Laporan Observasi Tugas Pemrograman 1

Diajukan Untuk Memenuhi Tugas Mata Kuliah Pengantar Kecerdasan Buatan

Dosen Pengampu: Muhammad Yuslan Abubakar, M.Kom.

Disusun Oleh:

Ni Made Dwipadini Puspitarini	(1301194141)
Iqbal Saviola Syah Billhaq	(1301190318)
Afrizal Syahruluddin Yusuf	(1301194288)

PROGRAM STUDI INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

2020/2021

DAFTAR ISI

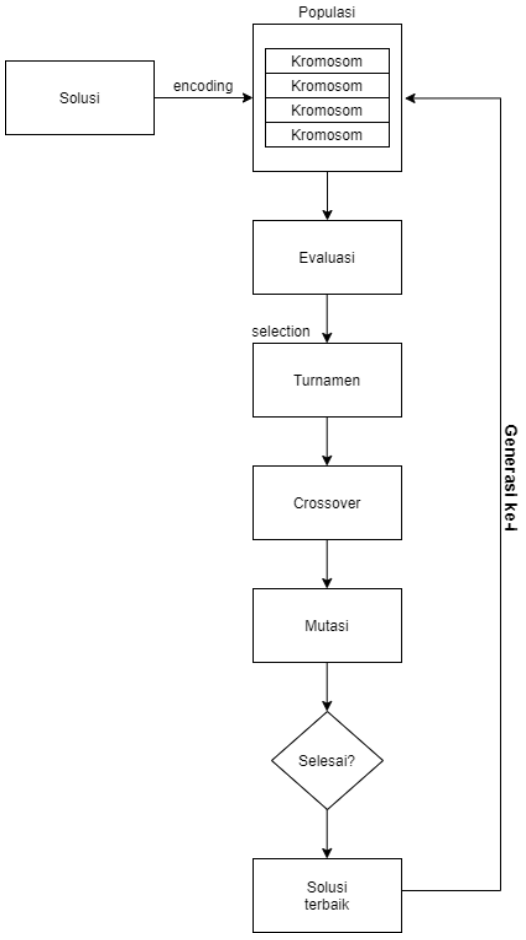
BAB I
PENDAHULUAN

1.1 Latar Belakang

Laporan hasil observasi mengenai *Genetic Algorithm* (GA) merupakan salah satu kegiatan wajib dalam rangka memenuhi penilaian pada mata kuliah Pengantar Kecerdasan Buatan. GA merupakan contoh dari metode heuristik yang digunakan untuk mengoptimasi suatu permasalahan dengan mencari solusi yang mendekati optimal.

Genetic Algorithm (GA) memiliki cara kerja yang hampir sama dengan hukum rimba. Dimana arti dari hukum rimba sendiri ialah siapa yang kuat, dialah pemenangnya. Secara konsep, GA terinspirasi dengan teori evolusi Darwin yang menjelaskan bahwa suatu individu dapat bertahan hidup dengan melewati beberapa proses yakni reproduksi (*crossover*) dan mutasi. Konsep tersebut kemudian diadaptasi dan diimplementasikan pada *Genetic Algorithm* (GA).

Dengan adanya penugasan mengenai *Genetic Algorithm* (GA) ini, diharapkan para mahasiswa sebagai *observer* dapat mengetahui bagaimana proses yang terjadi pada GA yang kemudian dapat mengimplementasikannya pada permasalahan yang diberikan. Sehingga dapat menemukan sebuah solusi yang mendekati optimal.



Gambar 1.1 Diagram Alir GA

1.2 Tujuan Observasi

Berdasarakan latar belakang yang telah kami uraikan di atas, maka tujuan dibuatnya laporan observasi ini adalah menemukan nilai maximum atau solusi yang mendekati optimal dari fungsi :

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

dengan menggunakan *Genetic Algorithm* (GA) ke dalam suatu program komputer dengan batasan $-1 \leq x \leq 2$ dan $-1 \leq y \leq 1$.

1.3 Manfaat Observasi

Manfaat dari pelaksanaan observasi ini adalah :

1. Meningkatkan pengetahuan dan pemahaman observer mengenai *Genetic Algorithm* (GA).
2. Meningkatkan keterampilan *coding* para observer.
3. Melatih *soft skill* berupa kerja sama para observer

1.4 Metode Observasi

Kami menggunakan bahasa pemrograman python untuk membangun proses seperti fungsi dan prosedur dengan bantuan notebook *google colaboratory* dan *visual studio code* sebagai teks editor.

