Laporan Tugas Kecerdasan Buatan 1

Disusun Oleh :

DEDE RAHMAT FITRIANSYAH
1302213121, S1 Rekayasa Perangkat Lunak, Fakultas Informatika, Universitas Telkom, dederf@student.telkomuniversity.ac.id

MUHAMMAD RISKY FARHAN
1302210056, S1 Rekayasa Perangkat Lunak, Fakultas Informatika, Universitas Telkom, skyfrhn@student.telkomuniversity.ac.id

FARHAN MULYA ARGYANTO
1302213073, S1 Rekayasa Perangkat Lunak, Fakultas Informatika, Universitas Telkom, fahanmulya@student.telkomuniversity.ac.id

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK TELKOM UNIVERSITY
KOTA BANDUNG
2023

Daftar Isi

Permasalahan

$$f(x_1, x_2) = -\left(sin(x_1)cos(x_2) + \frac{4}{5}exp\left(1 - \sqrt{x_1^2 + x_2^2}\right)\right)$$

dengan **domain** (batas nilai) untuk x_1 dan x_2 adalah

$$-10 \leq x_1 \leq 10 \text{ dan } -10 \leq x_2 \leq 10$$

Tujuan

Tujuannya adalah untuk mencari atau menemukan nilai x1 dan x2 yang akan menghasilkan dari fungsi f(x1, x2) dalam batasan domain yang telah ditentukan. Algoritma Genetika akan digunakan untuk mencari solusi optimal dengan menghasilkan populasi kromosom (individu), menggabungkan kromosom (crossover), menerapkan mutasi pada kromosom, dan memilih kromosom terbaik berdasarkan nilai fitness (nilai fungsi yang akan dioptimalkan). Proses ini akan berulang melalui beberapa generasi hingga kondisi berhenti terpenuhi, dengan tujuan mendekati nilai minimum dari fungsi objektif yang diberikan.

Penjelasan Program Yang Dibuat

Dalam penyelesaian tugas ini kami mengimplementasikan beberapa proses sebagai berikut:

```
# fungsi untuk memecah rumus math
def objectice_function(x1, x2):
    return -(math.sin(x1) * math.cos(x2) + 4/5 * math.exp(1 -
math.sqrt(x1**2 + x2**2)))
```

Fungsi ***objectice_function*** adalah fungsi matematis yang mengambil dua variabel, x1 dan x2, dan menghitung nilai objektif dengan menggunakan operasi matematika seperti sinus, cosinus, eksponensial, perkalian, dan pengurangan. Tujuan dari fungsi ini adalah untuk mencari nilai x1 dan x2 yang akan menghasilkan nilai objektif yang mendekati nilai minimum dalam konteks optimisasi. Fungsi ini digunakan dalam algoritma optimisasi, seperti Algoritma Genetika, untuk mencari solusi optimal dalam ruang pencarian.

```
# Fungsi untuk membuat kromosom
def create_kromosom():
    return [random.randint(0, 1) for _ in range(8)]
```

Fungsi ***create_kromosom*** digunakan untuk membuat kromosom dalam konteks Algoritma Genetika. Kromosom ini digunakan untuk merepresentasikan solusi potensial dalam ruang pencarian. Fungsi ini menghasilkan kromosom dengan panjang 8, dimana setiap elemen kromosom adalah bilangan acak 0 atau 1. Panjang 8 mungkin sesuai dengan jumlah

bit yang digunakan untuk merepresentasikan variabel dalam optimisasi. Kromosom ini akan menjadi salah satu anggota dalam populasi yang akan dievolusi selama proses optimisasi.

```
# Fungsi untuk membuat populasi
def create_populasi():
    populasi = []
    for _ in range(8):
        kromosom = create_kromosom()
        populasi.append(kromosom)
    return populasi
```

