A.2. Simpul (Node)

Simpul, dalam konteks graf, merujuk kepada titik atau entitas individu yang digunakan untuk menggambarkan objek atau elemen dalam suatu graf. Seringkali, istilah "node" juga digunakan untuk merujuk pada simpul. Setiap simpul memiliki kemampuan untuk menyimpan atribut atau informasi tambahan yang terkait dengan entitas yang diwakilinya.

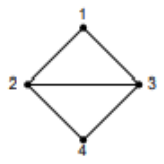


Gambar 3. Graf sederhana
Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf, diakses pada 02/12/2023

Dalam graf ini, “1, 2, 3, 4” merupakan simpul. Dalam berbagai aplikasi, simpul dapat mewakili berbagai hal, seperti lokasi, orang, konsep, maupun objek lainnya.

A.3. Sisi (Edge)

Sisi atau busur dalam konteks graf adalah hubungan antara dua simpul. Edge digunakan untuk mengilustrasikan keterkaitan atau hubungan antara elemen yang diwakili oleh simpul-simpul tersebut. Pada graf tak berarah, edge tidak memiliki arah tertentu, sehingga dapat diinterpretasikan sebagai koneksi dua arah antara dua simpul. Sebaliknya, pada graf berarah, edge memiliki arah yang mengindikasikan urutan dari simpul awal menuju simpul tujuan.

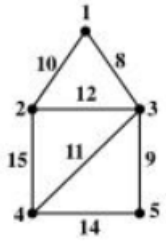


Gambar 3. Graf sederhana
Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf, diakses pada 02/12/2023

Dalam graf ini, garis menghubungkan simpul-simpul tersebut. Sebagai contoh, edge yang menghubungkan 1 dan 2 dapat dianggap sebagai koneksi antara simpul 1 dan simpul 2. Dalam graf tak berarah, hubungan antar simpul melalui sisi bersifat saling mengikat, sementara dalam graf berarah, sisi memiliki arah spesifik.

A.4. Graf Berbobot

Graf berbobot adalah suatu jenis graf di mana setiap edge (sisi atau busur) dilengkapi dengan nilai numerik yang disebut bobot atau weight. Bobot ini mencerminkan karakteristik atau informasi tambahan tentang hubungan antar simpul yang dihubungkan oleh edge tersebut. Bobot dapat mewakili berbagai konsep tergantung pada konteks aplikasinya, seperti jarak antara dua lokasi, biaya transportasi, kecepatan koneksi, atau nilai lainnya yang relevan.

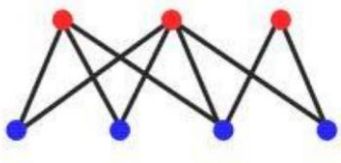


Gambar 4. Graf berbobot
Sumber:https://thumbs.slideserve.com/1_5650084.jpg

Dalam graf ini, angka yang berada di Tengah-tengah antara satu simpul dengan simpul lain menunjukan bobot dari masing-masing sisi. Bobot ini sendiri memberikan dimensi tambahan agar pemodelan bisa lebih realistis dan kompleks dalam berbagai aplikasi.

A.5. Graf Bipartit

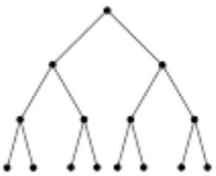
Graf bipartit adalah jenis graf yang memiliki dua himpunan simpul, di mana setiap sisi (edge) menghubungkan simpul dari himpunan yang berbeda. Dengan kata lain, tidak ada sisi yang menghubungkan dua simpul dalam himpunan yang sama. Himpunan simpul ini sering disebut sebagai "partisi."



Gambar 5. Graf Bipartit
Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian3-2023.pdf, diakses pada 09/12/2023

A.6. Graf Pohon

Graf pohon adalah jenis graf khusus yang membentuk struktur pohon, di mana setiap simpul (kecuali satu, yang disebut sebagai simpul akar) memiliki tepat satu simpul induk dan simpul tersebut terhubung oleh sisi arah. Setiap simpul yang terhubung secara langsung ke simpul induk disebut sebagai anak, dan simpul induknya disebut sebagai orang tua.



Gambar 6. Graf Pohon
Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Pohon-Bagian1-2023.pdf, diakses pada 09/12/2023

A.7. Multigraf

Multigraf adalah jenis graf di mana satu atau lebih pasangan simpul dapat terhubung oleh lebih dari satu sisi. Dengan kata lain, dalam multigraf, ada kemungkinan untuk memiliki beberapa sisi yang menghubungkan dua simpul yang sama.

B. Teori Spotify Wrapper

B.1. Definisi Spotify Wrapper

Spotify Wrapped, sebuah fitur tahunan dari layanan streaming musik Spotify yang pertama kali diperkenalkan pada tahun 2016, menjadi sorotan setiap akhir tahun bagi jutaan pengguna setia platform ini. Fitur ini merupakan bagian dari kampanye tahunan yang digelar pada bulan Desember, menawarkan pemetaan komprehensif kepada pengguna gratis dan premium mengenai kebiasaan mendengarkan mereka selama 12 bulan terakhir. Spotify Wrapped menggabungkan data mengenai artis yang paling sering didengarkan, lagu favorit, dominasi genre, dan bahkan mengajak pengguna untuk mengikuti kuis menyenangkan yang mengidentifikasi kepribadian pendengar.

