LAPORAN TUGAS PEMROGRAMAN 1
PENGANTAR KECERDASAN BUATAN

Disusun oleh:

Fendi Irfan Amorokhman	(1301191447)
Fadhil Wisnu Ramadhan	(1301190378)
Andre Agasi Simanungkalit	(1301190338)

PROGRAM STUDI INFORMATIKA

FAKULTAS INFORMATIKA

2020

Daftar Isi

BAB 1
PENDAHULUAN

Algoritma adalah metode langkah demi langkah pemecahan dari suatu masalah. Algoritma Genetika merupakan algoritma pencarian untuk menyelesaikan masalah yang didasarkan pada evolusi genetika yang terjadi pada makhluk hidup. Proses komputasi yang terjadi pada algoritma genetika dianalogikan dengan proses seleksi makhluk hidup dalam sebuah populasi. Dengan Algoritma Genetika, hal-hal yang perlu dihindarkan dalam pembuatan jadwal bisa dihilangkan, dan semua bentuk solusi yang menguntungkan pihak-pihak yang terkait akan lebih mudah untuk didapatkan.

Algoritma Genetika pertama kali ditemukan oleh John Holland, yang dapat dilihat dari bukunya yang berjudul Adaptation in Natural and Artificial Systems pada tahun 1960-an dan kemudian dikembangkan bersama murid dan rekan kerjanya di Universitas Michigan pada tahun 1960-an sampai 1970-an. Kemudian dipopulerkan oleh David Goldberg pada tahun 1980-an yang kemudian dalam bukunya, David Goldberg mendefinisikan Algoritma Genetika sebagai algoritma pencarian yang didasarkan pada mekanisme seleksi alamiah dan genetika alamiah. Algoritma Genetika merupakan metode pencarian yang disesuaikan dengan proses genetika dari organisme-organisme biologi yang berdasarkan pada teori evolusi Charles Darwin

1.1 KRITERIA SOAL

Pada tugas pemrograman 1 ini tiap kelompok diwajibkan melakukan analisis, desain, dan implementasi algoritma ***Genetic algorithm*** **(GA)** ke dalam suatu program komputer untuk menemukan **nilai maximum** dari fungsi:

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

dengan batasan $-1 \leq x \leq 2$ dan $-1 \leq y \leq 1$. Dengan memperhatikan beberapa hal yang harus di observasi seperti :

- Desain Kromosom dan Metode Pendekodean
- Ukuran Populasi
- Pemilihan orangtua
- Pemilihan dan teknik operasi genetik (*crossover* dan mutasi)
- Probabilitas operasi genetik (Pc dan Pm)
- Metode Pergantian Generasi (Seleksi Survivor)
- Kriteria Penghentian Evolusi

dan beberapa proses yang harus dibangun seperti :

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

- Mutasi
- Pergantian Generasi

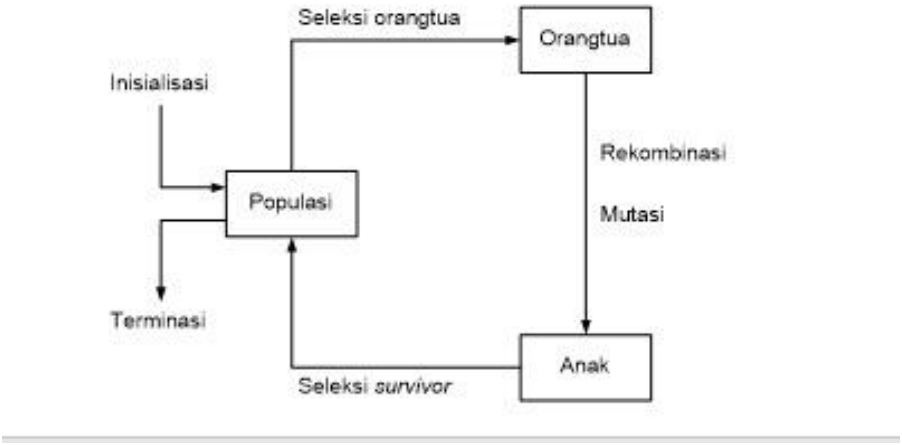
Output dari sistem adalah **kromosom terbaik** dan nilai $\boldsymbol{x}$ dan y hasil **dekode kromosom terbaik** tersebut.

BAB 2

METODE

Metode pada algoritma genetika memiliki peran penting dalam menentukan bagaimana sebuah algoritma akan beroperasi, dalam bentuk mencapai tujuan algoritma tersebut. Metode pemilihan bisa dikatakan sangat penting untuk menambah efisiensi dan ketepatan dari sebuah genetika algoritma. Dengan itu pemilihan dari algoritma tersebut perlu di konsiderasi.

Kita ketahui bahwa dalam kasus ini kita melakukan pemrograman EA, yang termasuk dalam teknik pencarian probabilistik, Skema EA memiliki beberapa proses dalam prosesnya.



(Gambar 2.1 Skema Proses Evolusi Generasi EA)

Dapat kita lihat pada skema tersebut, alur dari evolusi generasi EA.

1. Populasi
   Populasi merupakan kumpulan dari banyak individu, dan pemilihan metode yang digunakan dalam memilih individu pada algoritma ini adalah dengan menentukan jumlah gen pada setiap individu. Lalu, populasi pada algoritma dapat ditentukan dengan jumlah individu yang ada.
2. Representasi Individu dan Fungsi Fitness
   Pada algoritma ini terdapat halnya gen yang merupakan sebuah ciri ciri dari sebuah individu, dengan itu diperlukan juga cara untuk merepresentasikannya. Bentuk dari representasi biner dapat bervariasi dari biner, integer, dan real.
   Sedangkan Fungsi Fitness dari suatu kromosom adalah nilai kecocokan kromosom terhadap permasalahan. Semakin tinggi nilai fitness, seharusnya solusi tersebut semakin optimal.
3. Seleksi Orang Tua
   Proses ini bertujuan untuk memilih dua kromosom sebagai parent yang selanjutnya akan dilakukan proses crossover dan mutasi pada parent terpilih.
4. Rekombinasi
   proses rekombinasi / crossover adalah menyilangkan dua kromosom dengan harapan mendapatkan kromosom baru yang lebih baik daripada induk / orang tua nya
5. Mutasi

Mutasi pada algoritma genetika bertujuan untuk mengubah gen gen tertentu dalam sebuah kromosom yang dimana persentase kemungkinan terjadinya mutasi genetis sangat kecil. Mutasi dilakukan dengan tujuan untuk memperoleh kromosom kromosom baru agar mendapatkan nilai fitness yang lebih baik.

