ANALISIS, DESAIN, DAN IMPLEMENTASI ALGORITMA GENETIKA UNTUK MENYELESAIKAN MASALAH MAKSIMASI FUNGSI

Azriel Naufal Aulia [1], Muhammad Faiz Abdurrahman Djauhar [2], Ryan Abdurohman [3]
[1]1301190374, IF-43-10, S-1 Informatika,Telkom University
[2]1301190361, IF-43-10, S-1 Informatika,Telkom University
[3]1301191171, IF-43-10, S-1 Informatika,Telkom University

1. PENDAHULUAN

Algoritma genetika adalah algoritma pencarian solusi dari suatu masalah yang didasarkan pada mekanisme seleksi alamiah. Oleh karena itu, algoritma genetika merupakan salah satu metode heuristik, solusi yang dihasilkan dari algoritma genetika bukanlah yang terbaik, melainkan solusi yang paling mendekati optimal. Implementasi algoritma genetika sering ditemukan dalam berbagai masalah, seperti membuat solusi yang optimum, proses *machine learning,* peramalan, pemrograman otomatis, dan masalah lainnya.

Konsep algoritma genetika terinspirasi dari teori evolusi Darwin yang terkenal, yaitu konsep yang berbunyi *"the strongest species that survive"*. Yang dimana teori ini menjelaskan dari setiap seleksi alam, organisme dapat bertahan hidup dengan beberapa proses seperti, reproduksi *(crossover)*, mutasi, Seleksi survivor dan lainnya. Konsep tersebut kemudian diadaptasi pada algoritma komputasi untuk menemukan solusi yang paling optimum pada setiap masalah suatu fungsi objektif (Riady et al., 2020).

Selain untuk memperoleh nilai terbaik untuk tugas pemrograman 01 pada mata kuliah pengantar kecerdasan buatan, tujuan kami adalah untuk :

- Memahami aplikasi/implementasi dari algoritma genetika
- Lakukan analisis, desain, dan implementasi algoritma *Genetic algorithm* (GA) ke dalam suatu program komputer untuk menemukan nilai tertinggi dari fungsi:

$$h(x,y) = (\cos(x^2) \times \sin(y^2)) + (x+y)$$
dengan Batasan $-1 \le x \le 2$ dan $-1 \le y \le 1$.

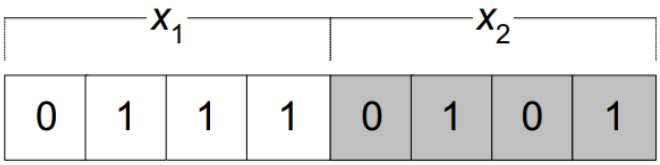
- Mengerti konsep pencarian solusi dari algoritma genetika dan dapat memanfaatkannya untuk pemecahan masalah yang lainnya.

2. METODOLOGI

2.1 Desain Kromosom dan Metode Pendekodean
2.1.1 Desain Kromosom
Untuk dapat memproses data pada algoritma genetika, diperlukan suatu individu yang merepresentasikan satu atau lebih kromosom (Suyanto, 2019). Walaupun memang tidak benar-benar seperti di dalam dunia biologi yang direpresentasikan oleh basa A, C, T, G. Dalam pengimplementasiannya, kami menggunakan satu individu yang membawa satu genome. Gnome ini membawa satu kromosom yang terdiri dari dua gen. Gen ini digunakan untuk merepresentasikan nilai X dan Y sebagai variabel pada fungsi. Masing-masing gen memiliki 10 *bases*. Sehingga dalam praktiknya, kami menggunakan struktur data list pada python dengan panjang list 20 untuk setiap informasi genotip kromosom. Index list 0 s.d. 9 untuk merepresentasikan gen X. Lalu, index list 10 s.d. 19 untuk merepresentasikan gen Y. Kami menggunakan teknik binary encoding sebagai *basis* untuk masing-masing gen.

