Laporan Tugas Pemrograman
Algoritma Genetika

Dosen Pembimbing:
IZZATUL UMMAH, S.T., M.T.
CII2M3-IF-44-11

Disusun oleh:
1301200240 Ramadhan Aditya Ibrahim
1301204125 Ryan Chandra Hadi

S1 Informatika
Universitas Telkom
Bandung, Jawa Barat 2022

Kata Pengantar

Dengan mengucapkan puji dan rasa syukur kepada Allah SWT yang telah memberikan rahmat sehingga kita dapat menyelesaikan tugas dari mata kuliah Pengantar Kecerdasan Buatan dengan tema “Algoritma Genetika” dengan benar dan tepat waktu.

Untuk memenuhi nilai tugas pada mata kuliah Pengantar Kecerdasan Buatan, maka dibuatkan tugas yang dapat kita selesaikan. Tidak hanya itu, tujuan dari pembuatan laporan dan pengerjaan tugas ini adalah untuk menambah wawasan tentang pembahasan Algoritma Genetika bagi kita semua.

Kami mengucapkan terima kasih kepada semua pihak dari mulai ibu Izzatul Ummah selaku dosen pembimbing yang telah memberikan kita tugas besar untuk membuat Algoritma Genetika.

Kami sangat menyadari laporan yang kami susun masih jauh dari kata sempurna. Tetapi kita akan terus berusaha untuk selalu menjadi lebih baik untuk kedepannya.

Bandung, 2 April 2022

Kelompok 6

BAB I
PENDAHULUAN

1. PERSOALAN

1.1 DEFINISI TUGAS

Lakukan analisis dan desain algoritma Genetika Algorithm (GA) serta mengimplementasikannya ke dalam suatu program komputer untuk mencari nilai *x* dan *y* sehingga diperoleh nilai minimum dari fungsi:

$$h(x,y) = \frac{(\cos x + \sin y)^2}{x^2 + y^2}$$

Dengan domain (batas nilai) untuk *x* dan *y*:

$$-5 \leq x \leq 5 \text{ dan } -5 \leq y \leq 5$$

Hal yang harus Anda analisis dan desain:

- Desain kromosom dan metode dekode-nya
- Ukuran populasi
- Metode pemilihan orangtua
- Metode operasi genetik (pindah silang dan mutasi)
- Probabilitas operasi genetik (*Pc* dan *Pm*)
- Metode pergantian generasi (seleksi survivor)
- Kriteria penghentian evolusi (loop)

Catatan: Poin-poin di atas harus ada di dalam Laporan Tugas!

Proses yang harus Anda **bangun/implementasikan** ke dalam baris-baris program:

- Dekode kromosom
- Perhitungan fitness
- Pemilihan orangtua
- Crossover (pindah silang)
- Mutasi
- Pergantian Generasi

Catatan: Proses-proses di atas harus dibangun tanpa menggunakan Library!

1.2 OUTPUT PROGRAM

Dengan masalah yang didefinisikan di atas, **output** dari program Anda:

- **kromosom terbaik**, dan
- **nilai *x* dan *y*** hasil dekode kromosom terbaik tersebut.

BAB II
PEMBAHASAN

2.1 Desain kromosom dan metode dekode-nya.

Pada pengerjaan soal ini, kelompok kami menggunakan metode random value yang tersedia pada library numpy yang sudah kami atur agar menghasilkan kromosom sebesar dua value yang nantinya akan di dekode dari satu value untuk x dan satu value untuk y. Dengan menggunakan batas yang sudah kami atur sesuai ketentuan dari soal yang diberikan (-5 <= x, y <= 5).

2.2 Ukuran Populasi

Pada pengerjaan tugas ini kami menggunakan ukuran populasi sebanyak 100 dengan tiap individu kromosom sebanyak 2 gen.

2.3 Metode pemilihan orangtua

Pada metode pemilihan orang tua, kelompok kami menggunakan metode Random Selection dengan memanfaatkan sebuah fungsi yang diambil dari library numpy yaitu, “np.random.permutation()” yang berfungsi untuk men-*generate* dua angka random pada setiap indeks yang kita miliki.

2.4 Crossover

Pada metode crossover ini kami menggunakan fungsi aljabar yang akan mencari rata-rata dari nilai x dan y yang nilai kromosomnya telah kita dapat dari pemilihan orang tua.

