Aplikasi Graf Berbobot dengan Algoritma A-star untuk Optimasi Urutan Daftar Putar Musik Berbasis Multiparameter Dinamis

Dzaky Aurelia Fawwaz - 13523065[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523065@std.stei.itb.ac.id ,[2]dzakyaureliafawwaz@gmail.com

***Abstract*— Makalah ini menyajikan optimasi urutan playlist musik menggunakan algoritma A* (A-star) dan graf berbobot. Dengan memanfaatkan multiple parameter musik seperti tempo, energi, danceability, key, dan mode, sistem ini mampu menghasilkan transisi musik yang lebih harmonis. Implementasi menggunakan representasi graf dimana vertex merepresentasikan lagu dan edge merepresentasikan transisi antar lagu dengan bobot yang dihitung dari parameter musik yang telah dinormalisasi. Algoritma A* dimodifikasi dengan heuristik khusus untuk mencari urutan optimal dengan mempertimbangkan keseluruhan parameter secara dinamis. Hasil pengujian menunjukkan bahwa metode ini berhasil menghasilkan urutan playlist yang optimal dengan total cost 0.323 dan rata-rata cost transisi 0.108. Dari analisis parameter musik, tempo menjadi faktor yang paling berpengaruh dengan impact score 5.700, diikuti oleh perubahan key dengan impact 0.350. Perbandingan dengan path alternatif memvalidasi optimalitas solusi yang dihasilkan, dimana path optimal berhasil mendistribusikan perubahan parameter secara lebih seimbang untuk menghindari transisi yang terlalu drastis.**

***Keywords*— Algoritma A-star , graf berbobot , music playlist optimization, dynamic parameters**

I. PENDAHULUAN

Perkembangan teknologi streaming musik telah secara fundamental mengubah cara orang mengkonsumsi musik. Platform seperti Spotify dan YouTube Music menyediakan akses ke jutaan lagu, memungkinkan pengguna membuat dan mengelola playlist mereka sendiri. Pengalaman mendengarkan yang optimal tidak hanya bergantung pada pemilihan lagu, tetapi juga pada urutan pemutarannya yang dapat mempengaruhi keseluruhan pengalaman pendengar [1].

Dalam konteks platform streaming musik modern, urutan pemutaran lagu menjadi aspek penting dalam menciptakan pengalaman mendengarkan yang kohesif. Transisi antar lagu yang harmonis dapat dicapai dengan mempertimbangkan berbagai parameter musik seperti tempo, energi, *danceability*, dan tonalitas [2]. Platform seperti Spotify menggunakan audio features dalam algoritma rekomendasi mereka untuk menganalisis karakteristik musik ini dan menghasilkan playlist yang lebih personal [1].

Optimasi urutan playlist dapat dimodelkan sebagai permasalahan pencarian jalur optimal dalam graf berbobot. Setiap lagu direpresentasikan sebagai vertex dengan atribut berupa parameter musiknya, sedangkan edge merepresentasikan transisi antar lagu dengan bobot yang menggambarkan kualitas transisi tersebut. Algoritma A* (A-star) menjadi pilihan yang menjanjikan untuk permasalahan ini karena kemampuannya dalam menemukan solusi optimal dengan menggunakan fungsi heuristik yang dapat disesuaikan dengan domain musik.

Makalah ini mengusulkan pendekatan untuk mengoptimalkan urutan playlist menggunakan algoritma A* dengan mempertimbangkan multiple parameter musik secara dinamis. Metode yang diusulkan memungkinkan penyesuaian bobot parameter sesuai dengan preferensi pengguna atau konteks pemutaran tertentu. Berbeda dengan implementasi playlist automation yang ada, pendekatan ini memberikan fleksibilitas dalam pembobotan parameter dan memastikan optimalitas urutan secara global melalui penggunaan algoritma A*.

Kontribusi utama makalah ini meliputi model graf berbobot yang mempertimbangkan multiple parameter musik, implementasi algoritma A* dengan heuristik yang disesuaikan untuk domain musik, dan sistem pembobotan dinamis yang memungkinkan adaptasi terhadap berbagai preferensi.

Pembahasan dalam makalah ini diorganisir sebagai berikut. Bagian II membahas landasan teori tentang graf berbobot dan algoritma A*. Bagian III menyajikan detail implementasi sistem, termasuk struktur data dan algoritma yang digunakan. Bagian IV mendiskusikan hasil pengujian dan analisis performa, termasuk perbandingan berbagai path yang mungkin. Terakhir, Bagian V menyimpulkan makalah dan menyarankan arah pengembangan di masa depan.