2.1.2 Metode Pengkodean
Seperti telah disinggung sebelumnya, kami menggunakan bilangan biner yaitu 0 dan 1 untuk merepresentasikan informasi dari suatu individu. Dalam praktiknya, kami membuat sebuah list yang memiliki panjang 20. Lalu, setiap indeks nya diisi dengan nilai 0 atau 1. Pemilihan angka 0 atau 1 ini dilakukan dengan memanggil pustaka random dengan fungsi choice di python. Berikut screenshot implementasi kode untuk menginstansiasi gen.

```
def generate_gen(length):
    return choices([0,1], k = length)
```



Ilustrasi 1 : representasi kromosom biner dengan panjang 4 gen
Dengan panjang 20, kromosom/solusi dalam representasi biner yang kita buat, akan menghasilkan populasi dengan kromosom-kromosom yang lebih variatif, sehingga solusi yang dibuat memiliki ketelitian yang lebih tinggi. Setelah kita membuat representasi kromosom biner dengan panjang 10 gen, kita akan melakukan binary encoding. Supaya variabel representasi biner kita menjadi nilai fenotip, sehingga dapat kita cari nilai fitnessnya.

Skema binary encoding, dimana sebuah individu berisi N gen dapat dilakukan dengan rumus :

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

```
def biner_encoding(gen, min,max):
    pyb = 0
    lef = 0
    for i in range(1,len(gen)+1):
        pyb = pyb + 2**(-1*i)
        lef = lef + gen[i-1]*2**(-1*i)
    #-----<>------
    xy = min + (max - min)*(lef)/pyb
    return xy
```

Fungsi fitness berdasarkan soal :

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

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

```
def fitness_function(genX,genY):
    a = cos(genX**2)+sin(genY**2)+(genX+genY)
# -1 <= x <= 2 and -1 <= y <= 1
    return a
```

Setelah kita, membuat representasi biner, mencari nilai fenotip dengan formula binary encoding, dan mencari nilai fitness, kita membuat fungsi generate_kromosom untuk mengembalikan dictionary yang berisi informasi-informasi tentang kromosom.

```
def generate_kromosom(lengthGen):
  x = generate_gen(lengthGen)
  y = generate_gen(lengthGen)
  c = x + y
  d = fitness_function(biner_encoding(x,batasX[0],batasX[1]),biner_encoding(y,batasY[0],batasY[1]))# -1 <= x <= 2 and -1 <= y <= 1
  e = 0
  kromosom = {
      'genX' : x ,
      'fenX' : biner_encoding(x, batasX[0], batasX[1]),
      'genY' : y ,
      'fenY' : biner_encoding(y, batasY[0], batasY[1])
      'krom' : c ,
      'fitVal' : d
      'prob' : e
  }
  return kromosom
```

2.2 Ukuran Populasi

```
def generate_populasi(maxPop,lenGen):
    Lkrom = []
    for _ in range(maxPop):
        a = generate_kromosom(lenGen)
        Lkrom.append(a)

    data = evaluateFitness(Lkrom)

    populasi = {
        'kumpulan_kromosom' : Lkrom ,
        'bestKromosom' : data
    }
    """
    return dictionary populasi berisi :
    1. list kromosom yang berisi dictionary gen
    2. bestkromosom
    """
    return populasi
```

Dalam fungsi generate_populasi, kita akan membuat dictionary kromosom sebanyak maxPop (pada parameter) dengan memanggil fungsi generate_kromosom. Pada parameter generate_kromosom kita akan mengisi panjang gen yaitu lenGen (didapat dari parameter fungsi). Setelah itu juga kita akan memanggil fungsi evaluatefitness untuk mencari kromosom dengan nilai fitness tertinggi diantara kromosom lainnya dalam satu populasi. Setelah selesai membuat satu populasi kromosom sebanyak maxPop dan mendapatkan kromosom dengan nilai fitness tertinggi, kita akan menyimpannya dalam satu dictionary bernama populasi.

```
def evaluateFitness(listGen):
    sem = 0
    for i in range(len(listGen)):
        if sem < listGen[i]['fitVal']:
            sem = listGen[i]['fitVal']
            indeks = i

    bestkromosom = {
        'kromosom' : listGen[indeks]['krom'] ,
        'Fitness_value' : listGen[indeks]['fitVal']
    }
    return bestkromosom
```