2.5 Mutasi

Pada metode ini kami menentukan probabilitas mutasi diantara (-0.1 sampai 0.1) dengan cara metode dengan memanfaatkan sebuah fungsi yang diambil dari library numpy yaitu, “random.uniform()” yang berfungsi untuk men-*generate* angka random dengan batas yang kita tentukan.

2.6 Probabilitas operasi genetik (*Pc* dan *Pm*)

Probabilitas crossover kami atur pada nilai 1 dengan harapan agar bisa mengurangi kegagalan dan probabilitas mutasi di antara -0.1 sampai 0.1. Dengan nilai

tersebut kita berharap mendapat generasi yang tidak sama dengan parent yang nantinya akan menghasilkan nilai fitness yang berbeda dari sebelumnya.

2.7 Metode pergantian generasi (seleksi survivor)

Pada metode pergantian generasi ini kami menggunakan sorting secara descending untuk menentukan kromosom mana yang akan kami pilih sebagai survivor. Setelah kita menggabungkan semua kromosom pada populasi awal, setelah di crossover, dan setelah di mutasi. Setelah itu kita lakukan sorting untuk melihat kromosom mana yang akan kami pilih dengan menentukan nilai yang kami tentukan yaitu sebanyak 100 kromosom.

2.8 Kriteria penghentian evolusi (loop)

Dalam kriteria penghentian evolusi ini kita akan melakukan evolusi sebanyak 20 generasi.

BAB III
IMPLEMENTASI PROGRAM

3.1 Library

```
import numpy as np
import pandas as pd
import math as ma
```

3.1.1 NumPy pada program kami berfungsi untuk membantu kami dalam men-*generate* angka yang bernilai float secara random, men-*generate* angka permutasi secara random, dan kita gunakan NumPy untuk menentukan nilai probabilitas mutasi.

3.1.2 Pandas pada program ini berfungsi merubah data array menjadi data berbentuk tabel.

3.1.3 Math pada program ini berfungsi untuk menghitung fungsi aljabar yang terdapat pada program yang kita buat semisal seperti pada saat kita menggunakan rumus untuk mencari h

3.2 Inisiasi

```
# minimum dan maksimum value yang ada di dalam fungsi
min = -5
max = 5

# menyatakan banyaknya variabel, panjang dari kromosom itu sendiri (x,y)
m = 2

# menyatakan banyaknya populasi
n = 10

# pc = 1
# pm = interval -0.1 s.d 0.1

# maksimum generasi
N = 20
```

Pada bagian ini kita membuat sebuah variabel batasan yang sudah ditentukan dari soal dan ada beberapa yang kita tentukan sendiri.

min & max = batas nilai kromosom yang ditentukan oleh soal
m = menyatakan nilai yang kita butuhkan x dan y
n = menyatakan banyaknya populasi yang ingin kita buat
N = menyatakan jumlah looping yang dilakukan untuk generasi selanjutnya

3.3 create population

```
def createPopulations(): # fungsi ini dibuat untuk meng-generate populasi
    populasi = np.random.rand(n, m)*(max - min)+ min
    pop = pd.DataFrame(populasi)
    pop.columns = ['x', 'y']

    return pop
```

Pada fungsi create population ini berfungsi untuk men-*generate* populasi, ukuran populasi sebanyak 100 dengan tiap individu kromosom sebanyak 2 gen.

3.4 fungsi fitness

$$f = \frac{1}{h + a}$$

```
[4]  def fitness(pop):
        h = (np.cos(pop['x']) + np.sin(pop['y']))**2 / ((pop['x'])**2 + (pop['y'])**2) # rumus untuk mencari nilai h(x, y)
        fitness = 1 / h + 1
        pop['Fitness'] = fitness
        pop['Nilai Fungsi'] = h

        return pop
```

Pada perintah soal, kita diharuskan untuk mencari nilai minimum dari sebuah fungsi yang telah tertera pada bab 2, maka dari itu kita menggunakan fungsi yang tertera diatas untuk mencapai nilai minimum dari sebuah fungsi (x, y).

3.5 fungsi random selection

```
def randomSelection(): # fungsi untuk meng-generate indeks random
    position = np.random.permutation(n)

    return position[0], position[1]
```

Pada fungsi random selection ini berfungsi untuk men-*generate* indeks random pada sebuah data yang nantinya kita bisa manfaatkan.