6. Seleksi Survivor

   Proses ini bertujuan untuk memilih N kromosom dari gabungan antara populasi sebelumnya dan offspring yang dihasilkan. Lalu, N kromosom yang terpilih akan digunakan sebagai populasi untuk perulangan atau iterasi berikutnya

BAB 3
DESAIN

Kita ketahui dari metode yang sudah di strategikan muncul juga desain desain yang akan dibuat untuk implementasi Algoritma GA. Desain dari algoritma dilakukan untuk mendapatkan hasil maksimum dari fungsi :

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

dengan batasan $-1 \leq x \leq 2$ dan $-1 \leq y \leq 1$.

Pemrograman dari desain dilakukan dalam bahasa python, dan penulisan algoritma juga dilakukan menggunakan platform Google Collab. Dari berbagai macam pendekatan yang di observasi didapat berbagai macam metode yang di strategikan untuk mencapai nilai maksimum tersebut.

3.1 Desain Kromosom dan Metode Pendekodean

Desain kromosom yang kami gunakan dalam algoritma adalah menggunakan bilangan integer, dengan itu proses pendekodean dari kromosom juga dilakukan menggunakan representasi biner. Penggunaan bilangan integer dalam proses dilakukan karena tidak seperti representasi biner yang hanya menggunakan 1/0, representasi integer dapat melakukan lebih dari dua instansi[3].

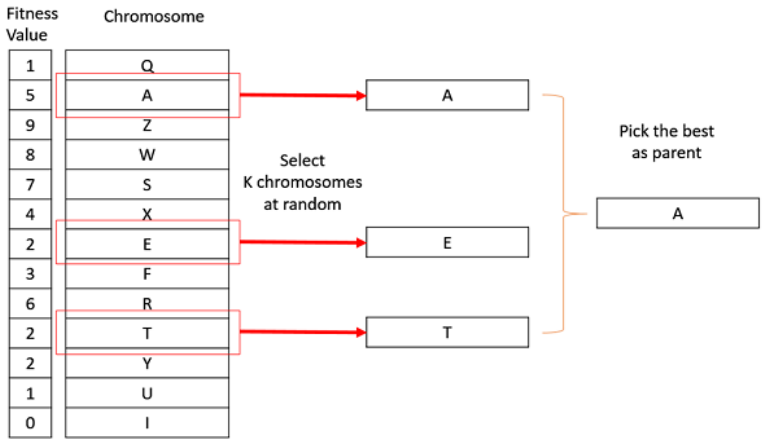
```
def individu(lenght,min,max):
  return [randint(min,max) for i in range(lenght)]

def population(count,lenght,min,max):
  return [individu(lenght,min,max) for i in range(count)]
```

3.2 Ukuran Populasi

Kami menggunakan populasi dengan jumlah individu sebanyak 100 dan tiap individu kromosom memiliki 24 gen. DeJong [4] suggested optimal range values for population size to be in the range of [50–100]

```
# Banyak gen pada kromosom (genap)
jumKromosom = 24

# Jumlah individu/populasi
jumIndividu = 100
```

3.3 Pemilihan orangtua



(Gambar 3.3.0 Pemilihan Kromosom Orang Tua Tournament Selection)

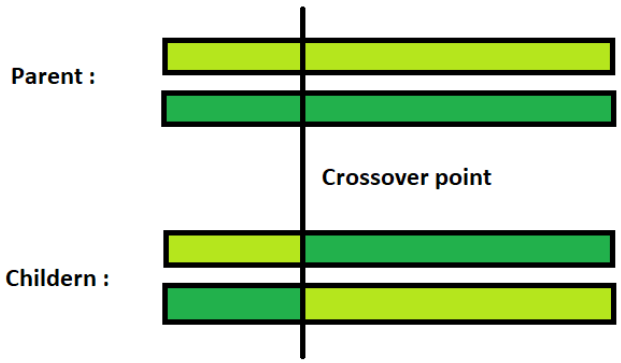
Metode pemilihan orangtua yang kami gunakan adalah metode tournament selection. Kenapa kami memilih metode tournament selection karena, hasil yang didapatkan lebih efektif dari pada metode roulette wheel, hasil tersebut didapatkan dalam observasi jurnal. "Tournament selection is more efficient than Proportionate roulette wheel selection in terms of Convergence rate"[1].

```
# Pemilihan Orang Tua
def tournamentSelection(df):
 parent = []
 jumSelection = math.floor((2*len(df))/ 3) # 2/3 dari jumlah populasi

 for k in range(0,jumSelection):
  # angka untuk memilih random index kromosom yang akan di bandingkan
  angka_random = [randint(0,len(df)-1) for i in range(0,3)]

  max = angka_random[0] # inisiasi

  # mencari index terbaik berdasarkan fitness
  for i in angka_random:
   if df['fitness'][max] < df['fitness'][i]:
    max = i
  parent.append(df['Kromosom'][max])

 return parent
```

3.4 Pemilihan dan teknik operasi genetik (*crossover* dan mutasi)

3.4.1 Crossover (rekombinasi satu titik)

Kami menggunakan metode single arithmetic crossover dimana Crossover ini melakukan penukaran gen-gen dari satu kromosom dengan kromosom lain untuk menghasilkan kromosom baru melalui satu titik potong. Titik potong didapatkan dengan cara membangkitkan random number dengan batasan 1 sampai n (panjang kromosom. Random number yang dihasilkan akan dijadikan sebagai titik potong kromosom.



