

Laporan Tugas Besar Kecerdasan Buatan II

Diajukan untuk memenuhi Tugas Besar
Mata Kuliah Kecerdasan Buatan



Disusun Oleh :

Glorious Satria Dhamang Aji
1302213105, S1 Rekayasa Perangkat Lunak, Fakultas Informatika, Universitas
Telkom,
glorioussatria@student.telkomuniversity.ac.id

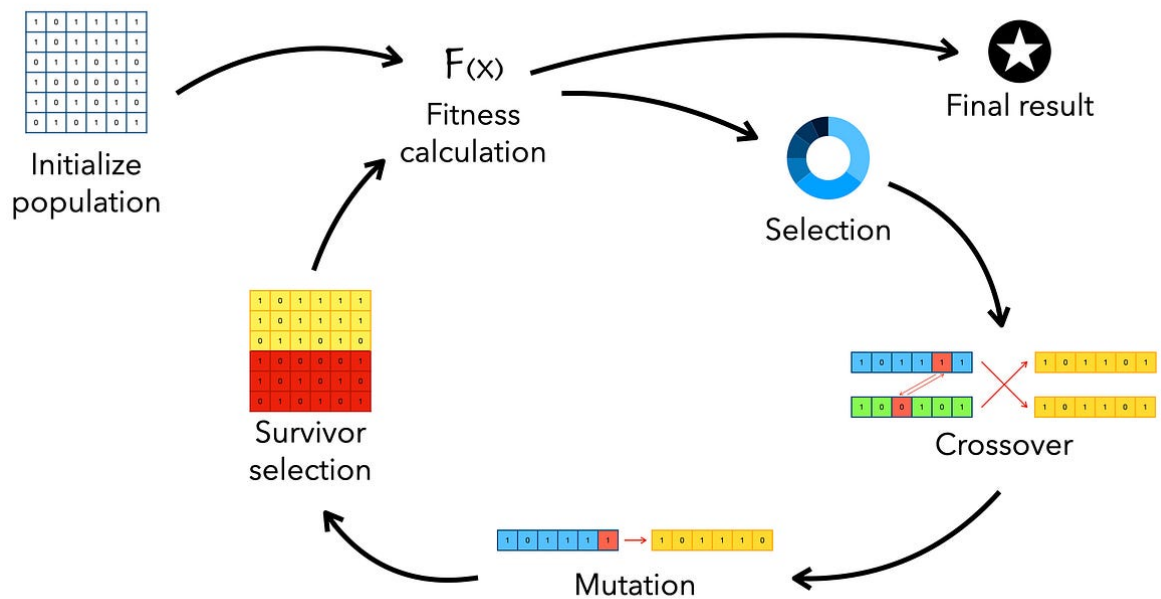
Kevin
1302210019, S1 Rekayasa Perangkat Lunak, Fakultas Informatika, Universitas
Telkom,
kevinwijaya@student.telkomuniversity.ac.id

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK
TELKOM UNIVERSITY
KOTA BANDUNG
2023

Pengenalan Algoritma Genetik.....	3
Algoritma Genetika untuk Pencarian Nilai Minimum.....	4
1. Ukuran populasi, rancangan kromosom, dan cara decode.....	5
a. Ukuran populasi.....	5
b. Rancangan kromosom.....	5
c. Cara decode.....	5
2. Metode Pemilihan Orang Tua.....	6
3. Metode Operasi Genetik.....	6
a. Crossover.....	6
b. Mutation.....	7
4. Probabilitas Metode Operasi Genetik (P_c dan P_m).....	7
a. Probabilitas Crossover.....	7
b. Probabilitas Mutasi.....	7
5. Metode pergantian generasi (seleksi survivor).....	8
6. Kriteria penghentian evolusi (loop).....	8
Proses yang Harus Diimplementasikan ke Dalam Baris-Baris Program.....	9
1. Inisialisasi populasi.....	9
2. Dekode kromosom.....	9
3. Perhitungan fitness.....	10
4. Pemilihan orangtua.....	10
5. Crossover (pindah silang).....	11
6. Mutasi.....	14
7. Pergantian Generasi.....	15
Hasil Observasi.....	16
1. Proses generate populasi, kromosom, decode, dan nilai fitness:.....	16
2. Proses berhenti dan penentuan kromosom terbaik:.....	18
3. Kesimpulan:.....	19
Lampiran.....	20