Fungsi ***create_populasi*** digunakan untuk menciptakan sebuah populasi awal dalam konteks Algoritma Genetika. Populasi ini terdiri dari beberapa kromosom yang merepresentasikan solusi potensial dalam ruang pencarian. Fungsi ini membuat populasi dengan 8 kromosom, di mana setiap kromosom dihasilkan menggunakan fungsi create_kromosom. Populasi ini akan menjadi awal dari proses evolusi dalam algoritma optimisasi, di mana kromosom-kromosom ini akan mengalami seleksi, crossover, dan mutasi untuk mencari solusi yang lebih baik.

```
def decode_chromosome(chromosome, a, b, n):
    # chromosome adalah kromosom yang ingin didekode
    # a dan b adalah batas bawah dan atas dari interval
     # n adalah jumlah bit yang digunakan untuk merepresentasikan setiap
variabel

    x1_bits = chromosome[:n]  # Ambil n bit pertama untuk x1
    x2_bits = chromosome[n:]  # Ambil n bit berikutnya untuk x2

    # Konversi bit ke bilangan desimal
     x1 = a + int("".join(map(str, x1_bits)), 2) * (b - a) / (2 ** n -
1)
     x2 = a + int("".join(map(str, x2_bits)), 2) * (b - a) / (2 ** n -
1)

    return x1, x2
```

Fungsi ***decode_chromosome*** digunakan untuk mengonversi kromosom (rangkaian bit) menjadi nilai-nilai variabel kontinu, dalam hal ini x1 dan x2. Fungsi ini memerlukan tiga parameter:

- chromosome: Kromosom yang akan didekode, yang terdiri dari rangkaian bit.

- a dan b: Batas bawah dan atas dari interval yang akan digunakan untuk mengonversi bit menjadi nilai variabel kontinu.
- n: Jumlah bit yang digunakan untuk merepresentasikan setiap variabel.

Proses Decodenya dilakukan sebagai berikut:

1. Kromosom dibagi menjadi dua bagian: n bit pertama untuk x1 dan n bit berikutnya untuk x2.
2. Setiap bagian kromosom (x1_bits dan x2_bits) dikonversi dari biner (bit) menjadi bilangan desimal.
3. Bilangan desimal yang dihasilkan untuk setiap variabel (x1 dan x2) dibaca sebagai representasi nilai dalam interval antara a dan b, dengan menggunakan rumus perhitungan yang melibatkan a, b, n, dan nilai biner yang telah diubah.

   Hasilnya adalah pasangan nilai x1 dan x2 yang dapat digunakan dalam evaluasi fungsi objektif atau dalam proses optimisasi untuk mencari solusi optimal dalam ruang pencarian.

```
#selection(mencari parent untuk disilangkan)
#dari evaluation tadi dicari yang paling rendah karena kasus min, kalau
rendah nilai fit pasti tinggi
def calculate_fitness(objective_values):
    fitness_values = []
    for obj_val in objective_values:
          fitness = 1 / (1 + obj_val) #+1 untuk menghindari pembagian
dengan 0
        fitness_values.append(fitness)
    return fitness_values
```

Fungsi **calculate_fitness** digunakan untuk menghitung nilai fitness dari sejumlah nilai objektif. Nilai objektif biasanya mengukur seberapa baik solusi potensial dalam konteks optimisasi, di mana nilai yang lebih rendah biasanya lebih baik (dalam kasus mencari nilai minimum).

Fungsi ini menerima daftar objective_values yang merupakan nilai-nilai objektif dari berbagai solusi. Untuk setiap nilai objektif, fungsi menghitung nilai fitness dengan rumus:

fitness = 1 / (1 + obj_val)

Rumus ini digunakan untuk menghasilkan nilai fitness yang lebih tinggi untuk nilai objektif yang lebih rendah, karena nilai fitness lebih tinggi mengindikasikan solusi yang lebih baik. Penambahan 1 dalam pembagi (denominator) digunakan untuk menghindari pembagian dengan 0 yang dapat menyebabkan masalah.

Hasilnya adalah daftar nilai fitness yang sesuai dengan nilai objektif, dan nilai-nilai fitness ini kemudian digunakan dalam seleksi dan evolusi populasi selama proses optimisasi.

```
#hitung nilai probability
def calculate_probability(fitness_values):
    total_fitness = sum(fitness_values)  # Langkah 1: Menghitung total
fitness

    probabilities = []
    for fitness in fitness_values:
        probability = fitness / total_fitness  # Langkah 2: Menghitung
probabilitas
        probabilities.append(probability)

    return probabilities
```

Fungsi ***calculate_probability*** digunakan untuk menghitung probabilitas munculnya setiap kromosom dalam populasi selama proses seleksi dengan Roulette Wheel Selection dalam Algoritma Genetika.

