LAPORAN TUGAS

Nama: Muhammad Dimas Maulana

NIM: 2109076022

Prodi: Teknik Elektro

**ALGORITMA GENETIK**

**KASUS SOLUSI PERSAMAAN**

Tentukan solusi untuk masalah optimasi persamaan **a+4b+2c+3d = 30**. Cari nilai a, b, c, dan d yang memenuhi persamaan diatas. Silahkan menggunakan bahasa pemrograman favorit kalian dalam menyelesaikannya. Buatlah akun Github dan buat repository dengan nama “NamaLengkap-NIM”. Kumpulkan solusi dalam bentuk file source code (\*.py, \*.c, atau \*.cpp atau ekstensi lainnya sesuai bahasa pemrograman yang Anda pilih) dan laporan tugas (\*.doc/\*.docx) pada folder “Kecerdasan Buatan 2023” dalam repository masing-masing.

Petunjuk penyelesaian:

1. **Pembentukan chromosome**

Karena yang dicari adalah nilai a, b, c, d maka variabel  a, b, c, d dijadikan sebagai gen-gen pembentuk chromosome. Batasan nilai variabel a adalah bilangan integer 0 sampai 30. Sedangkan batasan nilai variabel b, c, dan d adalah bilangan integer 0 sampai 10.

1. **Inisialisasi**

Proses inisialisasi dilakukan dengan cara memberikan nilai awal gen-gen dengan nilai acak sesuai batasan yang telah ditentukan. Misalkan tentukan jumlah populasi adalah 6.

1. **Evaluasi Chromosome**

Permasalahan yang ingin diselesaikan adalah  nilai variabel a, b, c, dan d yang memenuhi persamaan a+4b+2c+3d = 30, maka fungsi\_objektif yang dapat digunakan untuk mendapatkan solusi adalah fungsi\_objektif (chromosome) = | (a+4b+2c+3d) – 30 |

1. **Seleksi Chromosome**

Proses seleksi dilakukan dengan cara membuat chromosome yang mempunyai fungsi\_objektif kecil mempunyai kemungkinan terpilih yang besar atau mempunyai nilai probabilitas yang tinggi.

Cara seleksi adalah sebagai berikut:

1. Untuk itu dapat digunakan fungsi fitness = (1/(1+fungsi\_objektif)).
2. Hitung probabilitas tiap kromosom yaitu: P[i] = fitness[i]/total\_fitness
3. Hitung nilai kumulatif probabilitas:
4. Bangkitkan bilangan acak R antara 0-1, kemudian pilih chromosome ke-k sebagai induk dengan syarat **C[k-1] < R < C[k].**
5. Kromosom baru terbentuk
6. **Crossover**

Metode yang digunakan salah satunya adalah one-cut point, yaitu memilih secara acak satu posisi dalam chromosome induk kemudian saling menukar gen. Chromosome yang dijadikan induk dipilih secara acak dan jumlah chromosome yang mengalami crossover dipengaruhi oleh parameter crossover\_rate (**ρc**). Pseudo-code untuk proses crossover adalah sebagai berikut:

1. **Mutasi**

Jumlah chromosome yang mengalami mutasi dalam satu populasi ditentukan oleh parameter mutation\_rate. Proses mutasi dilakukan dengan cara mengganti satu gen yang terpilih secara acak dengan suatu nilai baru yang didapat secara acak.

1. Pertama hitung dahulu panjang total gen yang ada dalam satu populasi.
2. Misal ρm tentukan 10% maka diharapkan ada 10% dari total\_gen yang mengalami populasi

Jawaban

# Listing Program:

import random

a\_lower\_bound = 0

a\_upper\_bound = 30

bcd\_lower\_bound = 0

bcd\_upper\_bound = 10

population\_size = 6

chromosome\_length = 4

crossover\_rate = 0.8

mutation\_rate = 0.1

max\_generations = 100

def generate\_chromosome():

chromosome = [random.randint(a\_lower\_bound, a\_upper\_bound)]

for \_ in range(chromosome\_length - 1):

chromosome.append(random.randint(bcd\_lower\_bound, bcd\_upper\_bound))

return chromosome

def evaluate\_chromosome(chromosome):

equation\_result = chromosome[0] + 4 \* chromosome[1] + 2 \* chromosome[2] + 3 \* chromosome[3]

fitness = abs(equation\_result - 30)

return fitness

def selection(population):

fitness\_values = [1 / (1 + evaluate\_chromosome(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\_parents = []

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

random\_number = random.random()