3.6 fungsi crossover

```
def crossover(pop): # fungsi ini untuk melakukan pindah silang yang sudah
                    # dilakukan seleksi orang tua dengan menggunakan metode fungsi randomSelection
    popCrossover = pop.copy()
    for i in range(n):
        a, b = randomSelection()
        x = (pop.loc[a] + pop.loc[b])/2
        popc.loc[i] = x

    return popCrossover
```

Pada fungsi crossover kita melakukan pemilihan orang tua dengan memanfaatkan fungsi randomSelection(). Setelah kita mendapatkan angka indeks pada setiap parent, kita menggunakan metode rata-rata agar didapatkan nilai kromosom yang telah tercampur antara indeks[a] dan indeks[b].

3.7 fungsi mutasi

```
def mutasi(popCrossover): # fungsi ini kita pakai untuk membuat mutasi pada populasi
                          # yang telah kita lakukan crossover
    popMutasi = popCrossover.copy()

    for i in range(n):
        for j in popMutasi.columns:
            popMutasi.loc[i][j] += np.random.uniform(low=-0.1, high=0.1)

    return popMutasi
```

Pada fungsi mutasi ini kita melakukan penjumlahan setiap kromoson pada setiap populasi dengan probabilitas penjumlahan diantara -0.1 sampai 0.1 dengan memanfaatkan random.uniform di library numPy.

3.8 fungsi combinepop

```
def combinePop(pop, popCrossover, popMutasi): # Menggabungkan nilai populasi awal,
                                              # setelah crossover dan setelah mutasi
    popAll = pop.copy()
    popAll = popAll.append(popCrossover)
    popAll = popAll.append(popMutasi)

    popAll.index = range(len(popAll))

    return popAll
```

Pada fungsi combinepop ini kita menggabungkan nilai populasi awal, nilai populasi setelah crossover dan nilai populasi setelah dilakukan mutasi.

3.9 fungsi sort

```
def sort(popAll): # mengurutkan nilai values berdasarkan fitness secara descending
    popAll = popAll.sort_values(by=['Fitness'], ascending = False)
    popAll.index = range(len(popAll))

    return popAll
```

Pada fungsi sort kita mengurutkan data yang telah kita dapatkan setelah menggabungkan semua populasi menjadi data yang terurut secara descending berdasarkan values dari fitness.

3.10 fungsi elimiasi

```
def eliminasi(popAll):  # setelah kita lakukan sort, kita lakukan eliminasi sebanyak n = 100
    pop = popAll.head(n)

    return pop
```

Setelah kita melakukan sorting secara descending kita mengambil nilai berdasarkan fitness. Kita akan mengambil data tersebut hanya sebanyak 100.

3.11 output

```
pop = createPopulations()
pop
```

	x	y
0	-4.771656	2.551695
1	0.434645	-2.897186
2	-0.534261	-1.631470
3	0.562200	-0.337352
4	-1.810324	-3.689988
...	...	...
95	-1.749947	2.150254
96	0.137528	-0.890603
97	3.008970	-3.437181
98	-3.494101	3.396237
99	-4.383546	-0.942471

100 rows × 2 columns

Berfungsi untuk menampilkan setiap populasi dan kromosom yang ada.

```
pop = fitness(pop)
pop
```

	x	y	Fitness	Nilai Fungsi
0	-4.771656	2.551695	77.285829	0.012939
1	0.434645	-2.897186	19.405490	0.051532
2	-0.534261	-1.631470	155.848176	0.006417
3	0.562200	-0.337352	1.620197	0.617209
4	-1.810324	-3.689988	209.337981	0.004777
...	...	...	...	...
95	-1.749947	2.150254	17.721325	0.056429
96	0.137528	-0.890603	17.881638	0.055923
97	3.008970	-3.437181	42.598175	0.023475
98	-3.494101	3.396237	16.755001	0.059684
99	-4.383546	-0.942471	15.689653	0.063736

100 rows × 4 columns

Berfungsi untuk menampilkan nilai fitness pada setiap populasi.

```
popCrossover = crossover(pop)
print(popCrossover)
```

```
           x         y      Fitness  Nilai Fungsi
0  -0.284010 -0.059513    28.274334      0.037192
1  -0.752975  0.322256    56.549939      0.091886
2  -3.054758 -0.135929    85.720620      0.033074
3  -0.035844  4.581305    23.269829      0.043100
4   2.597668  1.964382    10.527406      0.184249
..       ...       ...          ...           ...
95 -1.660433  3.793133  7129.271581      0.005710
96  0.538110 -3.636869    12.914435      0.103607
97 -2.684613 -4.543498    20.134149      0.101706
98  2.368231  1.278077    51.019441      0.025844
99  0.148504 -2.091620     6.039091      0.470514

[100 rows x 4 columns]
```