BAB II
PROSES YANG DIBANGUN

2.1 Generate Populasi

Generate populasi merupakan langkah awal untuk menginisialisasi populasinya. Pada tahap ini ditentukan berapa panjang kromosom dan jumlah populasi yang akan digunakan. Semakin banyak jumlah populasinya maka akan lebih besar pula varian solusi yang dihasilkan. Hal ini akan memperbesar kemungkinan untuk mencapai solusi terbaik.

2.2 Dekode Kromosom

Decode kromosom merupakan suatu proses pengubahan bilangan dari satu bentuk ke bentuk lainnya. Misalnya bilangan biner, real, atau integer. Hasil dari decode kromosom sendiri merepresentasikan solusi dari suatu permasalahan.

2.3 Perhitungan Fitness

Fungsi Fitness

Maksimasi: $f = h$

Minimasi: $f = \frac{1}{(h+a)}$

Gambar 2.2 Fungsi Fitness

Nilai fitness merupakan nilai yang memperlihatkan bagaimana ketangguhan atau kualitas kromosom dalam beradaptasi. Perhitungan dalam nilai fitness bergantung kepada jenis masalah yang akan diselesaikan, misalnya masalah maksimasi atau minimasi. Maksimasi berarti mencari nilai maksimal dari suatu permasalahan, sedangkan minimasi artinya mencari nilai minimum.

Setiap jenis masalah, memiliki perhitungan fitness yang berbeda-beda. Jika tujuannya adalah memaksimalkan sebuah fungsi, maka fungsi *fitness* yang digunakan adalah fungsi yang dioptimasi tersebut. Perhitungan nilai *fitness* yang bisa digunakan adalah $f = h$. Tetapi, solusi masalah yang ingin dicapai bertujuan meminimalkan fungsi h, maka fungsi h tidak bisa digunakan secara langsung.

2.4 Pemilihan Orang Tua

Proses pemilihan orang tua, dapat dilakukan dengan beberapa cara. Tiga cara seleksi orang tua yang paling umum digunakan adalah :

1. Roullete Wheel

   Metode ini termasuk ke dalam teknik seleksi orangtua yang proporsional terhadap nilai *fitness*-nya. Semakin besar nilai *fitness* suatu individu, maka semakin besar pula peluangnya untuk terpilih sebagai orangtua. Penyeleksian dibuat dengan membangkitkan nilai *random* dari jarak semua nilai fitness suatu individu.

2. Baker’s SUS

   Metode ini merupakan pemodifikasian dari metode roullete wheel. Yang membedakan adalah jumlah jarum pada baker’s SUS lebih banyak daripada roullete wheel. Dimana semua jarum diletakkan secara merata dengan jarak yang sama pada roda roullete. Metode baker’s SUS dinilai lebih adil daripada roullete wheel karena, individu dengan fitness yang tinggi memiliki kesempatan terpilih lebih dari satu kali untuk menjadi orang tua.

3. Tournament Selection

   Metode ini merupakan teknik sampling atau seleksi. Penyeleksian dilakukan dengan memilih *a* kromosom secara acak dari sebuah *b* kromosom, kemudian akan dipilih kromosom dengan nilai fitness terbesar untuk dijadikan orang tua.

2.5 Elitisme

Elitisme mengandung kata elite yang berarti kumpulan individu dengan nilai fitness terbaik. Elitisme akan selalu menyimpan beberapa kromosom terbaik dengan nilai fitness terbaik pula. Jika dalam regenerasinya menemukan kromosom yang lebih baik, maka kromosom terbaik sebelumnya akan ditukar dengan kromosom terbaik yang baru.

2.6 Crossover (pindah silang)

Crossover adalah salah satu proses yang bisa digunakan untuk membentuk individu baru dalam algoritma genetika. Crossover dilakukan oleh 2 orang tua yang nantinya akan menghasilkan individu baru dengan sifat yang sama persis dengan orang tuanya.

