Optimasi Pembentukan Playlist Musik Personal Menggunakan Algoritma Backtracking dengan Multi-Constraint Satisfaction

Natalia Desiany Nursimin - 13523157
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: nataliadesianyy@gmail.com, 13523157@std.stei.itb.ac.id

***Abstrak*—Pembuatan playlist musik yang sesuai dengan preferensi personal merupakan tantangan kompleks di era digital, terutama saat mempertimbangkan berbagai kendala seperti target durasi, konsistensi mood, keragaman genre, dan batas maksimum lagu dari artis yang sama. Untuk mengatasi permasalahan ini, makalah ini mengusulkan sistem optimasi pembentukan playlist musik personal berbasis algoritma backtracking dengan pendekatan multi-constraint satisfaction untuk menghasilkan kombinasi lagu yang paling sesuai dengan preferensi pengguna. Algoritma backtracking digunakan untuk menjelajahi seluruh kemungkinan kombinasi lagu sambil memangkas cabang solusi yang tidak memenuhi kendala melalui teknik pruning yang efisien. Implementasi sistem dilakukan menggunakan dataset musik dengan beragam genre dan karakteristik mood yang berbeda untuk memvalidasi efektivitas pendekatan yang diusulkan. Hasil eksperimen menunjukkan bahwa metode ini mampu menghasilkan playlist yang lebih relevan dan sesuai dengan kebutuhan pengguna dibandingkan metode pemilihan acak, khususnya dalam skenario dengan banyak batasan dan variabel kompleks. Penelitian ini diharapkan dapat memberikan kontribusi nyata dalam pengembangan sistem rekomendasi musik yang lebih personal, fleksibel, dan adaptif terhadap kebutuhan pengguna dengan potensi diaplikasikan pada berbagai platform streaming musik modern.**

Kata kunci—playlist musik; algoritma backtracking; multi-constraint satisfaction; personalisasi; sistem rekomendasi

I. PENDAHULUAN

Dalam era digital saat ini, musik telah menjadi sebuah hal yang tak terpisahkan dari kehidupan sehari-hari masyarakat global. Perkembangan teknologi dan infrastruktur internet yang pesat telah mengantarkan konsumsi musik pada transformasi revolusioner, memungkinkan akses instan terhadap jutaan lagu melalui platform streaming seperti Spotify, Apple Music, YouTube Music, dan Joox. Meskipun kemudahan akses ini memberikan banyak keuntungan, muncul pula berbagai tantangan baru, yaitu bagaimana menyaring dan memilih lagu yang benar-benar sesuai dengan preferensi personal dari lautan lagu yang tersedia. Pemilihan musik secara manual menjadi tidak praktis dan sangat menguras waktu, terutama ketika pengguna tidak memiliki pengetahuan spesifik tentang artis atau judul lagu yang mereka inginkan.

Salah satu fitur paling umum yang ditawarkan oleh platform musik digital adalah sistem rekomendasi dan pembuatan playlist otomatis kepada para penggunanya. Secara teori, sistem ini bertujuan untuk membantu pengguna menemukan lagu-lagu yang cocok dengan selera dan kebutuhan mereka. Namun, dalam praktiknya, algoritma yang digunakan oleh sebagian besar sistem rekomendasi sering kali masih terbatas pada pendekatan statistik sederhana atau pembelajaran berbasis preferensi historis yang tidak selalu mampu mencerminkan kebutuhan pengguna secara real-time. Akibatnya, banyak pengguna yang merasa bahwa playlist yang diberikan terasa terlalu repetitif, tidak sesuai mood, atau gagal merepresentasikan spektrum preferensi personal mereka secara komprehensif.

Kompleksitas permasalahan ini semakin meningkat ketika pengguna memiliki *multiple constraints* atau banyak kriteria spesifik dalam menentukan sebuah playlist yang ideal. Misalnya, pengguna ingin playlist yang berdurasi total 30 menit, berisi lagu dari berbagai genre, namun tetap mempertahankan konsistensi mood tertentu seperti lagu yang "bersemangat" atau "sedih", serta tidak mengulang lagu dari artis yang sama lebih dari dua kali. Kebutuhan multi-dimensional semacam ini tidak dapat diselesaikan secara optimal oleh pendekatan konvensional seperti *collaborative filtering* atau *content-based recommendation* yang umumnya fokus pada satu atau dua dimensi preferensi. Sehingga, sangat diperlukan adanya pendekatan algoritmis baru yang mampu menelusuri kombinasi-kombinasi lagu secara sistematis sambil tetap mempertimbangkan banyak batasan secara bersamaan.

Dalam konteks ini, algoritma *backtracking* sebagai salah satu strategi fundamental dalam domain algoritma kombinatorial memiliki potensi besar untuk dimanfaatkan sebagai solusi yang tepat untuk menangani permasalahan *multi-constraint satisfaction* ini. Algoritma runut-balik memungkinkan dilakukannya eksplorasi sistematis terhadap seluruh ruang solusi yang mungkin dengan membangun solusi

secara incremental dan memanfaatkan teknik pemangkasan (*pruning)* terhadap cabang pencarian yang terbukti tidak *feasible* sejak tahap awal. Keunggulan utama pendekatan ini terletak pada kemampuannya untuk mengevaluasi kesesuaian berbagai batasan secara eksplisit dan dinamis, serta dapat menghasilkan sebuah solusi yang memenuhi seluruh batasan yang ditetapkan sambil tetap mempertahankan efisiensi komputasi melalui teknik *pruning* yang terintegrasi.

Signifikansi dari pengimplementasian ini juga tidak hanya terbatas pada kebutuhan individual, melainkan memiliki potensi aplikasi yang luas dalam konteks komersial seperti penyusunan playlist publik untuk restoran, kafe, gym, atau acara tertentu yang memiliki kebutuhan audio spesifik. Kemampuan untuk menyusun playlist berdasarkan multiple parameter membuat pendekatan ini sangat fleksibel dan dapat diadaptasi ke berbagai kebutuhan yang berbeda. Penelitian ini juga membuka ruang eksplorasi lebih lanjut terhadap bagaimana integrasi metode pencarian klasik seperti backtracking dengan data real-time dari perilaku pengguna dapat digunakan untuk menciptakan sistem rekomendasi yang cerdas dan *contextually-relevant*.

Berdasarkan latar belakang dan urgensi permasalahan tersebut, pembuatan makalah ini bertujuan untuk membantu mengatasi permasalahan yang ada dengan merancang, mengimplementasikan, dan mengevaluasi sistem pembentukan playlist musik personal yang menggabungkan algoritma *backtracking* dengan pendekatan *multi-constraint satisfaction*. Diharapkan, hasil dari penelitian ini dapat memberikan kontribusi dalam pengembangan sistem rekomendasi musik yang lebih akurat, fleksibel, dan memuaskan secara pengalaman pengguna.

