Algoritma Genetika dalam Penjadwalan Mata Kuliah: Eksplorasi Metode Crossover, Mutasi, dan Seleksi Terbaik

[1]Triyono*, [2]Kusrini
[1]Pascasarjana, Magister PJJ Informatika, Universitas Amikom, Yogyakarta, Indonesia
[2]Universitas Amikom, Yogyakarta, Indonesia
[1] triyono@students.amikom.ac.id* , [2]kusrini@amikom.ac.id

ABSTRAK

Penjadwalan mata kuliah adalah masalah kompleks yang dihadapi institusi pendidikan tinggi, memerlukan metode efisien dan efektif untuk menempatkan mata kuliah dengan mempertimbangkan berbagai kendala. Penelitian ini mengevaluasi pengaruh kombinasi metode crossover, mutasi, dan seleksi dalam algoritma genetika terhadap penjadwalan mata kuliah. Metode yang diuji meliputi Kombinasi Single Point, Cycle, dan Uniform Crossover; Binary dan Swap Mutation; serta Tournament, Ranking, dan Truncation Selection. Hasil penelitian menunjukkan bahwa metode Kombinasi Single Swap Tournament memiliki waktu eksekusi tercepat, sementara Kombinasi Cycle Swap Truncation paling lambat. Pada generasi pertama, Kombinasi Uniform Binary Ranking menunjukkan performa terbaik, sedangkan Kombinasi Single Binary Tournament terburuk. Pada generasi terakhir, Kombinasi Single Binary Truncation mencapai performa terbaik, sementara Kombinasi Single Swap Tournament dan Kombinasi Cycle Swap Tournament terburuk. Kombinasi Single Binary Tournament dan Kombinasi Single Swap Tournament menunjukkan performa yang konsisten dengan perbedaan signifikan dalam waktu eksekusi. Untuk aplikasi praktis, Kombinasi Single Swap Tournament direkomendasikan karena efisiensi waktunya. Penelitian lanjutan dapat mengembangkan kombinasi metode untuk meningkatkan performa keseluruhan, dan pengembangan teori baru dapat mengkaji faktor-faktor yang mempengaruhi efisiensi dan efektivitas algoritma genetika dalam penjadwalan kuliah.

Kata Kunci: Algoritma Genetika, Penjadwalan Mata Kuliah, Kombinasi Metode, Crossover, Mutasi.

PENDAHULUAN

Penjadwalan mata kuliah merupakan masalah kompleks yang sering dihadapi institusi pendidikan tinggi. Proses ini melibatkan penempatan mata kuliah pada waktu dan tempat yang tersedia, dengan memperhatikan kendala seperti ketersediaan dosen, kapasitas ruangan, dan preferensi mahasiswa. Penyelesaian manual sering memakan waktu lama dan rentan terhadap

kesalahan, sehingga diperlukan pendekatan yang lebih efisien dan efektif.

Algoritma genetika (AG) adalah metode komputasi evolusioner yang sering digunakan untuk menyelesaikan masalah optimasi, termasuk penjadwalan mata kuliah. AG meniru proses seleksi alam dengan mekanisme seperti crossover, mutasi, dan seleksi, yang memungkinkan eksplorasi ruang solusi secara luas dan menemukan solusi optimal.

Studi sebelumnya oleh Rudi et al. (2023) menunjukkan bahwa probabilitas crossover antara 0,85 hingga 0,95 menghasilkan waktu komputasi tercepat. Penelitian Fajar et al. (2023) menemukan bahwa kombinasi crossover dua-titik dengan probabilitas mutasi 3% paling efektif dalam penjadwalan mata kuliah. Hasil ini menunjukkan pentingnya pemilihan parameter yang tepat dalam AG.

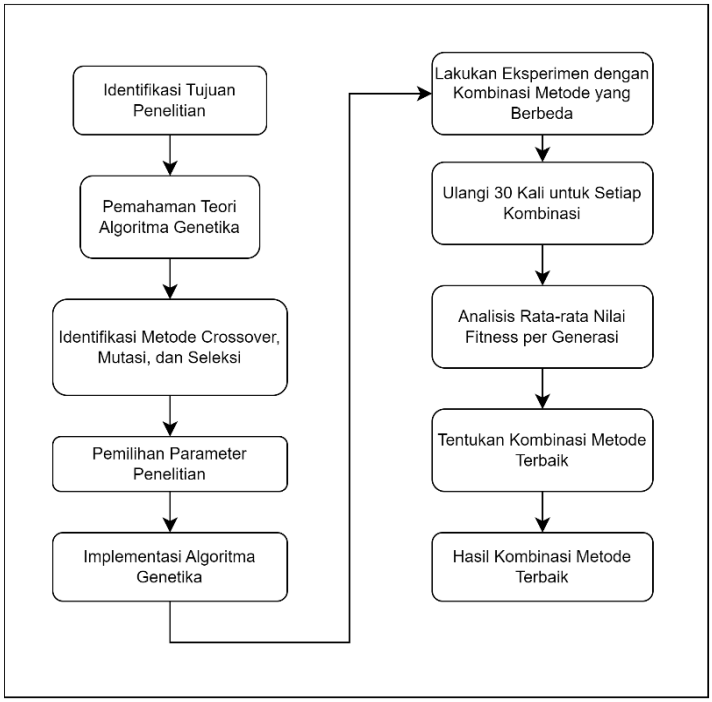
Penelitian ini bertujuan mengkaji pengaruh kombinasi metode crossover, mutasi, dan seleksi terhadap kinerja AG dalam penjadwalan mata kuliah. Metode yang diuji termasuk single point, cycle, dan uniform crossover; binary dan swap mutation; serta tournament, ranking, dan truncation selection. Setiap kombinasi diuji 30 kali, dan nilai fitness rata-rata dianalisis untuk menentukan metode paling efektif.

Diharapkan penelitian ini memberikan wawasan mendalam mengenai dampak berbagai metode tersebut terhadap performa AG, serta meningkatkan efisiensi dan efektivitas penjadwalan mata kuliah di institusi pendidikan tinggi.

METODE

1. Desain Penelitian

Penelitian ini menggunakan pendekatan kuantitatif dengan desain eksperimental. Fokus utama dari penelitian ini adalah untuk mengeksplorasi pengaruh berbagai kombinasi metode crossover, mutasi, dan seleksi dalam algoritma genetika untuk mengoptimalkan penjadwalan mata kuliah. Kombinasi metode yang diuji meliputi tiga metode crossover, dua metode mutasi, dan tiga metode seleksi.



