

Laporan Tugas Pemrograman 01 – Searching CII-2M3
Pengantar Kecerdasan Buatan
Semester Genap 2021/2022

Disusun Oleh:
Akmal Muhamad Firdaus-1301204188
Andre Eka Putra Simanjuntak-1301204173



PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
UNIVERSITAS TELKOM
BANDUNG
2022

1. PENDAHULUAN

Algoritma genetika adalah sebuah algoritma untuk melakukan suatu pencarian solusi dari suatu masalah yang didasarkan pada mekanisme seleksi alamiah. Oleh karena itu, algoritma genetika merupakan salah satu metode heuristik.

Konsep dari algoritma genetika ini dikembangkan pertama kali oleh John Holland dari New York, Amerika Serikat yang dipublikasikan dalam bukunya yang berjudul "*Adaptation in Natural and Artificial Systems*" tahun 1975. Solusi yang dihasilkan dari algoritma genetika bukanlah solusi yang terbaik, melainkan suatu solusi yang paling mendekati optimal. Implementasi algoritma genetika sering ditemukan dalam berbagai masalah yang memerlukan solusi kombinatorik seperti penjadwalan, peramalan, penentuan rute terpendek dan lain sebagainya.

1.1. Permasalahan

Diberikan suatu fungsi $h(x, y) = \frac{(\cos x + \sin y)^2}{x^2 + y^2}$ yang dimana akan dicari nilai minimum dari fungsi tersebut dengan domain $-5 \leq x \leq 5$ dan $-5 \leq y \leq 5$ menggunakan Genetic Algorithm.

1.2. Rumusan Masalah

- Mencari nilai minimum dari fungsi.
- Menentukan fungsi pada kode program yang akan digunakan.
- Menentukan metode pada setiap fungsi.

1.3. Tujuan

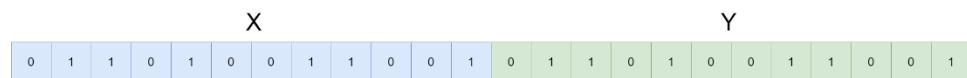
- Memahami aplikasi/implementasi dari algoritma genetika.
- Memahami konsep pencarian solusi dari algoritma genetika dan dapat memanfaatkannya untuk pemecahan masalah yang lainnya.

2. METODOLOGI

2.1. Desain Kromosom dan Metode Dekode-nya

2.1.1. Desain Kromosom

Untuk dapat memproses data pada algoritma genetika, diperlukan sesuatu yang merepresentasikan satu atau lebih kromosom. Dalam pengimplementasiannya, kami menggunakan satu individu yang membawa satu gnome. Gnome ini membawa satu kromosom yang terdiri dari 2 genotype yang merepresentasikan nilai X dan Y. Dalam 1 genotype terdiri dari 12 gen. Maka dari itu dalam 1 kromosom terdapat 24 gen / bit biner.



(Ilustrasi kromosom biner dengan panjang genotpe 12 gen)

2.1.2. Metode Dekode-nya

Seperti yang telah disinggung pada pembahasan sebelumnya, kami menggunakan bilangan biner(0 dan 1) untuk merepresentasikan suatu informasi individu. Sehingga dalam praktiknya kami menggunakan struktur *data list* pada python dengan panjang 24 (Index 0 s.d 23) untuk menampung data genotype. Maka dari itu struktur data list tersebut yang terdiri dari 24 index tersebut di isi oleh angka 0 s.d 1 dengan dilakukan pemanggilan *library random* menggunakan *choices* untuk mengisi ke 24 bit kromosom tersebut.

```
bitX = random.choices([0,1], k=length)
bitY = random.choices([0,1], k=length)
```

Skema *binary decoding*, dimana sebuah genotype yang terdapat pada suatu kromosom dapat diterjemahkan kedalam bilangan real dengan batas tertentu yang nantinya digunakan sebagai nilai *fenotype* untuk dimasukkan ke dalam *fitness function* menggunakan rumus berikut:

$$x = r_b + \frac{(r_a - r_b)}{\sum_{i=1}^N 2^{-i}} (g_1 \cdot 2^{-1} + g_2 \cdot 2^{-2} + \dots + g_N \cdot 2^{-N})$$

```
def biner_decoding(upper_range, lower_range, g):
    tp = [2**-i for i in range(1, len(g) + 1)]
    atas = lower_range + ((upper_range-lower_range)
    bawah = sum(tp) * sum([g[i] * tp[i] for i in range(len(g))]))
    return atas / bawah
```

Berdasarkan rumusan masalah yang diberikan, *fitness function* yang digunakan adalah meminimasi fungsinya, maka dari itu didapatlah rumus *fitness function* sebagai berikut :