Berdasarkan fungsi generate_kromosom di atas, di dalam dictionary kromosom yang dikembalikan, terdapat informasi tentang probabilitas yang disimpan didalam variabel prob. Disini nilai probabilitas digunakan nantinya untuk pemilihan 2 kromosom sebagai orang tua.

```
[9] def normalization_fitness(pop):
      fit = []
      fit_total = 0
      probability = []
      for i in pop['kumpulan_kromosom']:
        fit.append(i['fitVal']+10)
        fit_total = fit_total + (i['fitVal']+10)

      for j in range(len(fit)):
        probability.append(fit[j]/fit_total)

      return probability
```

Pada fungsi normalization_fitness, nilai probabilitas dihitung dengan cara menambahkan 10 seluruh nilai fitness masing-masing kromosom dan membaginya dengan nilai fitness total seluruh kromosom. Alasan mengapa ditambahkan 10 untuk masing-masing nilai fitness suatu kromosom antara lain untuk menghindari kemungkinan untuk nilai probabilitas memiliki nilai negatif, dari nilai fitnessnya yang negatif. Dari cara yang kita pakai ini, semakin besar nilai fitness suatu kromosom maka akan semakin besar pula nilai probabilitas yang diampu.

2.3 Pemilihan orangtua : roulette wheel versi modifikasi

```
[10] def parents(pop):
       seed()
       r = uniform(0, 1)
       index = 0
       while r > 0 and index < maks_populasi:
         r -= pop["kumpulan_kromosom"][index]["prob"]
         index+=1
       index-=1
       return pop["kumpulan_kromosom"][index], index
```

Pada percobaan kami, proses pemilihan orangtua dilakukan dengan metode roulette wheel yang dimodifikasi didalam fungi parents. Algoritma ini terinspirasi dari video youtube Daniel Shiffman dalam kanal The Coding Train (Shiffman, 2017). Pada fungsi ini, kita akan membuat angka random antara 0 sampai 1, yang mana angka tersebut digunakan untuk dibandingkan dengan nilai probabilitas dari setiap kromosom. Orang tua akan dipilih ketika nilai angka random tersebut bernilai negatif karena pengurangan terhadap nilai probabilitas kromosom dari setiap iterasi. Kromosom orang tua dipilih berdasarkan indeks terakhir tepat sebelum nilai random tersebut bernilai negatif.

2.4 Pemilihan dan teknik operasi genetik (crossover dan mutasi)
2.4.1 Crossover

```
[11] def crossover (laki, perempuan):
       singlepoint = randint(1, 2*panjang_gen-1)
       anak1 = []
       anak2 = []
       anak1 = laki["krom"][:singlepoint] + perempuan["krom"][singlepoint:]
       anak2 = laki["krom"][singlepoint:] + perempuan["krom"][:singlepoint]
       return anak1, anak2
```

Dengan fungsi parents, 2 individu sudah terpilih menjadi orang tua. Setelah didapatkan 2 individu sebagai orang tua, selanjutnya kita lakukan proses pindah silang *(crossover)*. Pada proses *crossover* terjadi kombinasi pewarisan gen-gen dari kromosom induknya. Sehingga dari hasil kombinasi gen-gen orang tua, didapat susunan kromosom yang baru dan lebih bervariasi. Pada fungsi crossover ini, kita menggunakan metode pindah silang satu titik *(single-point crossover)*. Dengan metode *single point crossover*, kita akan memilih satu titik secara acak untuk setiap pasangan orang tua, dan akan saling menggabungkan gennya, sehingga didapat 2 anak kromosom hasil warisan gen-gen pada kromosom orang tua.