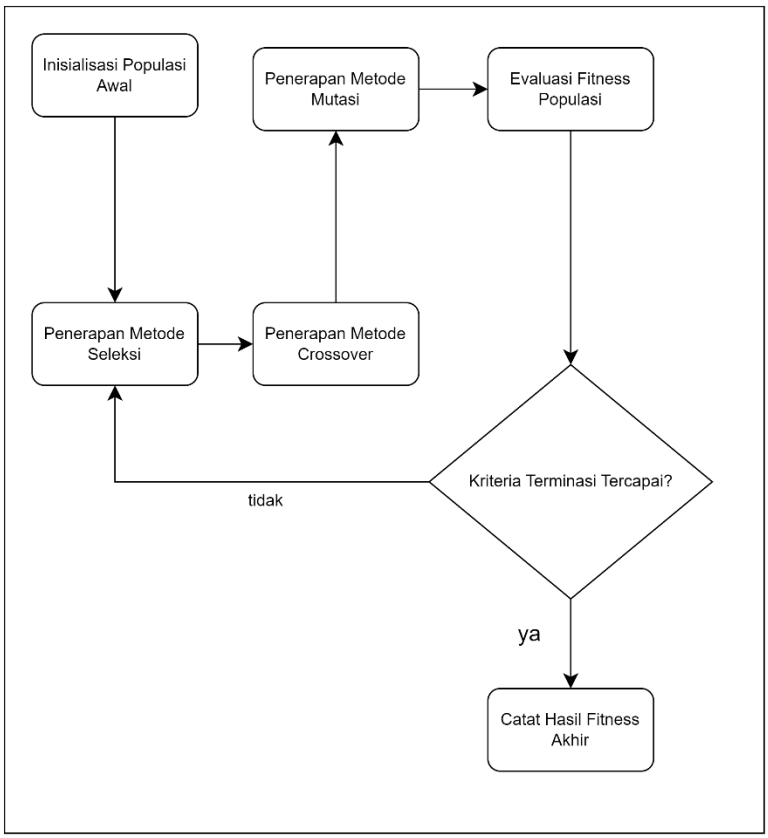
Gambar 1 Bagan alur penelitian

Pada gambar 1 dijelaskan untuk penelitian ini dimulai dengan mengidentifikasi tujuan utama, yaitu mengkaji pengaruh berbagai kombinasi metode dalam algoritma genetika terhadap penjadwalan mata kuliah. Selanjutnya, dilakukan pemahaman terhadap teori dasar algoritma genetika dan komponen utamanya, seperti crossover, mutasi, dan seleksi. Setelah itu, dipilih metode *crossover (single point, cycle, uniform)*, mutasi (*binary, swap),* dan seleksi *(tournament, ranking, truncation)* yang akan diuji. Parameter penelitian seperti ukuran populasi, probabilitas crossover dan mutasi, serta jumlah generasi ditentukan. Model penjadwalan mata kuliah kemudian dikembangkan dan dioptimasi menggunakan algoritma genetika. Implementasi algoritma meliputi inisialisasi populasi awal, penerapan metode seleksi, crossover, dan mutasi, evaluasi nilai fitness populasi, serta penentuan kriteria terminasi. Setiap kombinasi metode diuji sebanyak 30 kali untuk memastikan konsistensi hasil.

2 Metode Algoritma Genetika

Algoritma genetika adalah teknik pencarian dan optimasi yang

terinspirasi oleh prinsip-prinsip seleksi alam. Dalam penelitian ini, algoritma genetika diimplementasikan dengan tahapan pada gambar 2:



Gambar 2 Diagram alir Algoritma Genetika

Proses penelitian seperti pada gambar 2 dimulai dengan inisialisasi populasi awal, di mana solusi penjadwalan mata kuliah awal dibentuk. Langkah berikutnya adalah penerapan metode seleksi, di mana individu dari populasi dipilih untuk proses crossover berdasarkan kriteria seleksi yang telah ditentukan. Setelah seleksi, metode crossover diterapkan untuk menghasilkan individu baru melalui kombinasi gen dari pasangan induk yang terpilih. Selanjutnya, metode mutasi diterapkan pada individu baru untuk memperkenalkan variasi dan meningkatkan keberagaman genetik dalam populasi. Setelah mutasi, nilai fitness dari seluruh populasi dievaluasi untuk menilai kualitas setiap solusi penjadwalan. Pada titik ini, diperiksa apakah kriteria

terminasi telah tercapai, seperti jumlah generasi tertentu atau nilai fitness yang diinginkan. Jika kriteria terminasi belum tercapai, proses kembali ke langkah penerapan metode seleksi dan diulangi. Jika kriteria terminasi tercapai, hasil fitness akhir dicatat sebagai bagian dari hasil penelitian.

2.1 Inisialisasi Populasi

Inisialisasi populasi merupakan langkah awal dalam algoritma genetika yang bertujuan untuk menghasilkan sekumpulan solusi awal (individu) yang akan dievolusi melalui generasi berikutnya. Pada konteks penjadwalan, setiap individu dalam populasi mewakili satu kemungkinan jadwal yang terdiri dari alokasi waktu dan sumber daya untuk berbagai kelas atau kegiatan

Penentuan Ukuran Populasi:

$$P = \{S_1, S_2, \dots, S_N\}$$

Dimana:

a) P Adalah populasi
b) N adalah ukuran populasi, yaitu jumlah total solusi dalam populasi.
c) S_i adalah solusi ke-i dalam populasi.

Pembangkitan Solusi:

Setiap solusi S_i direpresentasikan sebagai jadwal dengan k kelas:

$$S_i = \{C_{i1}, C_{i2}, \dots, C_{iK}\}$$

Dimana:

a) S_i Adalah solusi ke- i
b) K adalah jumlah total kelas yang harus dijadwalkan..
c) C_{ij} adalah kelas ke- j dari solusi ke- i, yang mencakup informasi waktu, ruangan, dan dosen.

Pembangkitan Jadwal Kelas:

$$C_{ij} = \{Waktu_{ij}, Ruangan_{ij}, Dosen_{ij}\}$$

a) **Waktu** ($Waktu_{ij}$): Diambil secara acak dari slot waktu yang tersedia.

b) **Ruangan** ($Ruangan_{ij}$): Diambil secara acak dari ruangan yang tersedia.
c) **Dosen** ($Dosen_{ij}$): Diambil secara acak dari dosen yang tersedia.

2.2 Evaluasi Fitness

Fungsi fitness adalah cara untuk mengukur seberapa baik suatu solusi (individu) dalam populasi memenuhi tujuan atau menyelesaikan masalah yang diberikan

$$Fitness(x) = \omega_1 \times \frac{1}{1 + Conflict(x)} + \omega_2 \times \frac{1}{1 + Delay(x)} + \omega_3 \times ResourceUsage(x)$$

Di mana:

1. $\omega_1, \omega_2, \omega_3$ adalah bobot yang menunjukkan kepentingan relatif masing-masing faktor. Bobot ini digunakan untuk mengatur seberapa besar pengaruh masing-masing komponen dalam nilai fitness total.
2. $Conflict(x)$ mengukur jumlah konflik dalam jadwal x. Konflik dapat berupa dua kelas yang dijadwalkan pada waktu yang sama di ruangan yang sama atau dua kelas yang dijadwalkan pada waktu yang sama oleh profesor yang sama.
3. $Delay(x)$ mengukur total keterlambatan dalam jadwal x. Keterlambatan dapat berupa jeda yang terlalu lama antara kelas yang diajarkan oleh profesor yang sama atau kelas yang terlalu jauh jaraknya dari waktu yang diinginkan.
4. $ResourceUsage(x)$ mengukur tingkat pemanfaatan sumber daya dalam jadwal x. Ini dapat mencakup efisiensi penggunaan ruangan, laboratorium, dan waktu yang tersedia.

Rumus fitness ini dirancang untuk:

1. **Minimalkan Konflik**: Semakin sedikit konflik dalam jadwal, semakin tinggi nilai $\frac{1}{1+Conflict(x)}$.
2. **Minimalkan Keterlambatan**: Semakin sedikit keterlambatan dalam jadwal, semakin tinggi nilai $\frac{1}{1+Delay(x)}$.
3. **Maksimalkan Pemanfaatan Sumber Daya**: Semakin efisien penggunaan sumber daya, semakin tinggi nilai $ResourceUsage(x)$.