Langkah-langkah yang dilakukan dalam fungsi ini adalah:

1. Menghitung total fitness: Total fitness dalam populasi dihitung dengan menjumlahkan semua nilai fitness individu dalam populasi. Ini memberikan gambaran seberapa baik seluruh populasi dalam hal nilai fitness.
2. Menghitung probabilitas: Probabilitas munculnya setiap kromosom dihitung dengan membagi nilai fitness individu dengan total fitness. Ini dilakukan untuk setiap kromosom dalam populasi.

Hasil dari fungsi ini adalah daftar probabilitas munculnya setiap kromosom dalam populasi. Probabilitas ini kemudian digunakan dalam Roulette Wheel Selection untuk memilih kromosom-kromosom yang akan menjadi orang tua untuk reproduksi dalam generasi berikutnya. Kromosom dengan fitness yang lebih tinggi akan memiliki probabilitas yang lebih besar untuk dipilih.

```
def calculate_cumulative_probability(probabilities):
    cumulative_probabilities = [sum(probabilities[:i+1]) for i in
range(len(probabilities))]
    return cumulative_probabilities

def   select_chromosomes_using_roulette_wheel(cumulative_probabilities,
populasi):
    n = len(populasi)
    new_population = []
    for _ in range(n):
```

```
        random_value = random.uniform(0, 1)  # Menghasilkan nilai acak
antara 0 dan 1
        selected_chromosome = None
        for i, cumulative_prob in enumerate(cumulative_probabilities):
            if random_value <= cumulative_prob:
                selected_chromosome = populasi[i]
                break
        new_population.append(selected_chromosome)
    return new_population
```

Fungsi-fungsi diatas digunakan dalam tahap seleksi dengan Roulette Wheel Selection dalam Algoritma Genetika untuk memilih kromosom-kromosom yang akan menjadi orang tua untuk reproduksi dalam generasi berikutnya. Berikut penjelasannya:

1. ***calculate_cumulative_probability(probabilities):*** Fungsi ini mengambil daftar probabilitas munculnya kromosom (dihitung sebelumnya) dan menghitung probabilitas kumulatif. Probabilitas kumulatif adalah probabilitas akumulatif dari munculnya kromosom dalam urutan mereka. Hasilnya adalah daftar probabilitas kumulatif.
2. ***select_chromosomes_using_roulette_wheel(cumulative_probabilities, populasi):*** Fungsi ini mengambil daftar probabilitas kumulatif yang dihasilkan oleh fungsi sebelumnya dan populasi kromosom. Kemudian, ia melakukan seleksi kromosom menggunakan Roulette Wheel Selection. Ini dilakukan dengan menghasilkan nilai acak antara 0 dan 1 (disebut random_value), dan kemudian dari nilai Roulette Wheel tadi kita banding kan satu satu dengan nilai cumulative_prob dan bila nilai Roulette Wheel tersebut kurang dari nilai cumulative_prob dengan index tertentu maka index nilai Roulette Wheel yang lebih kecil tadi akan mencari index kromsom yang direplace kepopulasi baru dengan index nilai cumulative_prob yang lebih kecilnya tadi. dikasus ini populasi baru merupakan copyan dari populasi lama jadi isinya sama dan tergantung nilai randomnya ada yang direplace dan ada yang tidak di replace.

```
def crossover(new_population, crossover_rate):
            random_values = [random.uniform(0, 1) for _ in
range(len(new_population))]
    index_kurangdari_rate = [i for i, value in enumerate(random_values)
if value < crossover_rate]
    new_population_1 = new_population.copy()  # Create a copy of the
original population

    for i in range(0, len(index_kurangdari_rate), 2):
        index1 = index_kurangdari_rate[i]
        if i + 1 < len(index_kurangdari_rate):
            index2 = index_kurangdari_rate[i + 1]
        else:
```

```
            # If there's an odd number of selected parents, handle the
last one differently
            # You can choose to skip it, for example
            continue

        # Perform crossover on selected parents at index1 and index2
        parent1 = new_population[index1]
        parent2 = new_population[index2]

          # Implement crossover logic (you can use a custom crossover
function here)
            # For example, let's use a single-point crossover for
simplicity
        crossover_point = len(parent1) // 2
                        offspring1   =   parent1[:crossover_point]   +
parent2[crossover_point:]
                        offspring2   =   parent2[:crossover_point]   +
parent1[crossover_point:]

        # Replace the parents with offspring in the new_population_1
        new_population_1[index1] = offspring1
        new_population_1[index2] = offspring2

    return new_population_1, index_kurangdari_rate
```