Dengan cara ini, fitur ini tidak hanya memberikan hiburan semata, tetapi juga memberikan kesempatan kepada pengguna untuk merayakan dan berbagi cinta mereka pada dunia musik. Melalui hasil Spotify Wrapped, pengguna dapat dengan mudah membagikan preferensi musik mereka di platform media sosial, menciptakan pengalaman berbagi yang interaktif. Lebih dari sekadar alat untuk berbagi, fitur ini juga berfungsi sebagai sarana eksplorasi musik yang efektif, membantu pengguna menemukan artis dan genre baru yang mungkin sesuai dengan selera mereka.

B.2. API (Application Programming Interface)

API, singkatan dari Application Programming Interface, mengacu pada seperangkat panduan dan alat yang memudahkan interaksi antara aplikasi perangkat lunak. Pengembang dapat menggunakan API untuk mendapatkan akses ke fungsionalitas atau layanan yang disediakan oleh suatu sistem atau aplikasi tanpa harus terlibat dalam detail internalnya.

Sebuah API bisa berupa antarmuka perangkat lunak yang mengatur cara komunikasi antara dua komponen perangkat lunak. Pengembang memanfaatkan API untuk mengontrol atau menggunakan fitur tertentu dari aplikasi atau sistem eksternal.

B.3. Web Scraping

Web scraping adalah teknik pengumpulan data otomatis dari halaman web, di mana informasi diekstraksi secara otomatis dari struktur HTML atau XML menggunakan program atau skrip. Proses ini memberikan fleksibilitas kepada pengguna untuk mengambil data yang diinginkan dari beragam situs web, dengan tujuan penggunaan yang mencakup analisis data, pemantauan harga, riset pasar, atau pengumpulan informasi lainnya.

B.4. OAuth (Open Authorization)

OAuth, singkatan dari Open Authorization, adalah protokol otentikasi terbuka yang memungkinkan aplikasi pihak ketiga untuk mendapatkan akses terbatas ke sumber daya pengguna di suatu situs web tanpa perlu mengekspos kredensial login pengguna. Protokol ini didesain untuk memberikan otorisasi yang aman dan terbatas melalui protokol HTTP.

III. Penggunaan Graf pada Spotify Wrapped

Spotify Wrapped telah menjadi rujukan utama dalam menerapkan sistem rekomendasi yang canggih, menggabungkan teknik seperti Content-Based, Collaborative Filtering, dan unsur teori graf. Hal ini bertujuan memberikan pengalaman yang sangat pribadi dan mendalam kepada pengguna. Dalam kerangka Spotify Wrapped, setiap elemen musik seperti lagu, album, atau artis diwakili sebagai simpul dalam graf. Sementara itu, interaksi pengguna, seperti jumlah putaran dan waktu mendengarkan, diwakili sebagai sisi dengan bobot yang mencerminkan signifikansi lagu.

Pendekatan Collaborative Filtering memungkinkan sistem menganalisis pola interaksi pengguna untuk mengidentifikasi kesamaan preferensi antar pengguna. Ini berarti Spotify Wrapped dapat memberikan rekomendasi lagu berdasarkan pengalaman pengguna serupa. Teori graf menjadi pondasi kunci dalam membentuk struktur terorganisir Spotify Wrapped, memudahkan visualisasi dan analisis data yang kompleks. Analisis pola interaksi ini menghasilkan pengalaman pengguna yang personal dan reflektif, dengan statistik, infografis, dan rekomendasi lagu yang disesuaikan dengan pola unik setiap pengguna.

Dengan pendekatan ini, Spotify Wrapped tidak hanya menyajikan daftar lagu populer secara umum, melainkan berhasil memahami dan menganalisis preferensi musik individual pengguna. Spotify Wrapped menciptakan pengalaman mendalam yang memanfaatkan kekuatan sistem rekomendasi dan teori graf. Hasilnya adalah pengalaman yang kaya dan bermakna, memungkinkan pengguna merenungkan perjalanan musik mereka selama periode tertentu. Melalui integrasi teknologi canggih dan konsep teori graf, Spotify Wrapped berhasil menciptakan pengalaman pengguna yang unik dan berharga dalam konteks layanan streaming musik.

IV. Metodologi Penelitian

A. Content Based Filtering

Langkah awal dari algoritma Content-Based Filtering pada Spotify Wrapped melibatkan pengambilan fitur yang signifikan dari lagu-lagu yang telah diakses oleh pengguna. Fitur-fitur tersebut mencakup elemen seperti genre, artis, tempo, dan mungkin kata kunci atau lirik. Selanjutnya, pembentukan profil pengguna dilakukan dengan memperhitungkan preferensi musik sebelumnya, mengintegrasikan fitur-fitur tersebut dengan bobot yang mencerminkan tingkat preferensi. Proses selanjutnya melibatkan perbandingan antara item musik baru dengan profil pengguna, diukur berdasarkan seberapa cocok fitur-fitur tersebut. Hasilnya, item-item yang paling sesuai diurutkan dan direkomendasikan kepada pengguna sebagai bagian integral dari pengalaman Spotify Wrapped.