2.3 Seleksi

Metode seleksi digunakan untuk memilih individu yang akan menghasilkan generasi berikutnya. Tiga metode seleksi yang diuji dalam penelitian ini adalah:

1. **Tournament Selection:** Memilih sekelompok individu secara acak dan memilih individu terbaik dari kelompok tersebut.

Pemilihan K individu:

$$T = \{I_1, I_2, \ldots, I_k\}$$

Di mana T adalah himpunan individu yang dipilih secara acak dari populasi dan k adalah ukuran turnamen.

Penentuan pemenang:

$$I_{best} = \frac{\arg\max fitness(I)}{I \in T}$$

Di mana I_{best} adalah individu dengan nilai fitness tertinggi dalam himpunan T.

2. **Ranking Selection:** Mengurutkan individu berdasarkan nilai fitness mereka dan memilih individu dengan probabilitas yang sesuai dengan peringkatnya.

$$P(i) = \frac{2(N - Rank(i) + 1)}{N(N+1)}$$

Di mana:

a) $P(i)$ adalah probabilitas seleksi individu dengan peringkat i.
b) N adalah jumlah total individu dalam populasi.
c) $Rank(i)$ adalah peringkat individu i (dengan peringkat 1 untuk individu dengan fitness terendah dan N untuk individu dengan fitness tertinggi).

3. **Truncation Selection:** Memilih sejumlah individu teratas berdasarkan nilai fitness dan menggunakan individu tersebut untuk reproduksi.

Urutkan Individu Berdasarkan Fitness:

$$\{I_1, I_2, \ldots, I_N\} \rightarrow \{I'_1, I'_2, \ldots, I'_M\}$$

Di mana:
a) $\{I_1, I_2, \ldots, I_N\}$ adalah populasi awal.
b) $\{I'_1, I'_2, \ldots, I'_M\}$adalah populasi yang diurutkan berdasarkan nilai fitness, dengan I1′I'_1I1′ memiliki fitness tertinggi.

Menentukan Proporsi p dari Populasi yang akan dipilih:

$$p = \frac{M}{N}$$

Di mana:
a) M adalah jumlah individu yang akan dipilih untuk reproduksi.
b) N adalah ukuran total populasi.
c) p adalah proporsi populasi yang akan dipilih.

Pilih Individu Teratas untuk Reproduksi:

$$P_{selected} = \{I'_1, I'_2, \ldots, I'_M\}$$

Di mana:
a) $P_{selected}$ adalah subset dari populasi yang dipilih untuk reproduksi.
b) I'_i adalah individu dengan peringkat i tertinggi berdasarkan fitness.

2.4 Crossover

Tiga metode crossover yang diuji adalah:

1. **Single Point Crossover:** Memilih satu titik di sepanjang kromosom dan menukar bagian kromosom di atas titik tersebut antara dua individu.

Pilih Titik Crossover:

$$crossover_{point} = random\_integer(1, L-1)$$

Dimana L adalah panjang kromosom

Bentuk Keturunan Baru:
Misalkan dua orang tua P_1 dan P_2 dengan kromosom sebagai berikut:

$$P_1 = \{p_{1l}, p_{1l}, \dots, p_{1L}\}$$
$$P_2 = \{p_{2l}, p_{2l}, \dots, p_{2L}\}$$

Jika titik crossover dipilih pada posisi k, maka dua keturunan baru C_1 dan C_2 dibentuk sebagai berikut:

$$C_1 = \{p_{12}, p_{12}, \dots, p_{1k}, p_{2(k+1)}, p_{2(k+2)}, \dots, p_{2L}\}$$

$$C_2 = \{p_{21}, p_{22}, \dots, p_{2k}, p_{1(k+1)}, p_{1(k+2)}, \dots, p_{1L}\}$$

2. **Cycle Crossover:** Menghasilkan individu baru dengan mempertahankan siklus gen dari kedua orang tua.

Identifikasi Siklus:

$$g_i \rightarrow g_j \rightarrow g_k \rightarrow \cdots \rightarrow g_m \rightarrow g_i$$

dimulai dari gen g_i, bergerak melalui gen-gen g_j, g_k dan kembali ke gen awal g_i untuk menyelesaikan siklus.

Bentuk Jadwal Keturunan C_1:

$$C_1(i) = \begin{cases} P_1(i) \; jika \; i \in siklus \\ P_2(i) \; jika \; i \notin siklus \end{cases}$$

Bentuk Jadwal Keturunan C_2:

$$C_2(i) = \begin{cases} P_2(i) \; jika \; i \in siklus \\ P_1(i) \; jika \; i \notin siklus \end{cases}$$

3. **Uniform Crossover:** Menghasilkan individu baru dengan memilih gen dari kedua orang tua secara acak.

Pemilihan Acak Gen:

Untuk setiap gen i dalam kromosom keturunan C, pilih gen dari P_1 atau P_2 dengan probabilitas p.

$$C_1(i) = \begin{cases} P_1(i) \; dengan \; probabilitas \; p \\ P_2(i) \; dengan \; probabilitas \; 1-p \end{cases}$$

2.5 Mutasi

Dua metode mutasi yang diuji adalah:

1. **Binary Mutation:** Mengubah nilai gen menjadi 0 atau 1 dengan probabilitas tertentu.

Mutasi gen :

Untuk setiap gen i dalam kromosom P, ubah nilai gen dengan probabilitas p_m:

$$P'(i) = \begin{cases} 1 - P(i) \; dengan \; probabilitas \; p_m \\ P_1(i) \; dengan \; probabilitas \; 1-p_m \end{cases}$$

2. **Swap Mutation**: Menukar posisi dua gen dalam kromosom.

Mutasi gen:

Untuk setiap kromosom P, tukar posisi dua gen i dan j secara acak dengan probabilitas p_m:

$P'(i) = P(j)$ dan $P'(j) = P(i)$

3. Prosedur Eksperimen

Setiap kombinasi metode crossover, mutasi, dan seleksi diuji sebanyak 30 kali untuk memastikan konsistensi hasil. Prosedur eksperimen meliputi langkah-langkah berikut:

1. **Inisialisasi:** Membentuk populasi awal.
2. **Evaluasi:** Menghitung nilai fitness untuk setiap individu.
3. **Seleksi:** Memilih individu untuk crossover menggunakan metode yang dipilih.
4. **Crossover:** Menerapkan metode crossover untuk menghasilkan individu baru.
5. **Mutasi:** Menerapkan metode mutasi untuk memperkenalkan variasi.
6. **Evaluasi Fitness:** Menghitung nilai fitness untuk individu baru.
7. **Iterasi:** Mengulangi langkah 3-6 hingga mencapai kriteria terminasi

4. Analisis Data

Setelah eksperimen selesai, data yang diperoleh akan

dianalisis dengan menghitung rata-rata nilai fitness per generasi untuk setiap kombinasi metode. Hasil analisis ini digunakan untuk menentukan kombinasi metode yang paling efektif dalam mengoptimalkan penjadwalan mata kuliah. Analisis dilakukan menggunakan perangkat lunak statistik untuk memastikan keakuratan dan objektivitas hasil.