II. LANDASAN TEORI

A. Definisi Algoritma Runut-Balik (Backtracking)

Algoritma runut-balik (*backtracking*) merupakan salah satu teknik pemecahan masalah yang sangat efektif untuk menangani persoalan kombinatorial, seperti persoalan optimasi dan pencarian solusi dalam ruang yang sangat besar. Metode ini bekerja dengan cara membangun solusi secara bertahap, dan jika pada suatu titik solusi parsial ini tidak lagi menjanjikan, maka proses pencarian akan mundur (*backtrack*) ke keadaan sebelumnya untuk mencoba alternatif-alternatif lainnya.

Berbeda dengan pendekatan dengan metode *exhaustive search* yang memeriksa seluruh kemungkinan tanpa pengecualian, algoritma runut-balik hanya menelusuri cabang solusi yang dinilai akan berpotensi menghasilkan hasil yang valid. Proses penelusuran ini dikenal sebagai teknik *pruning*, yaitu teknik pemangkasan ruang solusi untuk meningkatkan efisiensi pencarian dan menghindari eksplorasi terhadap solusi yang jelas tidak memenuhi syarat.

Algoritma ini pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950 dan dikembangkan secara lebih lanjut oleh R.J. Walker, Golomb, dan Baumert. Prinsip dasar dari *backtracking* sangat berkaitan dengan traversal pohon status menggunakan pendekatan depth-first search (DFS). Dalam proses pencarian algoritma ini, setiap simpul mewakili sebuah status solusi parsial, dan pencarian dilakukan secara mendalam dari akar hingga ke simpul daun untuk menemukan solusi yang valid. Algoritma *backtracking* membangun solusi secara bertahap dengan membangkitkan simpul-simpul status secara sistematis. Hasil yang diperoleh akan berupa lintasan dari akar ke daun dalam sebuah *state space tree*, yang merepresentasikan seluruh kemungkinan solusi yang bisa dihasilkan.

B. Struktur Umum dan Komponen Algoritma Backtracking

Solusi dalam algoritma backtracking direpresentasikan sebagai vektor berukuran *n-tuple*, yaitu:

$$X = (x_1, x_2, ..., x_n)$$

di mana setiap komponen x_i akan dipilih dari himpunan kandidat solusi S_i. Dalam pembuatan sebuah *playlist musik*, elemen x_i dapat digunakan untuk merepresentasikan sebuah lagu yang dipilih pada posisi ke-*i* dalam playlist. Terdapat tiga komponen penting dalam algoritma ini, yaitu:

1. Fungsi pembangkit nilai (x_k)
   Komponen ini berfungsi untuk membangkitkan nilai-nilai kandidat untuk setiap komponen x_k berdasarkan keputusan pada posisi sebelumnya yang dapat direpresentasikan sebagai ekspresi dibawah ini:
   $$T(x_1, x_2, \dots, x_{k-1})$$

2. Fungsi pembatas (*bounding function)*
   Komponen ini berisi fungsi predikat yang menentukan apakah solusi parsial $(x_1, x_2, \dots, x_k)$ masih memungkinkan mengarah ke solusi akhir. Jika tidak, pencarian ini tidak akan diteruskan. Fungsi ini dapat dipresentasikan sebagai ekspresi dibawah ini:
   $$B(x_1, x_2, \dots, x_k) = true/false$$

3. Ruang solusi dan pohon status
   Seluruh kombinasi solusi dapat divisualisasikan dalam sebuah struktur pohon, yang disebut *state space tree*. Setiap simpul menyatakan keadaan parsial dari solusi, dan sisi yang menghubungkan menyatakan keputusan yang diambil. Pencarian berlangsung dari akar ke daun menggunakan strategi DFS, dan simpul akan "dimatikan" bila tidak menjanjikan (*dead node*).

C. Constraint Satisfaction Problem

Constraint Satisfaction Problem (CSP) dapat didefinisikan sebagai sebuah bentuk paradigma penyelesaian masalah yang melibatkan pencarian solusi dalam ruang yang dibatasi oleh sejumlah batasan. CSP menyediakan kerangka kerja matematis yang dapat digunakan untuk merepresentasikan dan menyelesaikan berbagai permasalahan kombinatorial dan optimasi diskret, terutama yang melibatkan pengambilan keputusan dengan banyak pembatas.

Permasalahan CSP dapat direpresentasikan dalam bentuk:

$$(X, D, C)$$

di mana:

- X = $(x_1, x_2, \dots, x_n)$ adalah himpunan finite variabel keputusan.
- D = $(D_1, D_2, \dots, D_n)$ adalah himpunan domain yang bersesuaian dengan setiap variable.
- C = $(c_1, c_2, \dots, c_m)$ adalah himpunan batasan *(constraints)* yang mendefinisikan relasi dan batasan antar variabel

Sebuah solusi dikatakan valid jika merupakan assignment lengkap dari nilai ke seluruh variabel dalam X, dan kombinasi nilai tersebut memenuhi semua constraint dalam C secara bersamaan.

D. Multi-Constraint Satisfaction Problem (Multi-CSP)

Multi-Constraint Satisfaction Problem (Multi-CSP) merupakan pengembangan dari model *dasar Constraint Satisfaction Problem* (CSP). Dalam model ini, permasalahan tidak hanya melibatkan sejumlah besar *constraint*, tetapi juga mencakup berbagai tipe constraint yang bersifat heterogen, memiliki karakteristik yang berbeda-beda, dan sering kali saling berkaitan.

Dalam Multi-CSP, sebuah variabel dapat dikenai lebih dari satu pembatas yang berasal dari jenis constraint yang berbeda, sehingga interaksi antar constraint menjadi lebih kompleks dan tidak linier. Secara umum, jenis-jenis *constraint* dalam Multi-CSP dapat diklasifikasikan ke dalam beberapa kategori berikut:

1. Constraint Unik
   Merupakan tipe batasan yang mengharuskan setiap nilai hanya muncul satu kali. Dalam konteks playlist, hal ini dapat dimisalkan, seperti tidak boleh ada lagu yang sama atau artis yang sama muncul lebih dari jumlah yang ditentukan.
2. Constraint Interval
   Merupakan tipe batasan di mana variabel harus berada dalam rentang nilai tertentu. Dalam konteks playlist, hal ini dapat dimisalkan, seperti durasi lagu harus berada dalam rentang 2 hingga 6 menit, atau suasana lagu harus berada dalam level tertentu yang disesuaikan dengan tema playlist.
3. Constraint Agregat
   Merupakan tipe batasan yang melibatkan perhitungan agregasi dari nilai-nilai variabel. Dalam konteks playlist, hal ini dapat dimisalkan, seperti total durasi dari seluruh lagu harus kurang dari 30 menit, atau rata-rata valensi seluruh lagu harus memenuhi ambang tertentu.
4. Constraint Kombinatorial
   Merupakan tipe batasan yang melibatkan aturan gabungan antar elemen. Dalam konteks playlist, hal ini dapat dimisalkan, seperti playlist harus mencakup minimal tiga genre berbeda, atau tidak boleh ada dua lagu berurutan dari genre yang sama.

