

**LAPORAN TUGAS PEMROGRAMAN GENETIC ALGORITHM  
PENGANTAR KECERDASAN BUATAN**



Disusun oleh :

1. M. Rifqi Arrahim N (1301190425)
2. Jane Raihan (1301194240)
3. Nur Qalbi (1301184194)

**Program Studi S1 Informatika  
Fakultas Informatika  
2021**

## ANALISIS MASALAH

### A. Desain Kromosom dan Metode Pendekodean

Diberikan sebuah fungsi  $h(x,y)=x^2 \cdot \sin y^2 +(x+y)$ , lalu temukan nilai maksimal dari fungsi tersebut menggunakan algoritma Genetic Algorithm dengan batasan  $-1 \leq x \leq 2$  dan  $-1 \leq y \leq 1$ . Untuk kasus seperti ini kita dapat merepresentasikan individu dengan menggunakan representasi biner, integer, real. Namun kita tidak bisa menggunakan representasi permutasi karena representasi permutasi digunakan untuk permasalahan mencari urutan, sedangkan masalah diatas mencari sebuah nilai. Sehingga pada tugas pemrograman ini kami menerapkan representasi integer untuk merepresentasikan individu. Berikut adalah algoritma fungsi kromosom dan nilai fungsi :

```
#batas
interval_x = [-1,2]
interval_y = [-1,1]
#kromosom
def kromosom(gen):
    tabkrom=[]
    for i in range(gen):
        tabkrom.append(random.randint(0,9))
    return tabkrom
```

```
#nilai fungsi
def fungsi (x,y):
    f = x**2 * math.sin(y**2)+(x+y)
    return f
```

### B. Ukuran Populasi

Ukuran populasi disesuaikan dengan jumlah yang diinputkan. Berikut adalah algoritma fungsi populasi :

```
#populasi
def populasi(pop,gen):
    tabpop=[]
    for i in range(pop):
        tabpop.append(kromosom(gen))
    return tabpop
```

### C. Fungsi Fitness

Fungsi fitness yang kita gunakan adalah fungsi maksimasi dengan rumus  $f = h$ , karena kita akan mencari nilai maksimal dari sebuah fungsi  $h(x,y)$ . Lalu ada fungsi hitungfitness untuk menghitung nilai fitness dari populasi yang kita cari. Berikut adalah algoritma fungsi nilai fitness dan hitung fitness :

```
#bagi array menjadi 2
def split(krom):
    return (krom[:len(krom)//2], krom[len(krom)//2:])

#decode kromosom
def decode(gamet, interval):
    pembagi = 0
    kali = 0
    #rumus integer
    for i in range(len(gamet)):
        n = gamet[i]
        kali += (n * (10**-(i+1)))
        pembagi += (9 * (10**-(i+1)))
    result = interval[0] + (((interval[1]-interval[0]) / pembagi) * kali)
    return result
```

```
#nilai fitness
def fit(f):
    return f

#hitung fitness
def hitungfitness(populasi):
    fitness_populasi=[]
    for i in range(len(populasi)):
        x,y = split(populasi[i])
        gamet_x = decode(x,interval_x)
        gamet_y = decode(y,interval_y)
        f = fungsi(gamet_x,gamet_y)
        fitness = fit(f)
        fitness_populasi.append(fitness)
    return fitness_populasi
```

#### D. Pemilihan Orangtua

Parent selection yang kami gunakan adalah Tournament Selection. Dimana cara kerja dari metode tournament selection ini adalah memilih sejumlah k kromosom secara acak dari sebuah populasi yang berjumlah N kromosom. Lalu dari k kromosom tersebut, pilih salah satu kromosom terbaik untuk dijadikan parent. Lakukan hal ini berulang-ulang, sampai dihasilkan N kromosom sebagai parent.