2.7 Mutasi

Cara kerja mutasi hampir sama dengan crossover. Mutasi merupakan operasi genetika yang mewakili proses mutasi dalam perjalanan hidup individu. Namun, mutasi tidak selalu terjadi karena memiliki kemungkinan yang kecil. Jika terjadi, kemungkinan besar mutasi itu tidak berguna. Hasil dari mutasi sendiri dapat menjadi individu yang lebih baik atau bahkan menjadi individu yang lebih buruk.

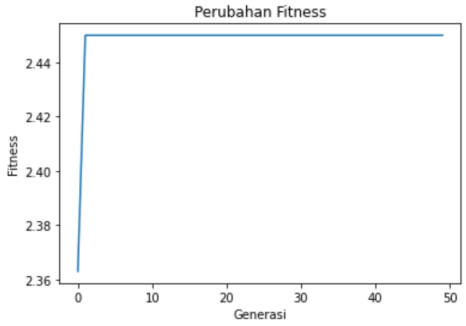
2.8 Pergantian Generasi

Pergantian generasi dibutuhkan untuk mendapatkan populasi berikutnya. Beberapa individu yang lemah atau memiliki nilai fitness yang rendah, pada proses ini akan dibuang dan digantikan oleh individu baru yang lebih unggul. Pergantian generasi dapat dilakukan dengan dua cara yaitu generational model dan steady state model.

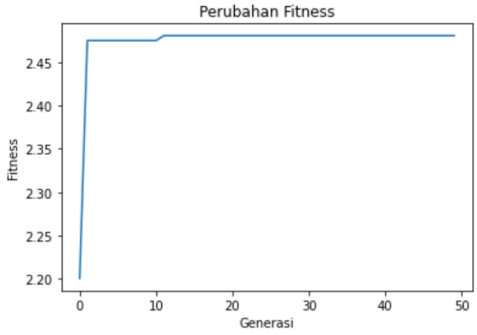
BAB III
HASIL OBSERVASI

3.1 Desain Kromosom dan Metode Pendekodean

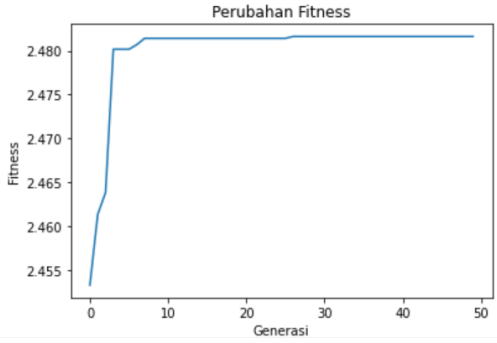
Desain kromosom yang kami gunakan di tugas pemograman kali ini adalah **representasi biner**. Kami memilih desain kromosom dengan metode pengkodean representasi biner agar lebih mudah untuk melakukan crossover dan mutasi pada proses kedepannya. Panjang kromosom yang kami gunakan sebesar 20, karena kami melihat perubahan fitness dengan panjang kromosom 20 lebih optimal jika dibandingkan dengan panjang kromosom sebesar 6 dan 10.



panjang kromosom 6



panjang kromosom 10



panjang kromosom 20

Untuk desain kromosomnya, kami melakukan pengisian gen melalui fungsi generate_populasi(lenKrom, n) dengan merandom angka 0 dan 1 (seperti langkah yang kami blok di bawah ini). Perandoman angka 0 dan 1 akan terus dilakukan sampai panjang kromosomnya terpenuhi. Kemudian, kumpulan gen tersebut akan ditambahkan ke dalam list krom yang kemudian akan menjadi populasi.

```
def generate_populasi(lenKrom, n):
    pop = []
    for i in range(n):
        krom = []
        for j in range(lenKrom):
            krom.append(random.randint(0, 1))
        pop.append(krom)
    return pop
```

Karena menggunakan representasi biner, maka rumus pendekodean yang kami gunakan adalah :

$$x = r_b + (r_a - r_b) / \Sigma^{n}_{(i=1)} . 2^{-i} \ (g_1 . 2^{-1} + g_2 . 2^{-2} + ... + g_n . 2^{-n})$$

Rumus pendekodean tersebut telah kami tuangkan ke dalam fungsi decode_kromosom(krom, xran, yran). Strategi yang kami gunakan yakni membagi rumus tersebut ke dalam beberapa bagian untuk memudahkan perhitungan. Berikut penjelasannya :