III. PEMBAHASAN

Menyusun sebuah playlist musik yang sesuai dengan preferensi personal telah menjadi tantangan yang semakin kompleks dalam era digital modern, ketika jutaan lagu tersedia secara instan melalui berbagai platform streaming. Kondisi ini menyebabkan banyak pengguna mengalami kesulitan dalam mencari dan merancang playlist yang tidak hanya sesuai dengan selera mereka, tetapi juga konsisten dalam hal mood, kualitas lagu, durasi total, serta keragaman genre. Salah satu pendekatan algoritmik yang relevan untuk menangani permasalahan ini adalah algoritma *backtracking*, yang memungkinkan eksplorasi ruang solusi secara sistematis dengan mempertimbangkan berbagai batasan pengguna. Oleh karena itu, pada bagian ini akan dibahas secara rinci kerangka penerapan algoritma backtracking dalam pembentukan playlist musik personal, yang diformulasikan sebagai permasalahan *Multi-Constraint Satisfaction* dengan tujuan menghasilkan playlist yang adaptif, berkualitas tinggi, dan sesuai dengan kriteria-kriteria yang diinginkan masing-masing pengguna.

A. Definisi Permasalahan

Pembentukan playlist musik personal yang optimal merupakan sebuah permasalahan kombinatorial kompleks yang dapat diformulasikan sebagai *Multi-Constraint Satisfaction Problem* (Multi-CSP). Masalah ini melibatkan pencarian kombinasi lagu yang memenuhi berbagai tipe batasan secara simultan, seperti:

- Constraint Durasi: Total durasi playlist harus berada dalam rentang target ± toleransi yang ditetapkan oleh pengguna.
- Constraint Mood: Persentase lagu dengan mood tertentu harus mencapai dan sesuai dengan threshold minimum.
- Constraint Kualitas: Rating rata-rata lagu harus memenuhi standar kualitas yang diinginkan oleh pengguna.
- Constraint Keragaman: Jumlah genre berbeda dan distribusi artis harus memenuhi kriteria diversitas yang diterapkan oleh pengguna.
- Constraint Batasan: Maksimum lagu per artis tidak boleh terlampaui. Artinya, dalam playlist yang dibentuk, tidak boleh ada lebih dari jumlah maksimum lagu yang ditentukan untuk setiap artis. Batasan ini dapat diterapkan sendiri oleh masing-masing pengguna.

B. Formulasi Logika

Formulasi logika untuk masalah pembentukan playlist dapat direpresentasikan dalam bentuk tupel, yang terdiri atas kombinasi dari atribut lagu serta batasan-batasan (*constrain*t) yang harus dipenuhi.

1. Definisi Ruang Status (*State Space*):

- Status (S): Playlist parsial yang terdiri dari 0 hingga *n* lagu.
- Status Awal (S_0): Playlist kosong ∅.
- Status Tujuan (S_m): Playlist lengkap yang memenuhi semua batasan.
- Aksi (A): Menambahkan satu lagu ke playlist saat ini.

2. Formalisasi Batasan (*Constraint*):

   Dalam proses pembentukan playlist musik personal berbasis algoritma *backtracking,* kita dapat mengklasifikasikan tipe batasan menjadi dua kategori utama, yaitu batasan keras dan batasan lunak.

   - Batasan Keras
     Batasan keras adalah batasan yang tidak dapat dilanggar oleh solusi yang dibentuk. Apabila salah satu dari batasan ini tidak terpenuhi, maka solusi tersebut secara otomatis dianggap tidak valid dan harus ditolak atau dihapus melalui proses *backtracking.* Berikut merupakan beberapa batasan keras yang harus diterapkan ke dalam algoritma penyusunan playlist:
     - Durasi: $|\sum \text{durasi}(s_i) - \text{durasi\_target}| \leq$ toleransi
     - Batasan Artis: Untuk setiap artis, jumlah lagunya ≤ batas maksimum lagu per artis
     - Ukuran Playlist: min_songs ≤ |playlist| ≤ max_songs
     - Keunikan: Tidak boleh ada lagu yang sama dalam playlist. Dapat diekspresikan dalam bentuk berikut: $\forall s_i, s_j \in$ playlist: $s_i \neq s_j$
   - Batasan Lunak
     Batasan lunak merupakan batasan yang tidak wajib sepenuhnya dipenuhi, namun menjadi faktor penting dalam meningkatkan kualitas dan relevansi playlist. Semakin tinggi tingkat pemenuhan batasan lunak, maka kualitas solusi akan semakin baik. Berikut merupakan beberapa batasan lunak yang harus diterapkan ke dalam algoritma penyusunan playlist:
     - Konsistensi Mood: (jumlah lagu dengan mood tertentu / total lagu) ≥ persentase minimum
     - Keberagaman Genre: Jumlah genre unik ≥ genre minimum
     - Kualitas Lagu: Rata-rata rating ≥ rating minimum
     - Kesesuaian Energi: rata-rata energi lagu berada dalam [energi_min, energi_max]

C. Algoritma Backtracking untuk Multi-Constraint Satisfactions

Algoritma backtracking diimplementasikan dengan pendekatan Depth-First Search (DFS) yang membangun solusi secara incremental. Struktur dasar algoritma ini dalam konteks penyusunan playlist meliputi komponen sebagai berikut:

- Pilihan (*Choice*): Memilih lagu dari kandidat berdasarkan prioritas
- Pemeriksaan Batasan (*Constraint Checking*): Memastikan lagu baru tidak melanggar constraint
- Eksplorasi: Melanjutkan pencarian secara rekursif
- Backtrack: Membatalkan keputusan jika hasil tidak valid

Kemudian, untuk dapat meningkatkan efisiensi pencarian, diperlukan pengimplementasian beberapa tingkat pruning, yaitu sebagai berikut:

a. Propagasi Batasan

   Setiap penambahan lagu memicu evaluasi batasan untuk mendeteksi pelanggaran sedini mungkin. Hal ini dilakukan dengan memeriksa beberapa komponen, seperti:

   - Pemeriksaan Jumlah Artis: Memastikan tidak melebihi batas lagu per artis
   - Pemeriksaan Batas Durasi: Memastikan durasi tidak melebihi batas atas
   - Pemeriksaan Batas Ukuran: Memastikan tidak melebihi maksimum lagu

b. Analisis Kelayakan

   Estimasi kemungkinan penyelesaian solusi parsial dengan menghitung berbagai komponen, seperti:

   - Ketersediaan Sumber Daya: Ketersediaan lagu dengan karakteristik yang dibutuhkan
   - Kelayakan Durasi: Kemungkinan mencapai target durasi dengan lagu tersisa
   - Potensi Pemenuhan Batasan: Potensi memenuhi batasan keragaman

c. Estimasi Batas

   Perhitungan batas atas skor yang mungkin dicapai untuk memangkas cabang yang tidak optimal. Perhitungan ini dilakukan dengan mempertimbangkan beberapa komponen, seperti:

   - Estimasi Skor Optimis: Perkiraan skor maksimum dengan lagu tersisa terbaik
   - Penalti Pelanggaran Batasan: Penalti untuk solusi yang berpotensi melanggar batasan

Selanjutnya, diperlukan pengimplementasian strategi pemilihan kandidat yang cerdas untuk mengurangi jumlah

branching factor. Hal ini dapat dilakukan dengan menerapkan dua strategi utama, yaitu:

a. Seleksi Berdasarkan Prioritas
   - Prioritas Mood: Prioritas tinggi untuk lagu dengan mood target jika persentase belum tercapai
   - Pengisian Genre: Prioritas untuk genre yang belum terwakili dalam playlist
   - Pengurutan Kualitas: Pengurutan berdasarkan rating dan popularitas untuk kualitas optimal

b. Penyaringan Berdasarkan Batasan
   - Batasan Artis: Eliminasi lagu dari artis yang sudah mencapai batas maksimum
   - Batasan Energi: Penyaringan berdasarkan rentang tingkat energi yang sesuai
   - Batasan Rating: Eliminasi lagu dengan rating di bawah minimum

Terakhir, diperlukan pengimplementasian optimasi strategi pencarian yang adaptif. Strategi pencarian ini dapat diimplementasikan melalui dua strategi, yaitu:

a. Pembatasan kedalaman pencarian berdasarkan kompleksitas batasan

   Algoritma yang dibuat harus dapat mengimplementasikan mekanisme penghentian dini untuk meningkatkan efisiensi komputasi. Pencarian akan dihentikan secara otomatis ketika ditemukan solusi dengan kualitas yang memadai, tanpa perlu mengeksplorasi seluruh ruang pencarian.

b. Penyesuaian dinamis berdasarkan tingkat pemangkasan yang dicapai

   Implementasi algoritma menerapkan strategi manajemen memori yang optimal untuk mengurangi overhead komputasi. State yang disimpan selama proses rekursi diminimalisasi hanya pada informasi yang benar-benar diperlukan untuk pengambilan keputusan dan backtracking.

D. Multi-Constraint Satisfaction dalam Domain Musik

1) Karakteristik Khusus Constraint Musik

a. Saling Ketergantungan Batasan:
- Hubungan Durasi-Jumlah: Durasi total berkorelasi dengan jumlah lagu
- Korelasi Mood-Energi: Mood tertentu cenderung memiliki tingkat energi spesifik
- Hubungan Genre-Artis: Distribusi artis dapat mempengaruhi keragaman genre

b. Hierarki Prioritas Batasan:
- Batasan Kritis: Batas durasi
- Batasan Penting: Konsistensi mood
- Batasan yang Diinginkan: Keragaman genre
- Batasan Opsional: Keragaman tahun

2) Praproses dan Pengindeksan untuk Efisiensi

Langkah-langkah untuk praproses dapat dibagi ke dalam tiga tahapan, yaitu:

a. Penyaringan Awal: Eliminasi lagu yang tidak memenuhi batasan dasar
b. Pengurutan Kualitas: Pengurutan berdasarkan rating dan popularitas
c. Pengayaan Metadata: Perhitungan atribut turunan untuk optimasi

3) Desain Fungsi Evaluasi

Fungsi evaluasi di desain sebagai sistem penilaian multi-kriteria. Sistem penilaian ini menggabungkan beberapa tujuan dalam satu skor:
- Kombinasi Linear Berbobot
- Normalisasi
- Integrasi Penalti
- Penyesuaian Berbasis Gradien

E. Analisis Kompleksitas dan Kinerja

1) Kompleksitas Teoretis

a. Kompleksitas Waktu:
- Kasus Terburuk: O (b^d), dimana b = faktor percabangan, d = kedalaman maksimum
- Dengan Pemangkasan: Kompleksitas efektif berkurang drastis tergantung tingkat pemangkasan
- Kasus Rata-rata: O(b^(d×p)) dimana p = faktor efisiensi pemangkasan

b. Kompleksitas Ruang:
- Tumpukan Rekursi: O(d) untuk kedalamanmaksimum
- Penyimpanan State: O(n) untuk struktur pengindeksan
- Memori Kerja: O(k) untuk daftar kandidat dan struktur sementara

2) Faktor yang Mempengaruhi Kinerja

a. Karakteristik Dataset:
- Dampak Ukuran: Linear pada praproses, eksponensial pada pencarian tanpa pemangkasan
- Distribusi: Keseimbangan mood/genre mempengaruhi ketersediaan kandidat
- Variasi Kualitas: Rentang rating mempengaruhi diskriminasi penilaian

b. Kekakuan Batasan *(Constraint Tightness)*:
- Batasan Longgar: Konvergensi cepat, banyak solusi valid
- Batasan Sedang: Kinerja seimbang dengan kualitas baik
- Batasan Ketat: Konvergensi lambat, tingkat pemangkasan tinggi
- Batasan Tidak Mungkin: Deteksi dini melalui analisis kelayakan

IV. IMPLEMENTASI

Program ini dirancang untuk menyusun playlist musik personal secara optimal dengan mempertimbangkan preferensi pengguna serta berbagai batasan kompleks, seperti konsistensi mood, target durasi, kualitas lagu, keragaman genre, dan jumlah maksimum lagu dari artis yang sama. Program ini memformulasikan permasalahan sebagai *Multi-Constraint Satisfaction Problem* (Multi-CSP), di mana setiap kombinasi lagu dievaluasi berdasarkan serangkaian constraint yang saling bergantung. Untuk menyelesaikan permasalahan tersebut, pendekatan algoritma backtracking digunakan karena dinilai mampu menelusuri ruang solusi secara sistematis dan efisien, sambil menerapkan teknik pruning untuk mengeliminasi jalur solusi yang tidak valid sejak awal.

A. Import Modules

Program mengimpor beberapa pustaka penting dari Python untuk membangun sistem optimasi pembentukan playlist musik ini, yaitu sebagai berikut:

Fungsi Modul:

- random: digunakan untuk melakukan pengacakan elemen lagu serta simulasi playlist.
- time: digunakan untuk mengukur waktu eksekusi algoritma.
- collections: digunakan untuk pembentukan struktur data.
- dataclasses: digunakan untuk membangun objek data seperti Song dan PlaylistConstraints secara efisien.
- typing: digunakan untuk menyediakan dukungan *type hinting* untuk meningkatkan keterbacaan dan validitas kode.

```
import random
import time
import json
from typing import List, Dict, Any, Optional, Tuple
from dataclasses import dataclass, asdict
from collections import defaultdict, Counter
```

Gambar 4.1. Modul yang diimpor

B. Arsitektur Sistem

Sistem pada program terdiri dari empat komponen utama, yaitu sebagai berikut:

1) Struktur Data Inti: komponen yang berfungsi untuk menyediakan representasi data lagu dan constraint pengguna sebagai dasar pembentukan playlist.