Pengenalan Algoritma Genetik

Algoritma genetika adalah algoritma komputasi yang diinspirasi teori evolusi yang kemudian diadopsi menjadi algoritma komputasi untuk mencari solusi suatu permasalahan dengan cara yang lebih “alamiah” (Hermawanto, hal. 1). Algoritma Genetika sering digunakan untuk masalah matematis dan pembuatan simulasi dari suatu model. Berikut adalah ilustrasi tahapan di dalam Algoritma Genetika.



Gambar 1.0 Ilustrasi Algoritma Genetika

Algoritma Genetika menggunakan prinsip-prinsip dasar kehidupan yang adalah representasi genetika dan dapat diambil dari proses perkawinan dari orang tua yang menghasilkan individu yang memiliki karakteristik tertentu. Proses inialisasi kumpulan individu yang akan menjadi orang tua sampai hingga mutasi dari individu yang akan menjadi anak dilakukan di dalam algoritma ini guna untuk mendapatkan hasil terbaik dari permasalahan yang ingin diselesaikan.

Algoritma Genetika untuk Pencarian Nilai Minimum

Penggunaan algoritma genetika sering diasosiasikan dengan penyelesaian masalah matematis. Sebagai contoh adalah pencarian nilai minimum yang sulit dipecahkan menggunakan metode matematika konvensional. Di dalam kasus ini, diberikan fungsi sebagai berikut:

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

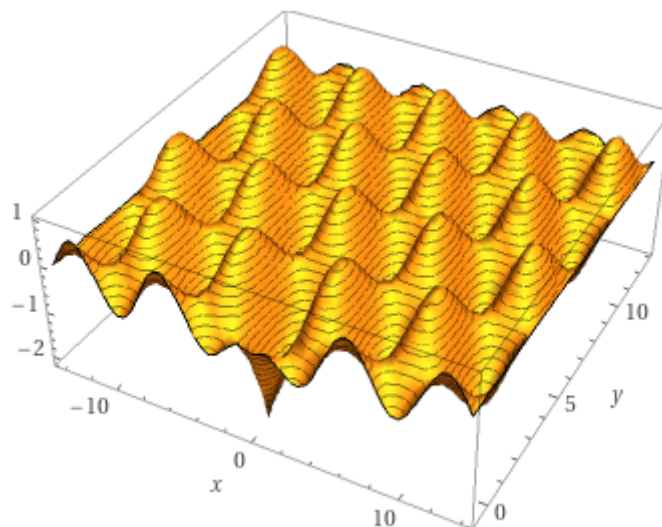
Gambar 2.0 Fungsi Matematis

Pencarian nilai minimum dapat dilakukan dengan menentukan batas dari nilai minimum tersebut. Nilai minimum yang diberikan adalah sebagai berikut:

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

Gambar 3.0 Batas Atas dan Batas Bawah

Sedemikian sehingga nilai minimum yang harus dicari harus berada di sekitar nilai -10 hingga 10 untuk kedua nilai x_1 dan x_2 . Nilai minimum ini memiliki visualisasi 3D plot sebagai berikut:



Gambar 4.0 3D Plot Nilai Minimum $f(x_1, x_2)$

Dengan kompleksitas tersebut, pemilihan metode algoritma genetika untuk penyelesaian fungsi tersebut adalah langkah yang tepat. Adapun langkah-langkah yang perlu dipertimbangkan dalam implementasi algoritma genetik ini adalah:

1. Ukuran populasi, rancangan kromosom, dan cara decode-nya
2. Metode pemilihan orangtua
3. Metode operasi genetik (pindah silang dan mutasi)
4. Probabilitas operasi genetik

Adapun pengambilan keputusan yang sudah dilakukan sehubungan dengan algoritma ini adalah sebagai berikut:

1. Ukuran populasi, rancangan kromosom, dan cara decode

a. Ukuran populasi

Untuk menghindari posibilitas hasil perhitungan jatuh ke dalam optimum lokal, maka ukuran populasi yang ditentukan adalah sebanyak 20 individu untuk 1 populasi. Dipilihnya angka 20 pun juga cara agar kompleksitas dari setiap populasi masih bisa diukur.