(Gambar 3.4 Single Point Crossover)

```
def singlePointCrossover(list_elitsme, TournamentSelection,Pc,jumChromosome,jumPopulasi):
 # point = math.floor(jumChromosome/ 3) #index point pada kromosom
 point = randint(12,jumChromosome-8) #index point pada kromosom
 # inisiasi awal elitsm akan termasuk ke dalam child
 child = []
 # proses perkawinan orang tua/kromosom yang akan menghasilkan child jumPopulasi - len(list_elitsme)
 while len(child) <= (jumPopulasi - len(list_elitsme)):
  # inisiasi variabel
  angkaRandomElitsm = randint(0,len(list_elitsme)-1)
  angkaRandomElitsm2 = randint(0,len(list_elitsme)-1)
  angkaRandomTournamentSelection = randint(0,len(TournamentSelection)-1)
  angkaRandomTournamentSelection2 = randint(0,len(TournamentSelection)-1)
  angkaRandom = random.uniform(0.0,1.0)

  # untuk mengecek bahwa Pc terpenuhi dari angkaRandom dan kromosoomnya tidak sama
  if (angkaRandom <= Pc) :
   angka_random_pemilihan_cara_crossOver = randint(0,2)
   # melakukan Single Point Crossover
   if (angka_random_pemilihan_cara_crossOver == 0) and (list_elitsme[angkaRandomElitsm] !=
TournamentSelection[angkaRandomTournamentSelection]): # kromosom yang di crossover yaitu
angkaRandomTournamentSelection dengan angkaRandomTournamentSelection
     child1 = list_elitsme[angkaRandomElitsm][:point] + TournamentSelection[angkaRandomTournamentSelection][point:]
#kromosom dari TournamentSelection menjadi ekor
     child2 = TournamentSelection[angkaRandomTournamentSelection][:point] + list_elitsme[angkaRandomElitsm][point:]
#kromosom dari TournamentSelection menjadi kepala
     # agar tidak ada yang samaantara child yang telah ada dengan child hasil crossover
     if child1 not in child:
      child.append(child1)
     if child2 not in child:
      child.append(child2)

   elif (angka_random_pemilihan_cara_crossOver == 1) and (TournamentSelection[angkaRandomTournamentSelection2] !=
TournamentSelection[angkaRandomTournamentSelection]) : # kromosom yang di crossover yaitu elitsm dengan
```

```
angkaRandomTournamentSelection
      child1 = TournamentSelection[angkaRandomTournamentSelection2][point:] +
TournamentSelection[angkaRandomTournamentSelection][point:] #kromosom dari TournamentSelection menjadi ekor
      child2 =  TournamentSelection[angkaRandomTournamentSelection][:point] +
TournamentSelection[angkaRandomTournamentSelection2][:point] #kromosom dari TournamentSelection menjadi kepala
      # agar tidak ada yang samaantara child yang telah ada dengan child hasil crossover
      if child1 not in child:
        child.append(child1)
      if child2 not in child:
        child.append(child2)

    elif (angka_random_pemilihan_cara_crossOver == 2) and (list_elitsme[angkaRandomElitsm] !=
list_elitsme[angkaRandomElitsm2]) : # kromosom yang di crossover yaitu elitsm dengan elitsm
      child1 = list_elitsme[angkaRandomElitsm2][:point] + list_elitsme[angkaRandomElitsm][point:] #kromosom dari
angkaRandomElitsm menjadi ekor
      child2 = list_elitsme[angkaRandomElitsm][:point] + list_elitsme[angkaRandomElitsm2][point:] #kromosom dari
angkaRandomElitsm menjadi kepala
      # agar tidak ada yang samaantara child yang telah ada dengan child hasil crossover
      if child1 not in child:
        child.append(child1)
      if child2 not in child:
        child.append(child2)
    # print('if')

    # print(len(child))
 return child[:jumPopulasi - len(list_elitsme)] #mengembalikan child dengan jumlah jumPopulasi - banyak elitsm
```

3.4.2 Mutasi

Metode yang kami gunakan pada bagian ini adalah metode pemilihan nilai secara acak.

```
# Mutasi
def randomMutation(Pm,min,max,jumKromosom,childList):
 # setiap kromosom akan di perhitungkan terjadi mutasi
 for i in range(len(childList)):
   angkaRandom = random.uniform(0.0,1.0)
   if angkaRandom <= Pm :
     # generate angka random
     angkaRandomValue = random.randint(min,max)
     # mencari angkaRandomValue yang berbeda dengan isi index ke 3 yang akan diganti jika sama
     while childList[i][3] == angkaRandomValue :
       angkaRandomValue = randint(min,max)
     childList[i][3] = angkaRandomValue
 return childList
```

3.5 Probabilitas Operasi Genetik (Pc dan Pm)

Dalam desain algoritma yang dibuat, probabilitas crossover dan probabilitas mutasi yang dipakai adalah 0.6 dan 0.06. The crossover used was based on one single point crossover to be around the rate of (0.6) [5]. Kemudian pada artikel “Effect of Mutation and Effective Use

of Mutation in Genetic Algorithm"[2], probabilitas mutasi dengan nilai 0.05-0.06, mendapatkan hasil yang paling baik dari yang lain.

```
# data Probabilitas Crossover dan Probabilitas Mutasi
Pc = 0.6
Pm = 0.06
```

3.6 Metode Pergantian Generasi (Seleksi Survivor)

Menggunakan pendekatan steady state model yaitu generasi setelahnya memliki beberapa kromosom dari generasi sebelumnya. Melalui proses seleksi elitsm (dimana individu – individu yang terpilih untuk menjadi generasi selanjutnya berdasarkan pada nilai fitness tertinggi) diambil 10 terbesar dari populasi sebelumnya ditambah dengan hasil crossover yang sudah di mutasi.

```
# Pergantian Generasi
def steadyStateModel(elitsm,child):
 return elitsm + child
```