2.4.2 Mutasi : pada tingkat bit

```
[12] def mutasi (krom):
       hasil = krom
       for i in range(2*panjang_gen):
         if uniform(0,1) < rate_mutasi:
           if hasil[i] == 0:  hasil[i] = 1
           else:  hasil[i] = 0
       return hasil
```

Mutasi diperlukan untuk mengembalikan informasi bit yang hilang akibat *crossover*. Mutasi diterapkan dengan probabilitas yang sangat kecil, karena jika mutasi terjadi terlalu sering, maka akan menghasilkan populasi yang berisi kromosom yang lemah (Suyanto, 2019). Kromosom dikatakan lemah karena kromosom dengan susunan gen yang unggul atau optimal memiliki kemungkinan untuk dirusak (dimutasi). Metode mutasi yang kami gunakan adalah mutasi pada tingkat gen, yang dimana semua bit dalam suatu gen akan berubah. Pada percobaan kami, kami membuat variabel global yaitu rate_mutasi sebesar 0.01 untuk dijadikan perbandingan dengan hasil angka random antara 0 sampai 1. Karena kami menggunakan representasi kromosom dengan biner, apabila terjadi kemungkinan suatu gen untuk bermutasi maka setiap gen 0 akan menjadi gen 1, dan juga sebaliknya.

2.5 Probabilitas operasi genetik (Pc dan Pm)

```
[ ]  maks_populasi = 150
     rate_mutasi = 0.01
     rate_crossover = 0.7
     batasX = [-1, 2]
     batasY = [-1, 1]
     panjang_gen = 10
     max_loop = 1000
     populasi = {}
```

Diatas merupakan variabel global dalam percobaan kami. Untuk probabilitas operasi genetik kami isi sebesar 0.01 untuk Pm (peluang terjadinya mutasi) dan 0.7 untuk Pc (peluang terjadinya crossover untuk pasangan orangtua).

2.6 Metode Pergantian Generasi (Seleksi Survivor)

```
[ ]  def survivorSelection(maks_populasi, bestKromosom, populasi):
       i = 0
       minimum = 0
       minIdx = 0

       while i < maks_populasi:
         suami, index = parents(populasi)
         temp = populasi["kumpulan_kromosom"][index]["prob"]
         populasi["kumpulan_kromosom"][index]["prob"] = 0
         istri, _ = parents(populasi)
         populasi["kumpulan_kromosom"][index]["prob"] = temp
         if uniform(0, 1) < rate_crossover:
           krAnak1, krAnak2 = crossover(suami, istri)
           krAnak1 = mutasi(krAnak1)
           krAnak2 = mutasi(krAnak2)
           anak1 = create_kromosom(krAnak1[:panjang_gen], krAnak1[panjang_gen:])
           anak2 = create_kromosom(krAnak2[:panjang_gen], krAnak2[panjang_gen:])
           populasi["kumpulan_kromosom"][i] = anak1
           populasi["kumpulan_kromosom"][i+1] = anak2
         if populasi["kumpulan_kromosom"][i]["fitVal"] < minimum:
           minimum = populasi["kumpulan_kromosom"][i]["fitVal"]
           minIdx = i
         if populasi["kumpulan_kromosom"][i+1]["fitVal"] < minimum:
           minimum = populasi["kumpulan_kromosom"][i+1]["fitVal"]
           minIdx = i+1
         i+=2
       populasi["kumpulan_kromosom"][minIdx] = bestKromosom
       return populasi
```

Kami menggunakan metode generational replacement sebagai metode pergantian generasi. Jadi kami akan mengganti seluruh populasi dengan individu baru. Namun, kami juga memakai prinsip elitisme, yaitu di setiap pergantian generasi, kami menganti individu yang memiliki nilai fitness terkecil dengan individu yang memiliki nilai fitness terbesar di generasi yang baru.