b. Rancangan kromosom

Ukuran kromosom pun ditentukan sepanjang array berjumlah 6 yang memiliki data integer agar tingkat kompleksitas dari hasil yang didapatkan masih dapat diukur dan dikendalikan.

c. Cara decode

Karena kromosom yang dihasilkan bertipe data integer, maka metode *decode* yang digunakan adalah representasi integer yang memiliki formula :

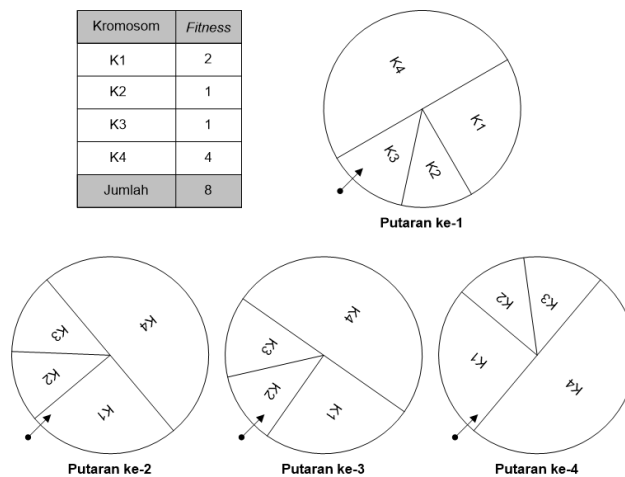
$$x = r_b + \frac{(r_a - r_b)}{\sum_{i=1}^N 9 \cdot 10^{-i}} (g_1 \cdot 10^{-1} + g_2 \cdot 10^{-2} + \dots + g_N \cdot 10^{-N})$$

Gambar 5.0 Formula Representasi Integer

Formula tersebut dibatasi dengan batas bawah dan batas atas yang sudah ditentukan juga sebelumnya di gambar 3.0.

2. Metode Pemilihan Orang Tua

Metode pemilihan orang tua yang digunakan adalah Roulette Wheel. Roulette wheel digunakan karena dianggap metode yang paling proporsional untuk nilai fitness yang sebelumnya sudah ditentukan.

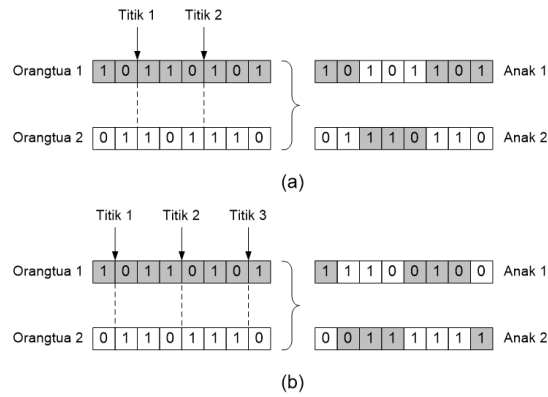


Gambar 6.0 Ilustrasi Roulette Wheel

3. Metode Operasi Genetik

a. Crossover

Metode Crossover yang digunakan adalah crossover banyak titik yang memiliki proses pembagian kedua orang tua untuk ditukar dan disatukan menjadi satu bentuk individu child yang baru. pewarisan gen-gen dilakukan secara menyilang.



Gambar 7.0 Ilustrasi Crossover

b. Mutation

Metode mutation yang digunakan adalah metode mutation representasi integer untuk mengcover tipe data kromosom yang digunakan yang adalah tipe data integer.