3.7 Kriteria Penghentian Evolusi

Dalam penghentian Evolusi kami menggunakan pendekatan penentuan jumlah generasi yang telah dipilih. Kami menggunakan 70 generasi dan kemudian akan dihentikan terjadinya evolusi kembali. Kami menggunakan pendekatan ini dikarenakan pengamatan perubahan kromosom mendekati generasi 70 telah stagnan tidak berubah.

```
# Banyak Generasi
jumGenerasi = 70
for z in range(jumGenerasi):
  data = {'Kromosom': populasi}
  # Create DataFrame
  df_population = pd.DataFrame(data)

  # Membuat dan mengisi fitness pada dataframe
  df_population = createFitness(df_population)

  # Seleksi Orang tua
  listElitisme,listNilaiFitnessTerbaik = getElitsm(df_population)

  Parent = tournamentSelection(df_population)

  # generate child yang berisi anak hasil persilangan elitsm dengan parent
(elitsm tidak termasuk kedalam child)
  child=singlePointCrossover(listElitisme,Parent,Pc,jumKromosom,jumIndividu)
  # Lakukan Mutasi pada child
  child = randomMutation(Pm,min,max,jumKromosom,child)
  # seleksi survivor
  populasi = steadyStateModel(listElitisme,child)
```

BAB 4
ANALISIS

4.1 Analisis Parameter Probabilitas Mutasi

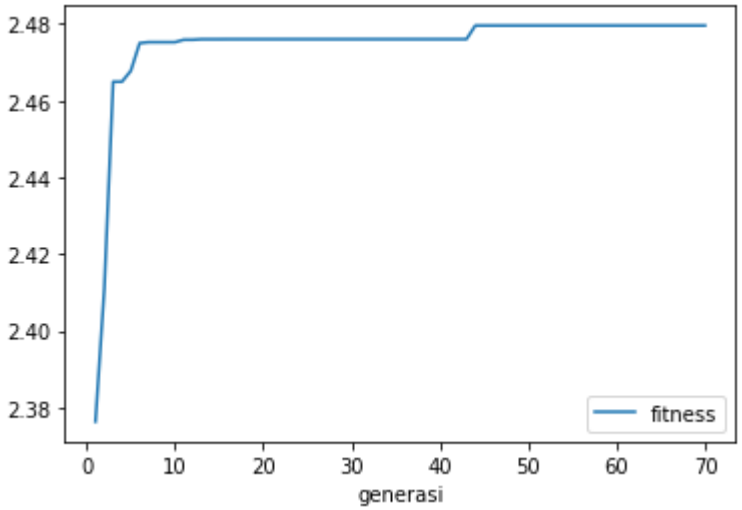
Kami melakukan pengamatan dan mengubah penggunaan parameter probabilitas mutasi terhadap keluaran dari model *Genetic Algorithm* yang kami kembangkan dan didapatkan hasil seperti pada tabel 4.1.0.

Probabilitas Mutasi	Maksimal	Mean	STD
0.02	2.481119601627994	2.474654	0.023358
0.03	2.474940307297132	2.460548	0.027800
0.04	2.4817033877053785	2.476455	0.026008
0.05	2.474940307297132	2.469126	0.022219
0.06	2.48162677313103	2.464152	0.009758
0.07	2.4815369293614316	2.475892	0.019216
0.08	2.4815803189917007	2.475368	0.030297
0.09	2.4817201899546517	2.476645	0.019297
0.1	2.474923564635092	2.464763	0.034360
0.2	2.481726344104677	2.460624	0.044972
0.3	2.481277386921542	2.479568	0.010988
0.4	2.4816252823404943	2.470245	0.023427
0.5	2.47842940114617	2.475332	0.014800
0.6	2.4811122980803333	2.471381	0.025003
0.7	2.477605311675321	2.46915	0.021621
0.8	2.4814403879114995	2.475842	0.021497
0.9	2.4817260483351538	2.469234	0.049072
1.0	2.480791193747425	2.459298	0.031604

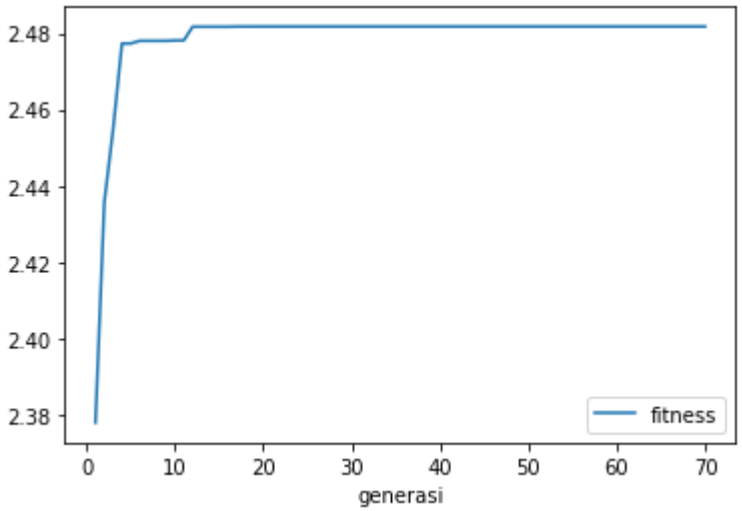
(Tabel 4.1.0 Hasil Analisis Penggunaan Probabilitas Mutasi)

Pada hasil tabel 4.1.0 terlihat bahwa penggunaan Probabilitas Mutasi 0.06 memiliki keluaran yang terbaik yang dilihat dari fitness maksimal yang didapatkan dan memiliki standar deviasi yang bahkan dibawah 0.01 yang berarti sebaran data dari fitness terbaik tiap generasi tidak

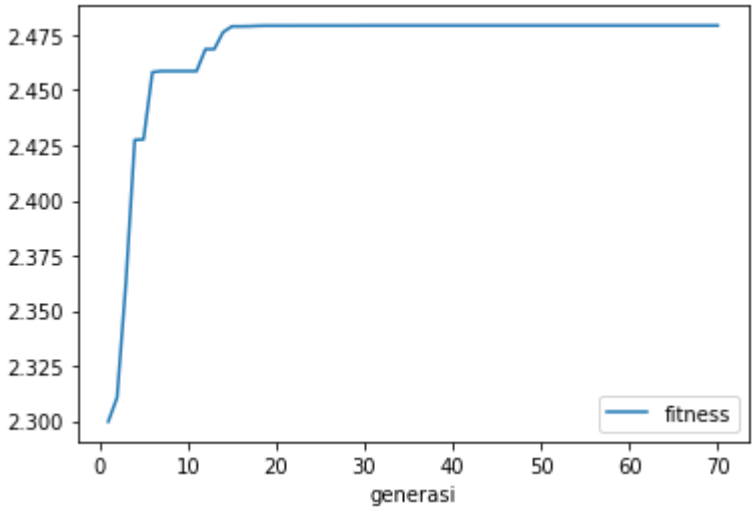
menyebar dan lebih menuju ke satu titik dibanding dengan nilai Probabilitas Mutasi lainnya yang memiliki standar deviasi lebih tinggi.