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

```
def f(x, y):
    #fungsi untuk minimasi
    a = random.uniform(0, 0.000000000000000009)
    return 1 / (h(x,y) + a)
```

2.2. Ukuran popoulasi

Pada algoritma genetika, diperlukan sekumpulan kromosom yang disebut sebagai populasi. Untuk menampung sekumpulan kromosom/individu, kami menggunakan data *list python* dengan panjang n (Index $0 \dots n-1$). Pada praktik ini, ukuran maksimum populasi untuk tiap generasi sejumlah 20 individu. Untuk mengisi *list* populasi tersebut, kami menggunakan pemanggilan fungsi membentuk individu sebanyak 20 atau n kali perulangan. Setelah itu populasi diurutkan berdasarkan nilai *fitness* masing-masing individu.

```
def create_population(pop):
    while len(populasi) < pop:
        populasi.append(generate_cromosome(length_gen))
    populasi.sort(key=sortFitness, reverse=True)
    return populasi
```

2.3. Metode Pemilihan Orangtua

Kami menggunakan metode *Tournament Selection* dalam pemilihan individu yang akan menjadi *parent*. Metode ini dilakukan dengan cara melakukan pengambilan N buah sampel dari populasi dan dilakukan pengurutan dari *fitness* tertinggi ke *fitness* terendah (*descending*), lalu diambil 2 *parent* dengan *fitness* terbaik.

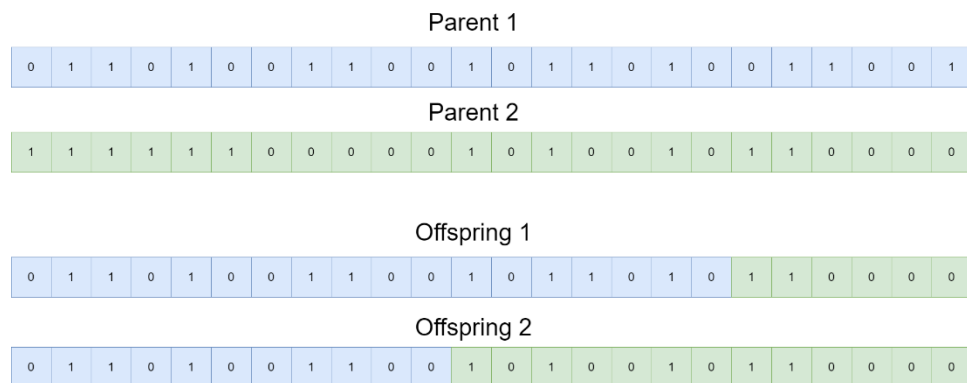
```
def parent_selection(populasi, parent_sample):
    sample1 = random.choices(populasi, k=parent_sample)
    sample1.sort(key=sortFitness, reverse=True)

    sample2 = random.choices(populasi, k=parent_sample)
    sample2.sort(key=sortFitness, reverse=True)
    return sample1[0], sample2[0]
```

2.4. Metode operasi genetik

2.4.1. Crossover

Setelah dilakukan pemilihan 2 individu yang dijadikan sebagai orang tua, kami melakukan *crossover* (pindah silang) terhadap ke-2 individu tersebut. Pada proses *crossover* terjadi kombinasi pewarisan gen antar induknya, sehingga dari hasil kombinasi orang tua tersebut didapat susunan kromosom baru yang lebih bervariasi. Pada proses ini kami menggunakan metode *random single point crossover* yaitu dengan cara memilih secara acak titik perpotongan untuk setiap pasangan parent dan akan saling menggabungkan gennya sehingga didapat 2 offspring(anak) dengan kromosom hasil warisan gen orang tuanya.



(Ilustrasi random single point crossover)

```
def crossover(parent1, parent2):
    point = random.randint(1, 2*length_gen-1)
    bit_child1 = parent2["cromosome"][:point] + parent1["cromosome"][point:]
    bit_child2 = parent2["cromosome"][point:] + parent1["cromosome"][:point]

    child1 = dict(parent1)
    child2 = dict(parent2)

    len_genotype = len(bit_child1)//2
    child1["genX"] = bit_child1[:len_genotype]
    child1["fenX"] = biner_encoding(upper_x, lower_x, child1["genX"])
    child1["genY"] = bit_child1[len_genotype:]
    child1["fenY"] = biner_encoding(upper_y, lower_y, child1["genY"])
    child1["cromosome"] = child1["genX"] + child1["genY"]
    child1["fitness"] = f(child1["fenX"], child1["fenY"])

    child2["genX"] = bit_child2[:len_genotype]
    child2["fenX"] = biner_encoding(upper_x, lower_x, child2["genX"])
    child2["genY"] = bit_child2[len_genotype:]
    child2["fenY"] = biner_encoding(upper_y, lower_y, child2["genY"])
    child2["cromosome"] = child2["genX"] + child2["genY"]
    child2["fitness"] = f(child2["fenX"], child2["fenY"])

    return child1, child2
```