1. Pertama kami men-*declare* satu variabel bernama "mid", dimana variabel tersebut akan di-*assign* menjadi nilai tengah kromosom.

```
def decode_kromosom(krom,xran,yran):
    mid = len(krom)//2 #titik potong untuk kromosom
```

2. Setelah itu, kami menghitung nilai ($g_1 . 2^{-1} + g_2 . 2^{-2} + ... + g_n . 2^{-n}$) untuk x. Pada bagian ini, kami mendifinisikan bagian kiri dari mid (:mid) merupakan biner yang merepresentasikan nilai x.

```
#menghitung rumus (g1.(2^-1) + g2.(2^-2) + ... + gi.(2^-i)) untuk x
gx = 0
mid_x = krom[:mid]
for i in range(1, mid+1):
    gx = gx + (mid_x[i-1] * (2**-i))
```

3. Selanjutnya, kami menghitung nilai ($g_1 . 2^{-1} + g_2 . 2^{-2} + ... + g_n . 2^{-n}$) untuk y. Pada bagian ini, kami mendifinisikan bagian kanan dari mid (mid:) merupakan biner yang merepresentasikan nilai y.

```
#menghitung rumus (g1.(2^-1) + g2.(2^-2) + ... + gi.(2^-i)) untuk y
gy = 0
mid_y = krom[mid:]
for i in range(1, mid+1):
    gy = gy + (mid_y[i-1] * (2**-i))
```

4. Selanjutnya, kami menyelesaikan rumus pendekodean bagian $\Sigma^{n}_{(i=1)} . 2^{-i}$ yang di assign ke dalam variabel total.

```
#menghitung nilai sum 2^-i
total = sum([2**-(i) for i in range(1, mid+1)])
```

5. Sebagai langkah terakhir, kami menggabungkan semua bagian rumus tersebut hingga menjadi rumus pendekodean yang valid untuk x dan y.

```
x = xran["min"] + (gx * (xran["max"] - xran["min"]) / total)
y = yran["min"] + (gy * (yran["max"] - yran["min"]) / total)
```

3.2 Ukuran Populasi

Kami melakukan perbandingan untuk mengetahui bagaimana pengaruh ukuran populasi terhadap pencarian solusi untuk permasalahan kami. Sebagai perbandingan, kami menggunakan ukuran populasi sebesar 20 dan 60. Untuk membuat populasi, kami menggunakan fungsi generate_populasi(lenKrom, n), dimana lenKrom merupakan panjang kromosom dan n merupakan ukuran populasinya. Secara garis besar, fungsi di bawah akan melakukan perulangan sebanyak ukuran populasinya. Di dalam perulangan tersebut, terdapat perulangan lagi untuk membentuk kumpulan gen sepanjang lenKrom yang nantinya akan ditambahkan ke dalam list pop sebagai populasi.

```
def generate_populasi(lenKrom, n):
    pop = []
    for i in range(n):
        krom = []
        for j in range(lenKrom):
            krom.append(random.randint(0, 1))
        pop.append(krom)
    return pop
```

Ketika kami menggunakan ukuran populasi sebesar 20 (gambar di bawah ini), nilai kromosom, fitnes, dan (x, y) terbaik baru mulai stabil pada generasi ke-44.

```
Gen ke : 44
Kromosom Terbaik : [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.4816153736443063, (x, y) : (0.8797653958944283, 1.0)
Gen ke : 45
Kromosom Terbaik : [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.4816153736443063, (x, y) : (0.8768328445747802, 1.0)
Gen ke : 46
Kromosom Terbaik : [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.4816153736443063, (x, y) : (0.8768328445747802, 1.0)
Gen ke : 47
Kromosom Terbaik : [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.4816153736443063, (x, y) : (0.8768328445747802, 1.0)
Gen ke : 48
Kromosom Terbaik : [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.4816153736443063, (x, y) : (0.8768328445747802, 1.0)
Gen ke : 49
Kromosom Terbaik : [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.4816153736443063, (x, y) : (0.8768328445747802, 1.0)
Gen ke : 50
Kromosom Terbaik : [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.4816153736443063, (x, y) : (0.8768328445747802, 1.0)
```