HASIL DAN PEMBAHASAN

Penelitian ini dilakukan untuk menentukan kombinasi metode algoritma genetika yang paling efektif dalam mengoptimalkan penjadwalan mata kuliah. Berikut ini adalah penjabaran rinci dari setiap langkah dalam metode penelitian dan hasil analisisnya.

Dataset yang digunakan untuk melakukan penjadwalan mata kuliah mencakup beberapa elemen kunci, yaitu ruangan, slot waktu, profesor, mata kuliah, dan departemen. Berikut adalah penjelasan dari setiap elemen tersebut:

1. Ruangan

Tabel 1 Data Ruangan

No	Nama Ruangan	Laboratorium
1	R1	Tidak
2	R2	Tidak
3	R3	Tidak
4	R4	Tidak
5	R5	Tidak
6	R6	Tidak
7	R7	Tidak
8	R8	Tidak
9	R9	Tidak
10	R10	Tidak
11	R11	Tidak
12	Lab 1	Ya
13	Lab 2	Ya

Pada tabel terdapat 13 ruangan yang tersedia untuk digunakan dalam penjadwalan, yang terdiri dari 11 ruangan reguler dan 2 laboratorium. Setiap ruangan memiliki karakteristik tersendiri

2. Alokasi Waktu

Penjadwalan dilakukan dalam 14 alokasi waktu yang tersedia, mencakup hari Senin hingga Jumat dengan berbagai rentang waktu, seperti pada tabel 2

Tabel 2 Data Alokasi Waktu

No	ID Slot Waktu	Waktu
1	T1	Senin 08:00 - 10:30
2	T2	Senin 10:30 - 13:00
3	T3	Senin 13:30 - 16:00
4	T4	Selasa 08:00 - 10:30
5	T5	Selasa 10:30 - 13:00
6	T6	Selasa 13:30 - 16:00
7	T7	Rabu 08:00 - 10:30
8	T8	Rabu 10:30 - 13:00
9	T9	Rabu 13:30 - 16:00
10	T10	Kamis 08:00 - 10:30
11	T11	Kamis 10:30 - 13:00
12	T12	Kamis 13:30 - 16:00
13	T13	Jumat 08:00 - 10:30

3. Dosen

Penelitian ini melibatkan empat orang dosen yang akan mengajar mata kuliah yang ditawarkan. Tabel 3 adalah data dosen yang terlibat

Tabel 3 Data Dosen

No	ID Dosen	Nama
1	P1	Dwi Hartanti
2	P2	Nurchim
3	P3	Joni Maulindar
4	P4	Nibras

4. Mata kuliah

Terdapat 10 mata kuliah yang ditawarkan oleh berbagai dosen, dengan beberapa di antaranya merupakan mata kuliah laboratorium yang memiliki preferensi waktu dan ruangan tertentu. Data mata kuliah disajikan dalam tabel 4 berikut

Tabel 4 Data Mata Kuliah

No	ID Mata Kuliah	Kode Mata Kuliah	ID Profesor	Mata Kuliah Laboratorium	Slot Waktu Preferensi	Ruangan Preferensi
1	C1	DS101	P1, P2	Ya	T2, T1	R12
2	C2	DS102	P1, P2, P4	Ya	T2, T3	Lab 1
3	C3	ML201	P3, P4	Ya	T3, T4	R7, Lab 1
4	C4	ML202	P3, P4	Tidak	T3, T4, T5	
5	C5	AI301	P1, P4	Tidak		
6	C6	BD401	P2, P3	Tidak	T1, T6	R7
7	C7	BD402	P2, P3	Tidak		
8	C8	NLP101	P3, P4	Tidak	T3, T5	R10
9	C9	CV101	P2, P3	Ya		
10	C10	DS201	P1, P2	Ya	T2, T4, T5	Lab 1

1. Inisialisasi Populasi Awal

Populasi awal diinisialisasi dengan 10 individu, masing-masing merepresentasikan solusi penjadwalan yang berbeda. Setiap individu dihasilkan secara acak untuk memastikan keragaman solusi awal. Berikut pseudocode untuk populasi awal:

Pseudocode untuk fungsi pada populasi awal:

```
KELAS Population:
   FUNGSI __init__(self, size, data):
     INISIALISASI atribut size dengan parameter size
     INISIALISASI atribut schedules sebagai list yang berisi
jadwal yang diinisialisasi dengan data yang diberikan, sebanyak
ukuran populasi
       UNTUK setiap elemen dalam range dari 0 hingga size - 1:
        TAMBAHKAN Schedule(data).initialize() ke dalam list
schedules

    FUNGSI get_schedules(self):
        KEMBALIKAN atribut schedules
```

Penjelasan Pseudocode

1. **Kelas `Population`**:
   a. Mewakili populasi jadwal dalam sistem penjadwalan.
   b. **Atribut**:
      i. `size`: Ukuran populasi.
      ii. `schedules`: Daftar jadwal dalam populasi.
2. **Fungsi `__init__`**:
   a. Menginisialisasi objek `Population`.
   b. **Parameter**:
      i. `size`: Ukuran populasi yang diinginkan.
      ii. `data`: Data untuk menginisialisasi setiap jadwal.
3. **Proses**:
   a. Menetapkan `size` sesuai parameter.
   b. Mengisi `schedules` dengan daftar jadwal yang diinisialisasi menggunakan `data`, sebanyak `size`.
4. **Fungsi `get_schedules`**:
   a. Mengembalikan daftar jadwal (`schedules`) dalam populasi.

2. Penerapan Metode Seleksi

Dalam algoritma genetika, seleksi individu merupakan langkah penting untuk menentukan individu mana yang akan digunakan dalam proses crossover guna membentuk generasi berikutnya. Penelitian ini menguji dua metode seleksi, yaitu

Tournament Selection dan *Ranking Selection*, untuk memilih individu berdasarkan performa fitness mereka.

Pseudocode untuk fungsi pada metode *selection*:

```
FUNGSI tournament_selection(populasi):
    INISIALISASI tournament_pop sebagai populasi kosong
dengan data dari populasi input
    UNTUK setiap i DARI 1 HINGGA
TOURNAMENT_SELECTION_SIZE:
        PILIH jadwal acak dari populasi
        TAMBAH jadwal ke tournament_pop

    URUTKAN tournament_pop berdasarkan fitness dari yang
tertinggi
    KEMBALIKAN jadwal dengan fitness tertinggi dari
tournament_pop

FUNGSI ranking_selection(populasi):
    URUTKAN jadwal dalam populasi berdasarkan fitness dari
yang tertinggi
    HITUNG rank_sum sebagai jumlah dari urutan 1 sampai
POPULATION_SIZE
    PILIH angka acak antara 0 dan rank_sum

    INISIALISASI current sebagai 0
    UNTUK setiap i, jadwal dalam ranked_schedules:
        TAMBAH (POPULATION_SIZE - i) ke current
        JIKA current lebih besar dari angka acak:
            KEMBALIKAN jadwal saat ini

FUNGSI truncation_selection(populasi):
    URUTKAN jadwal dalam populasi berdasarkan fitness dari
yang tertinggi
    PILIH top_schedules sebagai daftar jadwal dengan
fitness terbaik sebanyak TRUNCATION_SIZE
    PILIH jadwal acak dari top_schedules
    KEMBALIKAN jadwal yang dipilih
```

Penjelasan Pseudocode

1. **Fungsi `tournament_selection`:**

a) Membuat populasi turnamen kosong.
b) Memilih `TOURNAMENT_SELECTION_SIZE` jadwal acak dari populasi input dan menambahkannya ke populasi turnamen.
c) Mengurutkan populasi turnamen berdasarkan fitness.
d) Mengembalikan jadwal dengan fitness tertinggi dari populasi turnamen.