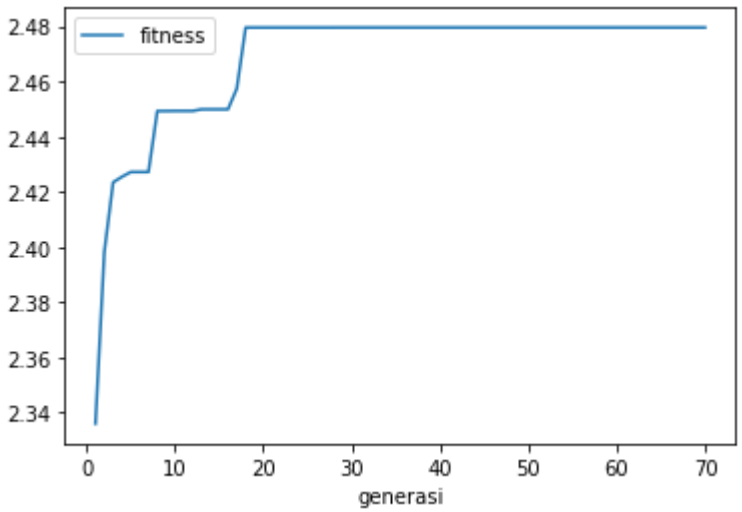
(Gambar 4.1.1 Fitness Terbaik dari Probabilitas Mutasi 0.02)



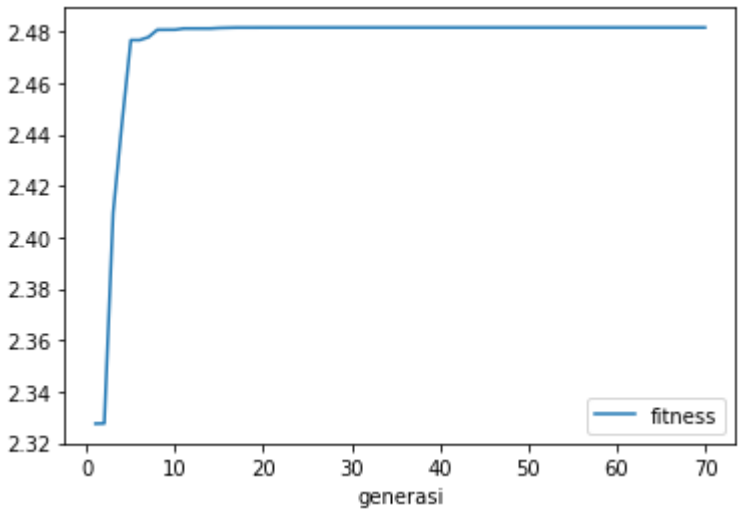
(Gambar 4.1.2 Fitness Terbaik dari Probabilitas Mutasi 0.03)



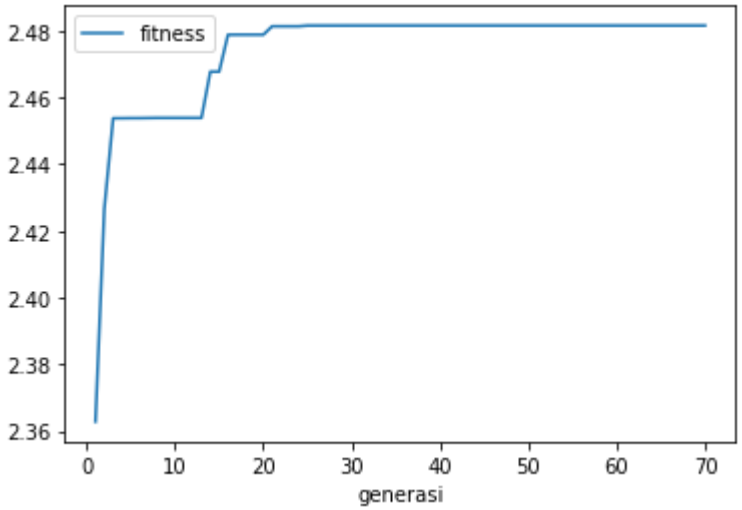
(Gambar 4.1.4 Fitness Terbaik dari Probabilitas Mutasi 0.04)



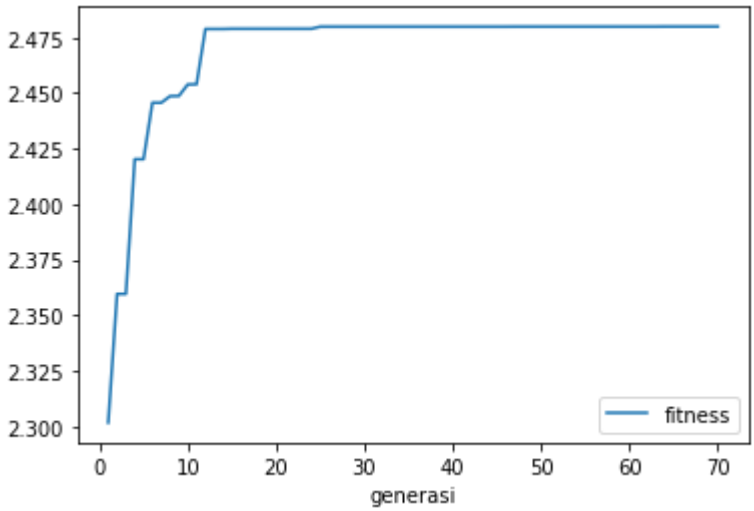
(Gambar 4.1.5 Fitness Terbaik dari Probabilitas Mutasi 0.05)