Berfungsi untuk menampilkan nilai setelah crossover.

```
popCrossover = fitness(popCrossover)
popCrossover
```

	x	y	Fitness	Nilai Fungsi
0	-0.284010	-0.059513	0.103848	9.629457
1	-0.752975	0.322256	0.612689	1.632150
2	-3.054758	-0.135929	7.299897	0.136988
3	-0.035844	4.581305	333198.546731	0.000003
4	2.597668	1.964382	2303.596121	0.000434
...	...	...	...	...
95	-1.660433	3.793133	35.400151	0.028248
96	0.538110	-3.636869	7.595894	0.131650
97	-2.684613	-4.543498	3565.347036	0.000280
98	2.368231	1.278077	123.765966	0.008080
99	0.148504	-2.091620	297.439157	0.003362

100 rows × 4 columns

Berfungsi untuk menampilkan *update* pada nilai fitness yang telah dilakukan crossover, karena pada gambar
sebelumnnya nilai fitness belum di *update.*

```
popMutasi = mutasi(popCrossover)

popMutasi
```

	x	y	Fitness	Nilai Fungsi
0	-0.205305	0.012330	0.115073	9.719344
1	-0.673165	0.415961	0.581240	1.552571
2	-3.041725	-0.049601	7.308667	0.097337
3	0.024349	4.551367	333198.533688	-0.050702
4	2.551339	1.998557	2303.651321	0.062630
...	...	...	...	...
95	-1.675862	3.831079	35.419312	0.014359
96	0.617943	-3.561594	7.668015	0.089076
97	-2.661885	-4.633478	3565.411029	-0.021520
98	2.446384	1.367732	123.717681	0.052446
99	0.213288	-2.062202	297.431863	0.012555

100 rows × 4 columns

Berfungsi untuk menampilkan hasil mutasi setelah dilakukan crossover.

```
popMutasi = fitness(popMutasi)

popMutasi
```

	x	y	Fitness	Nilai Fungsi
0	-0.205305	0.012330	0.043045	23.231315
1	-0.673165	0.415961	0.445229	2.246036
2	-3.041725	-0.049601	8.481194	0.117908
3	0.024349	4.551367	129667.523528	0.000008
4	2.551339	1.998557	1678.849474	0.000596
...	...	...	...	...
95	-1.675862	3.831079	31.844213	0.031403
96	0.617943	-3.561594	8.738458	0.114437
97	-2.661885	-4.633478	2370.304074	0.000422
98	2.446384	1.367732	175.556546	0.005696
99	0.213288	-2.062202	469.604326	0.002129

100 rows × 4 columns

Berfungsi untuk menampilkan *update* pada nilai fitness yang telah dilakukan mutasi di atas, karena pada gambar
sebelumnnya nilai fitness belum di *update.*

```
popAll = combinePop(pop, popCrossover, popMutasi)
popAll
```

	x	y	Fitness	Nilai Fungsi
0	-4.771656	2.551695	77.285829	0.012939
1	0.434645	-2.897186	19.405490	0.051532
2	-0.534261	-1.631470	155.848176	0.006417
3	0.562200	-0.337352	1.620197	0.617209
4	-1.810324	-3.689988	209.337981	0.004777
...	...	...	...	...
295	-1.675862	3.831079	31.844213	0.031403
296	0.617943	-3.561594	8.738458	0.114437
297	-2.661885	-4.633478	2370.304074	0.000422
298	2.446384	1.367732	175.556546	0.005696
299	0.213288	-2.062202	469.604326	0.002129

300 rows × 4 columns

Berfungsi untuk menampilkan semua nilai yang telah didapatkan tadi, mulai dari populasi awal, setelah di crossover, dan setelah dilakukannya mutasi.

```
popAll = sort(popAll)
popAll
```

	x	y	Fitness	Nilai Fungsi
0	-1.135202	-2.706725	1.983085e+07	5.042649e-08
1	-3.057472	-4.595952	2.911335e+06	3.434851e-07
2	-0.322891	-1.889633	2.247616e+06	4.449159e-07
3	-0.035844	4.581305	3.331985e+05	3.001214e-06
4	0.024349	4.551367	1.296675e+05	7.712031e-06
...	...	...	...	...
295	0.006202	0.545826	1.291185e-01	7.744823e+00
296	-0.284010	-0.059513	1.038480e-01	9.629457e+00
297	-0.284010	-0.059513	1.038480e-01	9.629457e+00
298	-0.260087	-0.093059	1.000192e-01	9.998085e+00
299	-0.205305	0.012330	4.304535e-02	2.323131e+01