Rumus :

Cosine Similarity(A, B) = $\frac{A \cdot B}{||A|| \times ||B||}$

Di mana :

- A· B adalah hasil perkalian titik vektor A dan B.
- $||A|| dan ||B||$ adalah borma Euclidean dari vektor A dan B

B. Item-Based Collaborative Filtering

Item-Based Collaborative Filtering di Spotify merupakan pendekatan rekomendasi yang bergantung pada kemiripan antar lagu. Dalam proses ini, Spotify menggunakan data perilaku mendengarkan pengguna untuk membentuk profil musik dan mengekstrak fitur-fitur dari setiap lagu. Matriks kemiripan antar item digunakan untuk mengukur sejauh mana lagu-lagu memiliki karakteristik serupa. Saat pengguna mendengarkan atau menyukai lagu, Spotify memanfaatkan matriks tersebut untuk mengidentifikasi lagu-lagu lain dengan kemiripan tinggi, yang selanjutnya diurutkan berdasarkan tingkat kesamaan. Hasilnya adalah rekomendasi lagu yang lebih pribadi dan sesuai dengan preferensi musik unik masing-masing pengguna, membantu mereka menemukan lagu baru yang mungkin disukai.

Rumus :

Similarity(i, j) = $$\frac{\sum u \in U_{ij} r_{ui} \cdot r_{uj}}{\sqrt{\sum u \in U_{ij} r_{ui}^2} \cdot \sqrt{\sum u \in U_{ij} r_{uj}^2}}$$

Di mana :

- i dan j adalah dua item yang dibandingkan
- U_{ij} adalah himpunan pengguna yang memberikan rating pada kedua item
- r_{ui} dan r_{uj} adalah peringkat item i dan j oleh masing-masing pengguna u.

C. Graph

Dalam konteks ini, setiap baris matriks mewakili satu pengguna, sedangkan setiap kolom mewakili satu lagu. Angka "1" yang ditempatkan pada perpotongan antara baris dan kolom tertentu menunjukkan bahwa pengguna tersebut mendengarkan lagu tersebut, sementara angka "0" menandakan bahwa pengguna tersebut tidak mendengarkan lagu tersebut.

Pembentukan matriks biner ini didasarkan pada data kehadiran atau ketiadaan suatu elemen, yaitu lagu, dalam kategori mendengarkan setiap pengguna. Matriks ini memberikan representasi visual yang efektif untuk melihat pola-pola preferensi mendengarkan secara menyeluruh di antara kelompok pengguna yang diberikan. Dengan demikian, matriks biner tersebut menjadi dasar untuk analisis lebih lanjut, seperti pembuatan grafik atau metode visualisasi lainnya, untuk memahami dan menggambarkan dinamika kehadiran lagu dalam konteks preferensi mendengarkan pengguna.

User /	Song A	Song B	Song C	Song D	Song E

Pengguna					
User 1	1 (5)	1 (3)	1 (2)	1 (4)	0
User 2	1 (4)	1 (3)	1 (2)	0	0
User 3	0	1 (3)	1 (2)	0	1 (1)
User 4	1 (3)	0	1 (4)	1 (2)	0
User 5	0	1 (4)	1 (3)	1 (1)	0
User 6	1 (2)	0	1 (3)	1 (4)	0
User 7	0	1 (5)	0	1 (3)	1 (1)
User 8	1 (4)	1 (2)	1 (3)	0	0
User 9	0	0	1 (4)	1 (2)	0
User 10	1 (3)	0	1 (2)	1 (4)	0

Tabel Graf antara Lagu dan User.

User / Pengguna	Genre A	Genre B	Genre C	Genre D	Genre E
User 1	1 (3)	0	1 (4)	1 (2)	0
User 2	1 (1)	1 (5)	1 (3)	0	0
User 3	0	1 (4)	1 (2)	1 (1)	1 (5)
User 4	1 (5)	0	1 (2)	0	0
User 5	0	1 (3)	0	1 (4)	0
User 6	1 (2)	1 (1)	0	0	0
User 7	0	1 (4)	0	1 (3)	1 (2)
User 8	1 (1)	0	1 (4)	0	0
User 9	0	0	1 (3)	0	0
User 10	1 (5)	0	1 (1)	1 (2)	1 (4)

Tabel Graf antara Genre dan User

User / Pengguna	Artist 1	Artist 2	Artist 3	Artist 4	Artist 5
User 1	1 (5)	1 (2)	0	1 (3)	0
User 2	1 (3)	0	1 (1)	0	0
User 3	1 (2)	1 (5)	1 (4)	0	1 (1)
User 4	1 (2)	0	0	1 (5)	0
User 5	0	1 (4)	0	1 (1)	0
User 6	1 (1)	0	1 (3)	1 (2)	1 (4)
User 7	0	1 (3)	0	1 (2)	1 (5)
User 8	1 (4)	1 (1)	1 (5)	1 (2)	0
User 9	0	0	1 (4)	0	0
User 10	1 (1)	0	1 (2)	0	1 (3)