2) Penyelesaian Constraints: komponen yang berfungsi untuk mengevaluasi apakah kombinasi lagu memenuhi batasan keras dan lunak secara simultan.

3) Algoritma Backtracking: komponen yang berfungsi untuk menelusuri seluruh ruang solusi secara rekursif dengan memanfaatkan algoritma *backtracking* sambil melakukan teknik *pruning* terhadap setiap kombinasi yang tidak valid.

4) Strategi Optimasi: komponen yang berfungsi untuk meningkatkan efisiensi pencarian solusi melalui pemilihan kandidat cerdas, evaluasi skor, dan penghentian dini.

1) Struktur Data Inti

Komponen ini merepresentasikan elemen kombinatorial berupa lagu-lagu yang akan digunakan sebagai bahan pembentuk playlist. Struktur data inti pada program ini terbagi menjadi dua kelas, yaitu class Song dan class PlaylistConstraints.

a. *Class Song*

Kelas ini berfungsi untuk mewakili tiap lagu sebagai unit kombinatorial dengan atribut: judul, artis, genre, mood, durasi (dalam detik), rating (1.0–5.0), tahun rilis, tingkat energi, dan popularitas. Berikut adalah pengimplementasian kodenya dalam program:

```
@dataclass
class Song:
    id: int
    title: str
    artist: str
    genre: str
    mood: str
    duration: int  # dalam detik
    rating: float  # skala 1.0 - 5.0
    year: int
    popularity: float = 0.0  # skor popularitas 0.0 - 1.0
    energy_level: float = 0.5  # tingkat energi 0.0 - 1.0

    def __str__(self):
        mins, secs = divmod(self.duration, 60)
        return f"{self.title} - {self.artist} ({mins}:{secs:02d})"

    def __hash__(self):
        return hash(self.id)

    def __eq__(self, other):
        return isinstance(other, Song) and self.id == other.id
```

Gambar 4.2. Class Song

b. *Class PlaylistConstraints*

Kelas ini berfungsi untuk menyimpan semua batasan playlist dari pengguna seperti mood target, durasi total, minimum rating, maksimal lagu per artis, hingga tingkat energi yang diinginkan. Berikut adalah pengimplementasian kodenya dalam program:

```
# Constraints yang diterapkan untuk pembentukan playlist
@dataclass
class PlaylistConstraints:
    target_duration: int        # target durasi dalam detik
    duration_tolerance: int     # toleransi durasi (±detik)
    min_rating: float           # rating minimum lagu yang diterima
    max_songs_per_artist: int   # batasan maksimum lagu per artis
    required_mood_percentage: float  # persentase mood dominan
    target_mood: str            # mood yang diinginkan
    min_genres: int             # minimum jumlah genre berbeda
    max_songs: int              # maksimum jumlah lagu dalam playlist
    min_songs: int              # minimum jumlah lagu dalam playlist
    year_diversity: bool        # keragaman tahun rilis
    min_energy_level: float = 0.0    # tingkat energi minimum
    max_energy_level: float = 1.0    # tingkat energi maksimum
```

Gambar 4.3. Class PlaylistConstraints

2) Penyelesaian Constraints

Komponen ini berfungsi untuk menangani batasan-*batasan secara simultan dengan pendekatan hierarchical constraint satisfaction.* Bagian ini terdiri dari *hard* dan *soft constraints*.

a. Hard Constraints

Hard constraints adalah batasan yang tidak boleh dilanggar dalam kondisi apapun. Jika salah satu hard constraint dilanggar, solusi tersebut langsung ditolak dan algoritma backtracking akan melakukan pruning pada cabang tersebut. Pada program ini terdapat beberapa batasan yang masuk ke dalam kategori *hard constraints*, yaitu seperti jumlah maksimum lagu per artist, maksimum jumlah lagu dalam playlist, durasi, serta duplikasi lagu. Fungsi _is_valid_partial_solution() bertugas memastikan playlist yang sedang dibentuk belum melanggar constraint keras. Berikut adalah pengimplementasian kodenya dalam program:

```python
# Untuk mengecek solusi parsial
def _is_valid_partial_solution(self, current_playlist: List[Song]) -> bool:
    if not current_playlist:
        return True

    # Cek maksimum lagu per artis
    artist_count = Counter(song.artist for song in current_playlist)
    if any(count > self.constraints.max_songs_per_artist for count in artist_count.values()):
        return False

    # Cek maksimum jumlah lagu
    if len(current_playlist) > self.constraints.max_songs:
        return False

    # Cek untuk memastikan durasi tidak melebihi batas atas
    current_duration = sum(song.duration for song in current_playlist)
    max_duration = self.constraints.target_duration + self.constraints.duration_tolerance
    if current_duration > max_duration:
        return False

    # Cek apakah terdapat duplikasi lagu
    if len(set(current_playlist)) != len(current_playlist):
        return False

    return True
```

Gambar 4.4. Hard Constraints

b. Soft Constraints

Soft constraints adalah batasan yang dioptimalkan tetapi tidak bersifat absolut. Pelanggaran soft constraint tidak menyebabkan immediate rejection, tetapi akan mengurangi skor dalam fungsi objektif. Pada program ini terdapat beberapa batasan yang masuk ke dalam kategoti *soft constraints,* yaitu seperti konsistensi mood, keragaman genre, keragaman artists. Berikut adalah pengimpementasian kodenya dalam program:

```python
def _is_complete_solution(self, playlist: List[Song]) -> bool:
    if len(playlist) < self.constraints.min_songs:
        return False

    stats = PlaylistStatistics(playlist, self.constraints)

    # Cek konsistensi mood
    if stats.target_mood_percentage < self.constraints.required_mood_percentage:
        return False

    # Cek keragaman genre
    if stats.unique_genres < self.constraints.min_genres:
        return False

    # Cek constraint artis
    max_artist_count = max(stats.artist_distribution.values())
    if max_artist_count > self.constraints.max_songs_per_artist:
        return False

    return True
```

Gambar 4.5. Soft Constraints

3) Algoritma Backtracking

Komponen ini berisi keseluruhan pengimplementasian algoritma *backtracking* pada program. Algoritma backtracking ini bekerja dengan cara melakukan eksplorasi ruang solusi secara rekursif sambil melakukan optimasi dan pruning untuk efisiensi. Proses dimulai dengan tracking iterasi dan kedalaman pencarian untuk monitoring kompleksitas, proses ini dilengkapi dengan timeout protection yang membatasi maksimal 15,000 iterasi untuk mencegah infinite loop.

Algoritma kemudian akan melakukan pengecekan base case untuk menentukan apakah playlist saat ini sudah lengkap dan memenuhi semua constraint. Jika ya, sistem menghitung skor playlist menggunakan multi-criteria evaluation dan melakukan update terhadap solusi terbaik jika skor yang diperoleh lebih tinggi dari sebelumnya. Proses ini memastikan algoritma selalu menyimpan solusi dengan kualitas optimal.