2. **Fungsi `ranking_selection`:**

a) Mengurutkan semua jadwal dalam populasi berdasarkan

fitness.

b) Menghitung `rank_sum` sebagai jumlah dari urutan 1 hingga ukuran populasi.
c) Memilih angka acak dalam rentang `rank_sum`.
d) Menggunakan angka acak untuk menentukan jadwal yang dipilih berdasarkan peringkat kumulatif.

3. **Fungsi `truncation_selection`:**

a) Mengurutkan jadwal dalam populasi berdasarkan fitness.
b) Memilih jadwal terbaik sebanyak `TRUNCATION_SIZE`.
c) Memilih salah satu dari jadwal terbaik secara acak dan mengembalikannya.

3. Penerapan Metode *Crossover*

Metode crossover digunakan untuk menggabungkan dua individu terpilih dan menghasilkan keturunan baru. Metode yang diuji meliputi *single point crossover, cycle crossover*, dan *uniform crossover*.

Pseudocode untuk fungsi pada metode *crossover*:

```
FUNGSI single_point_crossover(schedule1, schedule2):
    DATA = schedule1.data
    INISIALISASI crossover_schedule sebagai jadwal baru
dengan DATA
    parent1_classes = jadwal kelas dari schedule1
    parent2_classes = jadwal kelas dari schedule2

    crossover_point = pilih titik acak antara 0 dan
panjang kelas

    UNTUK setiap cls_id DARI 0 HINGGA panjang kelas - 1:
        JIKA cls_id lebih kecil dari crossover_point:
            ATUR ruangan, waktu, dan profesor
crossover_schedule dari parent2_classes
        SEBALIKNYA:
            ATUR ruangan, waktu, dan profesor
crossover_schedule dari parent1_classes

    KEMBALIKAN crossover_schedule

FUNGSI cycle_crossover(schedule1, schedule2):
    DATA = schedule1.data
    INISIALISASI crossover_schedule sebagai jadwal baru
dengan DATA
    parent1_classes = jadwal kelas dari schedule1
    parent2_classes = jadwal kelas dari schedule2
    size = panjang kelas
```

```
    child_classes = list kosong dengan ukuran size
    visited = list kosong dengan ukuran size diisi dengan
False
    cycle_num = 0

    SELAMA ada elemen None dalam child_classes:
        index = cari indeks pertama yang belum dikunjungi
dalam visited
        cycle_indices = list kosong

        SELAMA True:
            TAMBAH index ke cycle_indices
            SET visited[index] = True
            next_class_id = ID kelas dari parent2_classes
JIKA cycle_num genap, SEBALIKNYA dari parent1_classes
            index = cari indeks di parent1_classes atau
parent2_classes dengan ID kelas sesuai next_class_id

            JIKA index ada dalam cycle_indices:
                KELUAR dari loop

        UNTUK setiap i dalam cycle_indices:
            JIKA cycle_num genap:
                ATUR kelas di child_classes[i] dan
crossover_schedule dari parent1_classes
            SEBALIKNYA:
                ATUR kelas di child_classes[i] dan
crossover_schedule dari parent2_classes

        cycle_num += 1

    KEMBALIKAN crossover_schedule

FUNGSI uniform_crossover(schedule1, schedule2):
    DATA = schedule1.data
    INISIALISASI crossover_schedule sebagai jadwal baru
dengan DATA
    parent1_classes = jadwal kelas dari schedule1
    parent2_classes = jadwal kelas dari schedule2

    UNTUK setiap cls_id DARI 0 HINGGA panjang kelas - 1:
        JIKA angka acak kurang dari 0.5:
            ATUR ruangan, waktu, dan profesor
crossover_schedule dari parent1_classes
        SEBALIKNYA:
            ATUR ruangan, waktu, dan profesor
crossover_schedule dari parent2_classes

    KEMBALIKAN crossover_schedule
```

Penjelasan Pseudocode

1. **Fungsi `single_point_crossover`:**
   a) Menginisialisasi jadwal baru untuk hasil crossover.

b) Memilih titik crossover acak.
c) Untuk setiap kelas, menentukan apakah akan mengambil detail dari jadwal parent1 atau parent2 berdasarkan titik crossover.
d) Mengembalikan jadwal hasil crossover.

2. **Fungsi `cycle_crossover`:**
   a) Menginisialisasi jadwal baru untuk hasil crossover.
   b) Menggunakan pendekatan siklus untuk menggabungkan kelas dari dua jadwal.
   c) Mengunjungi setiap siklus kelas dan mengganti kelas sesuai dengan siklus yang ditemukan.
   d) Mengembalikan jadwal hasil crossover.
3. **Fungsi `uniform_crossover`:**
   a) Menginisialisasi jadwal baru untuk hasil crossover.
   b) Untuk setiap kelas, secara acak memilih apakah akan mengambil detail dari parent1 atau parent2.
   c) Mengembalikan jadwal hasil crossover.

4. Penerapan Metode Mutasi

Setelah crossover, mutasi diterapkan untuk memperkenalkan variasi dalam keturunan. Metode yang diuji meliputi binary mutation dan swap mutation.

Pseudocode untuk fungsi pada metode *mutation:*

```
FUNGSI binary_mutation(jadwal):
    DATA = jadwal.data

    UNTUK setiap cls dalam jadwal.get_classes():
        JIKA angka acak kurang dari MUTATION_RATE:
            ATUR waktu dari cls dengan waktu acak dari
DATA.get_timeslots()
            ATUR ruangan dari cls dengan ruangan acak dari
DATA.get_rooms()
            ATUR profesor dari cls dengan profesor acak
dari cls.get_course().get_professors()

    KEMBALIKAN jadwal


FUNGSI swap_mutation(jadwal):
    kelas_tmp = jadwal.get_classes()

    i = pilih angka acak antara 0 dan panjang kelas_tmp
    j = pilih angka acak antara 0 dan panjang kelas_tmp
```

```
    JIKA i tidak sama dengan j dan angka acak kurang dari
MUTATION_RATE:
        ATUR ruangan dari jadwal.get_classes()[i] dengan
ruangan dari jadwal.get_classes()[j]
        ATUR ruangan dari jadwal.get_classes()[j] dengan
ruangan dari kelas_tmp[i]

    KEMBALIKAN jadwal
```

Penjelasan Pseudocode

1. **Fungsi `binary_mutation`:**
   a) Menginisialisasi data dari jadwal yang diberikan.
   b) Untuk setiap kelas dalam jadwal, jika angka acak lebih kecil dari `MUTATION_RATE`, mengubah waktu, ruangan, dan profesor kelas tersebut dengan memilih secara acak dari daftar yang tersedia.
   c) Mengembalikan jadwal yang telah dimutasi.
2. **Fungsi `swap_mutation`:**
   a) Menyimpan daftar kelas dari jadwal ke dalam variabel sementara.
   b) Memilih dua indeks acak dari daftar kelas.
   c) Jika kedua indeks tersebut tidak sama dan angka acak kurang dari `MUTATION_RATE`, menukar ruangan antara kedua kelas tersebut.
   d) Mengembalikan jadwal yang telah dimutasi.