300 rows × 4 columns

Berfungsi untuk menampilkan semua data yang telah diurutkan dengan cara descending.

```
pop = createPopulations()
pop = fitness(pop)
print("Generasi Pertama")
print(pop)

for i in range(N):
    popCrossover = crossover(pop)
    popCrossover = fitness(popCrossover)

    popMutasi = mutasi(popCrossover)
    popMutasi = fitness(popMutasi)

    popAll = combinePop(pop, popCrossover, popMutasi)

    popAll = sort(popAll)
    pop = eliminasi(popAll)
    print()
    print(i)
    print(pop)

print()
print("Generasi Terakhir")
print(pop)
```

Berfungsi untuk menampilkan semua data dari mulai populasi awal, sampai ke data yang telah di eliminasi sebanyak n = 100 dan ditampilkan juga data terakhir setelah dilakukan kriteria dilakukannya evolusi sebanyak N = 20 kali.

```
0
           x         y       Fitness  Nilai Fungsi
0   2.647297  1.076925  2.021751e+08  4.946207e-09
1   3.307942  1.412953  6.865403e+06  1.456579e-07
2   1.613020  0.040641  1.040904e+06  9.607035e-07
3   0.940455  3.777162  8.525055e+05  1.173013e-06
4   3.330104  1.422810  2.847998e+05  3.511238e-06
..       ...       ...           ...           ...
95 -0.040052 -1.067723  7.534647e+01  1.327202e-02
96 -0.634013 -1.549781  7.440386e+01  1.344016e-02
97  3.056994 -3.603868  7.371104e+01  1.356649e-02
98  0.287464  3.695525  7.330538e+01  1.364156e-02
99 -3.885485  2.979460  7.265931e+01  1.376286e-02

[100 rows x 4 columns]
```

```
Generasi Terakhir
           x         y       Fitness  Nilai Fungsi
0  -2.668123  2.044266  3.528051e+18  2.834426e-19
1  -2.592618  2.119771  8.699462e+16  1.149496e-17
2  -2.616682  2.095707  6.098089e+16  1.639858e-17
3  -2.588617  2.123772  1.434952e+16  6.968873e-17
4  -2.609690  2.102699  4.020072e+15  2.487518e-16
..       ...       ...           ...           ...
95 -2.579447  2.132943  1.144999e+13  8.733631e-14
96 -2.607551  2.104840  1.070209e+13  9.343967e-14
97 -2.553748  2.158642  1.061767e+13  9.418260e-14
98 -2.598342  2.114045  1.057563e+13  9.455701e-14
99 -2.572589  2.139802  1.031800e+13  9.691797e-14

[100 rows x 4 columns]
```

BAB IV
KESIMPULAN

Setelah kita melakukan implementasi pada algoritma genetik nilai minimum, kita mendapati bahwasannya untuk minimasi sebuah fungsi akan menghasilkan nilai yang mendekati fungsi fitness yang sempurna. Jika kita perhatikan pada gambar bab 3 kita bisa melihat nilai fitness pada generasi pertama dan generasi terakhir memiliki nilai fitness yang sangat signifikan perbedaanya. Setelah kita tahu bahwa individu yang terbaik berada pada generasi terakhir kita bisa mengambil nilai minimum pada fungsi tersebut.

Nilai minimum = (-2.668123, 2.044266) = 3.528051e+18

Pembagian dalam pengerjaan tugas besar AI:

Ramadhan Aditya Ibrahim:

- Pembuatan Code pada bagian inisiasi sampai pada fungsi randomSelection
- Pembuatan Laporan dan crosscheck nilai fitness.

Ryan Chandra Hadi:

- Pembuatan Code pada bagian crossover sampai menentukan Kriteria penghentian evolusi.
- Pembuatan Laporan dan crosscheck nilai fungsi.

Berikut link video presentasi kelompok kami:

Via G-Drive:
https://drive.google.com/file/d/1G-x6uNKZkdYO_gnzpC1ZZfrMPTfJGRKC/view?usp=sharing