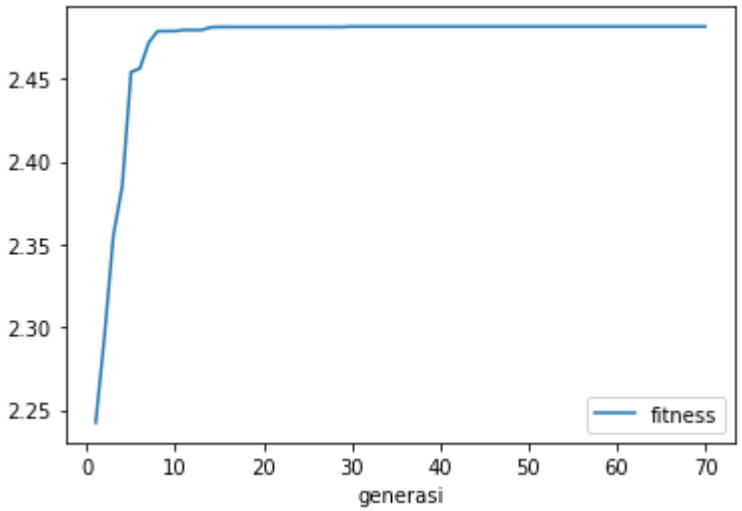
(Gambar 4.1.6 Fitness Terbaik dari Probabilitas Mutasi 0.06)



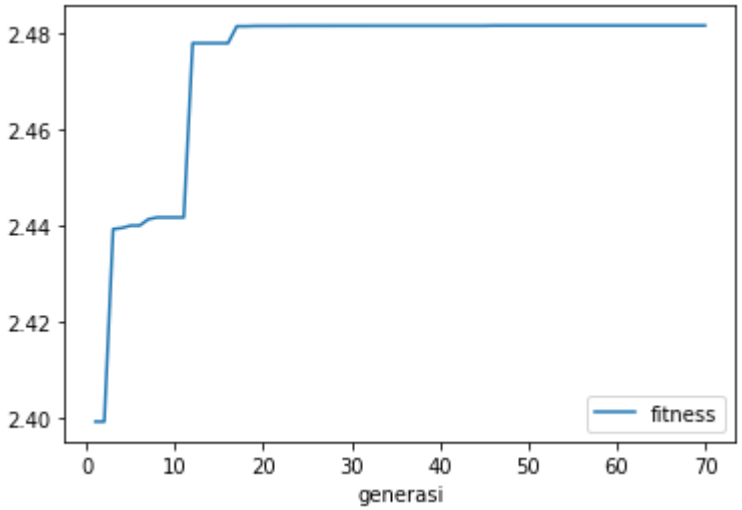
(Gambar 4.1.7 Fitness Terbaik dari Probabilitas Mutasi 0.07)



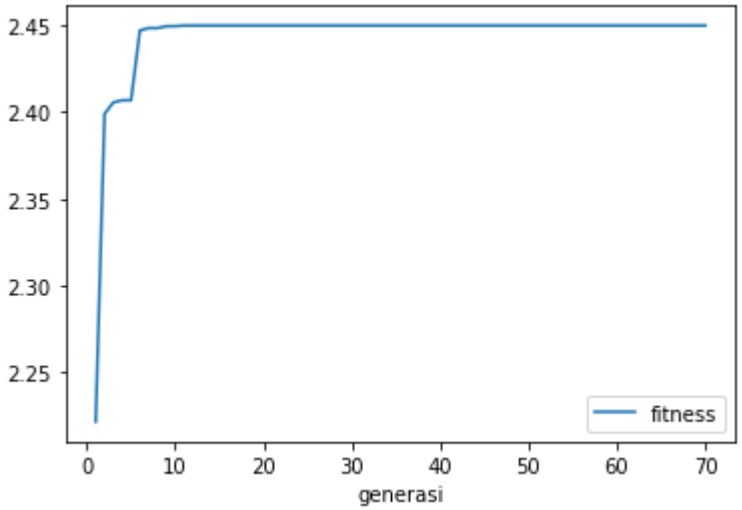
(Gambar 4.1.8 Fitness Terbaik dari Probabilitas Mutasi 0.08)



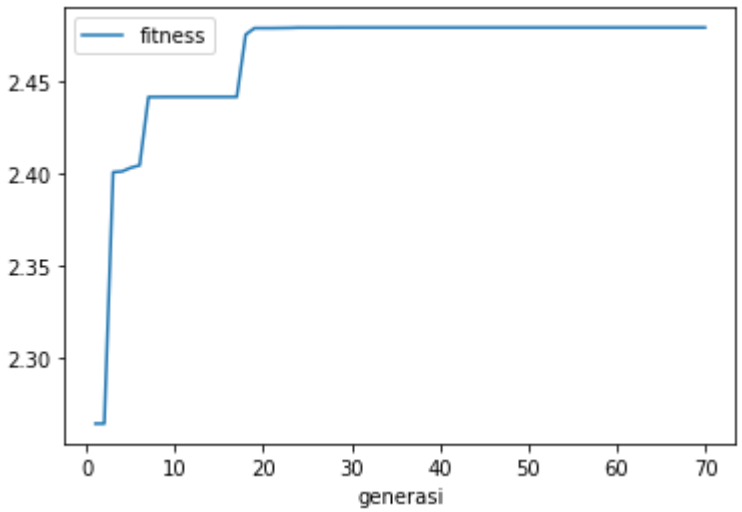
(Gambar 4.1.9 Fitness Terbaik dari Probabilitas Mutasi 0.09)