c. Elitisme

4. Probabilitas Metode Operasi Genetik (P_c dan P_m)

a. Probabilitas Crossover

Probabilitas crossover, ditentukan dengan $\text{probability_of_crossover} < 0.9$. Ini berarti ada 90% kemungkinan crossover terjadi antara dua orang tua saat menghasilkan anak.

```
def crossover(orangtua1, orangtua2):
    anak1, anak2, anakan = [], [], []
    probability_of_crossover = random.random()
    if probability_of_crossover < 0.9:
        anak1[:1], anak1[1:] = orangtua1[:1],
        orangtua2[1:], anak2[1:] = orangtua2[1:],
        orangtua1[1:],
        anakan.append(anak1)
        anakan.append(anak2)
    else:
        anakan.append(orangtua1)
        anakan.append(orangtua2)
    return anakan
```

b. Probabilitas Mutasi

Probabilitas mutasi untuk setiap kromosom anak, ditentukan dengan $\text{random.randint}(0, 9) < 0.1$. Ini berarti ada 10% kemungkinan mutasi terjadi pada setiap gen kromosom anak.

```
def mutation(anak1, anak2):
    for i in range(len(anak1)):
        probability_of_mutation_for_anak1 = random.randint(0,9)
        if probability_of_mutation_for_anak1 < 0.1:
            anak1[i] = random.randint(0,9)

        probability_of_mutation_for_anak2 = random.randint(0,9)
        if probability_of_mutation_for_anak2 < 0.1:
            anak2[i] = random.randint(0,9)

    return anak1, anak2
```

5. Metode pergantian generasi (seleksi survivor)

Seleksi dilakukan dengan membandingkan kromosom terbaik dari generasi sebelumnya dengan kromosom terburuk dari generasi saat ini. Jika kromosom terbaik dari generasi sebelumnya memiliki fitness yang lebih baik dari kromosom terburuk generasi saat ini, maka kromosom terburuk akan digantikan dengan kromosom terbaik dari generasi sebelumnya.

Metode pergantian generasi yang diterapkan terjadi di dalam loop yang mengatur evolusi populasi dari satu generasi ke generasi berikutnya. Ini termasuk pemilihan orang tua, crossover, mutasi anak-anak, dan penentuan populasi generasi berikutnya.

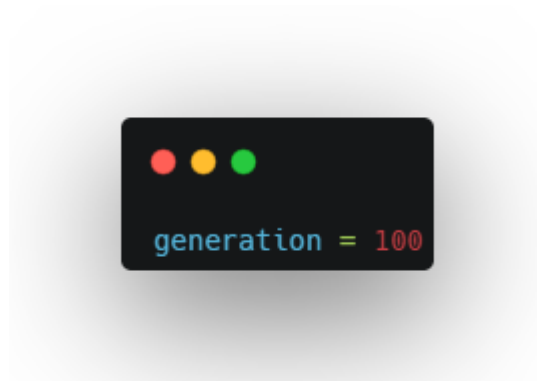
```
# Perulangan untuk melakukan seleksi orang tua, crossover dan mutasi anak untuk mendapatkan populasi
generasi selanjutnya
if genes != generation-1 :
    for i in range(population_total // 2):
        parent_1 = parent_roulette_selection(population, fitness_data, total_fitness)
        parent_2 = parent_roulette_selection(population, fitness_data, total_fitness)
        print(parent_1, parent_2)
        childs = crossover(parent_1, parent_2)
        print(childs)
        child_1, child_2 = mutation(childs[0], childs[1])

        new_population.append(child_1)
        new_population.append(child_2)

    population = new_population
```


6. Kriteria penghentian evolusi (loop)

Evolusi dilakukan selama sejumlah generasi tertentu, yaitu sebanyak generation yang ditentukan (100 dalam contoh). Setelah jumlah generasi ini tercapai, algoritma akan berhenti dan menghasilkan solusi terbaik yang ditemukan selama proses evolusi.



Proses yang Harus Diimplementasikan ke Dalam Baris-Baris Program

1. Inisialisasi populasi

Inisialisasi populasi adalah langkah awal di mana kita membuat sejumlah individu (20 dalam contoh) dengan panjang kromosom 6. Ini dilakukan dengan menggunakan fungsi generate population.

```
# Generate the population
jumlah_individu = 20
panjang_individu = 6
population =
generate_population(jumlah_individu,
panjang_individu)
```

2. Dekode kromosom

Dekode kromosom adalah proses mengubah rangkaian angka biner pada kromosom menjadi angka nyata yang sesuai dengan batas yang telah ditentukan (x1_batas dan x2_batas).

```
#decode cromosome
def decode(cromosome, batas):
    kali, pembagi = 0, 0
    for i in range(len(cromosome)) :
        num = cromosome[i]
        kali += num * (10**-(i+1))
        pembagi += 9 * (10**-(i+1))

    return batas[0] + (((batas[1] -
batas[0]) / pembagi) * kali)
```