Tabel Graf antara Artist dan User

PERHITUNGAN :

Contoh data yang diambil dari user 1 dan user 2

Content Based Similarity :

1. Pilihan lagu yang didengarkan kedua pengguna :
   Lagu A, Lagu B, Lagu C, Lagu D, Lagu E

2. Genre lagu :
   Missal :
   - Lagu A = pop
   - Lagu B = rock
   - Lagu C = jazz
   - Lagu D = klasik
   - Lagu E = R&B

3. Perhitungan
   Dari data tabel graf diatas bisa dibuat list seperti :
   - User 1 = [1,1,1,1,0]
   - User 2 = [1,1,1,0,0]

$$\text{Cosine similarity} = \frac{1.1+1.1+1.1+1.0+0.0}{\sqrt{1^2+1^2+1^2+1^2+0^2} \times \sqrt{1^2+1^2+1^2+0^2+0^2}}$$
$$= \frac{3}{2 \times \sqrt{3}} = 0.577$$

Maka kesamaan dalam pemilihan genre lagu antara user 1 dan user 2 adalah 0.577

Item-Based Collaborative Filtering :

1. Pilihan lagu yang didengarkan kedua pengguna :
   Lagu A, Lagu B, Lagu C

2. Matriks kesamaan lagu :
   Gunakan Jaccard Similarity(A, B) = $\frac{Jumlah\ elemen\ bersama}{jumlah\ elemen\ unik\ di\ A\ dan\ B}$
   - Similarity(A, B) = $\frac{1}{2}$
   - Similarity(A, C) = $\frac{1}{2}$
   - Similarity(B, C) = 0

3. Perhitungan kesamaan user:
   Similarity(User 1, User 2)
   $= \frac{Similarity(A,B)+Similarity(A,C)+Similiarity(B,C)}{3}$
   $= \frac{\frac{1}{2}+\frac{1}{2}+0}{3}$
   $= \frac{1}{3}$

Maka dalam hal lagu yang didengarkan, user 1 dan user 2 memiliki similaritas $\frac{1}{3}$.

Hal ini dilakukan dengan semua user dari 1 – 10 dan masing-masing akan mendapatkan hasil similaritas yang berbeda mulai dari perbandingan genre, dan lainnya. Dimana total user akan menjadi total node dalam graf dan setiap sisi memiliki bobot yaitu similaritas antar pengguna/user. Dan ketebalan sisi mempresentasikan seberapa besar kesamaan antar pengguna.