5. Evaluasi Fitness Populasi

Proses evaluasi fitness dilakukan untuk menilai kualitas setiap solusi (jadwal) yang dihasilkan oleh algoritma genetika. Fungsi *`calculate_fitness_with_soft_constraints`* digunakan untuk menghitung nilai fitness dengan memperhitungkan pelanggaran hard constraint dan soft constraint. Hard constraint mencakup kesesuaian ruangan dengan kebutuhan lab dan tidak adanya konflik timeslot untuk ruangan atau profesor yang sama. Soft constraint mencakup preferensi timeslot dan ruangan yang diinginkan oleh mata kuliah. Setiap pelanggaran constraint dihitung dan diberi bobot tertentu, kemudian dijumlahkan untuk mendapatkan nilai fitness. Semakin sedikit pelanggaran yang terjadi, semakin tinggi nilai fitness yang diperoleh, menunjukkan jadwal yang lebih optimal.

Pseudocode untuk fungsi pada evaluasi fitness yang mencakup hard constraint dan soft constraint:

```
FUNGSI hitung_fitness_dengan_soft_constraints()
    SET numOfConflicts KE 0
    SET pelanggaran_hard_constraint KE 0
    SET pelanggaran_soft_constraint KE 0
    SET kelas KE dapatkan_kelas()

    UNTUK i DARI 0 HINGGA panjang(kelas) - 1
        JIKA kelas[i].ruangan.isLab TIDAK SAMA DENGAN
kelas[i].mata_kuliah.perluLab
            INCREMENT pelanggaran_hard_constraint

        UNTUK j DARI i + 1 HINGGA panjang(kelas)
            JIKA kelas[i].timeslot SAMA DENGAN
kelas[j].timeslot DAN kelas[i].id TIDAK SAMA DENGAN
kelas[j].id
                JIKA kelas[i].ruangan SAMA DENGAN
kelas[j].ruangan
                    INCREMENT pelanggaran_hard_constraint
                JIKA kelas[i].profesor SAMA DENGAN
kelas[j].profesor
                    INCREMENT pelanggaran_hard_constraint

    UNTUK SETIAP cls DALAM kelas
        SET timeslot_preferensi KE
cls.mata_kuliah.timeslot_preferensi
        JIKA panjang(timeslot_preferensi) > 0 DAN
cls.timeslot.id TIDAK ADA DI timeslot_preferensi
            INCREMENT pelanggaran_soft_constraint

        SET ruangan_preferensi KE
cls.mata_kuliah.ruangan_preferensi
        JIKA panjang(ruangan_preferensi) > 0 DAN
nomor(cls.ruangan) TIDAK ADA DI ruangan_preferensi
            INCREMENT pelanggaran_soft_constraint

    SET numOfConflicts KE pelanggaran_hard_constraint *
PENALTI_HARD_CONSTRAINT + pelanggaran_soft_constraint *
PENALTI_SOFT_CONSTRAINT

    RETURN 0 - numOfConflicts
```

Penjelasan Evaluasi Fitness

Fungsi `hitung_fitness_dengan_soft_constraints` menghitung nilai fitness solusi penjadwalan dengan memperhitungkan pelanggaran terhadap hard constraint dan soft constraint.

1. **Inisialisasi Variabel:**
   a. `numOfConflicts`, `pelanggaran_hard_constraint`, dan `pelanggaran_soft_constraint` diatur ke 0.
   b. `kelas` menyimpan daftar kelas yang perlu dijadwalkan.

2. **Evaluasi Hard Constraint:**
   a. **Perulangan Pertama:** Memeriksa kesesuaian ruangan dengan kebutuhan laboratorium mata kuliah.
   b. **Perulangan Kedua:** Memeriksa tumpang tindih timeslot, ruangan, dan profesor antara dua kelas berbeda.
3. **Evaluasi Soft Constraint:**
   a. **Perulangan Kelas:** Memeriksa apakah timeslot dan ruangan kelas sesuai dengan preferensi yang ditentukan mata kuliah.
4. **Menghitung Skor Fitness:**
   a. `numOfConflicts` dihitung sebagai jumlah pelanggaran hard constraint dan soft constraint yang dikalikan dengan penalti masing-masing.
   b. Fungsi mengembalikan nilai negatif `numOfConflicts` sebagai nilai fitness (nilai negatif karena semakin sedikit konflik, semakin tinggi fitness).

6. Kriteria Terminasi

Algoritma berhenti setelah mencapai jumlah generasi maksimum dalam hal ini 200 generasi atau tidak ada peningkatan signifikan dalam fitness.

7. Algoritma Genetika

Algoritma Genetika adalah teknik optimisasi yang terinspirasi oleh prinsip-prinsip evolusi alami, seperti seleksi, mutasi, dan persilangan. Dalam konteks algoritma genetika, sebuah populasi solusi dievaluasi dan diperbaiki melalui generasi dengan tujuan menemukan solusi optimal untuk masalah yang kompleks. Pseudocode berikut menjelaskan struktur dasar dari algoritma genetika yang melibatkan tiga operasi utama: mutasi, crossover, dan seleksi.

Pseudocode untuk Algoritma Genetika

```
kelas GeneticAlgorithm:

    fungsi __init__(self, mutation_func, crossover_func,
selection_func):
        self.mutation_func = mutation_func
        self.crossover_func = crossover_func
        self.selection_func = selection_func
```

```
    fungsi evolve(self, population):
        kembalikan
self.mutate_population(self.crossover_population(populatio
n))

    fungsi mutate_population(self, population):
        untuk i dari NUMBER_OF_ELITE_SCHEDULES hingga
POPULATION_SIZE:
            population.get_schedules()[i]                =
self.mutation_func(population.get_schedules()[i])
        kembalikan population

    fungsi crossover_population(self, population):
        crossover_pop           =           Population(0,
population.get_schedules()[0].data)

        untuk i dari NUMBER_OF_ELITE_SCHEDULES:

crossover_pop.get_schedules().append(population.get_schedu
les()[i])

        untuk i dari NUMBER_OF_ELITE_SCHEDULES hingga
POPULATION_SIZE:
            schedule1 = self.selection_func(population)
            schedule2 = self.selection_func(population)
            jika rnd.random() < CROSSOVER_RATE:

crossover_pop.get_schedules().append(self.crossover_func(s
chedule1, schedule2))
            jika tidak:

crossover_pop.get_schedules().append(schedule1)
        kembalikan crossover_pop
```

Penjelasan Pseudocode:

1. **Kelas GeneticAlgorithm**: Kelas ini mengatur algoritma genetika dengan fungsi-fungsi untuk evolusi populasi solusi.
2. **Inisialisasi**: Konstruktor `__init__` mengatur fungsi untuk mutasi, crossover, dan seleksi yang digunakan dalam evolusi.
3. **Evolusi Populasi**: Fungsi `evolve` melakukan dua langkah: pertama melakukan crossover pada populasi, lalu menerapkan mutasi pada hasil crossover.
4. **Mutasi Populasi**: Fungsi `mutate_population` mengubah individu dalam populasi dengan probabilitas tertentu, kecuali individu yang terbaik.
5. **Crossover Populasi**: Fungsi `crossover_population` membuat individu baru dengan menggabungkan pasangan individu dari populasi lama, sebagian besar tetap dari individu elit, dan sisanya dari hasil crossover berdasarkan

probabilitas yang ditentukan.