3. Perhitungan fitness

Fitness dihitung berdasarkan fungsi objektif yang ditentukan (function(x, y)). Semakin kecil nilai fungsi objektif, semakin tinggi fitnessnya.

```
#rumus yang akan dicari nilai minimumnya
def fitness_calculation(x,y):
    return 1 /(((sin(x)*cos(y)+0.8*exp(1-sqrt(x**2+y**2)))) * -1) +
    0.0001)
```

4. Pemilihan orangtua

Pemilihan orang tua (parent selection) dilakukan dengan metode roulette wheel selection. Kromosom dengan fitness lebih tinggi memiliki probabilitas yang lebih besar untuk dipilih sebagai orang tua.

```
# function to do the parent selection process
def parent_roulette_selection(population, fitness,fitness_total):
    probability = random.random()
    index_parent = 0
    while probability > 0:
        probability -= fitness[index_parent]/fitness_total
        index_parent += 1
        if index_parent == len(population) - 1:
            break
    return population[index_parent]
```

5. Crossover (pindah silang)

Crossover adalah proses di mana dua orang tua menggabungkan sebagian dari kromosom mereka untuk menghasilkan anak. Kami mengidentifikasi beberapa metode crossover yang sesuai dengan representasi integer dan memilih metode crossover terbaik. Beberapa metode yang menjadi pertimbangan adalah order crossover, simple crossover, single arithmetic crossover, dan whole arithmetic crossover. Dan pada akhirnya, dipilih order crossover yang memiliki hasil paling baik dibandingkan dengan crossover yang lain.

```
# crossover and breed new individuals
def crossover(orangtua1, orangtua2):
    anak1, anak2, anakan = [], [], []
    probability_of_crossover = random.random()
    if probability_of_crossover < 0.9:
        anak1[:1], anak1[1:] = orangtua1[:1],
        orangtua2[:1], anak2[1:] = orangtua2[:1],
        orangtua1[1:],
        orangtua2[1:]
        anakan.append(anak1)
        anakan.append(anak2)
    else:
        anakan.append(orangtua1)
        anakan.append(orangtua2)
    return anakan
```

Metode yang menjadi Konsiderasi

a. Order Crossover

- Dalam metode ini, dua kromosom orang tua pertama kali dipotong pada dua titik acak.
- Kemudian, bagian tengah dari masing-masing kromosom dipertukarkan antara kedua orang tua untuk membuat dua anak.
- Ini menghasilkan dua anak yang mewarisi beberapa sifat dari masing-masing orang tua.

```

Orang Tua 1: 1 2 3 | 4 5 6
Orang Tua 2: 6 5 4 | 3 2 1

Anak 1: 1 2 3 | 3 2 1
Anak 2: 6 5 4 | 4 5 6

```

b. Simpel Crossover

- Metode ini memilih titik acak untuk memotong kromosom orang tua.
- Kemudian, bagian kiri dari salah satu orang tua digabungkan dengan bagian kanan orang tua lainnya untuk menghasilkan dua anak.
- Ini adalah metode crossover sederhana di mana sebagian besar sifat masing-masing orang tua ditransfer ke anak.

```

Orang Tua 1: 1 2 3 4 5 6
Orang Tua 2: 7 8 9 0 1 2

Anak 1: 1 2 3 | 0 1 2
Anak 2: 7 8 9 | 4 5 6

```

c. Single Arithmetic Crossover

- Dalam metode ini, setiap gen anak dihitung sebagai kombinasi tertentu dari gen orang tua menggunakan bobot.
- Sebuah titik crossover dipilih, dan anak-anak diperoleh dengan menggabungkan gen orang tua berdasarkan bobot.

```
Orang Tua 1: 2 3 4 5 6
Orang Tua 2: 7 8 9 0 1

Bobot: 0.4 0.6 0.5 0.3 0.7 (misalnya)

Anak 1: (0.4 * 2) + (0.6 * 7), (0.4 * 3) + (0.6 * 8),
Anak 2: (0.3 * 2) + (0.7 * 7), (0.3 * 3) + (0.7 * 8),
...
```