Selanjutnya dilakukan dua level pruning untuk meningkatkan efisiensi pencarian. Level pertama memvalidasi solusi parsial terhadap hard constraints seperti batasan artis, ukuran playlist, dan durasi. Level kedua melakukan feasibility analysis untuk memastikan solusi parsial masih mungkin diselesaikan dengan resource yang tersedia. Jika salah satu pruning gagal, algoritma langsung melakukan backtrack tanpa eksplorasi lebih lanjut, sehingga mengurangi search space secara signifikan.

Untuk menggenerasi kandidat, algoritma melakukan smart selection dengan memilih maksimal 15 lagu kandidat terbaik berdasarkan prioritas constraint yang belum terpenuhi. Kemudian dilakukan eksplorasi rekursif dimana setiap kandidat ditambahkan ke playlist, dilakukan recursive call untuk eksplorasi lebih dalam, dan kemudian lagu dihapus dari playlist (*backtrack*) untuk mencoba alternatif lainnya.

Algoritma juga dilengkapi dengan *early termination strategies* yang akan menghentikan pencarian jika sudah mendapat solusi berkualitas tinggi (skor > 80 dengan minimal 3 solusi atau skor > 85), sehingga mengoptimalkan waktu eksekusi tanpa mengorbankan kualitas solusi. Berikut adalah pengimplementasian algoritma *backtracking* dalam program:

```python
# Algoritma backtracking untuk mencari solusi dengan optimal
def backtrack(self, current_playlist: List[Song]) -> bool:
    self.iterations += 1
    self.search_depth += 1
    self.max_depth_reached = max(self.max_depth_reached, self.search_depth)

    # Timeout untuk membatasi iterasi dan mencegah infinite loop
    if self.iterations > 15000:
        print(f"  Timeout: Mencapai batas maksimum iterasi ({self.iterations})")
        return False

    if self.iterations % 1000 == 0:
        print(f"  Progress: {self.iterations:,} iterasi, kedalaman ({self.search_depth}), solusi: {self.solutions_found}")

    try:
        if self._is_complete_solution(current_playlist):
            score = self._calculate_playlist_score(current_playlist)
            if score > self.best_score:
                self.best_score = score
                self.best_playlist = current_playlist.copy()
                self.solutions_found += 1
                print(f"  --> Solusi baru! Skor: {score:.2f}, Lagu: {len(current_playlist)}")
            return True

        # Pruning: cek validitas solusi parsial
        if not self._is_valid_partial_solution(current_playlist):
            self.pruned_branches += 1
            return False

        # Pruning: cek kemungkinan completion
        if not self._can_complete_solution(current_playlist):
            self.pruned_branches += 1
            return False

        # Early termination jika sudah dapat solusi yang cukup baik
        if self.best_score > 80 and self.solutions_found >= 3:
            return True
```

```
        # Generate kandidkdat
        candidates = self._get_candidates(current_playlist)

        max_candidates = min(15, len(candidates))
        candidates = candidates[:max_candidates]

        found_solution = False

        for song in candidates:
            # Tambahkan lagu ke playlist
            current_playlist.append(song)

            # Rekursi
            if self.backtrack(current_playlist):
                found_solution = True

            # Backtrack, hapus lagu dari playlist
            current_playlist.pop()

            # Early exit jika sudah dapat solusi sesuai
            if self.best_score > 85:
                break

        return found_solution

    finally:
        self.search_depth -= 1
```

Gambar 4.6. Implementasi algoritma backtracking

4) Strategi Optimasi

Optimasi pada program ini dilakukan dengan fungsi optimize(). Fungsi ini bekerja dengan memulai pencarian dari playlist kosong, kemudian melakukan pruning untuk menghindari cabang solusi tidak valid, serta mencatat metrik penting seperti jumlah iterasi, efisiensi pemangkasan cabang, skor terbaik, dan kedalaman maksimum pencarian. Berikut adalah pengimplementasiannya pada kode program:

```
# Optimisasi
def optimize(self, max_iterations: int = 15000, time_limit: int = 30) -> Tuple[List[Song], float, Dict]:
    print(f"Memulai optimasi playlist untuk mood '{self.constraints.target_mood}'...")
    print(f"Target: {self.constraints.target_duration//60}:{self.constraints.target_duration%60:02d} menit")
    print(f"Timeout: {time_limit} detik atau {max_iterations:,} iterasi")

    start_time = time.time()

    # Reset state
    self.iterations = 0
    self.pruned_branches = 0
    self.solutions_found = 0
    self.best_playlist = []
    self.best_score = -1
    self.search_depth = 0
    self.max_depth_reached = 0

    # Mulai proses backtracking dari playlist kosong
    try:
        import signal

        def timeout_handler(signum, frame):
            raise TimeoutError("Optimasi timeout")

        try:
            signal.signal(signal.SIGALRM, timeout_handler)
            signal.alarm(time_limit)
        except:
            pass

        self.backtrack([])

    except (KeyboardInterrupt, TimeoutError):
        print(f"\n  Optimasi dihentikan ({timeout atau user interrupt})")
    except Exception as e:
        print(f"\n  Error during optimization: {e}")
    finally:
        try:
            signal.alarm(0)
        except:
            pass

    end_time = time.time()
    execution_time = end_time - start_time

    # Kompilasi statistik hasil eksekusi
    stats = {
        'iterations': self.iterations,
        'pruned_branches': self.pruned_branches,
        'solutions_found': self.solutions_found,
        'execution_time': execution_time,
        'songs_explored': len(self.valid_songs),
        'best_score': self.best_score,
        'max_depth_reached': self.max_depth_reached,
        'pruning_efficiency': (self.pruned_branches / max(self.iterations, 1)) * 100
    }

    print(f"\nOptimasi selesai!")
    print(f"Waktu eksekusi: {execution_time:.2f} detik")
    print(f"Total iterasi: {stats['iterations']:,}")
    print(f"Cabang dipangkas: {stats['pruned_branches']:,} ({stats['pruning_efficiency']:.1f}%)")
    print(f"Solusi ditemukan: {stats['solutions_found']}")

    if self.best_playlist:
        print(f"Skor terbaik: {self.best_score:.2f}")
    else:
        print("Tidak ditemukan solusi yang memenuhi semua batasan!")
        print("Coba dengan kriteria yang lebih fleksibel.")

    return self.best_playlist, self.best_score, stats
```

Gambar 4.7. Pengimplementasian optimasi

V. PENJELASAN HASIL UJI COBA

Berikut merupakan hasil uji coba dari program pembentukan playlist musik personal yang telah dikembangkan. Pengujian dilakukan untuk melihat bagaimana sistem memenuhi berbagai batasan pengguna serta menilai efektivitas algoritma backtracking dalam menghasilkan playlist yang optimal dan sesuai preferensi pengguna.