Berbeda ketika kami menggunakan ukuran populasi sebesar 60 (gambar di bawah ini), nilai kromosom, fitness, dan (x, y) terbaik sudah mulai stabil pada generasi ke-11.

```
Gen ke : 11
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 12
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 13
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 14
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 15
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 16
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 17
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 18
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 19
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 20
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 21
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
```

Dari percobaan di atas dapat dilihat bahwa menggunakan ukuran populasi sebesar 60 memberikan solusi terbaik lebih cepat daripada ukuran populasi 20. Maka, kami dapat menarik kesimpulan bahwa ukuran populasi mempengaruhi lama waktu eksekusi dalam pencarian solusi terbaik. Hal ini dapat disebabkan oleh semakin banyaknya ukuran populasi, maka semakin besar pula kemungkinan untuk mencapai solusi terbaik. Atas dasar perbedaan

tersebut, maka kami memutuskan pada tugas pemrograman ini untuk menggunakan ukuran populasi sebesar 60.

3.3 Pemilihan Orang Tua

Teknik pemilihan orang tua yang digunakan pada tugas pemograman kali ini adalah tournament selection. Kami memilih tournament selection, karena konsepnya yang mengadopsi seleksi alami di dunia nyata. Pada seleksi alami, individu pemenanglah yang dapat melakukan persilangan. Metode pemilihan orang tua lain seperti roullete wheel tidak dapat mengakomodasikan permasalahan ini.

```
def tournament_selection(pop, fitness):
    bestparent = random.sample(list(zip(pop,fitness)),random.randint(2, jmlPop))
    bestparent = sorted(bestparent,key= lambda k:k[1],reverse=True)
    return [bestparent[0][0], bestparent[1][0]]
```

Kami mengimplementasikan konsep tournament selection ke dalam fungsi tournament_selection(pop, fitness). Yang dilakukan oleh fungsi ini adalah :

1. Mengambil sampling kromosom beserta nilai fitnessnya secara acak.

```
    bestparent = random.sample(list(zip(pop,fitness)),random.randint(2, jmlPop))
```

2. Kemudian akan dilakukan pengurutan pada sampling tersebut secara descending berdasarkan nilai fitnessnya. Sehingga kromosom dengan nilai fitness tertinggi akan menempati posisi teratas.

```
    bestparent = sorted(bestparent,key= lambda k:k[1],reverse=True)
```

3. Lalu kami mengambil 2 individu terbaik dengan nilai fitness tertinggi untuk menjadi orang tua *(best parent)* yang nantinya akan melanjutkan proses ke crossover.

```
    return [bestparent[0][0], bestparent[1][0]]
```

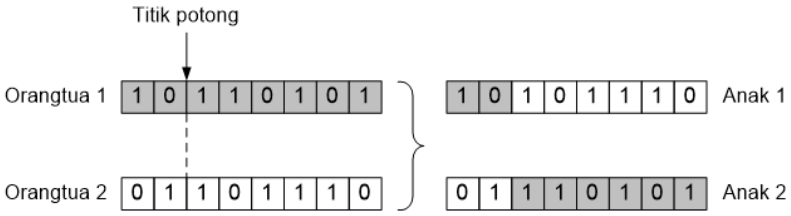
3.4 Pemilihan dan Teknik Operasi Genetik (Crossover dan Mutasi)

- 1-Point Crossover

Untuk crossover, kami memilih teknik *one point crossover*, dimana kami hanya membutuhkan 1 titik potong (*cut off*). Kami menerapkan konsep *one point crossover* ke dalam fungsi crossover(parent1, parent2, probCross). Dimana parent1 dan parent2 merupakan orang tua terbaik yang dihasilkan melalui tournament selection serta probCross adalah probabilitas crossovernya sebesar 0.8.

```
#Teknik one point crossover
def crossover(parent1, parent2, probCross):
    child1 = parent1
    child2 = parent2
    if random.random() <= probCross:
        crossPoint = random.randint(1, (lenKrom)-1) #menentukan 1 titik crossover
        child1 = parent1[:crossPoint] + parent2[crossPoint:]
        child2 = parent2[:crossPoint] + parent1[crossPoint:]
    return child1, child2
```

Cara kerja dari fungsi tersebut ialah terlebih dahulu mengecek apakah kondisi dalam if bernilai true. Jika bernilai true, maka crossover akan terjadi. Sebelum crossover terjadi, dilakukan perandoman dulu untuk menentukan titik potongnya dari 1 - 19. Kemudian setelah titik potong berhasil ditentukan, barulah terjadi crossover dengan konsep seperti gambar di bawah ini. Hasil dari crossover antara 2 parent terbaik akan menghasilkan individu baru yang sifatnya sama persis dengan orang tuanya. Namun apabila kondisi if bernilai false, maka crossover tidak akan terjadi.