Fungsi ***crossover*** digunakan untuk melakukan operasi crossover pada populasi kromosom dengan tingkat crossover yang telah ditentukan. Berikut penjelasannya:

1. Fungsi menerima dua parameter:
    - ***new_population***: Populasi kromosom yang akan mengalami crossover.
    - ***crossover_rate***: Tingkat crossover, yaitu probabilitas bahwa dua kromosom akan mengalami crossover.
2. Pertama, fungsi menghasilkan sejumlah nilai acak antara 0 dan 1 (disebut ***random_values***) untuk setiap kromosom dalam populasi. Ini digunakan untuk menentukan apakah suatu kromosom akan mengalami crossover atau tidak.

3. Fungsi kemudian mencari indeks kromosom yang memiliki nilai acak kurang dari ***crossover_rate***. Ini dilakukan dengan membuat daftar ***index_kurangdari_rate*** yang berisi indeks kromosom yang memenuhi syarat.

4. Selanjutnya, populasi baru (***new_population_1***) dibuat sebagai salinan dari populasi awal. Ini dilakukan untuk menjaga populasi awal tetap tidak berubah.

5. Fungsi melakukan iterasi melalui daftar indeks yang memenuhi syarat dalam ***index_kurangdari_rate***. Untuk setiap pasangan indeks (jika ada), fungsi melakukan operasi crossover antara dua kromosom yang dipilih.

6. Implementasi crossover yang digunakan dalam contoh ini adalah single-point crossover, yang membagi dua kromosom pada titik tengah dan menukar bagian-bagian kromosom antara kedua orang tua.

7. Hasil dari operasi crossover adalah dua kromosom anak (offspring), yang kemudian menggantikan orang tua mereka dalam populasi baru (***new_population_1***).

8. Hasil akhir dari fungsi adalah populasi baru setelah operasi crossover, bersama dengan daftar indeks yang memenuhi syarat untuk crossover. Populasi baru ini akan digunakan dalam tahap berikutnya dalam algoritma optimisasi.

```
def mutate_chromosome(chromosome, mutation_rate):
    mutated_chromosome = []
    for bit in chromosome:
        if random.random() < mutation_rate:
            mutated_bit = 1 - bit  # Flip the bit
        else:
            mutated_bit = bit
        mutated_chromosome.append(mutated_bit)
    return mutated_chromosome

# Function to apply mutation to the entire population
def mutate_population(population, mutation_rate):
    mutated_population = []
    for chromosome in population:
                mutated_chromosome  =  mutate_chromosome(chromosome,
mutation_rate)
        mutated_population.append(mutated_chromosome)
    return mutated_population
```

1. ***mutate_chromosome(chromosome, mutation_rate)***: Fungsi ini menerima satu kromosom (runtutan bit) dan tingkat mutasi. Operasi mutasi pada kromosom dilakukan dengan cara berikut:

    - Fungsi melakukan iterasi melalui setiap bit dalam kromosom.
    - Untuk setiap bit, dihasilkan nilai acak antara 0 dan 1 (menggunakan ***random.random()***).

- Jika nilai acak tersebut kurang dari ***mutation_rate***, maka bit tersebut mengalami mutasi. Dalam kasus mutasi, bit tersebut dibalik (dari 0 menjadi 1 atau sebaliknya, untuk program ga ini kita memakai mutation rate 0,25).
- Jika nilai acak lebih besar atau sama dengan ***mutation_rate***, bit tersebut tetap tidak berubah.
- Bit-bit yang telah dimutasi atau tidak berubah kemudian digabungkan kembali untuk membentuk kromosom yang dimutasi.

2. ***mutate_population(population, mutation_rate)***: Fungsi ini digunakan untuk menerapkan operasi mutasi ke seluruh populasi kromosom.

    - Fungsi menerima populasi kromosom dan tingkat mutasi.
    - Ia melakukan iterasi melalui setiap kromosom dalam populasi dan menerapkan fungsi ***mutate_chromosome*** pada masing-masing kromosom.
    - Hasil dari operasi mutasi adalah populasi kromosom yang telah dimutasi.