A. Tampilan Awal Program

```
SISTEM OPTIMASI PLAYLIST MUSIK PERSONAL
Menggunakan Algoritma Backtracking dengan Multi-Constraint Satisfaction

Memuat database musik...
Database berhasil dimuat: 75 lagu tersedia

MENU UTAMA:
1. Lihat Informasi Database Musik
2. Buat Playlist Personal
3. Keluar

Pilih opsi (1-3):
```

Gambar 5.1. Tampilan awal pada program

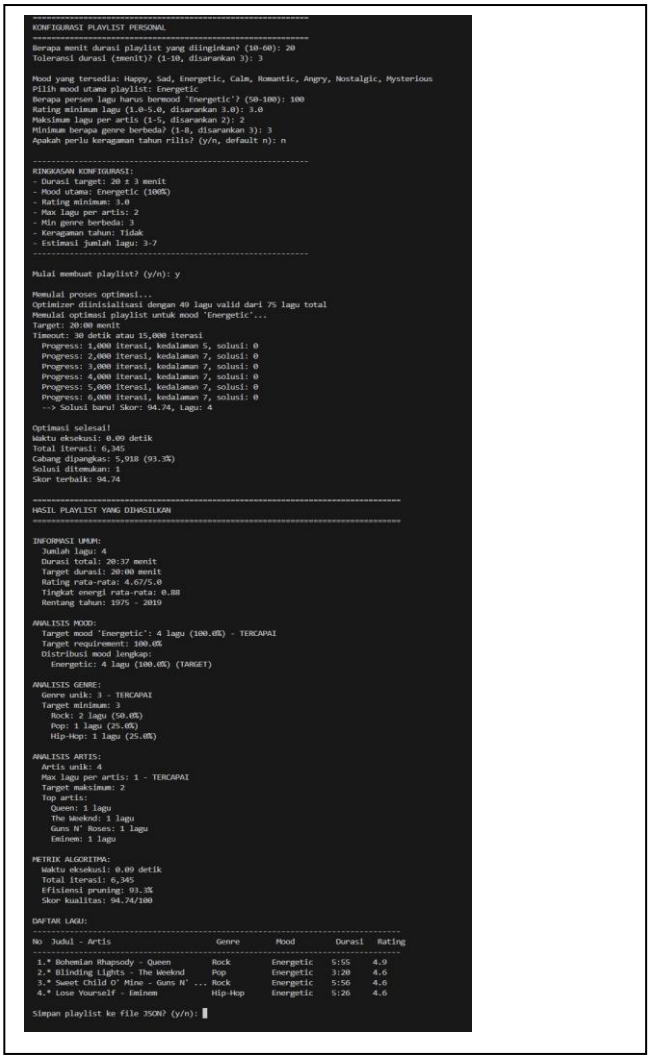
B. Tampilan Informasi Database Musik

```
Pilih opsi (1-3): 1

============================================================
INFORMASI DATASET MUSIK
============================================================
Total lagu dalam database: 75

Distribusi Genre:
  Pop: 10 lagu (13.3%)
  Rock: 5 lagu (6.7%)
  Hip-Hop: 5 lagu (6.7%)
  Electronic: 5 lagu (6.7%)
  R&B: 5 lagu (6.7%)
  Jazz: 5 lagu (6.7%)
  Classical: 5 lagu (6.7%)
  Country: 5 lagu (6.7%)
  Alternative: 5 lagu (6.7%)
  Reggae: 5 lagu (6.7%)
  Latin: 5 lagu (6.7%)
  Blues: 5 lagu (6.7%)
  Folk: 5 lagu (6.7%)
  Funk: 4 lagu (5.3%)
  Disco: 1 lagu (1.3%)

Distribusi Mood:
  Happy: 22 lagu (29.3%)
  Energetic: 16 lagu (21.3%)
  Sad: 10 lagu (13.3%)
  Nostalgic: 8 lagu (10.7%)
  Calm: 8 lagu (10.7%)
  Romantic: 7 lagu (9.3%)
  Mysterious: 2 lagu (2.7%)
  Angry: 2 lagu (2.7%)

Distribusi Era:
  1900-1959: 9 lagu (12.0%)
  1960-1979: 18 lagu (24.0%)
  1980-1999: 5 lagu (6.7%)
  2000-2009: 6 lagu (8.0%)
  2010-2024: 33 lagu (44.0%)

Kualitas Lagu:
  Rating rata-rata: 4.33/5.0
  Lagu berkualitas tinggi (≥4.0): 71 lagu (94.7%)
```

Gambar 5.2. Tampilan Informasi Database

C. Uji Coba 1 Pembuatan Playlist Personal

```
MENU UTAMA:
1. Lihat Informasi Database Musik
2. Buat Playlist Personal
3. Keluar

Pilih opsi (1-3): 2
```



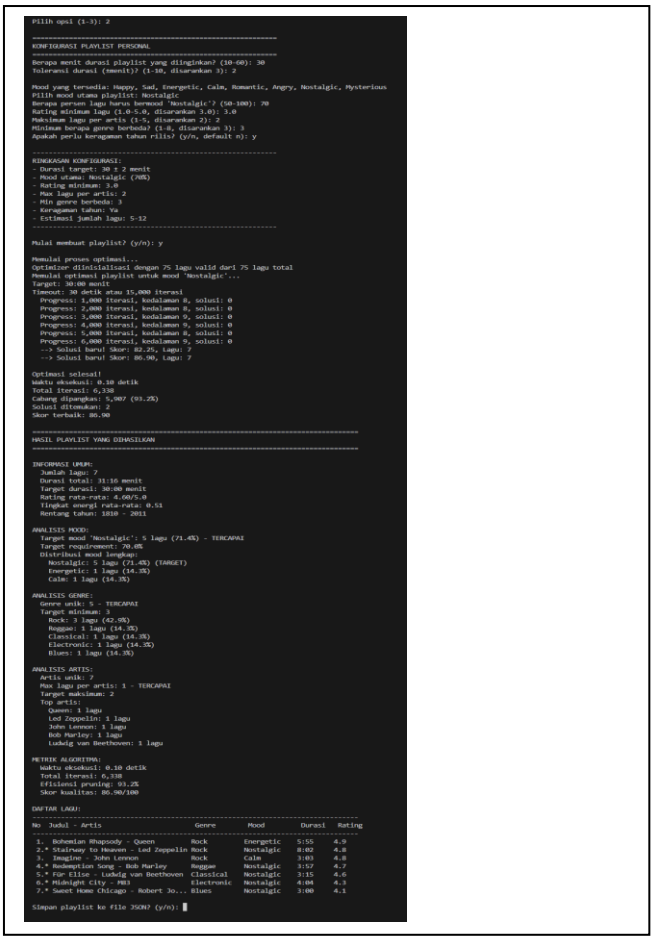
Gambar 5.3. Hasil uji coba 1

Gambar 5.4. Hasil penyimpanan uji coba 1 dalam file json

Pada uji coba 1, pengguna mengonfigurasi playlist dengan durasi target 20 menit ± 3 menit, mood utama Energetic dengan 100% lagu harus memiliki mood tersebut, serta batasan tambahan seperti rating minimum 3.0, maksimal 2 lagu per artis, dan minimal 3 genre berbeda. Dapat dilihat bahwa dari total 75 lagu dalam database, sistem berhasil menyaring 49 lagu valid yang memenuhi syarat awal. Proses optimasi dilakukan dengan menggunakan algoritma backtracking untuk menghasilkan playlist dengan 4 lagu, total durasi 20:37 menit, dan rating rata-rata 4.67, dengan seluruh lagu bermood *Energetic*. Playlist ini juga mencakup 3 genre unik dan 4 artis berbeda, sepenuhnya memenuhi seluruh batasan *hard* dan *soft constraints* yang dimasukkan pengguna.