d. Whole Arithmetic Crossover

- Mirip dengan Single Arithmetic Crossover, tetapi dalam metode ini, seluruh kromosom anak-anak dihitung sebagai kombinasi tertentu dari kromosom orang tua menggunakan bobot.
- Tidak ada titik potong dalam metode ini.

```
Orang Tua 1: 2 3 4 5 6
Orang Tua 2: 7 8 9 0 1

Bobot: 0.4 0.6 0.5 0.3 0.7 (misalnya)

Anak 1: (0.4 * 2) + (0.6 * 7), (0.4 * 3) + (0.6 * 8),
Anak 2: (0.4 * 2) + (0.6 * 7), (0.4 * 3) + (0.6 * 8),
...
```

6. Mutasi

Mutasi adalah proses acak di mana satu atau beberapa gen dalam kromosom anak dapat berubah. Seperti halnya crossover, Kami mengidentifikasi setiap jenis mutasi yang cocok dengan representasi integer untuk mendapatkan hasil paling optimal untuk algoritma ini. Metode mutasi tersebut antara lain adalah mutasi acak, mutasi membalik integer, dan mutasi perlahan

```
# Mutation and add new individuals to the population
def mutation(anak1, anak2):
    for i in range(len(anak1)):
        probability_of_mutation_for_anak1 = random.randint(0,9)
        if probability_of_mutation_for_anak1 < 0.1:
            anak1[i] = random.randint(0,9)

        probability_of_mutation_for_anak2 = random.randint(0,9)
        if probability_of_mutation_for_anak2 < 0.1:
            anak2[i] = random.randint(0,9)

    return anak1, anak2
```

7. Pergantian Generasi

Pergantian generasi adalah proses utama algoritma genetika. Populasi mengalami evolusi sebanyak jumlah generasi yang ditentukan (generation, dalam contoh ini 50).

Setiap generasi melibatkan perhitungan fitness, seleksi orang tua, crossover, mutasi, dan pembentukan populasi generasi berikutnya.

```
for genes in range(generation):
    # Inisialisasi variabel untuk proses perhitungan algoritma genetika
    kromosom_data, best_kromosom, bad_kromosom, fitness_data, new_population, child = [], [], [], [],
    [], []
    total_fitness, count_kromosom, index = 0, 999, 0

    print('\n=====')
    print('Generasi', genes+1)
    print('=====')

    # Perulangan untuk mencari nilai phenotype dan nilai fungsi / fitness pada setiap kromosom
    for i, kromosom in enumerate(population):
        kromosom_a, kromosom_b = split_kromosom(kromosom)
        x1 = decode(kromosom_a, x1_batas)
        x2 = decode(kromosom_b, x2_batas)

        fitness_value = function(x1, x2)
        fitness_data.append(fitness_value)
        total_fitness += fitness_value

    # Pencarian Fitness Terkecil Dalam Suatu Generasi
    if genes != 0 and fitness_value < count_kromosom:
        count_kromosom = fitness_value
        bad_kromosom = [fitness_value, kromosom, i]

    # Pemilihan Kromosom Dengan Fitness Terbaik
    best_kromosom = best_kromosom_selection(population)

    print("Kromosom Terbaik :", best_kromosom[0])
    print("Fitness Terbaik :", best_kromosom[1])

    # Proses Elitisme untuk memasukkan kromosom terbaik pada generasi sebelumnya
    if genes != 0:
        most_best = sorted(best_kromosom_generation, key=lambda x: x[1], reverse=True)[0]
        population, total_fitness = elitisme(population, most_best, bad_kromosom, total_fitness)

    best_kromosom_generation.append(best_kromosom)

    # Perulangan untuk melakukan seleksi orang tua, crossover dan mutasi anak untuk mendapatkan
    populasi generasi selanjutnya
    if genes != generation-1:
        for i in range(population_total // 2):
            parent_1 = parent_roulette_selection(population, fitness_data, total_fitness)
            parent_2 = parent_roulette_selection(population, fitness_data, total_fitness)
            print(parent_1, parent_2)
            childs = crossover(parent_1, parent_2)
            print(childs)
            child_1, child_2 = mutation(childs[0], childs[1])

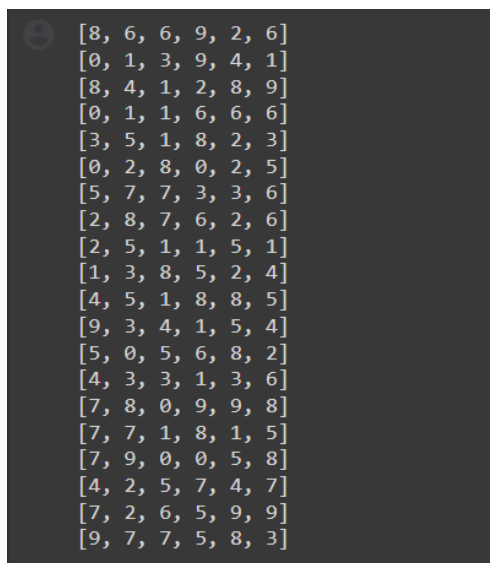
            new_population.append(child_1)
            new_population.append(child_2)

        population = new_population
```