II. LANDASAN TEORI

A. Graf

Graf didefinisikan sebagai sebuah pasangan (V,E) di mana V adalah himpunan tidak-kosong dari simpul-simpul (vertices atau nodes) dan E adalah himpunan sisi (edges and arcs) yang menghubungkan sepasang simpul [3]. Simpul pada suatu graf umumnya dinyatakan dengan huruf a, b, c, ..., dengan bilangan asli 1, 2, 3, ..., atau gabungan keduanya, sedangkan sisi yang menghubungkan simpul x dan simpul y dinyatakan sebagai pasangan (x,y) atau dengan lambang e1, e2, e3, dst [3]

Graf dapat digolongkan berdasarkan ada tidaknya gelang atau sisi ganda menjadi dua jenis utama. Graf sederhana merupakan

graf yang tidak mengandung gelang maupun sisi ganda, dimana sisi yang menghubungkan simpul x dan simpul y merupakan pasangan tak-terurut {x,y}. Sementara itu, graf tidak sederhana merupakan graf yang mengandung gelang, sisi ganda, atau keduanya. Graf tidak sederhana dapat dibagi lagi menjadi graf-ganda (*multigraph*) yang mengandung sisi ganda namun tidak mengandung gelang, dan graf semu (pseudograph) yang mengandung gelang (loop) [3].

Graf berbobot merupakan pengembangan konsep graf dimana setiap sisinya memiliki sebuah nilai yang disebut bobot (weight). Bobot suatu sisi dapat merepresentasikan kuat hubungan antara kedua simpul yang bersisian, jarak antara kedua simpul, dan sebagainya. Dalam konteks optimasi *playlist* musik yang dibahas dalam makalah ini, bobot *edge* merepresentasikan *cost* transisi antar lagu yang dihitung berdasarkan parameter-parameter musik seperti tempo, energi, *danceability*, *key,* dan *mode [2]*.

Beberapa konsep penting dalam teori graf yang relevan dengan implementasi optimasi playlist musik meliputi kebertetanggaan (adjacency) yang menunjukkan hubungan antar simpul yang terhubung langsung oleh sisi, lintasan (path) yang merupakan urutan simpul yang terhubung oleh sisi, cost lintasan yang merepresentasikan jumlah bobot sisi dalam suatu lintasan pada graf berbobot, dan lintasan optimal yang merupakan lintasan dengan total cost minimum [3]. Konsep-konsep ini menjadi dasar dalam implementasi dimana vertex merepresentasikan lagu, edge merepresentasikan transisi antar lagu, bobot edge merepresentasikan cost transisi, dan path optimal merepresentasikan urutan playlist yang optimal.

B. Parameter Musik Digital

Parameter musik digital merupakan atribut-atribut terukur yang dapat diekstrak dari file musik. Beberapa parameter kunci yang digunakan dalam optimasi playlist meliputi:

Tempo (BPM - Beats Per Minute) mengukur kecepatan lagu, umumnya berkisar antara 60-200 BPM. Perbedaan tempo yang signifikan antar lagu dapat menghasilkan transisi yang tidak mulus [2].

Energy merupakan parameter yang menggambarkan intensitas dan aktivitas dalam lagu, diukur dalam skala 0.0 hingga 1.0. Energy menggabungkan berbagai aspek seperti *dynamic range, perceived loudness, timbre, onset rate,* dan *general entropy [2]*.

Danceability mengindikasikan seberapa cocok lagu untuk menari, berdasarkan kombinasi elemen musik termasuk tempo, rhythm stability, beat strength, dan overall regularity.[4]

Musical key merepresentasikan tonalitas lagu dalam skala 0-11 (C=0, C#=1, dst). Transisi antar lagu dengan key yang harmonis (misalnya mengikuti circle of fifths) cenderung lebih enak didengar [2].

C. Algoritma A*

Algoritma A* adalah algoritma pencarian heuristik yang menggunakan fungsi evaluasi[5] ,dimana

$$f(n) = g(n) + h\,(n)$$

g(n) adalah biaya sebenarnya dari titik awal ke node n, dan h(n) adalah estimasi heuristik dari node n ke tujuan. Dalam konteks optimasi playlist, g(n) merepresentasikan total cost transisi yang sudah terjadi, sementara h(n) mengestimasi cost transisi yang tersisa [5].