Secara algoritmik, sistem melakukan 6.345 iterasi, namun berhasil memangkas 93,3% cabang tidak layak melalui strategi pruning, dan menyelesaikan pencarian hanya dalam 0,09 detik, dengan skor kualitas playlist mencapai 94.74/100. Hal ini menunjukkan bahwa algoritma berhasil menghasilkan sebuah playlist yang sangat relevan dan sesuai dengan keinginan pengguna.

D. Uji Coba 2 Pembuatan Playlist Personal



Gambar 5.5. Hasil uji coba 2

Pada uji coba 2, terlihat bahwa pengguna menginginkan playlist berdurasi 30 menit ±2 menit, dengan minimal 70% lagu bermood *Nostalgic*, rating lagu ≥ 3.0, maksimal 2 lagu per artis, dan minimal 3 genre berbeda serta keragaman tahun rilis. Dari 75 lagu, semua lolos tahap penyaringan awal. Setelah melalui 6.338 iterasi, sistem menemukan 2 solusi valid, dan menghasilkan playlist dengan 7 lagu berdurasi total 31:16 menit, mood *Nostalgic* mencapai 71.4%, serta mencakup 5 genre unik dan 7 artis berbeda. Proses berjalan sangat efisien, memangkas 93,2% cabang tidak layak dan memperoleh skor kualitas 86.90/100, hasil ini menunjukkan agoritma berhasil menghasil playlist yang relevan dan memenuhi kriteria yang diinginkan pengguna.

VI. KESIMPULAN

Penerapan algoritma backtracking dengan pendekatan *Multi-Constraint Satisfaction* terbukti efektif dalam menyelesaikan permasalahan pembentukan playlist musik personal yang kompleks dan multidimensional. Dengan kemampuan eksplorasi solusi secara sistematis dan penerapan teknik pruning yang efisien, algoritma ini mampu menghasilkan kombinasi lagu yang tidak hanya memenuhi batasan keras seperti durasi, jumlah lagu per artis, dan keunikan lagu, tetapi juga mengoptimalkan batasan lunak seperti konsistensi mood, keberagaman genre, dan rata-rata kualitas lagu. Hasil implementasi menunjukkan bahwa sistem mampu membentuk playlist yang lebih relevan dan adaptif dibandingkan pendekatan acak atau sistem rekomendasi tradisional.

Melalui desain algoritma yang dirancang dengan memperhatikan dinamika preferensi pengguna serta karakteristik data musik yang heterogen, sistem ini dapat berfungsi secara fleksibel pada berbagai skenario. Penelitian ini juga membuka peluang untuk pengembangan lebih lanjut, seperti integrasi dengan data perilaku pengguna secara real-time atau penerapan pada skala platform streaming besar. Dengan demikian, sistem optimasi pembentukan playlist musik berbasis *backtracking* dan *Multi-Constraint Satisfaction* ini dapat memberikan kontribusi nyata dalam menciptakan pengalaman musik yang lebih personal, terkontrol, dan relevan dengan kebutuhan pengguna.

LAMPIRAN

Berikut merupakan link dari github repository untuk mengakses kode program: https://github.com/nataliadesiany/Makalah-Stima.git.

Berikut merupakan link video YouTube penjelasan makalah: https://youtu.be/6-A8Eqk5Dqc?si=GxWIFuZjwub0p1Cz

UCAPAN TERIMA KASIH

Pertama-tama, penulis ingin mengucapkan puji syukur yang sebesar-besarnya kepada Tuhan Yang Maha Esa atas segala rahmat dan kasih karunia-Nya, sehingga makalah ini dapat diselesaikan dengan baik dan tepat waktu. Penulis menyampaikan rasa terima kasih kepada Bapak Monterico Adrian, S.T., M.T., selaku dosen mata kuliah IF2211 Strategi Algoritma, atas seluruh ilmu, bimbingan, dan arahannya yang sangat berharga selama satu semester ini. Penulis juga menyampaikan terima kasih yang mendalam kepada kedua orang tua yang senantiasa memberikan semangat, doa, serta motivasi selama proses penyusunan makalah ini. Akhir kata, penulis mengucapkan apresiasi kepada para pembaca dan berharap semoga isi serta pembahasan dalam makalah ini dapat memberikan manfaat, memperluas wawasan, dan mendorong eksplorasi lebih lanjut di bidang yang dibahas.

REFERENCES

[1] R. Munir, "Algoritma backtracking (2025) bagian 1," 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/15-Algoritma-backtracking-(2025)-Bagian1.pdf. (Diakses 22 Juni 2025).

[2] R. Munir, "Algoritma backtracking (2025) bagian 2," 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/16-Algoritma-backtracking-(2025)-Bagian2.pdf. (Diakses 22 Juni 2025).

[3] A. Khansa and T. Sutabri, "Implementation of the backtracking algorithm for optimizing work shift scheduling," International Journal Scientific and Professional, vol. 4, no. 2, pp. 501-508, 2025. https://www.researchgate.net/profile/Ainna-Khansa/publication/392030941_Implementation_of_the_Backtracking_Algorithm_for_Optimizing_Work_Shift_Scheduling/links/68314873d1054b0207f20508/Implementation-of-the-Backtracking-Algorithm-for-Optimizing-Work-Shift-Scheduling.pdf. (Diakses 22 Juni 2025).

[4] M. Schedl, H. Zamani, C. W. Chen, Y. Deldjoo, and M. Elahi, "Current challenges and visions in music recommender systems research," International Journal of Multimedia Information Retrieval, vol. 7, pp. 95-116, 2018. https://sci-hub.se/10.1007/s13735-018-0154-2. (Diakses 22 Juni 2025).

[5] S. M. A. France, "A combinatorial approach to content-based music selection," 1999. https://web.archive.org/web/20030327033803id_/http://www.csl.sony.fr:80/downloads/papers/1999/pachet99c.pdf. (Diakses 22 Juni 2025).

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, 24 Juni 2025

Natalia Desiany Nursimin
13523157