2.7 Kriteria Penghentian Evolusi

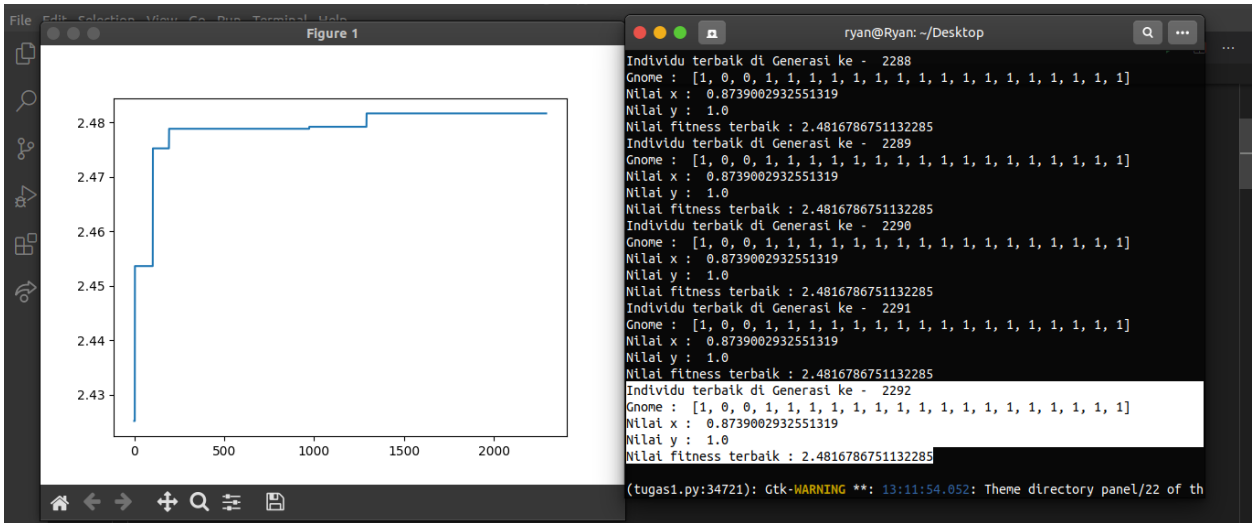
```
[ ]  maks_populasi = 150
     rate_mutasi = 0.01
     rate_crossover = 0.7
     batasX = [-1, 2]
     batasY = [-1, 1]
     panjang_gen = 10
     max_loop = 1000
     populasi = {}
```

Kriteria penghentian evolusi kami adalah ketika selama max_loop generasi, nilai fitness untuk best kromosomnya (kromosom dengan nilai fitness tertinggi) memiliki nilai yang konstan. Apabila nilai max_loop tersebut sebesar 1000, maka evolusi dari pembuatan generasi kromosom yang baru akan berakhir ketika sudah memiliki 1000 generasi baru dengan nilai konstan.

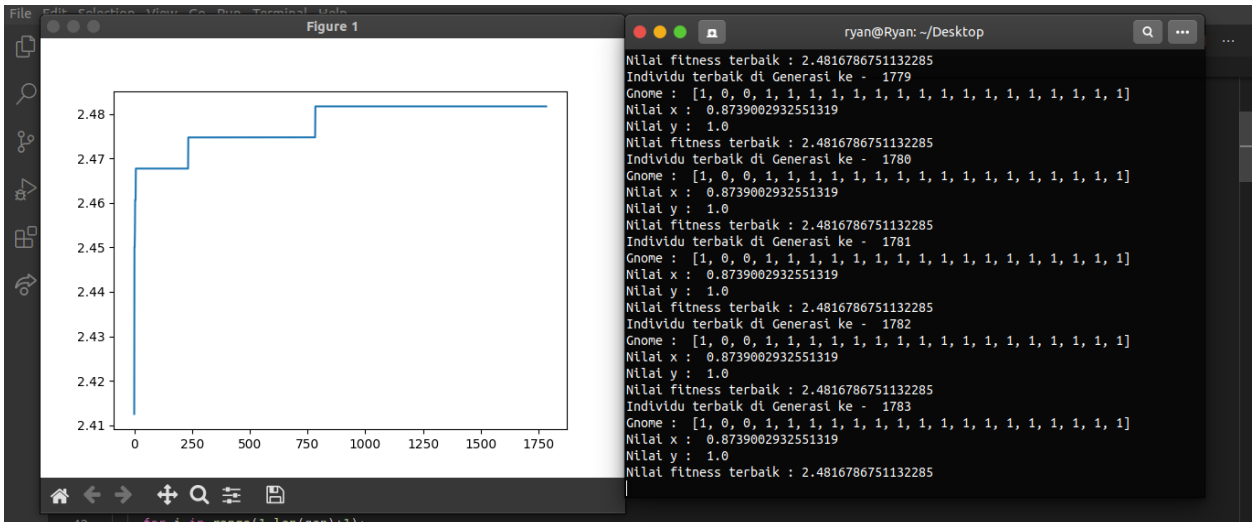
3. HASIL DAN PEMBAHASAN

Berikut screenshot dari hasil running algoritma genetika yang kita rancang. Hasil output mengeluarkan Kromosom dengan nilai fitness tertinggi, nilai x, nilai y dan nilai fitnessnya untuk setiap generasi.