    Operasi mutasi adalah salah satu komponen penting dalam Algoritma Genetika, yang membantu dalam menjaga keragaman genetik dalam populasi dan mencegah stagnasi dalam pencarian solusi. Mutasi secara acak mengubah beberapa bit dalam kromosom, memberikan peluang bagi solusi baru untuk dieksplorasi.

```
def main():
    # Inisialisasi variabel
    non_improvement_count = 0
     max_non_improvement = 5   # Jumlah generasi tanpa peningkatan yang
diperbolehkan
    generation = 0
     best_fitness_1 = float("-inf")   # Awalnya diatur ke negatif tak
terhingga
    best_chromosome_1 = None
    populasi = create_populasi()


print("+-------------------------------------------------------------------
--------------------------------+")
    print("| Generation | Best Chromosome                                |
Best Fitness      | x1      | x2      |")

print("+-------------------------------------------------------------------
--------------------------------+")

          while generation < 100 and non_improvement_count <
max_non_improvement:
```

```
        a = -10.0  # Batas bawah interval
        b = 10.0  # Batas atas interval
         n = 4  # Jumlah bit yang digunakan untuk merepresentasikan x1
dan x2
        decoded_populasi = []
        nilai_objective = []

        for chromosome in populasi:
            x1, x2 = decode_chromosome(chromosome, a, b, n)
            objective_value = objectice_function(x1, x2)
            nilai_objective.append(objective_value)
            decoded_populasi.append((x1, x2))

        # ... Seleksi, crossover, mutasi, dan perhitungan fitness ...
        fitness_values = calculate_fitness(nilai_objective)
        best_fitness = max(fitness_values)
        best_index = fitness_values.index(best_fitness)
        best_chromosome = populasi[best_index]
        probabilities = calculate_probability(fitness_values)
                                            cumulative_probabilities     =
calculate_cumulative_probability(probabilities)
                                                    new_population       =
select_chromosomes_using_roulette_wheel(cumulative_probabilities,
populasi)
        crossover_rate = 0.25  # 25%
                  new_population_with_crossover,index_kurangdari_rate  =
crossover(new_population, crossover_rate)
        mutation_rate = 0.25
                                        new_population_with_crossover    =
mutate_population(new_population_with_crossover, mutation_rate)

        decoded_populasi = []
        nilai_objective = []
        populasi = new_population_with_crossover
        for chromosome in populasi:
            x1, x2 = decode_chromosome(chromosome, a, b, n)
            objective_value = objectice_function(x1, x2)
            nilai_objective.append(objective_value)
            decoded_populasi.append((x1, x2))

        fitness_values_new = calculate_fitness(nilai_objective)
        best_fitness_new = max(fitness_values_new)
        best_index_new = fitness_values_new.index(best_fitness_new)
```

```
                                                best_chromosome_new      =
new_population_with_crossover[best_index_new]
        if best_fitness_new > best_fitness_1:
            best_fitness_1 = best_fitness_new
            best_chromosome_1 = best_chromosome_new
            non_improvement_count = 0  # Reset count
        else:
            non_improvement_count += 1

        x1, x2 = decode_chromosome(best_chromosome_1, a, b, n)
        best_chromosome_1_str = ", ".join(map(str, best_chromosome_1))
               print("|  {:<10}  |  {:<40}  |  {:<11}  |  {:<7}  |  {:<7}
|".format(generation, best_chromosome_1_str, best_fitness_1, x1, x2))
        generation += 1

    print("+-------------------------------------+")
    print("Best Chromosome:", best_chromosome_1)
    print("Best Fitness:", best_fitness_1)
    print("x1=", x1)
    print("x2=", x2)

if __name__ == "__main__":
    main()
```

Ini adalah fungsi ***main*** yang menjalankan seluruh algoritma optimisasi menggunakan Algoritma Genetika. Berikut penjelasannya:

1. Inisialisasi variabel:
    - ***non_improvement_count***: Jumlah generasi tanpa peningkatan yang diperbolehkan.
    - ***max_non_improvement***: Jumlah maksimum generasi tanpa peningkatan.
    - generation: Generasi saat ini.
    - ***best_fitness_1***: Nilai fitness terbaik awalnya diatur ke negatif tak terhingga.
    - ***best_chromosome_1***: Kromosom terbaik.
2. Membuat populasi awal menggunakan fungsi ***create_populasi***.
3. Melakukan iterasi sebanyak 100 generasi atau tidak terjadi kenaikan nilai fit dibestkromosom generasi baru selama 5 iterasi.
4. Untuk setiap generasi, melakukan operasi:
    - Mendekode kromosom-kromosom dalam populasi awal untuk mendapatkan nilai x1 dan x2.
    - Menghitung nilai objektif (nilai fungsi objektif) untuk setiap kromosom.
    - Menghitung fitness untuk setiap kromosom menggunakan ***calculate_fitness***.
    - Memilih kromosom terbaik dalam populasi saat ini.

5. Menghitung probabilitas munculnya kromosom dengan ***calculate_probability***, dan menghitung probabilitas kumulatif dengan ***calculate_cumulative_probability***.
6. Memilih kromosom-kromosom orang tua untuk reproduksi menggunakan Roulette Wheel Selection.
7. Melakukan operasi crossover pada kromosom-kromosom yang dipilih.
8. Melakukan operasi mutasi pada populasi yang telah mengalami crossover.
9. Menghitung kembali fitness untuk populasi yang telah dimutasi.
10. Menyimpan kromosom terbaik dalam generasi saat ini jika memiliki fitness yang lebih baik dari kromosom terbaik sebelumnya.
11. Menampilkan hasil setiap generasi, termasuk generasi, kromosom terbaik, nilai fitness terbaik, nilai x1, dan nilai x2.
12. Setelah selesai, menampilkan kromosom terbaik dan nilai fitness terbaik.