2.4.2. Mutasi

Mutasi diperlukan untuk mengembalikan informasi bit yang hilang akibat *crossover*. Mutasi diterapkan dengan probabilitas yang sangat kecil, karena jika terlalu sering dilakukan mutasi, maka akan menghasilkan populasi berisi kromosom yang lemah. Kromosom dikatakan lemah karena kromosom dengan susunan gen yang unggul atau optimal dari hasil *crossover* bisa saja rusak (dimutasi). Metode mutasi yang kami gunakan adalah mutasi pada tingkat gen, yang dimana semua bit dalam suatu gen akan berubah. Pada praktiknya, kami membuat variable global yang bernama “mutation_rate” sebesar 0.2 untuk dijadikan perbandingan dengan angka acak antara 0 s.d. 1. Karena kami menggunakan kromosom biner, apabila terjadi mutasi suatu gen, maka setiap gen yang berangka 0 akan berubah menjadi angka 1 dan juga sebaliknya.

```
def mutation(child, mutation_rate):
    offspring = dict(child)
    offspring["cromosome"] = list(child["cromosome"])
    for i in range(len(offspring["cromosome"])):
        if random.uniform(0,1) <= mutation_rate:
            if offspring["cromosome"][i] == 0:
                offspring["cromosome"][i] = 1
            else:
                offspring["cromosome"][i] = 0
    len_genotype = len(offspring["cromosome"]) // 2
    offspring["genX"] = offspring["cromosome"][:len_genotype]
    offspring["genY"] = offspring["cromosome"][len_genotype:]

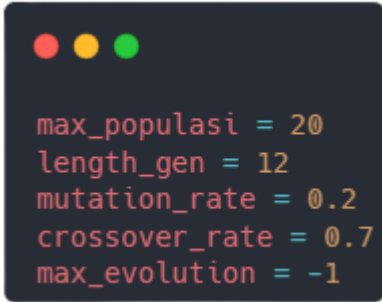
    offspring["fenX"] = biner_encoding(upper_x, lower_x, offspring["genX"])
    offspring["fenY"] = biner_encoding(upper_y, lower_y, offspring["genY"] )

    offspring["fitness"] = f(offspring["fenX"], offspring["fenY"])

    return offspring
```

2.5. Probabilitas operasi genetik

Dalam algoritma genetika, terdapat dua jenis operasi genetika, yaitu perkawinan dan mutasi. Sesuai dengan prinsip pada biologi yang dimana terdapat kemungkinan terjadinya perkawinan antar individu dan mutasi pada individu. Kemungkinan terjadinya perkawinan atau *crossover* pada *parent* sebesar 70%, sedangkan peluang terjadinya mutasi pada individu hasil *crossover* sebesar 10%.



```
max_populasi = 20
length_gen = 12
mutation_rate = 0.2
crossover_rate = 0.7
max_evolution = -1
```

2.6. Metode pergantian generasi

Setelah dilakukan penambahan kromosom baru hasil dari perkawinan antar dua *parent* dan individu hasil mutasi, maka diperlukan pergantian generasi pada populasi. Untuk menjaga kualitas individu yang diteruskan pada generasi selanjutnya, kami menggunakan metode *elitism*. Dengan membandingkan seluruh *children* dan populasi, metode ini mempertahankan individu dengan *fitness* yang mendekati tujuan akhir kami.



```
def regeneration(children, populasi):
    for i in children:
        if i["fitness"] >= populasi[-1]["fitness"]:
            populasi[-1] = i
    populasi.sort(key=sortFitness, reverse=True)
    return populasi
```


2.7. Kriteria penghentian evolusi

Proses penambahan generasi dan evolusi dilakukan berulang hingga prosesnya dihentikan. Pada algoritma genetika yang kami rancang, proses terminasi evolusi terbagi menjadi 2 opsi yaitu, yang pertama adalah jika variabel "max_evolution" bernilai -1, maka terminasi akan dilakukan ketika nilai fitness dari semua individu pada populasi bernilai konstan, dan untuk yang kedua yaitu ditentukan berdasarkan maksimal evolusi yang terjadi. Misalkan variabel "max_evolution" bernilai 1000, maka evolusi akan berhenti digenerasi ke-1000.

```
def evaluation(population, generasi):  
    if len(population) > 0 and max_evolution == -1:  
        return all(round(x["fitness"]) == round(population[0]["fitness"]) for x in population)  
    elif max_evolution <= generasi:  
        return True  
    else:  
        return False
```

3. HASIL DAN PEMBAHASAN

Berikut ini adalah *screenshot* dari hasil *running code* algoritma genetika yang kami rancang. *Output* mengeluarkan kromosom dengan nilai *fitness* paling optimum dengan data detail sebagai berikut.