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

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

selected\_parents.append(population[i])

break

return selected\_parents

def crossover(parent1, parent2):

crossover\_point = random.randint(1, chromosome\_length - 1)

child1 = parent1[:crossover\_point] + parent2[crossover\_point:]

child2 = parent2[:crossover\_point] + parent1[crossover\_point:]

return child1, child2

def mutation(chromosome):

mutated\_chromosome = chromosome[:]

gene\_to\_mutate = random.randint(0, chromosome\_length - 1)

if gene\_to\_mutate == 0:

mutated\_chromosome[gene\_to\_mutate] = random.randint(a\_lower\_bound, a\_upper\_bound)

else:

mutated\_chromosome[gene\_to\_mutate] = random.randint(bcd\_lower\_bound, bcd\_upper\_bound)

return mutated\_chromosome

def genetic\_algorithm():

population = [generate\_chromosome() for \_ in range(population\_size)]

for generation in range(max\_generations):

fitness\_values = [evaluate\_chromosome(chromosome) for chromosome in population]

if 0 in fitness\_values:

index = fitness\_values.index(0)

solution = population[index]

print("Solution found in generation", generation)

print("Chromosome:", solution)

return solution

parents = selection(population)

offspring = []

for i in range(0, len(parents), 2):

if random.random() < crossover\_rate:

child1, child2 = crossover(parents[i], parents[i+1])

offspring.extend([child1, child2])

for i in range(len(offspring)):

if random.random() < mutation\_rate:

offspring[i] = mutation(offspring[i])

population = parents + offspring

print("Solution not found")

return None

genetic\_algorithm()

# Penjelasan:

Listing Program di atas adalah implementasi algoritma genetika untuk memecahkan masalah optimasi. Algoritma genetika adalah metode yang terinspirasi oleh teori evolusi dalam biologi, yang digunakan untuk mencari solusi yang optimal dalam ruang pencarian yang besar atau kompleks.

Berikut adalah penjelasan lengkap mengenai listing program diatas:

1. Mengimpor modul **random** yang digunakan untuk menghasilkan angka acak.
2. Mendefinisikan batas bawah (**a\_lower\_bound**) dan batas atas (**a\_upper\_bound**) untuk variabel **a**, serta batas bawah (**bcd\_lower\_bound**) dan batas atas (**bcd\_upper\_bound**) untuk variabel **b**, **c**, dan **d**.
3. Mendefinisikan parameter-parameter penting seperti ukuran populasi (**population\_size**), panjang kromosom (**chromosome\_length**), tingkat crossover (**crossover\_rate**), tingkat mutasi (**mutation\_rate**), dan jumlah generasi maksimum (**max\_generations**).
4. Membuat fungsi **generate\_chromosome()** yang menghasilkan kromosom acak. Kromosom terdiri dari bilangan acak dalam rentang batas yang telah ditentukan sebelumnya.
5. Membuat fungsi **evaluate\_chromosome(chromosome)** yang mengevaluasi kromosom berdasarkan persamaan yang diberikan. Fitness kromosom dihitung sebagai selisih absolut antara hasil persamaan dengan nilai target (30 dalam kasus ini).
6. Membuat fungsi **selection(population)** untuk melakukan seleksi orang tua berdasarkan fitness kromosom. Probabilitas seleksi didasarkan pada nilai fitness relatif dari setiap kromosom dalam populasi. Metode seleksi yang digunakan dalam coding ini adalah seleksi roulette-wheel, di mana probabilitas seleksi kromosom berbanding lurus dengan fitness relatifnya.
7. Membuat fungsi **crossover(parent1, parent2)** untuk melakukan operasi crossover antara dua kromosom orang tua. Poin crossover dipilih secara acak, dan anak-anak dihasilkan dengan menggabungkan bagian kromosom dari kedua orang tua.
8. Membuat fungsi **mutation(chromosome)** untuk melakukan mutasi pada kromosom. Gen yang akan dimutasi dipilih secara acak, dan nilainya diganti dengan bilangan acak baru sesuai dengan batas yang ditentukan.
9. Membuat fungsi **genetic\_algorithm()** untuk menjalankan algoritma genetika secara keseluruhan. Pada setiap generasi, fitness kromosom dievaluasi, dan jika ditemukan solusi optimal (fitness = 0), program akan menghentikan perhitungan dan mengeluarkan solusi tersebut. Jika solusi tidak ditemukan setelah mencapai jumlah generasi maksimum, program akan memberikan pesan bahwa solusi tidak ditemukan.
10. Memanggil fungsi **genetic\_algorithm()** untuk menjalankan algoritma genetika dan mencari solusi untuk masalah optimasi yang diberikan.

Dalam konteks ini, algoritma genetika digunakan untuk mencari nilai-nilai variabel **a**, **b**, **c**, dan **d** yang memenuhi persamaan yang diberikan (a + 4b + 2c + 3d = 30) dengan meminimalkan selisih absolut dari hasil persamaan dengan nilai target (30).