Dengan menggunakan :

```python
# Import libraries
import networkx as nx
import matplotlib.pyplot as plt

# Function to create the song-user graph with weights
def create_song_user_graph_with_weights_song(user_data):
    G = nx.Graph()

    # Add user nodes
    G.add_nodes_from(user_data['Users'], bipartite=0)

    # Add song nodes
    G.add_nodes_from(user_data['Songs'], bipartite=1)

    # Add edges with weights representing the number of times a user played a song
    for user, songs in user_data['Listening History'].items():
        for song, play_count in songs.items():
            G.add_edge(user, song, weight=play_count)

    return G

# Function to create the song-user graph with weights
def create_genre_user_graph_with_weights_song(user_data):
    G = nx.Graph()

    # Add user nodes
    G.add_nodes_from(user_data['Users'], bipartite=0)

    # Add song nodes
    G.add_nodes_from(user_data['Genres'], bipartite=1)

    # Add edges with weights representing the number of times a user played a song
    for user, songs in user_data['Listening History'].items():
        for song, play_count in songs.items():
            G.add_edge(user, song, weight=play_count)

    return G
```

```python
# Function to create the song-user graph with weights
def create_artist_user_graph_with_weights_song(user_data):
    G = nx.Graph()

    # Add user nodes
    G.add_nodes_from(user_data['Users'], bipartite=0)

    # Add song nodes
    G.add_nodes_from(user_data['Artists'], bipartite=1)

    # Add edges with weights representing the number of times a user played a song
    for user, songs in user_data['Listening History'].items():
        for song, play_count in songs.items():
            G.add_edge(user, song, weight=play_count)

    return G

# Function to analyze the similarity between users (without using weights)
def user_similarity_analysis(G, user1, user2):
    common_songs = set(G.neighbors(user1)).intersection(set(G.neighbors(user2)))
    total_songs = set(G.neighbors(user1)).union(set(G.neighbors(user2)))

    similarity = len(common_songs) / len(total_songs)

    return similarity
```

```python
# Function to calculate and print the most played song by all users with weights
def most_played_song_by_all_users_weighted(G, user_data):
    song_count = {}

    # Iterate over all users
    for user in user_data['Users']:
        # Get the songs listened by the user with weights
        user_songs = {song: G[user][song]['weight'] for song in G.neighbors(user)}

        # Update the song count
        for song, weight in user_songs.items():
            song_count[song] = song_count.get(song, 0) + weight

    # Find the song with the highest total weight
    most_played_song = max(song_count, key=song_count.get)

    print(f"The most played song by all users is: {most_played_song} (played {song_count[most_played_song]} times)")

# Function to calculate and print the most played genre by all users with weights
def most_played_genre_by_all_users_weighted(G, user_data):
    genre_count = {}

    # Iterate over all users
    for user in user_data['Users']:
        # Get the genres listened by the user with weights
        user_genres = {song: G[user][song]['weight'] for song in G.neighbors(user)}

        # Update the genre count
        for genre, weight in user_genres.items():
            genre_count[genre] = genre_count.get(genre, 0) + weight

    # Find the genre with the highest total weight
    most_played_genre = max(genre_count, key=genre_count.get)

    print(f"The most played genre by all users is: {most_played_genre} (played {genre_count[most_played_genre]} times)")
```

```python
# Function to calculate and print the most listened artist by all users with weights
def most_listened_artist_by_all_users_weighted(G, user_data):
    artist_count = {}

    # Iterate over all users
    for user in user_data['Users']:
        # Get the artists listened by the user with weights
        user_artists = {artist: G[user][artist]['weight'] for artist in G.neighbors(user)}

        # Update the artist count
        for artist, weight in user_artists.items():
            artist_count[artist] = artist_count.get(artist, 0) + weight

    # Find the artist with the highest total weight
    most_played_artist = max(artist_count, key=artist_count.get)

    print(f"The most listened artist by all users is: {most_played_artist} (played {artist_count[most_played_artist]} times)")

# Function to print the most listened-to songs by each user
def most_listened_songs_by_user(G, user_data, user):
    user_songs = {song: G[user][song]['weight'] for song in G.neighbors(user)}
    most_listened_song = max(user_songs, key=user_songs.get)
    play_count = user_songs[most_listened_song]
    print(f"Most listened-to song by {user}: {most_listened_song} (played {play_count} times)")

# Function to print the most listened-to genres by each user
def most_listened_genres_by_user(G, user_data, user):
    user_genres = {genre: G[user][genre]['weight'] for genre in G.neighbors(user)}
    most_listened_genre = max(user_genres, key=user_genres.get)
    play_count = user_genres[most_listened_genre]
    print(f"Most listened-to genre by {user}: {most_listened_genre} (played {play_count} times)")

# Function to print the most listened-to artists by each user
def most_listened_artists_by_user(G, user_data, user):
    user_artists = {artist: G[user][artist]['weight'] for artist in G.neighbors(user)}
    most_listened_artist = max(user_artists, key=user_artists.get)
    play_count = user_artists[most_listened_artist]
    print(f"Most listened-to artist by {user}: {most_listened_artist} (played {play_count} times)")
```