Alasan kami memilih metode tournament selection adalah metode ini bekerja lebih baik dibandingkan roulette wheel dan baker USS. Metode roulette wheel memiliki kelemahan yaitu kromosom yang dipilih untuk dijadikan parent belum tentu memiliki nilai fitness yang paling optimal, karena pemilihan parent dilakukan secara acak. Metode baker USS memiliki kelemahan yaitu tidak bisa memberikan solusi yang optimal ketika semua kromosom memiliki nilai fitness yang hampir sama atau ada sebuah kromosom yang memiliki nilai fitness yang sangat besar dibandingkan dengan kromosom yang lainnya. Berikut adalah algoritma fungsi tournament selection :

```
#tournament selection
def select (populasi):
    calon = []
    fitness_calon = []
    for i in range(4):
        n = random.randint(0,len(populasi)-1)
        calon.append(populasi[n])
    fitness_calon = hitungfitness(calon)
    parent_1 = max(fitness_calon)
    idx_parent_1 = fitness_calon.index(parent_1)
    fitness_calon.remove(parent_1)
    parent_2 = max(fitness_calon)
    idx_parent_2 = fitness_calon.index(parent_2)
    return (calon[idx_parent_1],calon[idx_parent_2])
```

#### E. Crossover dan Mutasi

Rekombinasi untuk representasi biner dan integer dapat menggunakan rekombinasi satu titik, rekombinasi banyak titik, rekombinasi seragam. Rekombinasi untuk representasi real dapat menggunakan discrete crossover, single arithmetic crossover,

simple arithmetic crossover, whole arithmetic crossover. Pada tugas ini kami menggunakan rekombinasi satu titik dengan cara memilih satu titik potong pada kromosom. Karena cenderung lebih mudah dalam implementasinya. Misalnya, terpilih titik potong dengan posisi T, lalu untuk anak pertama akan diwarisi gen parent pertama dari awal hingga titik T dan titik T hingga akhir akan diwarisi gen parent kedua. Sebaliknya anak kedua akan diwarisi gen parent kedua dari awal hingga posisi T dan posisi T hingga akhir akan diwarisi gen parent pertama. Selanjutnya proses mutasi, mutasi pada representasi integer terdapat beberapa cara seperti membalik nilai integer, memilih nilai secara acak, dan mutasi creep. Pada tugas ini kami memilih mutasi dengan memilih nilai secara acak dengan cara memilih satu gen secara acak dalam kromosom, lalu nilainya akan diganti dengan nilai yang baru secara acak dalam interval [0,9]. Proses terjadinya mutasi dipengaruhi dari probabilitas terjadinya mutasi.

```
#rekombinasi
def cross(krom1,krom2,prob):
    child1 = []
    child2 = []
    #n adalah titik silang
    n = random.randint(1,len(krom1)-1)
    if random.random() <= prob:
        child1[:n]=krom1[:n]
        child1[n:]=krom2[n:]
        child2[:n]=krom2[:n]
        child2[n:]=krom1[n:]
    else:
        child1 = krom1
        child2 = krom2
    return (child1,child2)
```

```
#mutasi
def mutation(krom,prob):
    if random.random()<=prob:
        #posisi mutasi
        n = random.randint(0,len(krom)-1)
        #nilai mutasi
        m = random.randint(0,9)
        krom[n]=m
    return krom
```

#### F. Probabilitas Operasi Genetik

Semakin tinggi angka probabilitas mutasi, maka semakin besar kemungkinan sebuah gen dalam kromosom terjadi mutasi. Sebaliknya semakin kecil angka probabilitas mutasi, maka semakin kecil kemungkinan sebuah gen dalam kromosom untuk bermutasi. Kelompok kita menggunakan 0.01 untuk angka probabilitas mutasi dan 0.8 untuk angka probabilitas mutasi.

#### G. Seleksi Survivor

Pada proses seleksi survivor kita menggunakan metode Generational Model dimana suatu populasi berukuran N kromosom akan diganti dengan N kromosom baru pada generasi berikutnya. Namun untuk menjaga kromosom terbaik sehingga bisa menghasilkan solusi yang optimal kita menggunakan elitisme yaitu memilih sebuah kromosom terbaik dari generasi sebelumnya dan memindahkannya ke generasi selanjutnya. Dengan cara tersebut minimal kita akan mempunyai satu solusi terbaik dari generasi sebelumnya.

```
#seleksi survivor
def elitism(populasi):
    new_pop=[]
    fitness_populasi = hitungfitness(populasi)
    best = max(fitness_populasi)
    idx_best = fitness_populasi.index(best)
    new_pop.append(populasi[idx_best])
    return new_pop
```

