Nama: vincent candi

NIM: 2109076011

# 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 adalah implementasi algoritma genetika untuk menyelesaikan sebuah masalah optimasi., yang digunakan untuk mencari solusi yang optimal dalam ruang pencarian yang besar atau kompleks.

1. Import modul random: Digunakan untuk menghasilkan angka acak yang diperlukan dalam algoritma genetika.
2. Deklarasi variabel batas bawah dan batas atas:

a\_lower\_bound = 0 dan a\_upper\_bound = 30: Menentukan batas bawah dan batas atas untuk variabel a.

bcd\_lower\_bound = 0 dan bcd\_upper\_bound = 10: Menentukan batas bawah dan batas atas untuk variabel b, c, dan d.

1. Pengaturan parameter algoritma genetika:

population\_size = 6: Menentukan ukuran populasi, yaitu jumlah kromosom dalam satu generasi.

chromosome\_length = 4: Menentukan panjang kromosom, yaitu jumlah gen dalam satu kromosom.

crossover\_rate = 0.8: Menentukan probabilitas terjadinya operasi crossover antara dua orangtua.

mutation\_rate = 0.1: Menentukan probabilitas terjadinya operasi mutasi pada setiap gen dalam kromosom.

max\_generations = 100: Menentukan jumlah maksimum generasi yang akan dievaluasi sebelum algoritma berhenti.

1. Fungsi generate\_chromosome(): Menghasilkan satu kromosom baru dengan menggunakan angka acak sesuai dengan batas bawah dan batas atas yang telah ditentukan. Kromosom terdiri dari empat gen, dengan gen pertama diambil dari rentang a\_lower\_bound hingga a\_upper\_bound, dan gen berikutnya diambil dari rentang bcd\_lower\_bound hingga bcd\_upper\_bound.
2. Fungsi evaluate\_chromosome(chromosome): Menghitung nilai fitness (kesesuaian) dari suatu kromosom. Dalam kasus ini, fungsi ini menghitung hasil persamaan dengan mengalikan setiap gen dengan faktor tertentu dan menjumlahkannya. Kemudian, nilai kesesuaian dihitung dengan menghitung selisih absolut antara hasil persamaan dengan nilai target 30.
3. Fungsi selection(population): Melakukan seleksi orangtua berdasarkan nilai fitness. Fungsi ini menghitung probabilitas seleksi berdasarkan nilai fitness relatif setiap kromosom, membangun distribusi kumulatif probabilitas, dan memilih orangtua berdasarkan angka acak yang dibangkitkan.
4. Fungsi crossover(parent1, parent2): Melakukan operasi crossover antara dua orangtua untuk menghasilkan dua anak. Pada kasus ini, titik pemotongan crossover dipilih secara acak, dan gen-gene setelah titik tersebut ditukar antara kedua orangtua untuk menghasilkan anak-anak baru.
5. Fungsi mutation(chromosome): Melakukan operasi mutasi pada suatu kromosom dengan probabilitas mutasi yang telah ditentukan. Fungsi ini memilih secara acak satu gen dalam kromosom, dan jika gen tersebut adalah gen pertama, maka digantikan dengan

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).