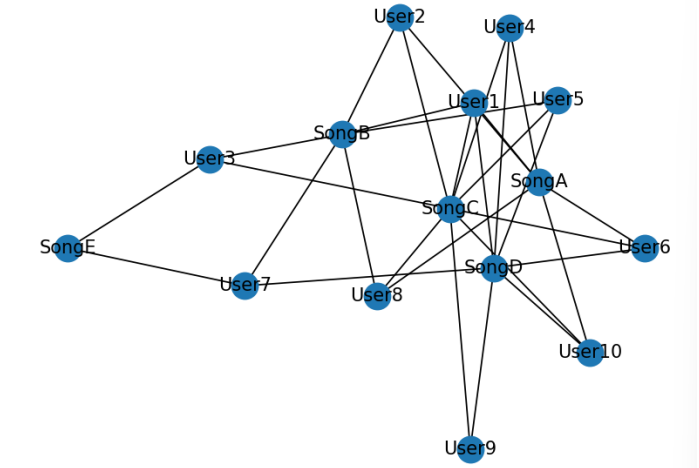
```python
# User data song with weights
user_data_with_weights_song = {
    'Users': ['User1', 'User2', 'User3', 'User4', 'User5', 'User6', 'User7', 'User8', 'User9', 'User10'],
    'Songs': ['SongA', 'SongB', 'SongC', 'SongD', 'SongE'],
    'Listening History': {
        'User1': {'SongA': 5, 'SongB': 3, 'SongC': 2, 'SongD': 4},
        'User2': {'SongA': 4, 'SongB': 3, 'SongC': 2},
        'User3': {'SongB': 3, 'SongC': 2, 'SongE': 1},
        'User4': {'SongA': 3, 'SongC': 4, 'SongD': 2},
        'User5': {'SongB': 4, 'SongC': 3, 'SongD': 1},
        'User6': {'SongA': 2, 'SongC': 3, 'SongD': 4},
        'User7': {'SongB': 5, 'SongD': 3, 'SongE': 1},
        'User8': {'SongA': 4, 'SongB': 2, 'SongC': 3},
        'User9': {'SongC': 4, 'SongD': 2},
        'User10': {'SongA': 3, 'SongC': 2, 'SongD': 4}
    }
}

# User data genre with weights
user_data_with_weights_genre = {
    'Users': ['User1', 'User2', 'User3', 'User4', 'User5', 'User6', 'User7', 'User8', 'User9', 'User10'],
    'Genres': ['GenreA', 'GenreB', 'GenreC', 'GenreD', 'GenreE'],
    'Listening History': {
        'User1': {'GenreA': 3, 'GenreC': 4, 'GenreD': 2},
        'User2': {'GenreA': 1, 'GenreB': 5, 'GenreC': 1},
        'User3': {'GenreB': 4, 'GenreC': 2, 'GenreD': 1, 'GenreE': 5},
        'User4': {'GenreA': 5, 'GenreC': 2},
        'User5': {'GenreB': 3, 'GenreD': 4},
        'User6': {'GenreA': 2, 'GenreB': 1},
        'User7': {'GenreB': 4, 'GenreD': 3, 'GenreE': 2},
        'User8': {'GenreA': 1, 'GenreC': 4},
        'User9': {'GenreC': 3},
        'User10': {'GenreA': 5, 'GenreC': 1, 'GenreD': 2, 'GenreE': 4}
    }
}
```

```python
# User data artist with weights
user_data_with_weights_artist = {
    'Users': ['User1', 'User2', 'User3', 'User4', 'User5', 'User6', 'User7', 'User8', 'User9', 'User10'],
    'Artists': ['Artist1', 'Artist2', 'Artist3', 'Artist4', 'Artist5'],
    'Listening History': {
        'User1': {'Artist1': 5, 'Artist2': 2, 'Artist4': 3},
        'User2': {'Artist1': 3, 'Artist3': 1},
        'User3': {'Artist1': 2, 'Artist2': 5, 'Artist3': 4, 'Artist5': 1},
        'User4': {'Artist1': 2, 'Artist4': 5},
        'User5': {'Artist2': 4, 'Artist4': 1},
        'User6': {'Artist1': 1, 'Artist3': 3, 'Artist4': 2, 'Artist5': 4},
        'User7': {'Artist2': 3, 'Artist4': 2, 'Artist5': 5},
        'User8': {'Artist1': 4, 'Artist2': 1, 'Artist3': 5, 'Artist4': 2},
        'User9': {'Artist3': 4},
        'User10': {'Artist1': 1, 'Artist3': 2, 'Artist5': 3}
    }
}

# Create the graph with weights
song_user_graph_with_weights_song = create_song_user_graph_with_weights_song(user_data_with_weights_song)
genre_user_graph_with_weights_song = create_genre_user_graph_with_weights_song(user_data_with_weights_genre)
artist_user_graph_with_weights_song = create_artist_user_graph_with_weights_song(user_data_with_weights_artist)

# Visualize the graph
nx.draw(song_user_graph_with_weights_song, with_labels=True)
plt.show()
nx.draw(genre_user_graph_with_weights_song, with_labels=True)
plt.show()
nx.draw(artist_user_graph_with_weights_song, with_labels=True)
plt.show()
```

```
# Analyze the similarity between users (without using weights)
for i in range(2, 11):
    similarity = user_similarity_analysis(song_user_graph_with_weights_song, 'User1', f'User{i}')
    print(f"Song similarity between User1 and User{i}: {similarity}")

most_played_song_by_all_users_weighted(song_user_graph_with_weights_song, user_data_with_weights_song)
most_listened_songs_by_user(song_user_graph_with_weights_song, user_data_with_weights_song, 'User1')

for i in range(2, 11):
    similarity = user_similarity_analysis(genre_user_graph_with_weights_song, 'User1', f'User{i}')
    print(f"Genre similarity between User1 and User{i}: {similarity}")

most_played_genre_by_all_users_weighted(genre_user_graph_with_weights_song, user_data_with_weights_genre)
most_listened_genres_by_user(genre_user_graph_with_weights_song, user_data_with_weights_genre, 'User1')

for i in range(2, 11):
    similarity = user_similarity_analysis(artist_user_graph_with_weights_song, 'User1', f'User{i}')
    print(f"Artist similarity between User1 and User{i}: {similarity}")

most_listened_artist_by_all_users_weighted(artist_user_graph_with_weights_song, user_data_with_weights_artist)
most_listened_artists_by_user(artist_user_graph_with_weights_song, user_data_with_weights_artist, 'User1')
```

Gambar 7. Algoritma membentuk graf, perhitungan similarity, most song/genre/artist, most listened-to song/genre/artist

Graf Lagu dan User :

