**LAPORAN TUGAS ALGORITMA GENETIK**

**MATA KULIAH KECERDASAN BUATAN**

Nama : Muhammad Nur Ihsan

NIM : 2109076026

Prodi : Teknik Elektro

1. Algoritma

Langkah-langkah untuk permasalahan optimasi pada persamaan **a+4b+2c+3d=30**.

* 1. Mendefinisikan variabel **a, b c,** dan **d** sebagai gen kromosom dengan batasan nilai **a** adalah 0 sampai 30, batasan nilai **b, c,** dan **d** adalah 0 sampai 10.
  2. Menentukan jumlah populasi sebanyak 10.
  3. Menetukan iterasi maksimal yaitu 100.
  4. Menghasilkan kromosom dengan menginisiasi gen-gen kromosom secara acak.
  5. Mengevaluasi dengan menggunakan kromosom **a, b c,** dan **d** untuk persamaan **a+4b+2c+3d=30.**
  6. Menyeleksi kromosom dengan menghitung total fitness dengan persamaan (1/(1+fungsi\_objektif)). Lalu menghitung probabilitas pada tiap kromosom dengan persamaan P[i] = fitness[i]/total\_fitness. Hitung nilai kumulatif probabilitas
  7. Memangkitkan bilangan acak R antara 0-1, kemudian pilih chromosome ke-k sebagai induk dengan syarat C[k-1] < R < C[k].
  8. Melakukan proses mutase.

1. Listing Program pada Bahasa *phyton*

import random

# Batasan nilai variabel

a\_min = 0

a\_max = 30

b\_min = 0

b\_max = 10

c\_min = 0

c\_max = 10

d\_min = 0

d\_max = 10

# Jumlah populasi dan iterasi

jumlah\_populasi = 10

max\_iterasi = 100

def generate\_chromosome():

    kromosom = {

        'a': random.randint(a\_min, a\_max),

        'b': random.randint(b\_min, b\_max),

        'c': random.randint(c\_min, c\_max),

        'd': random.randint(d\_min, d\_max)

    }

    return kromosom

def evaluate(kromosom):

    hasil\_perhitungan = kromosom['a'] + 4 \* kromosom['b'] + 2 \* kromosom['c'] + 3 \* kromosom['d']

    fitness = abs(hasil\_perhitungan - 30)

    return fitness

def calculate\_fitness(populasi):

    total\_fitness = sum(1 / (1 + evaluate(kromosom)) for kromosom in populasi)

    return [1 / (1 + evaluate(kromosom)) / total\_fitness for kromosom in populasi]

def select\_parent(populasi, fitness\_values):

    probabilitas\_kumulatif = [sum(fitness\_values[:i+1]) for i in range(len(fitness\_values))]

    nilai\_acak = random.random()

    for i, probability in enumerate(probabilitas\_kumulatif):

        if nilai\_acak <= probability:

            return populasi[i]

def crossover(parent1, parent2):

    cut\_point = random.randint(1, 3)  # Memilih posisi pemotongan acak

    child1 = {key: parent1[key] if index < cut\_point else parent2[key] for index, key in enumerate(parent1)}

    child2 = {key: parent2[key] if index < cut\_point else parent1[key] for index, key in enumerate(parent2)}

    return child1, child2

def mutate(kromosom):

    mutated\_gene = random.choice(['a', 'b', 'c', 'd'])

    if mutated\_gene == 'a':

        kromosom['a'] = random.randint(a\_min, a\_max)

    elif mutated\_gene == 'b':

        kromosom['b'] = random.randint(b\_min, b\_max)

    elif mutated\_gene == 'c':

        kromosom['c'] = random.randint(c\_min, c\_max)

    elif mutated\_gene == 'd':

        kromosom['d'] = random.randint(d\_min, d\_max)

    return kromosom

def genetic\_algorithm():

    # Inisialisasi populasi awal

    populasi = [generate\_chromosome() for \_ in range(jumlah\_populasi)]

    for iterasi in range(max\_iterasi):

        fitness\_values = calculate\_fitness(populasi)

        # Seleksi orang tua

        parents = [select\_parent(populasi, fitness\_values) for \_ in range(jumlah\_populasi)]

        # Crossover

        offspring = []

        for i in range(0, jumlah\_populasi, 2):

            parent1 = parents[i]

            parent2 = parents[i+1]

            child1, child2 = crossover(parent1, parent2)

            offspring.append(child1)

            offspring.append(child2)

        # Mutasi

        for i in range(jumlah\_populasi):

            if random.random() < nilai\_mutasi:

                offspring[i] = mutate(offspring[i])

        # Menggabungkan populasi sebelumnya dengan offspring

        populasi += offspring

        # Memilih populasi baru berdasarkan fitness

        populasi = sorted(populasi, key=lambda x: evaluate(x))

        populasi = populasi[:jumlah\_populasi]

        kromosom\_terbaik = populasi[0]

        best\_fitness = evaluate(kromosom\_terbaik)

        print(f"Iterasi {iterasi+1}: Kromosom Terbaik: {kromosom\_terbaik}, Fitness: {best\_fitness}")

        if best\_fitness == 0:

            break

    return kromosom\_terbaik

# Menjalankan algoritma genetika

nilai\_mutasi = 0.1

solusi\_terbaik = genetic\_algorithm()

print("Solusi Terbaik:", solusi\_terbaik)