Fungsi heuristik h(n) harus memenuhi sifat *admissible*, artinya tidak boleh overestimasi biaya sebenarnya ke tujuan. Pemilihan fungsi heuristik yang tepat sangat mempengaruhi efisiensi algoritma dalam menemukan urutan optimal. Implementasi A* dalam konteks musik menggunakan heuristik berdasarkan parameter musik yang telah dinormalisasi [5].

D. Sistem Rekomendasi Musik

Sistem rekomendasi musik modern menggunakan kombinasi pendekatan content-based dan collaborative filtering. Content-based filtering mengandalkan analisis parameter musik intrinsik, sementara collaborative filtering memanfaatkan data historis pendengar. Optimasi urutan playlist menggunakan graf dan A* termasuk dalam kategori *content-based recommendation*, dimana rekomendasi didasarkan pada karakteristik musik yang dapat diukur secara objektif [6].

III. Implementasi

A. Dataset Generatif

Implementasi optimasi playlist musik menggunakan dataset generatif yang dirancang untuk mencakup variasi parameter musik yang merepresentasikan berbagai genre dan karakteristik musik. Dataset ini dibuat dengan mempertimbangkan perbedaan parameter penting seperti tempo, energi, *danceability*, *key*, dan *mode* untuk menguji kehandalan algoritma.

Berikut contoh dataset yang digunakan :

```
songs = [
Song("1", "Bohemian Rhapsody", "Queen",
72, 0.9, 0.7, 0, 1, 354000),
Song("2", "Uptown Funk", "Mark Ronson",
115, 0.8, 0.9, 4, 1, 270000),
Song("3", "Sweet Child O' Mine", "Guns N'
Roses", 125, 0.7, 0.6, 7, 1, 356000),
Song("4", "Billie Jean", "Michael
Jackson", 117, 0.8, 0.9, 2, 1, 294000)
    ]
```

Dataset ini dirancang dengan mempertimbangkan:

1. Variasi tempo (72-128 BPM) untuk menguji transisi antar kecepatan
2. Range energi (0.4-0.9) untuk menguji perubahan intensitas
3. Perbedaan danceability (0.4-0.9) untuk variasi karakteristik
4. Kombinasi key berbeda untuk menguji transisi harmonik
5. Variasi mode (major/minor) untuk kelengkapan pengujian

B. Struktur Data Utama

Implementasi menggunakan dua struktur data utama untuk merepresentasikan komponen sistem. Pertama, class Song yang menyimpan informasi dan parameter setiap lagu. Setiap objek Song mengandung identifikasi unik lagu (id), informasi dasar

seperti judul dan artis, serta parameter musik seperti tempo dalam *beats* per minute (range 60-200), *energy level* dan *danceability* dalam *range* 0.0-1.0, *key* dalam representasi numerik 0-11 yang merepresentasikan nada dasar dari C hingga B, serta mode yang mengindikasikan tangga nada major (1) atau minor (0).

```
class Song:
  def__init__(self, id, title, artist,
tempo, energy, danceability, key, mode,
duration_ms):
  self.id = id
  self.title = title
  self.artist = artist
  self.tempo = tempo
  self.energy = energy
  self.danceability
  self.key = key
self.mode = mode
```

Struktur data kedua adalah class PlaylistNode yang berperan krusial dalam proses pencarian path optimal menggunakan algoritma A*. PlaylistNode merepresentasikan sebuah state dalam ruang pencarian, menyimpan urutan lagu yang telah dipilih dalam bentuk list serta lagu terakhir dalam urutan tersebut. Implementasi operator *less than* (**lt**) memungkinkan penggunaan PlaylistNode dalam priority queue, dimana urutan prioritas ditentukan berdasarkan panjang path saat ini.

```
class PlaylistNode:
   def__init__(self, songs: List[Song],
current_song: Song):
      self.songs = songs
      self.current_song = current_song

   def __lt__(self, other):
        return     len(self.songs)     <
len(other.songs)
```

Dalam sistem optimasi playlist, kedua struktur data ini bekerja secara terintegrasi. Song berperan sebagai bangunan dasar yang menyediakan semua informasi yang diperlukan untuk perhitungan transisi antar lagu, sementara PlaylistNode memfasilitasi pencarian A* dengan menyimpan dan mengelola state pencarian. Interaksi antara kedua struktur ini memungkinkan sistem untuk menghitung cost transisi antar lagu dan melakukan pencarian path optimal yang mempertimbangkan semua parameter musik yang relevan.