8. Eksperimen dengan kombinasi
Pseudocode untuk fungsi eksperimen:

```
FUNGSI   run_experiment(metode_crossover,   metode_mutasi,
metode_seleksi):
    INISIALISASI data
    INISIALISASI  avg_fitness_per_generation  sebagai  array
nol dengan ukuran MAX_GENERATIONS
    INISIALISASI total_runtime sebagai 0

    UNTUK setiap run DARI 1 HINGGA NUM_RUNS:
        MULAI timer
        INISIALISASI populasi
        INISIALISASI algoritma_genetika dengan metode yang
diberikan
        INISIALISASI fitness_per_generation sebagai daftar
kosong

        UNTUK    setiap    generasi    DARI    1    HINGGA
MAX_GENERATIONS:
            URUTKAN populasi berdasarkan fitness dari yang
tertinggi
            AMBIL fitness terbaik dari individu teratas
            TAMBAH        fitness        terbaik        ke
fitness_per_generation

            UPDATE populasi menggunakan algoritma_genetika

        HENTIKAN timer
        HITUNG runtime sebagai waktu yang telah berlalu
        TAMBAH runtime ke total_runtime
        TAMBAH          fitness_per_generation          ke
avg_fitness_per_generation

    HITUNG  rata-rata  avg_fitness_per_generation  dengan
membagi total fitness per generasi dengan NUM_RUNS
    HITUNG    rata-rata    avg_runtime    dengan    membagi
total_runtime dengan NUM_RUNS

    KEMBALIKAN (avg_fitness_per_generation, avg_runtime)
```

Penjelasan Pseudocode:

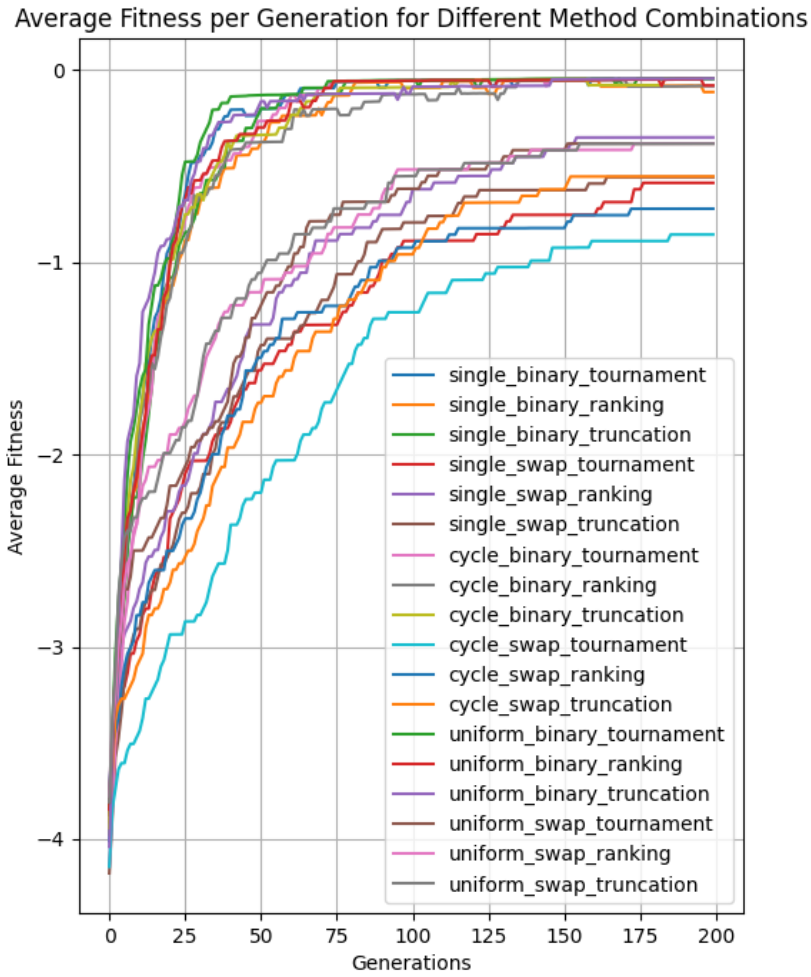
1. **Inisialisasi Variabel:**
   a) `data` berisi data yang diperlukan untuk eksperimen.
   b) `avg_fitness_per_generation` adalah array yang menyimpan total fitness per generasi dari semua eksperimen.
   c) `total_runtime` untuk menyimpan waktu total yang

digunakan untuk eksperimen.

2. **Loop Eksperimen:**
   a) Untuk setiap iterasi eksperimen (jumlah eksperimen adalah `NUM_RUNS`), inisialisasi populasi dan algoritma genetika dengan metode yang diberikan.
   b) Untuk setiap generasi (hingga `MAX_GENERATIONS`), urutkan populasi berdasarkan fitness, simpan fitness terbaik, dan evolusi populasi menggunakan algoritma genetika.
3. **Pengukuran Waktu dan Pembaruan Data:**
   a) Ukur waktu yang dibutuhkan untuk menjalankan eksperimen.
   b) Tambahkan waktu yang diukur dan fitness per generasi ke total runtime dan `avg_fitness_per_generation`.
4. **Perhitungan Rata-rata:**
   a) Hitung rata-rata fitness per generasi dengan membagi total fitness per generasi dengan jumlah eksperimen.
   b) Hitung rata-rata waktu eksekusi dengan membagi total runtime dengan jumlah eksperimen.
5. **Pengembalian Hasil:**
   a) Kembalikan fitness rata-rata per generasi dan rata-rata waktu eksekusi sebagai hasil dari fungsi.

Analisis Hasil Eksperimen

Setelah eksperimen selesai, data dianalisis dengan menghitung rata-rata nilai fitness per generasi untuk setiap kombinasi metode. Hasil analisis ini digunakan untuk menentukan kombinasi metode yang paling efektif dalam mengoptimalkan penjadwalan mata kuliah.



Gambar 3 Rata-rata fitness per generasi dengan berbagai metode

Gambar 3 menunjukkan rata-rata fitness per generasi menggunakan berbagai metode. Dari gambar tersebut, terlihat bahwa setiap metode memiliki pola peningkatan fitness yang berbeda seiring bertambahnya generasi. Pada generasi awal, beberapa metode seperti Single Binary Tournament dan Cycle Swap Tournament menunjukkan performa yang kurang optimal dibandingkan metode lainnya. Namun, seiring berjalannya waktu, beberapa metode mulai menunjukkan peningkatan signifikan dalam nilai fitness. Metode Uniform Binary Ranking dan Single Binary Truncation, misalnya, menunjukkan tren peningkatan yang lebih konsisten dan mencapai nilai fitness yang lebih tinggi pada generasi akhir. Hal ini menunjukkan bahwa

pilihan metode memiliki dampak signifikan terhadap efektivitas dan efisiensi algoritma genetika dalam mencapai solusi optimal. Penelitian lebih lanjut diperlukan untuk memahami faktor-faktor yang mempengaruhi performa masing-masing metode serta kemungkinan kombinasi metode untuk hasil yang lebih optimal.