Hasil Observasi

1. Proses generate populasi, kromosom, decode, dan nilai fitness:

Awalnya, kami menginisialisasi populasi dengan 20 individu, masing-masing memiliki kromosom dengan panjang 6 digit. Kromosom-kromosom ini diisi dengan angka-angka acak antara 0 hingga 9.



```
[8, 6, 6, 9, 2, 6]
[0, 1, 3, 9, 4, 1]
[8, 4, 1, 2, 8, 9]
[0, 1, 1, 6, 6, 6]
[3, 5, 1, 8, 2, 3]
[0, 2, 8, 0, 2, 5]
[5, 7, 7, 3, 3, 6]
[2, 8, 7, 6, 2, 6]
[2, 5, 1, 1, 5, 1]
[1, 3, 8, 5, 2, 4]
[4, 5, 1, 8, 8, 5]
[9, 3, 4, 1, 5, 4]
[5, 0, 5, 6, 8, 2]
[4, 3, 3, 1, 3, 6]
[7, 8, 0, 9, 9, 8]
[7, 7, 1, 8, 1, 5]
[7, 9, 0, 0, 5, 8]
[4, 2, 5, 7, 4, 7]
[7, 2, 6, 5, 9, 9]
[9, 7, 7, 5, 8, 3]
```

Setiap individu dalam populasi kemudian dipecah menjadi dua bagian, yang mewakili dua variabel, x dan y. Setiap bagian kromosom diubah menjadi nilai numerik menggunakan fungsi decode dengan memperhitungkan batas nilai yang telah ditentukan.

```
Original Chromosome: [8, 6, 6, 9, 2, 6]
Split Chromosome A: [8, 6, 6]
Split Chromosome B: [9, 2, 6]
Decoded x1: 7.337337337337338
Decoded x2: 8.538538538538539

Original Chromosome: [0, 1, 3, 9, 4, 1]
Split Chromosome A: [0, 1, 3]
Split Chromosome B: [9, 4, 1]
Decoded x1: -9.73973973973974
Decoded x2: 8.83883883883884

Original Chromosome: [8, 4, 1, 2, 8, 9]
Split Chromosome A: [8, 4, 1]
Split Chromosome B: [2, 8, 9]
Decoded x1: 6.836836836836838
Decoded x2: -4.214214214214214

Original Chromosome: [0, 1, 1, 6, 6, 6]
Split Chromosome A: [0, 1, 1]
Split Chromosome B: [6, 6, 6]
Decoded x1: -9.77977977977978
Decoded x2: 3.3333333333333357

Original Chromosome: [3, 5, 1, 8, 2, 3]
Split Chromosome A: [3, 5, 1]
Split Chromosome B: [8, 2, 3]
Decoded x1: -2.972972972972972
Decoded x2: 6.476476476476478
```