C. Algoritma A*

Algoritma A* diimplementasikan untuk menemukan urutan optimal playlist musik dengan mempertimbangkan berbagai parameter musik. Implementasi dimulai dengan mendefinisikan struktur data PlaylistNode yang merepresentasikan state dalam ruang pencarian. Setiap node menyimpan informasi tentang urutan lagu yang telah dipilih (path) dan lagu terakhir dalam urutan tersebut, yang akan digunakan untuk menghitung transisi ke lagu berikutnya.

Perhitungan *cost* transisi antara dua lagu merupakan komponen krusial dalam implementasi ini. *Cost* dihitung dengan mempertimbangkan lima parameter musik utama: tempo, energi, *danceability*, *key*, dan *mode*. Setiap parameter dinormalisasi ke dalam range 0-1 untuk memastikan perbandingan yang adil. Tempo dinormalisasi dengan membagi nilai BPM dengan 200, key dihitung menggunakan jarak terpendek dalam circle of fifths (dibagi 12), sedangkan energy dan danceability sudah dalam range yang sesuai. Transisi antara mode major dan minor dihitung sebagai perbedaan boolean.

```
def calculate_transition_cost(self,
song1: Song, song2: Song) -> float:
    costs = {
      'tempo':abs(song1.tempo
song2.tempo) / 200.0,
      'energy': abs(song1.energy -
            song2.energy),
      'danceability':
      abs(song1.danceability           -
      song2.danceability),
      'key': min((song1.key - song2.key)
           % 12,(song2.key -
        song1.key) % 12) / 12.0,
    'mode': abs(song1.mode - song2.mode)
                  }
 return  sum(self.weights[param]  *  cost
for param, cost in costs.items())
```

Fungsi heuristik dalam algoritma A* dirancang untuk memenuhi sifat *admissible* dengan mengestimasi *cost* minimum yang tersisa untuk mencapai *state* tujuan. Implementasi heuristik mencari *cost* transisi minimum dari lagu saat ini ke semua lagu yang tersisa, kemudian mengalikannya dengan jumlah lagu yang belum masuk dalam urutan. Pendekatan ini menjamin bahwa estimasi tidak akan melebihi *cost* sebenarnya.

```
def   heuristic(self,   current:   Song,
remaining: Set[Song]) -> float:
      if not remaining:
          return 0
      min_cost = float('inf')
      for song in remaining:
          cost                          =
self.calculate_transition_cost(current,
song)
          min_cost = min(min_cost, cost)
    return min_cost * len(remaining)
```

Proses pencarian *path* optimal menggunakan *priority queue* untuk mengelola open set, memastikan node dengan f-score terendah selalu dievaluasi terlebih dahulu. Setiap node dalam *open se*t menyimpan f-score yang merupakan kombinasi dari g-*score* (*cost path* sejauh ini) dan h-score (estimasi *cost* tersisa). Pencarian dilanjutkan hingga ditemukan urutan lengkap yang

mencakup semua lagu atau tidak ada lagi node yang dapat dieksplorasi.

```
def optimize_playlist(self, start_song: Song) -> List[Song]:
    open_set = []
    closed_set = set()
    start_node = PlaylistNode([start_song], start_song)
    heapq.heappush(open_set, (self.heuristic(start_song, remaining_songs), start_node))

    while open_set:
        current_f, current_node = heapq.heappop(open_set)
        if len(current_node.songs) == len(self.songs):
            return current_node.songs
```

Selama proses pencarian, algoritma menjaga track node yang sudah dieksplorasi menggunakan closed set untuk menghindari eksplorasi berulang. State yang sudah dievaluasi ditandai dengan kombinasi unik dari lagu saat ini dan urutan lagu sebelumnya, memastikan efisiensi pencarian tanpa kehilangan kemungkinan path optimal.

Implementasi ini memungkinkan pencarian urutan playlist yang optimal dengan mempertimbangkan kompleksitas transisi musik secara menyeluruh, sambil menjaga efisiensi komputasi melalui penggunaan struktur data dan fungsi heuristik yang tepat.

