**LAPORAN TUGAS**

**Aki Ananta**

**2009076026**

**Teknik Elektro**

1. *Listing Program*

import random

# Pembentukan Chromosome

def generate\_chromosome():

a = random.randint(0, 30)

b = random.randint(0, 10)

c = random.randint(0, 10)

d = random.randint(0, 10)

return [a, b, c, d]

# Inisialisasi Populasi

def initialize\_population(population\_size):

population = []

for \_ in range(population\_size):

chromosome = generate\_chromosome()

population.append(chromosome)

return population

# Evaluasi Chromosome

def fitness\_function(chromosome):

a, b, c, d = chromosome

equation\_result = a + 4 \* b + 2 \* c + 3 \* d

return abs(equation\_result - 30)

# Seleksi Chromosome

def selection(population):

fitness\_values = [1 / (1 + fitness\_function(chromosome)) for chromosome in population]

total\_fitness = sum(fitness\_values)

probabilities = [fitness / total\_fitness for fitness in fitness\_values]

cumulative\_probabilities = [sum(probabilities[:i+1]) for i in range(len(probabilities))]

selected\_population = []

for \_ in range(len(population)):

random\_number = random.random()

for i in range(len(cumulative\_probabilities)):

if random\_number <= cumulative\_probabilities[i]:

selected\_population.append(population[i])

break

return selected\_population

# Reproduksi (Crossover)

def crossover(parent1, parent2):

crossover\_point = random.randint(1, len(parent1)-1)

offspring1 = parent1[:crossover\_point] + parent2[crossover\_point:]

offspring2 = parent2[:crossover\_point] + parent1[crossover\_point:]

return offspring1, offspring2

# Mutasi

def mutation(chromosome, mutation\_rate):

mutated\_chromosome = chromosome.copy()

for i in range(len(mutated\_chromosome)):

if random.random() < mutation\_rate:

mutated\_chromosome[i] = generate\_chromosome()[i]

return mutated\_chromosome

# Algoritma Genetika

def genetic\_algorithm(population\_size, num\_generations, mutation\_rate):

population = initialize\_population(population\_size)

for generation in range(num\_generations):

print("Generation:", generation+1)

# Evaluasi populasi

evaluated\_population = [(chromosome, fitness\_function(chromosome)) for chromosome in population]

evaluated\_population.sort(key=lambda x: x[1])

best\_chromosome = evaluated\_population[0][0]

best\_fitness = evaluated\_population[0][1]

print("Best Solution:", best\_chromosome)

print("Best Fitness:", best\_fitness)

print()

# Seleksi populasi

selected\_population = selection([chromosome for chromosome, \_ in evaluated\_population])

# Reproduksi (Crossover)

offspring\_population = []

while len(offspring\_population) < population\_size:

parent1, parent2 = random.choices(selected\_population, k=2)

offspring1, offspring2 = crossover(parent1, parent2)

offspring\_population.append(offspring1)

offspring\_population.append(offspring2)

# Mutasi

mutated\_population = [mutation(chromosome, mutation\_rate) for chromosome in offspring\_population]

population = mutated\_population

return best\_chromosome

# Main program

population\_size = 6

num\_generations = 10

mutation\_rate = 0.1

best\_solution = genetic\_algorithm(population\_size, num\_generations, mutation\_rate)

print("Final Solution:", best\_solution)

1. Penjelasan
2. Fungsi ‘**generate\_chromosome()**’: Fungsi ini menghasilkan satu chromosome acak dengan nilai a, b, c, dan d yang sesuai dengan batasan yang ditentukan.
3. Fungsi ‘**initialize\_population(population\_size)**’: Fungsi ini membuat populasi awal dengan jumlah chromosome sebanyak population\_size. Setiap chromosome dihasilkan menggunakan fungsi generate\_chromosome().
4. Fungsi ‘**fitness\_function(chromosome)**’: Fungsi ini menghitung nilai fungsi objektif dari suatu chromosome, yaitu hasil dari persamaan a + 4b + 2c + 3d - 30. Semakin kecil nilai fungsi objektif, semakin baik chromosome tersebut memenuhi persamaan.
5. Fungsi ‘**selection(population)**’: Fungsi ini melakukan seleksi chromosome menggunakan pendekatan Roulette Wheel Selection. Pertama, nilai fitness dihitung untuk setiap chromosome dengan menggunakan fungsi fitness\_function(). Kemudian, probabilitas seleksi dihitung berdasarkan nilai fitness. Dalam implementasi ini, nilai fitness diubah menjadi probabilitas dengan membaginya dengan total fitness. Nilai kumulatif probabilitas juga dihitung untuk memudahkan seleksi berdasarkan probabilitas.
6. Fungsi ‘**crossover(parent1, parent2)**’: Fungsi ini melakukan operasi crossover antara dua chromosome induk (parent1 dan parent2). Posisi titik crossover dipilih secara acak, lalu offspring baru terbentuk dengan menggabungkan gen-gen dari kedua induk.
7. Fungsi mutation’**(chromosome, mutation\_rate)**’: Fungsi ini melakukan operasi mutasi pada suatu chromosome dengan probabilitas mutation\_rate. Pada setiap gen, jika angka acak yang dihasilkan lebih kecil dari mutation\_rate, maka gen tersebut akan diubah menjadi nilai acak yang sesuai dengan batasan.
8. Fungsi ‘**genetic\_algorithm(population\_size, num\_generations, mutation\_rate)**’: Fungsi ini merupakan inti dari algoritma genetika. Di dalamnya, dilakukan iterasi sebanyak ‘**num\_generations**’ untuk evolusi populasi. Pada setiap generasi, dilakukan evaluasi populasi, seleksi, reproduksi (crossover), dan mutasi. Kemudian, populasi yang baru terbentuk digunakan untuk generasi berikutnya.
9. Main program: Pada bagian ini, Anda dapat menentukan parameter seperti ‘**population\_size**’ (ukuran populasi awal), num\_generations (jumlah generasi), dan ‘**mutation\_rate**’ (tingkat mutasi). Kemudian, dipanggil fungsi genetic\_algorithm() dengan parameter tersebut untuk menjalankan algoritma genetika dan mendapatkan solusi terbaik.

Selama proses iterasi, hasil terbaik (chromosome dengan nilai fitness terkecil) dicetak pada setiap generasi. Setelah iterasi selesai, solusi terbaik yang ditemukan akan dicetak sebagai "Final Solution".