Tabel 5 Metrik dan ringkasan dari hasil algoritma

Metric	**Value**
Average Runtime	
Min	0.08
Max	0.10
Mean	0.08
Std Dev	0.01
Generasi 1	
Min	-4.01
Max	-3.48
Mean	-3.78
Std Dev	0.19
Generasi 200	
Min	-0.59
Max	-0.04
Mean	-0.26
Std Dev	0.20

Hasil penelitian pada tabel 5 menunjukkan variasi dalam runtime dan performa metode di berbagai generasi. Rata-rata runtime berkisar antara 0.08 hingga 0.10 detik, dengan nilai rata-rata sebesar 0.08 detik dan deviasi standar 0.01 detik. Pada generasi pertama, performa metode bervariasi dengan nilai minimum -4.01 dan nilai maksimum -3.48, serta nilai rata-rata -3.78 dengan deviasi standar 0.19. Pada generasi ke-200, performa metode mengalami peningkatan yang signifikan dengan nilai minimum -0.59 dan nilai maksimum -0.04, serta nilai rata-rata -0.26 dengan deviasi standar 0.20. Perubahan nilai dari generasi pertama hingga generasi ke-200 menunjukkan peningkatan dalam performa algoritma, meskipun terdapat variasi dalam nilai minimum, maksimum, dan rata-rata yang tercatat.

Tabel 6 Performa Rata-rata per Generasi untuk Setiap Metode

Metode	Gen 1	Gen 50	Gen 100	Gen 150	Gen 200
single_binary_tournament	-4.01	-2.36	-1.10	-0.04	-0.04
single_binary_ranking	-3.91	-2.37	-1.29	-0.08	-0.11
single_binary_truncation	-3.87	-2.36	-1.28	-0.04	-0.04
single_swap_tournament	-3.87	-3.04	-1.79	-0.59	-0.59
single_swap_ranking	-3.77	-2.87	-1.55	-0.35	-0.35
cycle_binary_tournament	-4.00	-2.57	-1.37	-0.04	-0.04
cycle_binary_ranking	-3.92	-2.58	-1.38	-0.08	-0.11
cycle_binary_truncation	-3.88	-2.57	-1.37	-0.04	-0.04
cycle_swap_tournament	-3.87	-3.26	-1.86	-0.59	-0.59
cycle_swap_ranking	-3.77	-2.90	-1.62	-0.35	-0.35
uniform_binary_tournament	-3.97	-2.47	-1.28	-0.04	-0.04
uniform_binary_ranking	-3.87	-2.47	-1.38	-0.08	-0.11
uniform_binary_truncation	-3.87	-2.47	-1.37	-0.04	-0.04
uniform_swap_tournament	-3.87	-3.18	-1.86	-0.59	-0.59
uniform_swap_ranking	-3.77	-2.93	-1.63	-0.35	-0.35
uniform_swap_truncation	-3.87	-2.57	-1.37	-0.04	-0.04

Tabel 6 menunjukkan kombinasi metode dengan runtime rata-rata terendah adalah kombinasi Single Swap Tournament dengan 0.08 detik, sementara yang tertinggi adalah kombinasi Cycle Swap Truncation dengan 0.10 detik. Pada generasi pertama, metode Single Binary Tournament memiliki performa terburuk dengan nilai -4.01, sedangkan Uniform Binary Ranking memiliki performa terbaik dengan nilai -3.48. Pada generasi terakhir, metode Single Swap Tournament dan Cycle Swap Tournament memiliki performa terburuk dengan nilai -0.59, sementara Single Binary Truncation memiliki performa terbaik dengan nilai -0.04. Metode Single Binary Tournament dan Single Swap Tournament menunjukkan performa yang konstan baik di awal maupun di akhir, tetapi dengan perbedaan yang signifikan dalam runtime rata-rata.

KESIMPULAN

Metode Single Swap Tournament menunjukkan waktu eksekusi tercepat, sementara Cycle Swap Truncation paling lambat. Pada generasi pertama, Uniform Binary Ranking memiliki performa terbaik dan Single Binary Tournament terburuk, sedangkan pada generasi terakhir, Single Binary Truncation mencapai performa terbaik dan Single Swap Tournament serta Cycle Swap Tournament terburuk. Single Binary Tournament dan Single Swap Tournament konsisten

dalam performa awal dan akhir, tetapi berbeda signifikan dalam waktu eksekusi.

Untuk aplikasi praktis yang memerlukan efisiensi waktu, metode Single Swap Tournament direkomendasikan. Penelitian lebih lanjut dapat mengembangkan kombinasi metode untuk meningkatkan performa keseluruhan, dan pengembangan teori baru dapat mengkaji faktor-faktor yang mempengaruhi efisiensi dan efektivitas algoritma genetika dalam penjadwalan kuliah.

DAFTAR PUSTAKA

Andriyadi, A., & Halimah, H. (2022). Optimasi Algoritma Genetika dalam Perancangan Sistem Informasi Penjadwalan Seminar dan Sidang Skripsi Mahasiswa Institut Informatika dan Bisnis (IIB) Darmajaya. *TEKNIKA: Jurnal Ilmiah Bidang Ilmu Rekayasa*, *16*(1), 133-140.

Banihashemian, S. S., & Adibnia, F. (2021). A novel robust soft-computed range-free localization algorithm against malicious anchor nodes. *Cognitive Computation*, *13*(4), 992-1007.

Chen, S. H., & Chen, M. C. (2011, November). Operators of the two-part encoding genetic algorithm in solving the multiple traveling salesmen problem. In *2011 International Conference on Technologies and Applications of Artificial Intelligence* (pp. 331-336). IEEE.

Ewi, E. I., & Radiles, H. (2023). Mitigasi Premature Convergence Pada Genetic Algorithm Menggunakan Metoda Dynamics Growth Population Dalam Kasus University Course Scheduling. *JEKIN-Jurnal Teknik Informatika*, *3*(1), 33-44.

Fajarlestari, M. K., & Suban, I. B. (2023). Kombinasi Crossover dan Mutasi Terbaik pada Algoritma Genetika dalam Penjadwalan Mata Kuliah. *Techno. com*, *22*(4).

John, H. (1992). Holland. genetic algorithms. *Scientific american*, *267*(1), 44-50.

Raisanen, L. (2005). *Multi-objective site selection and analysis for GSM cellular network planning*. Cardiff University (United Kingdom).

Rimalia, W. (2023). Penerapan Algoritma Genetika serta Metode TOPSIS Sebagai Solusi Penjadwalan Mata Kuliah. *Journal of System and Computer Engineering (JSCE)*, *4*(1), 92-105.

Salman, R. (2023). A, Analisis Pengaruh Probabilitas Crossover Terhadap Kinerja Algoritma Genetika Dalam Optimasi Penjadwalan Matakuliah. Jurnal Teknoif Teknik Informatika Institut Teknologi Padang, 11(2), 69-74.

Syawal, M., Belluano, P. L. L., & Manga, A. R. (2021). Implementasi Algoritma Genetika Untuk Penjadwalan Laboratotium Fakultas Ilmu Komputer Universitas Muslim Indonesia. *Indonesian Journal of Data and Science*, *2*(1), 29-37.

Yuan, Y., Wang, W., Coghill, G. M., & Pang, W. (2021). A novel genetic algorithm with hierarchical evaluation strategy for hyperparameter optimisation of graph neural networks. *arXiv preprint arXiv:2101.09300*.