**SEARCHING : GENETIC ALGORITHM**

**Laporan Observasi**

**oleh: Muhammad Shulhannur**

**PROBLEM :**

Bangunlah sebuah algoritma genetika yang menghasilkan nilai minimum dari fungsi :

$$h(x_1, x_2) = \cos(x_1)\sin(x_2) - \frac{x_1}{(x_2^2 + 1)}$$

dengan batasan $-1 \leq x_1 \leq 2$ dan $-1 \leq x_2 \leq 1$.

**HAL-HAL YANG DIOBSERVASI :**

1. Penggunaan Bahasa Pemrograman, Tools, dan Library

Penulis menggunakan bahasa pemrograman Python, dengan compiler online Google Colab, dan memanggil library yang terdiri atas :

```
[122] import random
      import numpy as np
      np.random.seed(1234)
```

Justifikasi yang dapat penulis berikan mengenai preferensi tersebut adalah karena penggunaan Python tidak memerlukan compile, dan Google Colab memberikan kemudahan dalam menulis dokumentasi, markdown dan notes. Library tersebut berfungsi untuk me-random angka yang dibutuhkan, dan memastikan bahwa angka yang di-random tidak predictable.

2. Desain Kromosom dan Metode Pendekodean

Kromosom yang penulis desain terbuat dari array dengan index 10, yang berisi 10 bilangan biner sebagai gen.

Justifikasi yang dapat penulis berikan mengenai preferensi tersebut adalah karena penggunaan encoding method menggunakan binary representation memudahkan array slicing ketika decoding method dijalankan. Serta karena penggunaan method tersebut adalah yang pertama kali saya coba, dan langsung berhasil. Sedangkan metode lainnya harus melalui beberapa attempt

3. Ukuran Populasi

Populasi yang penulis desain terbatas pada 50 kromosom, dan justifikasi yang dapat penulis berikan mengenai preferensi tersebut adalah karena 50 adalah bilangan genap, sehingga mempermudah keadaan ketika harus melakukan Tournament Selection Method pada proses pemilihan orangtua.

4. Pemilihan orangtua

Metode yang penulis gunakan ketika melakukan pemilihan 2 kromosom yang akan dijadikan predecessor generasi berikutnya adalah Tournament Selection Methtod.

Justifikasi yang dapat penulis berikan mengenai preferensi tersebut adalah karena populasi yang penulis ciptakan terdiri atas 50 kromosom (bilangan genap) sehingga sangat mudah untul menemukan 2 pemenang akhir dari pool tersebut.

5. Pemilihan dan teknik operasi genetik (crossover dan mutasi)

Metode yang penulis gunakan dalam melakukan crossover dari 2 parent yang dipilih adalah single-point crossover, yang menghasilkan 2 offspring dan justifikasi yang dapat penulis berikan mengenai preferensi tersebut adalah karena pada kromosom yang penulis ciptakan, terdapat 10 indeks yang secara simetris dapat dilakukan satu kali array slicing, dan menghasilkan dua offspring berarti melakukan pengisian generasi dua kali lebih cepat daripada menghasilkan satu offspring .

Sedangkan metode yang penulis gunakan dalam melakukan mutasi dari 2 offspring yang telah dihasilkan adalah swapping pada array index kedua dan ketiga, dan justifikasi yang dapat penulis berikan mengenai preferensi tersebut adalah karena metode tersebut adalah teknik yang cukup berbahaya, sehingga lebih mampu menghasilkan populasi yang lebih diverse.

6. Probabilitas operasi genetik (Pc dan Pm)

Probabilitas yang penulis set, akan terjadinya crossover dari kedua parent yang telah di-select adalah 99%. Dengan demikian populasi tersebut dapat mencapai limit generation lebih cepat daripada men-set probabilitas terjadinya crossover yang mendekati 0%.

Sedangkan probabilitas yang penulis set, akan terjadinya mutasi dari kedua offspring yang telah dibuat adalah 99%. Dengan demikian populasi tersebut dapat memiliki diversity lebih tinggi daripada men-set probabilitas terjadinya mutasi yang mendekati 0%.

7. Metode Pergantian Generasi (Seleksi Survivor)

Metode yang penulis gunakan dalam melakukan Seleksi Survivor dari populasi yang ditargetkan untuk mencapai generation optimum adalah Generational Replacement.

Justifikasi yang dapat penulis berikan mengenai preferensi tersebut adalah karena tidak perlu melakukan re-sorting tiap menciptakan generasi baru.