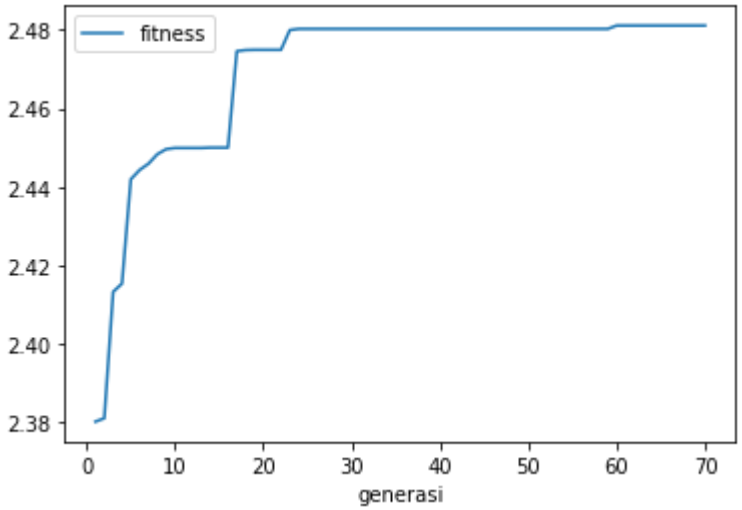
(Gambar 4.1.10 Fitness Terbaik dari Probabilitas Mutasi 0.1)



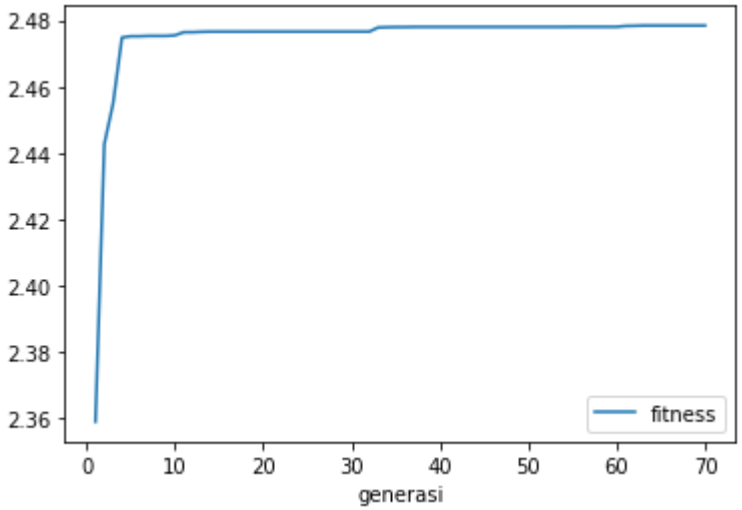
(Gambar 4.1.11 Fitness Terbaik dari Probabilitas Mutasi 0.2)



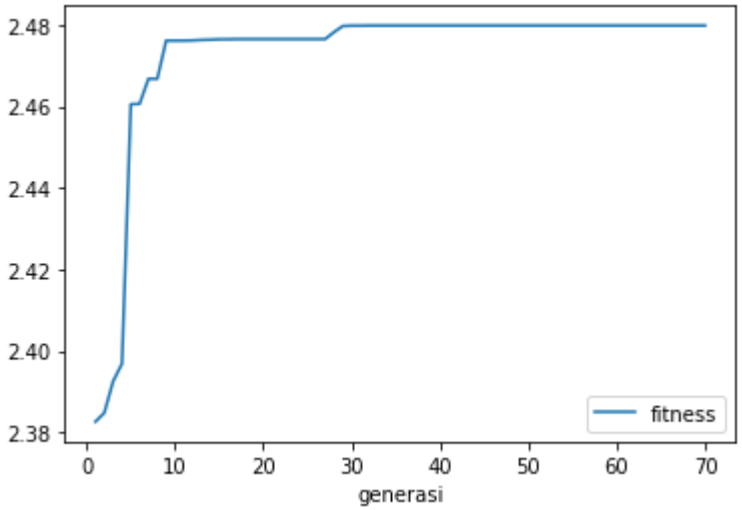
(Gambar 4.1.12 Fitness Terbaik dari Probabilitas Mutasi 0.3)



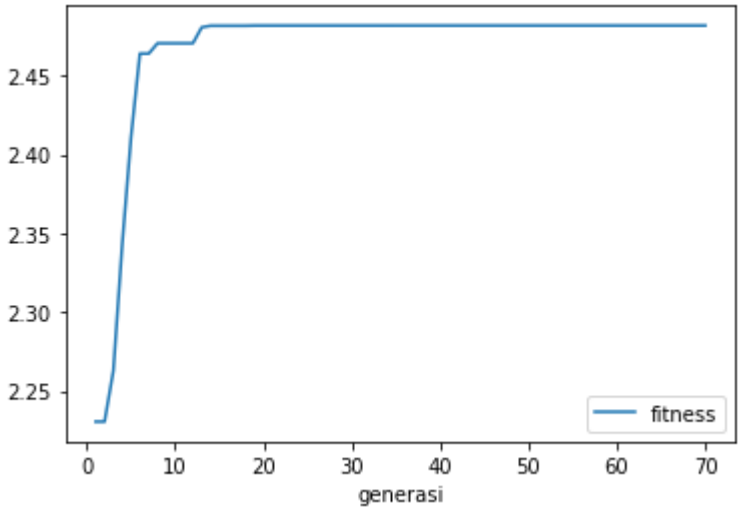
(Gambar 4.1.13 Fitness Terbaik dari Probabilitas Mutasi 0.4)



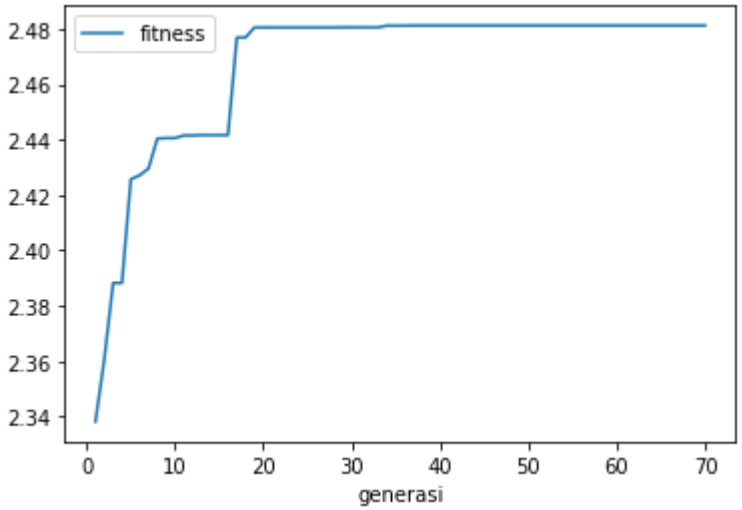
(Gambar 4.1.14 Fitness Terbaik dari Probabilitas Mutasi 0.5)