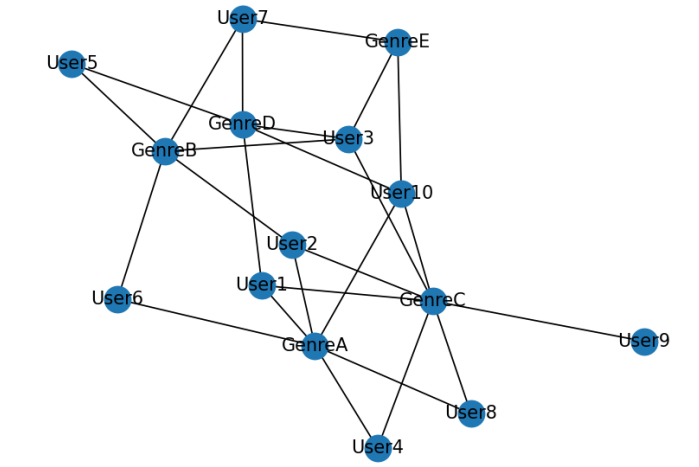


Gambar 8. Hasil Graf User dan Song

```
Song similarity between User1 and User2: 0.75
Song similarity between User1 and User3: 0.4
Song similarity between User1 and User4: 0.75
Song similarity between User1 and User5: 0.75
Song similarity between User1 and User6: 0.75
Song similarity between User1 and User7: 0.4
Song similarity between User1 and User8: 0.75
Song similarity between User1 and User9: 0.5
Song similarity between User1 and User10: 0.75
The most played song by all users is: SongC (played 25 times)
Most listened-to song by User1: SongA (played 5 times)
```

Gambar 9. Hasil song similarity User1 dan User lainnya ,serta most played song dan most listened-to song

Graf Genre dan User :

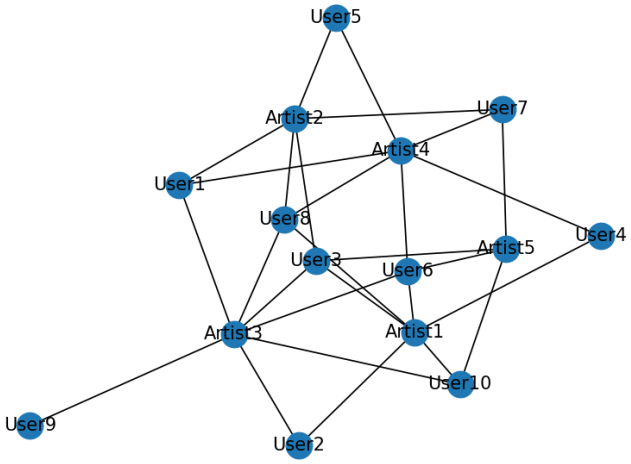


Gambar 10. Hasil Graf User dan Genre

```
Genre similarity between User1 and User2: 0.5
Genre similarity between User1 and User3: 0.4
Genre similarity between User1 and User4: 0.6666666666666666
Genre similarity between User1 and User5: 0.25
Genre similarity between User1 and User6: 0.25
Genre similarity between User1 and User7: 0.2
Genre similarity between User1 and User8: 0.6666666666666666
Genre similarity between User1 and User9: 0.3333333333333333
Genre similarity between User1 and User10: 0.75
The most played genre by all users is: GenreC (played 19 times)
Most listened-to genre by User1: GenreC (played 4 times)
```

Gambar 11. Hasil genre similarity User1 dan User lainnya ,serta most played genre dan most listened-to genre

Graf Artist dan User :



Gambar 12. Hasil Graf User dan Artist

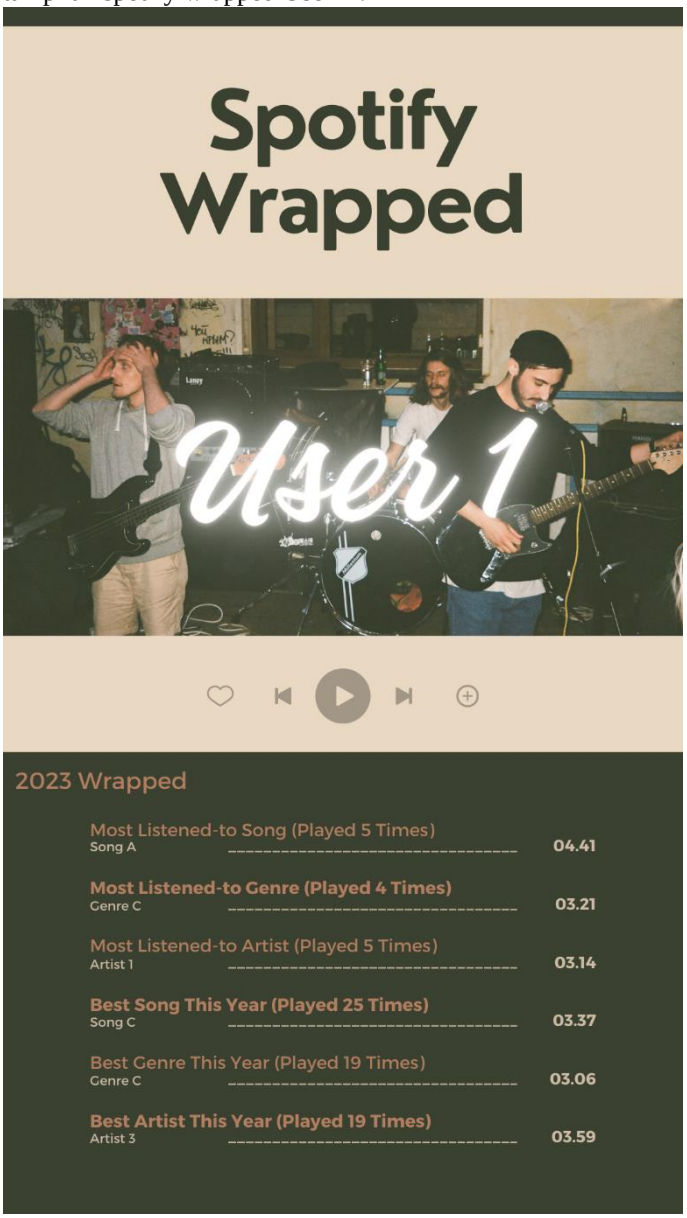
```
Artist similarity between User1 and User2: 0.25
Artist similarity between User1 and User3: 0.4
Artist similarity between User1 and User4: 0.6666666666666666
Artist similarity between User1 and User5: 0.6666666666666666
Artist similarity between User1 and User6: 0.4
Artist similarity between User1 and User7: 0.5
Artist similarity between User1 and User8: 0.75
Artist similarity between User1 and User9: 0.0
Artist similarity between User1 and User10: 0.2
The most listened artist by all users is: Artist3 (played 19 times)
Most listened-to artist by User1: Artist1 (played 5 times)
```