8. Kriteria Penghentian Evolusi

Penulis memberikan kriteria spesifik berupa pencapaian generasi ke-50 untuk langsung meng-evaluate fitness score dari ke-50 generasi tersebut.

**PROSES YANG DIBANGUN**

1. Dekode kromosom

▾ IMPORTED LIBRARY

```
[ ] import random
    import numpy as np
    np.random.seed(1234)
```

▾ CONSTANTS AND GLOBAL VARIABLES

```
[ ] total_individual = 50
    chromosome_length = 10
    pc = 0.99
    pm = 0.99
    max_gen = 500
    population = []
    pool_filial = []
    mutants = []
```

▾ 1ST GENERATION

```
[ ] def generate_chromosome(n):
      return np.random.randint(0,2,n)
```

▾ CHROMOSOME DECODING

```
[ ] def x1(chromosome):
      return (-1+(2-(-1)/(2**-1+2**-2+2**-3+2**-4+2**-5))*(chromosome[0]*2**-1 + chromosome[1]*2**-2 + chromosome[2]*2**-3 + chromosome[3]*2**-4 + chromosome[4]*2**-5))


    def x2(chromosome):
      return (-1+(1-(-1)/(2**-1+2**-2+2**-3+2**-4+2**-5))*(chromosome[5]*2**-1 + chromosome[6]*2**-2 + chromosome[7]*2**-3 + chromosome[8]*2**-4 + chromosome[9]*2**-5))
```

2. Perhitungan fitness

▾ H FUNCTION

```
[ ] def h(x1,x2):
      return np.cos(x1)*np.sin(x2)-(x1/(x2**2+1))
```

▾ FITNESS FUNCTION

```
[ ] def fitness(chromosome):
      return (-h(x1(chromosome),x2(chromosome)))
```

3. Pemilihan orangtua

▾ PARENTAL SELECTION

```
[ ] def parental_selection(pool):
      tournamently_selected = []
      total = 0
      for i in range(total_individual):
        total = total + fitness(pool[i])
      for j in range(total_individual):
        r = random.random()
        individual = 0
        while (r>0):
          r = r - fitness(pool[i])
          individual = individual + 1
          tournamently_selected.append(pool[individual])
      return tournamently_selected
```

4. Crossover (pindah silang)

▾ CROSSOVER

```
[ ] def crossover(selected1,selected2):
      n = np.random.randint(1,len(selected1))
      offspring1, offspring2 = [],[]
      for i in range(n):
        offspring1.append(selected1[i])
        offspring2.append(selected2[i])
      while(n<len(selected1)):
        offspring1.append(selected2[n])
        offspring2.append(selected1[n])
        n+=1
      return offspring1,offspring2
```

5. Mutasi

▾ MUTATION

```
[ ] def mutation(filial):
      mutant = filial
      r1 = np.random.random()
      if (r1<=pm):
        r2 = np.random.randint(0,len(filial))
        r3 = np.random.randint(0,2)
        mutant[r2] = r3
      return mutant
```

6. Pergantian Generasi

▾ SURVIVOR SELECTION

```
[ ] def survivor_selection(population1):
      generational_replacement = []
      next_gen = sorted(population1,key=fitness,reverse=True)
      for i in range(total_individual):
        generational_replacement.append(next_gen[i])
      return generational_replacement
```

7. Program Utama

▾ MAIN

```
[ ] for i in range(total_individual):
      population.append(generate_chromosome(chromosome_length))

    for j in range(15):
      #parental_selection
      parent = parental_selection(population)

      #crossover
      k = 0
      while (k<total_individual):
        filial1, filial2 = crossover(parent[k],parent[k+1])
        pool_filial.append(filial1)
        pool_filial.append(filial2)
        k += 2

      #mutation
      for l in range(total_individual):
        mutants.append(mutation(pool_filial[l]))

      #survivor_selection
      generasi = population + mutants
      population = survivor_selection(generasi)
      elitism = max(population,key=fitness)

    print("best solution     : ",elitism)
    print("fitness score     : ",fitness(elitism))
    print("x1                : ",x1(elitism))
    print("x2                : ",x2(elitism))
    print("function minimum  : ",h(x1(elitism),x2(elitism)))
```

8. Hasil Akhir

```
best solution     :  [1 1 1 1 1 1 0 0 1 0]
fitness score     :  1.9497449360779626
x1                :  1.9375
x2                :  0.1431451612903225
function minimum  :  -1.9497449360779626
```