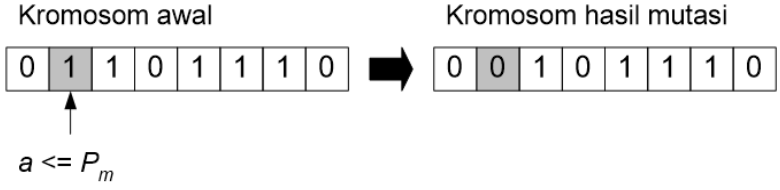


- Mutasi untuk representasi biner

Kami menggunakan representasi biner dalam desain kromosom dan metode pengkodean, sehingga teknik yang kami pilih untuk mutasi yakni mutasi untuk representasi biner pula. Kami menuangkan konsep mutasi tersebut ke dalam fungsi mutasi(krom, probMut) dimana krom merupakan kromosom dan probMut adalah probabilitas mutasi sebesar 0.2.

```
def mutasi(krom, probMut):
    for i in range(len(krom)):
        if random.random() <= probMut:
            if krom[i] == 0:
                krom[i] = 1
            else:
                krom[i] = 0
    return krom
```

Secara sederhana, fungsi di atas melakukan perulangan sebanyak jumlah panjang kromosomnya. Apabila kondisi if bernilai true, maka mutasi akan terjadi dengan membalik setiap bit 1 menjadi 0 begitu pula sebaliknya (seperti gambar di bawah ini). Namun apabila kondisi if bernilai false, maka mutasi tidak akan terjadi. Faktanya, mutasi jarang terjadi karena kemungkinannya kecil. Hasil dari mutasi sendiri dapat menjadi individu yang lebih baik atau bahkan menjadi individu yang lebih buruk.



3.5 Probabilitas Operasi Genetik (Pc dan Pm)

- Probabilitas crossover

Nilai probabilitas persilangan yang baik berada pada range 60%-80%. Pada tugas pemrograman ini, kami mendefinisikan Pc sebagai probCross dengan mengeset nilai Pc sebesar 80% (0.8). Karena crossover atau persilangan pasti akan menghasilkan individu baru yang unik.

- Probabilitas mutasi

Pada tugas pemrograman ini, kami mendefinisikan Pm sebagai probMut dengan mengeset nilai Pm sebesar 20% (0.2). Karena mutasi jarang terjadi dan mutasi tidak selalu menghasilkan individu yang lebih baik namun dapat juga menghasilkan individu yang lebih buruk.

3.6 Metode Pergantian Generasi (Seleksi Survivor)

Metode pergantian generasi (seleksi suvivor) yang kami gunakan adalah generational model. Pada generational model, semua populasi pada suatu generasi akan digantikan oleh populasi baru pada generasi berikutnya. Kami menambahkan fungsi elitisme(pop, fitness) untuk menjaga kromosom terbaik.

```
for i in range(gen):
    fitness = []
    for krom in pop:
        x, y = decode_kromosom(krom, xran, yran)
        fitness.append(fitness_score(x, y))
    bestfitness.append(max(fitness))
    newPop = elitisme(pop, fitness)
```

Fungsi elitisme(pop, fitness) bekerja dengan cara melakukan pengurutan populasi sekaligus nilai fitnessnya secara descending berdasarkan nilai fitness. Sehingga fungsi elitisme akan menyimpan 2 kromosom teratas yang memiliki nilai fitness tertinggi pada setiap generasinya.

```
def elitisme(pop, fitness):
    elite = zip(pop, fitness)
    elite = sorted(elite,key= lambda k:k[1],reverse=True)
    return [elite[0][0], elite[1][0]]
```

3.7 Kriteria Penghentian Evolusi

Untuk kriteria penghentian evolusi, kami menggunakan metode iterasi maksimum. Kami memberikan batas regenerasi pada angka sampai 50 generasi. Angka 50 generasi tidak terlalu tinggi agar running time tidak terlalu lama dan tidak telalu rendah agar hasil tetap akurat.