Penjelasan Garis Besar Kode Program

```
import random
import math

# fungsi untuk memecah rumus math
def objectice_function(x1, x2):
    return -(math.sin(x1) * math.cos(x2) + 4/5 * math.exp(1 -
math.sqrt(x1**2 + x2**2)))

# Fungsi untuk membuat kromosom
def create_kromosom():
    return [random.randint(0, 1) for _ in range(8)]

# Fungsi untuk membuat populasi
def create_populasi():
    populasi = []
    for _ in range(8):
        kromosom = create_kromosom()
        populasi.append(kromosom)
    return populasi

def decode_chromosome(chromosome, a, b, n):
    # chromosome adalah kromosom yang ingin didekode
    # a dan b adalah batas bawah dan atas dari interval
    # n adalah jumlah bit yang digunakan untuk merepresentasikan setiap
variabel

    x1_bits = chromosome[:n]  # Ambil n bit pertama untuk x1
```

```
    x2_bits = chromosome[n:]  # Ambil n bit berikutnya untuk x2

    # Konversi bit ke bilangan desimal
    x1 = a + int("".join(map(str, x1_bits)), 2) * (b - a) / (2 ** n -
1)
    x2 = a + int("".join(map(str, x2_bits)), 2) * (b - a) / (2 ** n -
1)

    return x1, x2

#selection(mencari parent untuk disilangkan)
#dari evaluation tadi dicari yang paling rendah karena kasus min, kalau
rendh nilai fit pasti tinggi
def calculate_fitness(objective_values):
    fitness_values = []
    for obj_val in objective_values:
        fitness = 1 / (1 + obj_val) #+1 untuk menghindari pembagian
dengan 0
        fitness_values.append(fitness)
    return fitness_values

#hitung nilai probability
def calculate_probability(fitness_values):
    total_fitness = sum(fitness_values)  # Langkah 1: Menghitung total
fitness

    probabilities = []
    for fitness in fitness_values:
        probability = fitness / total_fitness  # Langkah 2: Menghitung
probabilitas
        probabilities.append(probability)

    return probabilities

#memilih parent dengan roulette whell, harus menghitung cumulative
probability values
#Selanjutnya generate nilai random tadi sesuai dengan jumlah kromom
def calculate_cumulative_probability(probabilities):
    cumulative_probabilities = [sum(probabilities[:i+1]) for i in
range(len(probabilities))]
    return cumulative_probabilities
```

```
def select_chromosomes_using_roulette_wheel(cumulative_probabilities, populasi):
    n = len(populasi)
    new_population = []
    for _ in range(n):
        random_value = random.uniform(0, 1)  # Menghasilkan nilai acak antara 0 dan 1
        selected_chromosome = None
        for i, cumulative_prob in enumerate(cumulative_probabilities):
            if random_value <= cumulative_prob:
                selected_chromosome = populasi[i]
                break
        new_population.append(selected_chromosome)
    return new_population

#crossover dengan rate 25%

def crossover(new_population, crossover_rate):
    random_values = [random.uniform(0, 1) for _ in range(len(new_population))]
    index_kurangdari_rate = [i for i, value in enumerate(random_values) if value < crossover_rate]
    new_population_1 = new_population.copy()  # Create a copy of the original population

    for i in range(0, len(index_kurangdari_rate), 2):
        index1 = index_kurangdari_rate[i]
        if i + 1 < len(index_kurangdari_rate):
            index2 = index_kurangdari_rate[i + 1]
        else:
            # If there's an odd number of selected parents, handle the last one differently
            # You can choose to skip it, for example
            continue

        # Perform crossover on selected parents at index1 and index2
        parent1 = new_population[index1]
        parent2 = new_population[index2]

        # Implement crossover logic (you can use a custom crossover function here)
        # For example, let's use a single-point crossover for simplicity
```

```
        crossover_point = len(parent1) // 2
        offspring1 = parent1[:crossover_point] +
parent2[crossover_point:]
        offspring2 = parent2[:crossover_point] +
parent1[crossover_point:]

        # Replace the parents with offspring in the new_population_1
        new_population_1[index1] = offspring1
        new_population_1[index2] = offspring2

    return new_population_1, index_kurangdari_rate

# Mutation function
def mutate_chromosome(chromosome, mutation_rate):
    mutated_chromosome = []
    for bit in chromosome:
        if random.random() < mutation_rate:
            mutated_bit = 1 - bit  # Flip the bit
        else:
            mutated_bit = bit
        mutated_chromosome.append(mutated_bit)
    return mutated_chromosome

# Function to apply mutation to the entire population
def mutate_population(population, mutation_rate):
    mutated_population = []
    for chromosome in population:
        mutated_chromosome = mutate_chromosome(chromosome,
mutation_rate)
        mutated_population.append(mutated_chromosome)
    return mutated_population

# MAIN

def main():
    # Inisialisasi variabel
    non_improvement_count = 0
    max_non_improvement = 5  # Jumlah generasi tanpa peningkatan yang
diperbolehkan
    generation = 0
    best_fitness_1 = float("-inf")  # Awalnya diatur ke negatif tak
terhingga
    best_chromosome_1 = None
```

```
    populasi = create_populasi()


print("+-------------------------------------------------------------------
--------------------------------+")
    print("| Generation | Best Chromosome                          |
Best Fitness      | x1      | x2      |")

print("+-------------------------------------------------------------------
--------------------------------+")

    while generation < 100 and non_improvement_count <
max_non_improvement:
        a = -10.0  # Batas bawah interval
        b = 10.0  # Batas atas interval
        n = 4  # Jumlah bit yang digunakan untuk merepresentasikan x1
dan x2
        decoded_populasi = []
        nilai_objective = []

        for chromosome in populasi:
            x1, x2 = decode_chromosome(chromosome, a, b, n)
            objective_value = objectice_function(x1, x2)
            nilai_objective.append(objective_value)
            decoded_populasi.append((x1, x2))

        # ... Seleksi, crossover, mutasi, dan perhitungan fitness ...
        fitness_values = calculate_fitness(nilai_objective)
        best_fitness = max(fitness_values)
        best_index = fitness_values.index(best_fitness)
        best_chromosome = populasi[best_index]
        probabilities = calculate_probability(fitness_values)
        cumulative_probabilities =
calculate_cumulative_probability(probabilities)
        new_population =
select_chromosomes_using_roulette_wheel(cumulative_probabilities,
populasi)
        crossover_rate = 0.25  # 25%
        new_population_with_crossover,index_kurangdari_rate =
crossover(new_population, crossover_rate)
        mutation_rate = 0.25
        new_population_with_crossover =
mutate_population(new_population_with_crossover, mutation_rate)
```

```
        decoded_populasi = []
        nilai_objective = []
        populasi = new_population_with_crossover
        for chromosome in populasi:
            x1, x2 = decode_chromosome(chromosome, a, b, n)
            objective_value = objectice_function(x1, x2)
            nilai_objective.append(objective_value)
            decoded_populasi.append((x1, x2))

        fitness_values_new = calculate_fitness(nilai_objective)
        best_fitness_new = max(fitness_values_new)
        best_index_new = fitness_values_new.index(best_fitness_new)
        best_chromosome_new =
new_population_with_crossover[best_index_new]
        if best_fitness_new > best_fitness_1:
            best_fitness_1 = best_fitness_new
            best_chromosome_1 = best_chromosome_new
            non_improvement_count = 0  # Reset count
        else:
            non_improvement_count += 1

        x1, x2 = decode_chromosome(best_chromosome_1, a, b, n)
        best_chromosome_1_str = ", ".join(map(str, best_chromosome_1))
        print("| {:<10} | {:<40} | {:<11} | {:<7} | {:<7}
|".format(generation, best_chromosome_1_str, best_fitness_1, x1, x2))
        generation += 1

    print("+--------------------------------------+")
    print("Best Chromosome:", best_chromosome_1)
    print("Best Fitness:", best_fitness_1)
    print("x1=", x1)
    print("x2=", x2)

if __name__ == "__main__":
    main()
```