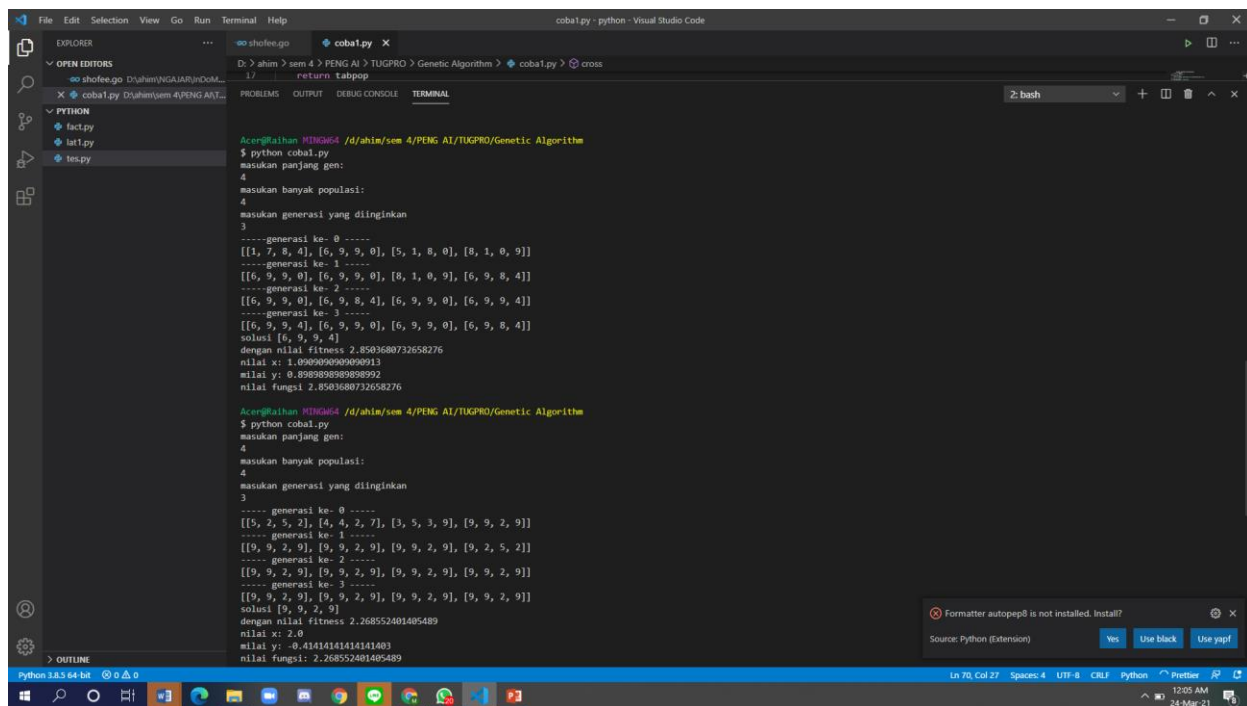
## H. Evolusi

1. Model populasi: *generational model*
2. Seleksi orangtua: *tournament selection*
3. Rekombinasi: satu titik potong
4. Mutasi: Memilih nilai secara acak
5. Seleksi *Survivor*: dengan elitisme

## FAKTOR YANG MEMPENGARUHI HASIL

Ada beberapa faktor yang dapat mempengaruhi hasil akhir yaitu metode pemilihan orangtua, metode rekombinasi, metode mutasi, metode seleksi survivor, panjang gen yang di generate, banyaknya kromosom dalam suatu populasi, angka probabilitas rekombinasi, angka probabilitas mutasi, dan berapa banyak generasi yang dijalankan.