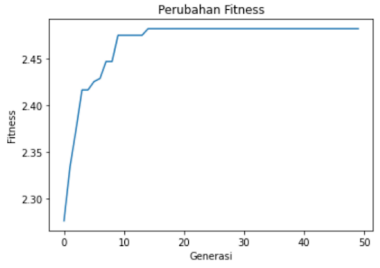
```
for i in range(gen):
    fitness = []
    for krom in pop:
        x, y = decode_kromosom(krom, xran, yran)
        fitness.append(fitness_score(x, y))
    bestfitness.append(max(fitness))
    newPop = elitisme(pop, fitness)
    print(f"Gen ke : {i+1}")
    print(f"Kromosom Terbaik : {newPop[0]}, fitness : {max(bestfitness)}, (x, y) : {decode_kromosom(pop[0], xran, yran)}")
    while len(newPop) < jmlPop:
        [parent1, parent2] = tournament_selection(pop.copy(), fitness)
        child1, child2 = crossover(parent1.copy(), parent2.copy(), probCross)
        child1 = mutasi(child1, probMut)
        child2 = mutasi(child2, probMut)
        newPop.append(child1)
        newPop.append(child2)
    pop = newPop

print(" ")
print("============================BEST SOLUTION============================")
best = list(zip(pop, fitness))
print("Fitness Terbaik : ", best[0][1])
print("Kromosom Terbaik : ", best[0][0])
```

Kami merancang program utama yang akan terus melakukan perulangan apabila jumlah iterasi nya belum mencapai jumlah generasinya. Dalam perulangan tersebut, program akan terus melakukan regenerasi dan menampilkan kromosom, fitness , dan nilai (x, y) yang terbaik di setiap generasinya. Program akan berhenti apabila nilai iterasinya telah mencapai batas generasi. Ketika berhenti, program akan menampilkan solusi terbaik (best solution) secara umum berupa nilai fitness, kromosom, dan nilai (x, y) yang terbaik serta menampilkan grafik perubahan fitnessnya.

```
Gen ke : 50
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)

============================BEST SOLUTION============================
Fitness Terbaik :  2.481727432155131
Kromosom Terbaik :  [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
(x, y) :  (0.8680351906158357, 1.0)
```



BAB IV
HASIL DAN KESIMPULAN

4.1 Hasil

Berdasarkan proses GA yang telah kami rancang, kami mencoba melakukan running untuk mengamati hasilnya. Berikut ini merupakan hasil running dari program Genetic Algorithm yang kami buat

```
Gen ke : 1
Kromosom Terbaik : [1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1], fitness : 2.276199191347529, (x, y) : (0.02052785923753664, -0.3822
Gen ke : 2
Kromosom Terbaik : [1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1], fitness : 2.333760853125886, (x, y) : (1.9501466275659824, 0.644183
Gen ke : 3
Kromosom Terbaik : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0], fitness : 2.373737239019057, (x, y) : (1.9736070381231672, 0.738025
Gen ke : 4
Kromosom Terbaik : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1], fitness : 2.4162683091402544, (x, y) : (1.997067448680352, 0.642228
Gen ke : 5
Kromosom Terbaik : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1], fitness : 2.4162683091402544, (x, y) : (1.997067448680352, 0.894428
Gen ke : 6
Kromosom Terbaik : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0], fitness : 2.4251708687034292, (x, y) : (1.997067448680352, 0.894428
Gen ke : 7
Kromosom Terbaik : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1], fitness : 2.4285082580383626, (x, y) : (2.0, 0.89247311827957)
Gen ke : 8
Kromosom Terbaik : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1], fitness : 2.446633297946429, (x, y) : (2.0, 0.9100684261974585)
Gen ke : 9
Kromosom Terbaik : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1], fitness : 2.446633297946429, (x, y) : (2.0, 0.9882697947214076)
Gen ke : 10
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1], fitness : 2.4747232242858512, (x, y) : (2.0, 0.9882697947214076)
Gen ke : 11
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1], fitness : 2.4747232242858512, (x, y) : (0.8680351906158357, 0.99608
Gen ke : 12
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1], fitness : 2.4747232242858512, (x, y) : (0.8680351906158357, 0.99608
Gen ke : 13
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1], fitness : 2.4747232242858512, (x, y) : (0.8680351906158357, 0.99608
Gen ke : 14
```