D. Evaluasi Lintasan

Analisis hasil optimasi playlist dilakukan melalui serangkaian evaluasi komprehensif terhadap lintasan yang dihasilkan. Implementasi analisis mencakup perhitungan statistik untuk setiap transisi dan keseluruhan lintasan, serta visualisasi hasil untuk memudahkan interpretasi. Fungsi analyze_path mengkalkulasi metrik-metrik penting seperti total *cost path*, *cost* per transisi, dan rata-rata cost keseluruhan.

```
def analyze_path(self, playlist: List[Song]) -> Dict:
    transitions = []
    total_cost = 0

    for i in range(len(playlist)-1):

        cost=self.calculate_transition_cost(playlist[i], playlist[i+1])
        total_cost += cost
        transitions.append({
          'from': playlist[i].title,
          'to': playlist[i+1].title,
          'cost': cost,
          'parameters': {

            'tempo_diff': abs(playlist[i].tempo - playlist[i+1].tempo),

            'key_distance': min((playlist[i].key - playlist[i+1].key) % 12,

            (playlist[i+1].key - playlist[i].key) % 12),

            'energy_change': playlist[i+1].energy - playlist[i].energy
                }
            })
```

Analisis setiap transisi memberikan pandangan mendalam tentang perubahan parameter musik antar lagu, memungkinkan evaluasi kualitas transisi secara individual. Parameter yang dianalisis mencakup perubahan tempo, dan perubahan level energi, memberikan pemahaman komprehensif tentang *smoothness* transisi.

E. Visualisasi Hasil

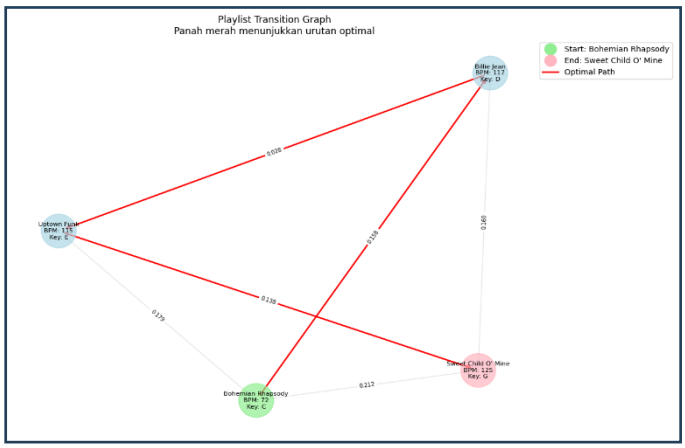
Visualisasi hasil optimasi diimplementasikan menggunakan kombinasi networkx untuk representasi graf dan matplotlib untuk rendering visual. Graf hasil optimasi menampilkan lagu sebagai nodes dan transisi sebagai *edges* berbobot, dengan informasi parameter musik yang relevan.

```
def visualize_graph(self, playlist: List[Song]):
  G = nx.DiGraph()
  for song in playlist:
      G.add_node(song.title,
      tempo=song.tempo,
      key=get_key_name(song.key),
      energy=song.energy)
  for i in range(len(playlist)-1):
      cost
      =self.calculate_transition_cost(playlist[i], playlist[i+1])
     G.add_edge(playlist[i].title,
     playlist[i+1].title,
     weight=cost,
     color=get_edge_color(cost))
```

Visualisasi menampilkan informasi dalam format yang intuitif, dengan warna e*dge* merepresentasikan cost transisi (merah untuk *cost* tinggi, hijau untuk *cost* rendah) dan ukuran node menunjukkan karakteristik musik seperti energi atau tempo. Layout graf dioptimasi menggunakan *spring layout* untuk memaksimalkan keterbacaan dan meminimalkan *edge* yg berpotongan.

IV. HASIL DAN PEMBAHASAN

A. Analisis Path Optimal



Gambar 1 Graph Transisi Playlist
Sumber : Output Implementasi Program

Berdasarkan hasil optimasi menggunakan algoritma A*, diperoleh urutan optimal playlist yang dimulai dari lagu Bohemian Rhapsody dan berakhir di Sweet Child O' Mine. Dari visualisasi graf yang ditampilkan, dapat dilihat bahwa algoritma A* berhasil menemukan lintasan optimal yang ditunjukkan dengan panah merah. Pemilihan urutan ini terbukti optimal karena memanfaatkan transisi dengan cost rendah (0.028) antara Billie Jean dan Uptown Funk, serta mendistribusikan perubahan parameter yang lebih ekstrem (terutama tempo) ke beberapa transisi untuk menghindari perubahan yang terlalu drastis dalam satu langkah.