Gambar 13. Hasil artist similarity User1 dan User lainnya ,serta most listened artist dan most listened-to artist

Dalam spotify wrapped nantinya setiap pengguna akan diberi informasi tentang artis favorit yang pengguna dengarkan, lagu favorit yang pengguna dengarkan, popularitas lagu yang didengarkan, popularitas genre lagu, dan lainnya. Semua aspek itu dapat ditampilkan dengan membaca graf yang sudah dibuat beserta perhitungannya, dan pastinya spotify wrapped setiap pengguna akan berbeda dengan pengguna lain.

Dalam kasus ini saya hanya mengambil contoh User 1, Dimana pada spotify wrapped User 1 akan menampilkan lagu-lagu dan genre-genre yang memiliki nilai similarity terbesar dengan User 1 dan juga akan menampilkan lagu yang paling sering diputar semua User, Genre yang paling sering diputar semua User, dan Artist yang paling sering didengarkan semua User.

Dari hasil perhitungan dan implementasi graf maka tampilan spotify wrapped User 1 :



Gambar 14. Hasil Ilustrasi Spotify Wrapped User 1

V. KESIMPULAN

Penelitian ini menunjukkan bahwa mengintegrasikan teori graf dan wrapper dalam lingkungan Spotify dapat memberikan dampak positif pada kemampuan platform dalam memberikan rekomendasi lagu yang lebih cerdas dan pribadi. Melalui penerapan pendekatan Content-Based dan Item-Based Collaborative Filtering, Spotify Wrapped mampu menghasilkan pengalaman pengguna yang unik dan bermakna.

Pengguna dapat mengalami rekomendasi lagu yang lebih sesuai dengan preferensi pribadi mereka, sementara hasil dari Spotify Wrapped memberikan pemahaman yang lebih mendalam tentang preferensi musik individual. Oleh karena itu, penelitian ini diharapkan dapat menjadi solusi inovatif untuk meningkatkan kepuasan pengguna, menciptakan pengalaman mendengarkan yang lebih mendalam, dan mengatasi tantangan yang dihadapi dalam mengonsumsi musik digital pada era layanan streaming.

REFERENCES

[1] Anonim, “Discrete Mathematics : Definition and Types of Graphs”, 2021. Tersedia pada https://www.belajarstatistik.com/blog/2021/10/02/definisi-dan-jenis-jenis-graf/, Diakses pada 10 desember 2023.

[2] Munir, Rinaldi, “IF2120 Matematika Diskrit – Semester I Tahun 2023/2024”, 2023. Tersedia pada https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/matdis23-24.htm, Diakses pada 10 desember 2023.

[3] Sania, “Apa itu Spotify Wrapped? Ini Cara Membuatnya”, 2023. Tersedia pada: https://www.rukita.co/stories/spotify-wrapped/, Diakses pada 10 desember 2023.

[4] Lawrence, Alexandromeo, “API: Pengertian, Fungsi, dan Cara Kerjanya”, 2020. Terssedia pada https://www.niagahoster.co.id/blog/api-adalah/, Diakses pada 10 desember 2023.

[5] Stanley, E. J. , “What is Web Scraping? A Complete Guide”, 2020. Tersedia pada https://www.fortra.com/resources/guides/what-is-web-scraping, Diakses pada 10 desember 2023.

[6] Cobb, Michael, Stephanie Mann, “OAuth”, 2020. Tersedia pada https://www.techtarget.com/searchapparchitecture/definition/OAuth, Diakses pada 10 desember 2023.

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, 10 Desember 2023

Nama dan NIM
Jason Fernando / 13522156