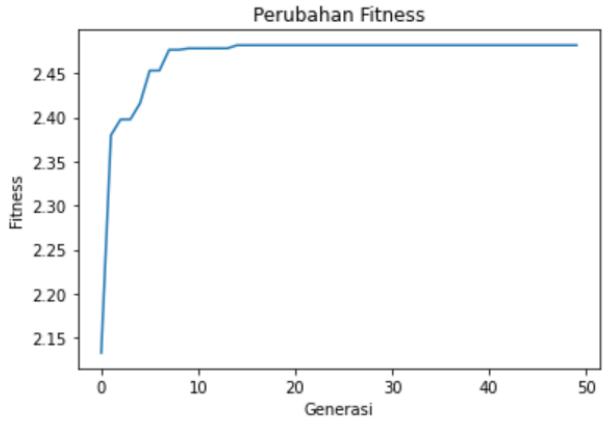
```
Gen ke : 15
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.48171600024987, (x, y) : (0.8739002932551319, 0.9980449
Gen ke : 16
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.48171600024987, (x, y) : (0.8709677419354838, 1.0)
Gen ke : 17
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8709677419354838, 1.0)
Gen ke : 18
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 19
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 20
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 21
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 22
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 23
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 24
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 25
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 26
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 27
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 28
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
```

```
Gen ke : 29
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 30
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 31
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 32
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 33
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 34
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 35
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 36
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 37
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 38
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 39
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 40
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 41
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 42
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)

Gen ke : 43
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 44
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 45
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 46
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 47
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 48
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 49
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
Gen ke : 50
Kromosom Terbaik : [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], fitness : 2.481727432155131, (x, y) : (0.8680351906158357, 1.0)
```

Program tersebut menampilkan nilai fitness dan x dan y dan melakukan regerenerasi sebanyak 50 generasi, agar mendapatkan solusi yang optimal.

```
==============================BEST SOLUTION==============================
Fitness Terbaik :  2.481727432155131
Kromosom Terbaik :  [1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
(x, y) :  (0.8680351906158357, 1.0)
```



Kami mendapatkan nilai fitness terbaik di angka 2.48 dan nilai x di angka 0.86 dan y di angka 1.00. Pada observasi kali ini, kami mendapatkan hasil yang optimal pada generasi ke-17.

4.2 Kesimpulan

- Algoritma Genetika dapat digunakan untuk proses pencarian dan proses optimasi.
- Untuk menggunakan Algoritma Genetika ada beberapa langkah yang harus diperhatikan : mendefinisikan individu, mendefinisikan nilai fitness, pembangkitan populasi awal, menentukan proses seleksi yang akan digunakan, menentukan proses perkawinan silang (cross-over) dan mutasi gen yang akan digunakan.
- Beberapa hal yang perlu diperhatikan dalam pemakaian Algoritma Genetika adalah: Algoritma Genetika adalah algoritma yang dikembangkan dari proses pencarian solusi menggunakan pencarian acak, ini terlihat pada proses pembangkitan populasi awal yang menyatakan sekumpulan solusi yang dipilih secara acak.
- Berikutnya pencarian dilakukan berdasarkan proses-proses dari teori genetika yang memperhatikan pemikiran bagaimana memperoleh individu yang lebih baik, sehingga dalam proses evolusi dapat diharapkan diperoleh individu yang terbaik.

Daftar Pustaka

References

Abdy, M., Wahyuni, M. S., & Ilmi, N. (2017). Algoritma Genetika Dan Penerapannya dalam Mencari Akar Persamaan Polinomial. *SAINTIFIK*, *2*(1), 1-8.

https://www.geeksforgeeks.org/genetic-algorithms/

Lampiran

Link video presentasi :
https://youtu.be/XMZeVwyUg5w

Link Google Colab :

https://colab.research.google.com/drive/1obXRKB_vURSH-cG7Xy8v5bRz7RQsiIdt?usp=sharing