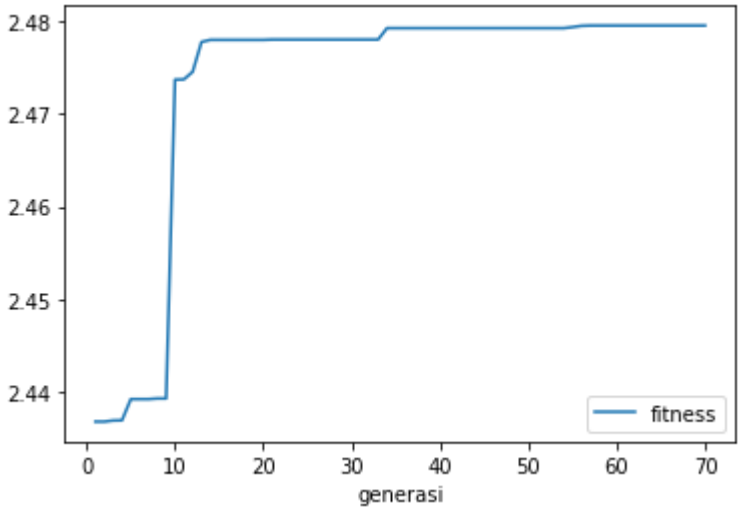
(Gambar 4.1.15 Fitness Terbaik dari Probabilitas Mutasi 0.6)



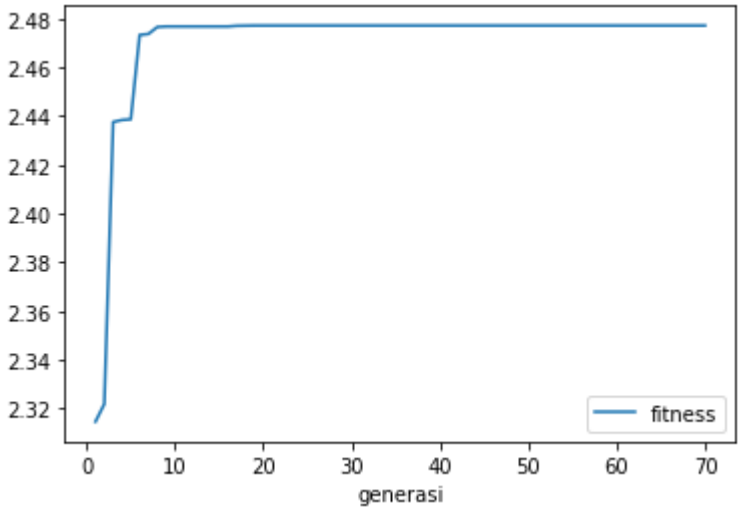
(Gambar 4.1.16 Fitness Terbaik dari Probabilitas Mutasi 0.07)



(Gambar 4.1.17 Fitness Terbaik dari Probabilitas Mutasi 0.8)



(Gambar 4.1.18 Fitness Terbaik dari Probabilitas Mutasi 0.9)



(Gambar 4.1.19 Fitness Terbaik dari Probabilitas Mutasi 1.0)

Pada gambar 4.1.0 sampai 4.1.19 kami mendapatkan bahwa pemilihan Probabilitas Mutasi (Pm) mempengaruhi nilai maksimal dari fitness dan juga pergerakan nilai fitness tiap generasi. Contoh pada Pm = 0.8 butuh sampai 20 generasi agar nilai maksimal fitnes terbaik dapat memulai flat curve nya sedangkan pada Pm = 0.06 dibutuhkan kurang dari 10 generasi untuk mendapatkan flatt curve nya.

KESIMPULAN

Pada penelitian ini kami mengembangkan *genetic algorithm* (GA) untuk menyelesaikan permasalahan pencarian nilai x dan y yang mengakibatkan didapatkan nilai maksimal terhadap rusmus h(x,y) = (cos(x^2) ∗sin(y^2)) + (x+y). Dalam membangun model GA kami menggunakan pendekatan dan parameter seperti pada bab 3 dan mendapatkan hasil bahwa nilai x = 0.8600000000000003 dan y = 1.0 merupakan kombinasi nilai x dan y yang menghasilkan nilai maksimal dengan nilai h(x,y) = 2.48162677313103. Kami juga mencoba menganalisis pencarian nilai terbaik pada probabilitas Mutasi (Pm) untuk model GA yang telah kami kembangkan pada bab 4 dan mendapatkan bahwa Pm yang terbaik adalah 0.06 sama seperti Pm awal yang kami pilih berdasarkan literatur yang ada. Kedepannya kami berharap akan adanya pengembangan pada model GA kami dan pencarian nilai parameter terbaik khusus terhadap model GA kami.

Daftar Pustaka

[1] Kora, P dan Yadlapalli, P . 2017 . Crossover Operators in Genetic Algorithms: A Review , Volume 162 – No 10, Hal, 35
[2] Dassanayake, P . 2015 . Effect of Mutation and Effective Use of Mutation in Genetic Algorithm, Hal, 22
[3] Tutorials Point . Genotype Representation . https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_genotype_representation.html
[4] Schlierkamp-Voosen, D. Optimal interaction of mutation and crossover in the breeder genetic algorithm. In International Conference on Genetic Algorithms; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1993; 648p
[5] Hassanat, D dkk.2019. Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach

Lampiran

Video Presentasi

Fadhil Wisnu Ramadhan - 1301190378
https://drive.google.com/file/d/1c0hPS4uHNGSVTiZAUTnP8WrZC5haTHKt/view?usp=sharing

Andre Agasi Simanungkalit - 1301190338
https://drive.google.com/file/d/1auyuJorRJHTUGxll5bHM4nRbqS6LjLBL/view?usp=sharing

Fendi Irfan Amorokhman - 1301191447
https://drive.google.com/drive/folders/13O6YpZzPbfQSLFw5M4BfvHVgyZVesIx5?usp=sharing