Tabel I Detail Path Optimal dan Transisi

Urutan	Lagu	Lagu Selanjutnya	Cost	Detail Parameter
1	Bohemian Rhapsody	Billie Jean	0.158	Δ Tempo: +45 BPM, ΔKey: +2
2	Billie Jean	Uptown Funk	0.028	Δ Tempo: +2 BPM, ΔKey: +2
3	Uptown Funk	Sweet Child O' Mine	0.138	Δ Tempo: +10 BPM, ΔKey: +3

Pada Tabel I dapat dilihat bahwa urutan optimal yang dihasilkan mempertimbangkan perubahan parameter musik antar lagu yang berurutan.

Transisi pertama dari Bohemian Rhapsody ke Billie Jean memiliki cost 0.158, yang merupakan cost tertinggi dalam path ini. Hal ini disebabkan oleh perbedaan tempo yang signifikan, yaitu kenaikan 45 BPM dari tempo 72 BPM ke 117 BPM, serta perubahan key sebanyak 2 langkah dari C ke D. Meskipun cost transisi ini tinggi, algoritma memilih path ini karena memberikan total cost yang lebih rendah untuk keseluruhan urutan.

Transisi kedua dari Billie Jean ke Uptown Funk memiliki cost terendah yaitu 0.028. Cost yang rendah ini dicapai karena kedua lagu memiliki karakteristik yang sangat mirip, dengan perbedaan tempo yang minimal (hanya 2 BPM) meskipun terdapat perubahan key sebanyak 2 langkah dari D ke E. Transisi yang smooth ini menunjukkan keberhasilan algoritma dalam menemukan urutan lagu yang memiliki kemiripan parameter musik.

Pada transisi terakhir dari Uptown Funk ke Sweet Child O' Mine, cost kembali meningkat menjadi 0.138. Peningkatan ini terutama disebabkan oleh kenaikan tempo sebesar 10 BPM dan perubahan key yang lebih jauh yaitu 3 langkah dari E ke G. Meskipun demikian, cost ini masih lebih rendah dibandingkan transisi pertama karena perubahan parameternya lebih moderat

B. Perbandingan Path

Tabel II Perbandingan Path

Path	Total Cost	Avg Cost	Max Cost	Min Cost
Bohemian R →Billie Jea →Uptown Fun →Sweet Chil	0.323	0.108	0.158	0.028
Bohemian R →Sweet Chil → Billie Jea →Uptown Fun	0.400	0.133	0.212	0.028
Sweet Chil →Uptown Fun → Billie Jea → Bohemian R	0.323	0.108	0.158	0.028

Keterangan :
"→" memiliki arti "lanjut ke"

Analisis perbandingan berbagai path yang mungkin disajikan dalam Tabel II menunjukkan tiga alternatif urutan playlist dengan karakteristik transisi yang berbeda. Menariknya, ditemukan dua path yang menghasilkan total cost yang sama yaitu 0.323, namun dengan urutan lagu yang berbeda.

Lintasan optimal yang ditemukan algoritma A* (Bohemian Rhapsody → Billie Jean → Uptown Funk → Sweet Child O' Mine) mencapai total cost 0.323 dengan rata-rata cost 0.108 per transisi. Path ini memiliki cost maksimum 0.158 yang terjadi pada transisi awal dan cost minimum 0.028 pada transisi antara Billie Jean dan Uptown Funk. Distribusi cost yang seimbang ini menunjukkan bahwa algoritma berhasil mendistribusikan perubahan parameter musik secara efektif.

Lintasan alternatif pertama (Bohemian Rhapsody → Sweet Child O' Mine → Billie Jean → Uptown Funk) menghasilkan total cost yang lebih tinggi yaitu 0.400 dengan rata-rata 0.133. Path ini memiliki cost maksimum tertinggi di antara ketiga alternatif yaitu 0.212, yang disebabkan oleh transisi langsung dari Bohemian Rhapsody ke Sweet Child O' Mine yang memiliki perbedaan parameter yang signifikan. Meskipun path ini juga mencapai cost minimum 0.028 pada transisi antara

Billie Jean dan Uptown Funk, total costnya yang lebih tinggi membuatnya menjadi pilihan yang kurang optimal.