Nilai x dan y digunakan untuk menghitung nilai fitness menggunakan rumus yang didefinisikan dalam fungsi function.

```
Original Chromosome: [8, 6, 6, 9, 2, 6]
Split Chromosome A: [8, 6, 6]
Split Chromosome B: [9, 2, 6]
Decoded x1: 7.337337337337338
Decoded x2: 8.538538538538539
Objective Function : 0.5497711192120478
Fitness Value : 1.8186079702330542

Original Chromosome: [0, 1, 3, 9, 4, 1]
Split Chromosome A: [0, 1, 3]
Split Chromosome B: [9, 4, 1]
Decoded x1: -9.73973973973974
Decoded x2: 8.83883883883884
Objective Function : 0.2581024581607867
Fitness Value : 3.8729298207427774

Original Chromosome: [8, 4, 1, 2, 8, 9]
Split Chromosome A: [8, 4, 1]
Split Chromosome B: [2, 8, 9]
Decoded x1: 6.836836836836838
Decoded x2: -4.214214214214214
Objective Function : 0.25053073704490353
Fitness Value : 3.98993360427631

Original Chromosome: [0, 1, 1, 6, 6, 6]
Split Chromosome A: [0, 1, 1]
Split Chromosome B: [6, 6, 6]
Decoded x1: -9.77977977977978
Decoded x2: 3.3333333333333357
Objective Function : 0.34115128126351957
Fitness Value : 2.9303919278995596
```

2. Proses berhenti dan penentuan kromosom terbaik:

```
# Memanggil fungsi untuk menentukan kromosom terbaik pada keseluruhan generasi
print('\n=====')
print('Hasil Akhir Kromosom Terbaik')
print('=====')
print('Kromosom Terbaik      = ', most_best[0])
print('Phenotype x           = ', most_best[2])
print('Phenotype y           = ', most_best[3])
print('Nilai Fungsi / Fitness = ', most_best[1])
print('=====')
```

```
=====
Hasil Akhir Kromosom Terbaik
=====
Kromosom Terbaik      = [2, 1, 7, 8, 7, 8]
Phenotype x           = 4.154154154154156
Phenotype y           = 7.577577577577578
Nilai Fungsi / Fitness = -0.1604013680760776
=====
```

Pada bagian ini, terjadi proses akhir dari algoritma genetika yang telah dijalankan selama sejumlah generasi yang ditentukan (dalam kasus ini, 100 generasi). Proses ini bertujuan untuk menentukan kromosom terbaik dari keseluruhan generasi dan menampilkan hasil akhir.

"Kromosom Terbaik" menampilkan kromosom terbaik yang telah ditemukan. Kromosom ini adalah representasi genetik dari solusi yang mendekati solusi optimal.

"Phenotype x" dan "Phenotype y" adalah hasil decode dari kromosom terbaik untuk menghasilkan dua fenotipe, x dan y. Fenotipe ini adalah nilai-nilai yang sesuai dengan

representasi kromosom dan dapat digunakan untuk menginterpretasikan solusi dalam konteks permasalahan yang ada. "Nilai Fungsi / Fitness" adalah nilai dari fungsi objektif yang dihitung berdasarkan fenotipe x dan y yang ditemukan melalui decode kromosom terbaik. Nilai ini mengindikasikan seberapa baik solusi ini dalam konteks permasalahan yang diberikan.

3. Kesimpulan:

```
=====
Hasil Akhir Kromosom Terbaik
=====
Kromosom Terbaik      = [2, 1, 7, 8, 7, 8]
Phenotype x           = 4.154154154154156
Phenotype y           = 7.577577577577578
Nilai Fungsi / Fitness = -0.1604013680760776
=====
```

Proses ini adalah implementasi algoritma genetika untuk mencari nilai minimum dari fungsi objektif yang kompleks. Dalam setiap generasi, populasi dianalisis, dan individu dengan nilai fitness terbaik disimpan. Proses seleksi orang tua, crossover, dan mutasi digunakan untuk menghasilkan generasi berikutnya. Setelah 50 generasi, kromosom terbaik dari seluruh proses evolusi adalah yang memiliki nilai fitness terendah. Hasil akhir adalah kromosom terbaik, nilai x dan y yang sesuai, serta nilai minimum dari fungsi objektif yang telah dicapai melalui algoritma genetika.

Lampiran

Source Code :

https://colab.research.google.com/drive/1Ng_nAih2F7SwZnxQbFm-hCMWsYCNBwLB?usp=sharing#scrollTo=NhlvSMYR8Emj