1. Import Library:
   Program mengimpor dua library, yaitu random dan math.
2. Fungsi Objektif (***objective_function***):

Ini adalah fungsi yang akan dioptimalkan. Fungsi ini mengambil dua parameter x1 dan x2 dan menghitung nilai objektif berdasarkan rumus matematika tertentu.

3. Membuat Kromosom (***create_chromosome***):

   Fungsi ini digunakan untuk membuat kromosom acak yang terdiri dari 8 bit (dalam hal ini, panjang kromosom adalah tetap).

4. Membuat Populasi Awal (***create_population***):

   Fungsi ini digunakan untuk membuat populasi awal. Populasi terdiri dari beberapa kromosom yang dihasilkan oleh fungsi ***create_chromosome***.

5. Decode Kromosom (***decode_chromosome***):

   Fungsi ini digunakan untuk mengurai kromosom menjadi dua nilai riil, yaitu x1 dan x2, yang berada dalam interval tertentu.

6. Menghitung Fitness (***calculate_fitness***):

   Fungsi ini digunakan untuk menghitung nilai fitness untuk setiap kromosom dalam populasi. Nilai fitness dihitung berdasarkan nilai objektif, dan semakin tinggi nilainya, semakin baik.

7. Menghitung Probabilitas (***calculate_probability***):

   Fungsi ini menghitung probabilitas munculnya setiap kromosom berdasarkan nilai fitness-nya. Probabilitas dihitung sebagai rasio fitness individu terhadap total fitness populasi.

8. Menghitung Probabilitas Kumulatif (***calculate_cumulative_probability***):

   Fungsi ini menghitung probabilitas kumulatif, yang akan digunakan dalam seleksi orang tua menggunakan metode "Roulette Wheel Selection."

9. Seleksi Orang Tua (***select_chromosomes_using_roulette_wheel***):

   Fungsi ini digunakan untuk memilih kromosom orang tua dari populasi dengan menggunakan "Roulette Wheel Selection," di mana probabilitas kumulatif digunakan untuk memilih kromosom.

10. Crossover:

    Crossover dilakukan dengan tingkat (rate) sebesar 25% (0.25). Ini berarti bahwa sekitar 25% dari populasi akan mengalami crossover untuk menghasilkan keturunan.

11. Mutasi:

    Mutasi dilakukan pada populasi yang telah mengalami crossover. Tingkat mutasi adalah 25% (0.25). Ini berarti sekitar 25% bit dalam setiap kromosom akan mengalami perubahan acak.

12. Algoritma Genetika:

    Seluruh algoritma Genetika dijalankan dalam fungsi main. Ini termasuk iterasi melalui generasi, menghitung nilai fitness, seleksi, crossover, mutasi, dan evaluasi hasil.

13. Output:

    Hasil terbaik, termasuk kromosom terbaik, nilai fitness terbaik, nilai x1, dan nilai x2, dicetak setiap generasi. Juga, program menghentikan iterasi jika tidak ada peningkatan dalam beberapa generasi (menggunakan ***max_non_improvement***).