Lintasan alternatif kedua (Sweet Child O' Mine → Uptown Funk → Billie Jean → Bohemian Rhapsody) menarik karena menghasilkan total *cost* yang sama dengan lintasan optimal yaitu 0.323. Lintasan ini bahkan memiliki distribusi *cost* yang serupa dengan *cost* maksimum 0.158 dan minimum 0.028. Namun, algoritma A* tetap memilih path pertama sebagai solusi karena memenuhi *constraint* dimulai dari Bohemian Rhapsody sesuai dengan *input* yang diberikan.

Hasil perbandingan ini memvalidasi keputusan algoritma A* dalam memilih lintasan optimal, dimana urutan yang dipilih tidak hanya mempertimbangkan total *cost* terendah tetapi juga memenuhi *constraint* yang ditetapkan serta menjaga keseimbangan transisi antar lagu.

C. Analisis Parameter

Tabel III Analisis Parameter Musik

Parameter	Bobot	Rata-Rata	Std Dev	Impact
Tempo	0.30	19	18.673	5.7
Energy	0.25	0.067	0.047	0.017
Dance	0.20	0.167	0.125	0.033
Key	0.15	2.333	0.471	0.356
Mode	0.10	0	0	0

Berdasarkan Tabel III, dapat dilakukan analisis mendalam terhadap kontribusi masing-masing parameter musik dalam optimasi urutan playlist. Setiap parameter memiliki bobot dan pengaruh yang berbeda terhadap kualitas transisi antar lagu.

Tempo merupakan parameter dengan bobot tertinggi (0.30) dan menunjukkan pengaruh yang paling signifikan. Dengan rata-rata perubahan 19 BPM dan standar deviasi 18.673, parameter ini memiliki variasi yang cukup tinggi dalam dataset. *Impact score* sebesar 5.700 mengindikasikan bahwa perbedaan tempo antar lagu menjadi faktor dominan dalam menentukan *cost* transisi. Hal ini sesuai dengan intuisi musikal dimana perubahan tempo yang drastis dapat mengganggu pengalaman mendengarkan.

Parameter *energy* dengan bobot 0.25 menunjukkan perubahan yang relatif kecil dengan rata-rata 0.067 dan standar deviasi 0.047. Impact score yang rendah (0.017) mengindikasikan bahwa lagu-lagu dalam playlist memiliki level energi yang cukup konsisten, sehingga tidak terlalu mempengaruhi cost transisi secara keseluruhan.

Danceability (bobot 0.20) memiliki rata-rata perubahan 0.167 dengan standar deviasi 0.125, menghasilkan impact score 0.033. Nilai ini menunjukkan bahwa variasi *danceability* lebih signifikan dibandingkan energy, namun tetap lebih rendah dibandingkan pengaruh tempo.

Perubahan key (bobot 0.15) menunjukkan rata-rata 2.333 langkah dalam dengan standar deviasi yang relatif kecil (0.471). *Impact score* 0.350 mengindikasikan bahwa meskipun bobotnya lebih rendah, perubahan key memberikan kontribusi yang cukup signifikan pada cost transisi.

Mode (bobot 0.10) memiliki nilai perubahan dan impact 0 karena seluruh lagu dalam dataset menggunakan mode yang sama (major). Hal ini menunjukkan bahwa dalam kasus ini, parameter mode tidak mempengaruhi optimasi urutan playlist.

Analisis ini menunjukkan bahwa algoritma A* berhasil mengoptimalkan urutan playlist dengan mempertimbangkan terutama perubahan tempo dan key, sementara menjaga konsistensi *energy* dan *danceability*. Dominasi parameter tempo dalam penentuan *cost* sesuai dengan praktik musik dimana perubahan tempo yang halus umumnya lebih diutamakan untuk menciptakan transisi yang mulus antar lagu.

V. KESIMPULAN

Makalah ini telah menyajikan implementasi algoritma A* untuk optimasi urutan playlist musik menggunakan pendekatan graf berbobot dengan parameter dinamis. Berdasarkan hasil pengujian dan analisis yang telah dilakukan, dapat disimpulkan beberapa poin utama.