## HASIL DAN KESIMPULAN



```
Acorn@rahman-W106664 /d/ahim/sem 4/PENG AI/TUGPRO/Genetic Algorithm
$ python cobal.py
masukan panjang gen:
4
masukan banyak populasi:
4
masukan generasi yang diinginkan
3
-----generasi ke- 0 -----
[[1, 7, 8, 4], [6, 9, 9, 0], [5, 1, 0, 0], [8, 1, 0, 9]]
-----generasi ke- 1 -----
[[6, 9, 9, 0], [6, 9, 9, 0], [8, 1, 0, 9], [6, 9, 8, 4]]
-----generasi ke- 2 -----
[[6, 9, 9, 0], [6, 9, 8, 4], [6, 9, 9, 0], [6, 9, 9, 4]]
-----generasi ke- 3 -----
[[6, 9, 9, 4], [6, 9, 9, 0], [6, 9, 9, 0], [6, 9, 8, 4]]
solusi [6, 9, 9, 4]
dengan nilai fitness 2.8503680732658276
nilai x: 1.0909090909090913
nilai y: 0.890909090909092
nilai fungsi 2.8503680732658276

Acorn@rahman-W106664 /d/ahim/sem 4/PENG AI/TUGPRO/Genetic Algorithm
$ python cobal.py
masukan panjang gen:
4
masukan banyak populasi:
4
masukan generasi yang diinginkan
3
-----generasi ke- 0 -----
[[5, 2, 5, 2], [4, 4, 2, 7], [3, 5, 3, 9], [9, 9, 2, 9]]
-----generasi ke- 1 -----
[[9, 9, 2, 9], [9, 9, 2, 9], [9, 9, 2, 9], [9, 2, 5, 2]]
-----generasi ke- 2 -----
[[9, 9, 2, 9], [9, 9, 2, 9], [9, 9, 2, 9], [9, 9, 2, 9]]
-----generasi ke- 3 -----
[[9, 9, 2, 9], [9, 9, 2, 9], [9, 9, 2, 9], [9, 9, 2, 9]]
solusi [9, 9, 2, 9]
dengan nilai fitness 2.268552401405489
nilai x: 2.0
nilai y: -0.4141414141414141
nilai fungsi: 2.268552401405489
```

```
D:\> ahim > sem 4 > PENG AI > TUGPRO > Genetic Algorithm > cobal.py > cross
1/ - return tabpop

Acer@kaihan MINGW64 /d/ahim/sem 4/PENG AI/TUGPRO/Genetic Algorithm
$ python cobal.py
masukan panjang gen:
4
masukan banyak populasi:
4
masukan generasi yang diinginkan
3
----- generasi ke- 0 -----
[[7, 6, 8, 5], [9, 0, 0, 7], [7, 7, 6, 2], [3, 0, 0, 2]]
----- generasi ke- 1 -----
[[9, 0, 0, 7], [9, 0, 0, 7], [3, 0, 0, 2], [9, 0, 0, 7]]
----- generasi ke- 2 -----
[[9, 0, 0, 7], [9, 0, 0, 7], [9, 0, 0, 7], [9, 0, 0, 2]]
----- generasi ke- 3 -----
[[9, 0, 0, 2], [9, 0, 0, 7], [9, 0, 0, 7], [9, 0, 0, 7]]
solusi [9, 0, 0, 2]
dengan nilai fitness 3.1428204504847246
nilai x: 1.7272727272727275
nilai y: -0.9595959595959596
nilai fungsi: 3.1428204504847246

Acer@kaihan MINGW64 /d/ahim/sem 4/PENG AI/TUGPRO/Genetic Algorithm
$
```

Dari hasil 3 kali percobaan diatas dengan panjang gen 4, jumlah kromosom 4, dan digenerate sebanyak 3 kali didapatkan hasil seperti berikut:

- Nilai x [1.09,2.0]
- Nilai y [-0.94,0.89]
- Nilai fungsi [2.26,3.14]
- Nilai fitness [2.26,3.14]

```
D:\> ahim > sem 4 > PENG AI > TUGPRO > Genetic Algorithm > cobal.py > ...
192 - tabpop = populasi(pop,ren)

Acer@kaihan MINGW64 /d/ahim/sem 4/PENG AI/TUGPRO/Genetic Algorithm
$ python cobal.py
masukan panjang gen:
100
masukan banyak populasi:
100
masukan generasi yang diinginkan
100
solusi [9, 9, 4, 7, 2, 6, 3, 9, 3, 7, 4, 8, 8, 5, 4, 6, 7, 8, 7, 2, 0, 2, 5, 8, 0, 3, 8, 8, 4, 4, 5, 6, 3, 0, 7, 2, 0, 0, 3, 5, 4, 6, 6, 3, 7, 3, 0, 0, 3, 9, 9, 3, 5, 9, 9, 8, 9, 6, 6, 8, 8, 2, 4, 7, 6, 3, 2, 6, 2, 2, 7, 0, 7, 3, 4, 3, 7, 7, 6, 3, 9, 0, 7, 7, 7, 8, 7, 6, 5, 4, 4, 8, 0, 0, 6, 3, 7, 0, 4, 7]
dengan nilai fitness 6.2290491933894785
nilai x: 1.9841791812465637
nilai y: 0.9871997933764953
nilai fungsi 6.2290491933894785

Acer@kaihan MINGW64 /d/ahim/sem 4/PENG AI/TUGPRO/Genetic Algorithm
$ python cobal.py
masukan panjang gen:
100
masukan banyak populasi:
100
masukan generasi yang diinginkan
100
solusi [9, 7, 9, 8, 6, 6, 6, 8, 7, 8, 7, 5, 6, 0, 3, 5, 5, 9, 9, 4, 7, 3, 9, 1, 3, 6, 3, 3, 3, 4, 6, 8, 4, 2, 8, 8, 0, 4, 1, 9, 4, 8, 0, 8, 1, 8, 0, 8, 6, 2, 9, 9, 3, 4, 3, 4, 7, 4, 9, 8, 5, 5, 3, 9, 6, 8, 7, 4, 6, 8, 6, 9, 2, 3, 4, 6, 0, 1, 4, 1, 9, 2, 9, 0, 7, 1, 2, 7, 2, 0, 9, 7, 0, 7, 3, 2, 1, 9, 5, 5]
dengan nilai fitness 6.038026529883883
nilai x: 1.9396086362681866
nilai y: 0.988604997169924
nilai fungsi 6.038026529883883

Acer@kaihan MINGW64 /d/ahim/sem 4/PENG AI/TUGPRO/Genetic Algorithm
$ python cobal.py
masukan panjang gen:
100
masukan banyak populasi:
100
masukan generasi yang diinginkan
100
solusi [9, 9, 9, 2, 1, 9, 5, 5, 0, 2, 1, 6, 7, 8, 5, 8, 9, 4, 0, 8, 7, 6, 4, 7, 7, 7, 9, 1, 7, 5, 2, 4, 4, 5, 5, 8, 6, 8, 7, 6, 1, 1, 4, 0, 1, 4, 6, 0, 9, 6, 2, 9, 3, 8, 9, 8, 9, 6, 5, 1, 0, 6, 6, 7, 5, 6, 0, 5, 3, 9, 9, 1, 8, 7, 9, 3, 2, 2, 5, 7, 2, 7, 4, 9, 1, 4, 8, 0, 7, 8, 7, 9, 5, 6, 5, 9]
dengan nilai fitness 5.940642179156338
nilai x: 1.997658656565375
nilai y: 0.9258779793021332
nilai fungsi 5.940642179156338
```

Dari hasil 3 kali percobaan diatas dengan panjang gen 100, jumlah kromosom 100, dan digenerate sebanyak 100 kali didapatkan hasil seperti berikut:

- Nilai x [1.93,1.99]
- Nilai y [0.92,0.98]
- Nilai fungsi [5.94,6.22]
- Nilai fitness [5.94,6.22]

Kesimpulan yang dapat diperoleh dari tugas pemrograman genetic algorithm adalah dengan menggunakan metode pemilihan orang tua, metode mutasi, metode seleksi survivor yang tepat dan panjang gen dan jumlah kromosom yang banyak dan angka probabilitas mutase yang semakin kecil, maka semakin besar kemungkinan untuk kita mendapatkan hasil yang optimal.

Link Google Colab : <https://colab.research.google.com/drive/102nfqUdqXYrvXwmWCq-Wk5SZHKW3VxFA?usp=sharing&authuser=1#scrollTo=KzaXRqNIYJzM>

Link Video :  
[https://drive.google.com/drive/folders/16N3\\_BOAnMvQBuA5SyUyIMFrEQDI8XmPb?usp=sharing](https://drive.google.com/drive/folders/16N3_BOAnMvQBuA5SyUyIMFrEQDI8XmPb?usp=sharing)