Generasi : 6164

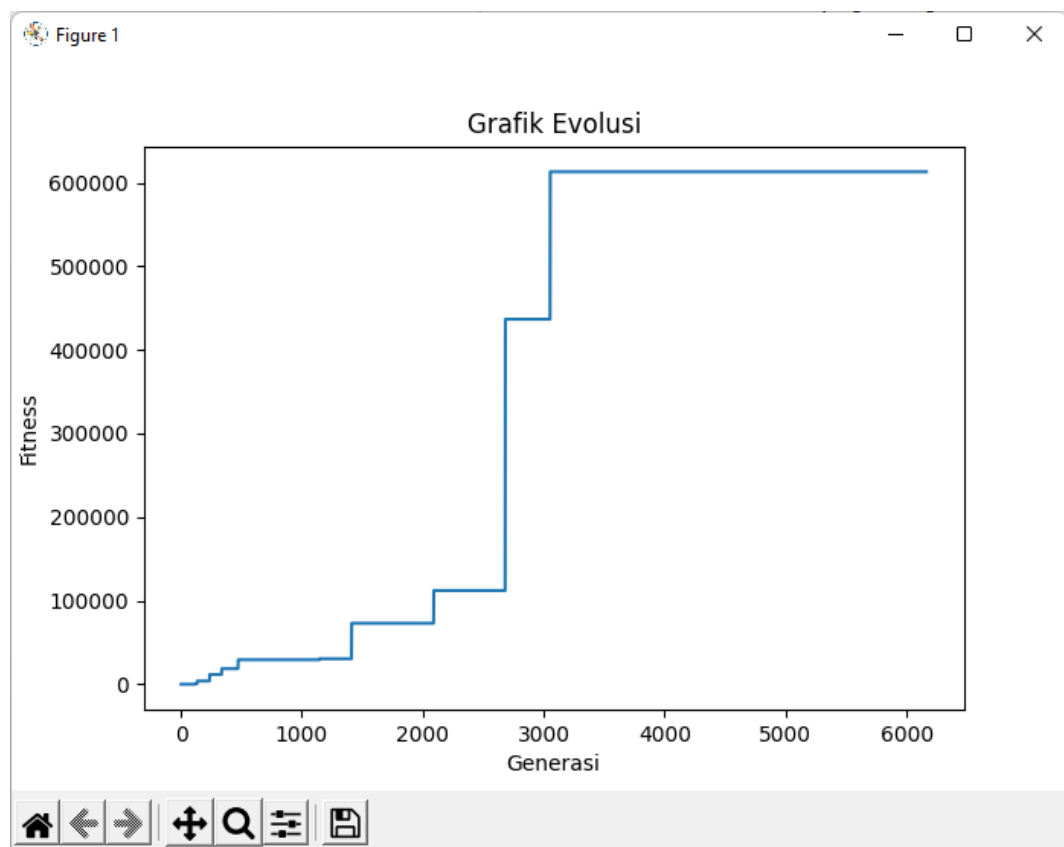
Best Fitness : $61.30394348638011 \times 10^4$

Best X : -1.5714285714285716

Best Y : 0.0012210012210012167

Cromosome : [0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

```
Generasi      : 6164
Best Fitness  : 61.30394348638011 x 10^4
Best X        : -1.5714285714285716
Best Y        : 0.0012210012210012167
Cromosome     : [0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```



Pada hasil uji coba tersebut, kami menggunakan parameter yang sudah disebutkan pada sub-bab sebelumnya. Menurut kami, parameter tersebut sudah optimum untuk digunakan dalam algoritma genetika ini.

4. SIMPULAN

Dari percobaan yang telah kami lakukan, kami dapat mengambil kesimpulan bahwa algoritma genetika yang kami desain dapat menemukan hasil yang mendekati nilai optimum dengan total generasi yang berbeda pada setiap percobaannya, tergantung pada nilai random pada algoritma untuk menemukan nilai *fitness* yang konstan pada tiap individu.

Video Presentasi : <https://www.youtube.com/watch?v=uuG2T02qTWk>

Peran anggota :

- Akmal Muhamad Firdaus (1301204188) :
 - Desain kromosom dan decode kromosom
 - Membuat fungsi Generate kromosom
 - Membuat fungsi crossover
 - Membuat fungsi evolution
 - Membuat fungsi logging
 - Membuat fungsi sorting fitness
 - Membuat laporan
- Andre Eka Putra Simanjuntak(1301204173):
 - Membuat fungsi Parent selection
 - Membuat fungsi mutation
 - Membuat fungsi create population
 - Membuat fungsi regeneration
 - Membuat laporan
 - Membuat PPT