Algoritma A* terbukti efektif dalam menemukan urutan playlist optimal dengan total cost 0.323 dan rata-rata cost transisi 0.108. Hal ini dicapai melalui pertimbangan beberapa parameter musik seperti tempo, *energy, danceability, key*, dan *mode*, dimana setiap parameter diberi bobot yang berbeda sesuai dengan tingkat kepentingannya dalam menentukan kualitas transisi.

Dari analisis parameter musik, tempo menjadi faktor yang paling berpengaruh dengan impact score 5.700, diikuti oleh perubahan *key* dengan *impact* 0.350. Hal ini menunjukkan bahwa algoritma berhasil mengoptimalkan urutan dengan memprioritaskan kehalusan transisi tempo sambil tetap mempertahankan harmonisasi antar lagu. Perbandingan dengan path alternatif memvalidasi optimalitas solusi yang dihasilkan, dimana path optimal berhasil mendistribusikan perubahan parameter secara lebih seimbang untuk menghindari transisi yang terlalu drastis.

Implementasi ini dapat dikembangkan lebih lanjut dengan menambahkan parameter musik lainnya, mengimplementasikan pembobotan dinamis berdasarkan preferensi pengguna, atau mengintegrasikan algoritma dengan platform streaming musik yang ada. Pengembangan ke depan juga dapat fokus pada optimasi performa untuk dataset yang lebih besar dan penanganan genre yang lebih beragam.

VI. LAMPIRAN

Program lengkap dari makalah ini dapat diakses melalu link repository github berikut:
https://github.com/WwzFwz/playlist_optimizer

Kemudian video penjelasan makalaj ini dapat diakses melalui link ini :
https://drive.google.com/file/d/1Lql5DEbxKerRR6bT2ELvdyn-kDM-UReT/view?usp=drive_link

VII. UCAPAN TERIMAKASIH

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas rahmat dan karunia-Nya sehingga makalah ini dapat diselesaikan dengan baik. Penulis mengucapkan terima kasih kepada:

1. Bapak Dr. Rinaldi Munir, M.T selaku dosen pengajar mata kuliah IF1220 Matematika Diskrit, atas bimbingan dan ilmu yang diberikan.
2. Kedua orang tua penulis atas dukungan yang tiada

henti.
3. Teman-teman yang telah membantu dalam proses penyusunan makalah ini.
4. Platform streaming musik dan komunitas pengembang yang telah menyediakan dataset untuk pengujian.
5. Para peneliti dan penulis yang karyanya menjadi referensi dalam penulisan makalah ini.

Akhir kata, penulis berharap makalah ini dapat memberikan kontribusi dalam pengembangan sistem optimasi playlist musik dan bermanfaat bagi pembaca.

DAFTAR PUSTAKA

[1] S. Y. M. Netti dan I. Irwansyah, "Spotify: Aplikasi Music Streaming untuk Generasi Milenial," *Jurnal Komunikasi*, vol. 10, no. 1, hlm. 1, Jul 2018, doi: 10.24912/jk.v10i1.1102.

[2] S. Ikeda, K. Oku, dan K. Kawagoe, "Music Playlist Recommendation Using Acoustic-Feature Transition Inside the Songs," dalam *Proceedings of the 15th International Conference on Advances in Mobile Computing & Multimedia - MoMM2017*, New York, New York, USA: ACM Press, 2017, hlm. 216–219. doi: 10.1145/3151848.3151880.

[3] Rinaldi Munir, "20-Graf-Bagian1-2024." Diakses: 8 Januari 2025. [Daring]. Tersedia pada: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

[4] D. Västfjäll, "Emotion induction through music: A review of the musical mood induction procedure," *Musicae Scientiae*, vol. 5, no. 1_suppl, hlm. 173–211, Sep 2001, doi: 10.1177/10298649020050S107.

[5] J. Yao, B. Zhang, dan Q. Zhou, "The Optimization of A* Algorithm in the Practical Path Finding Application," dalam *2009 WRI World Congress on Software Engineering*, IEEE, 2009, hlm. 514–518. doi: 10.1109/WCSE.2009.412.

[6] S. Sneha, D. S. Jayalakshmi, J. Shruthi, dan U. R. Shetty, "Recommending Music by Combining Content-Based and Collaborative Filtering with User Preferences," 2014, hlm. 507–515. doi: 10.1007/978-81-322-1157-0_52.

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, 8 Januari 2024

Dzaky Aurelia Fawwaz
13523065