Individu terbaik di Generasi ke - 2292
Gnome : [1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Nilai x : 0.8739002932551319
Nilai y : 1.0
Nilai fitness terbaik : 2.4816786751132285



Individu terbaik di Generasi ke - 1783
Gnome : [1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Nilai x : 0.8739002932551319
Nilai y : 1.0
Nilai fitness terbaik : 2.4816786751132285

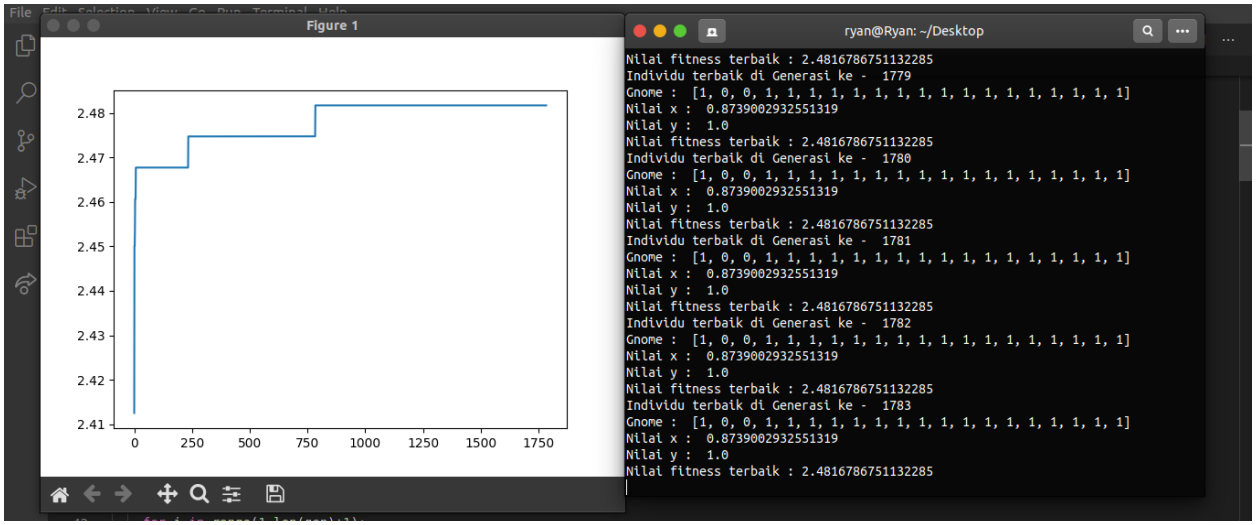


Pada hasil uji coba di atas, kami menggunakan parameter-parameter yang sudah disebutkan di sub bab sebelumnya. Menurut kami, parameter tersebut sudah optimum untuk digunakan.

4. SIMPULAN

Dari percobaan yang telah kami lakukan, kami dapat mengambil kesimpulan bahwa dengan menggunakan algoritma genetika, kami memperoleh :

Nilai x : 0.8739002932551319
Nilai y : 1.0
Nilai fitness terbaik : 2.4816786751132285



Secara rata-rata, hasil terbaik pasti akan didapat pada lebih dari generasi sebanyak max_loop (variabel yang kami gunakan untuk menentukan seberapa konstan nilai fitness di sejumlah generasi). Pada kasus ini, kami menggunakan parameter sebanyak 1000. Artinya, nilai fitness terbaik yang didapat haruslah bertahan di 1000 generasi.

5. REFERENSI

Riady, A., Alvin Setiabudi, C., Andrian Nugroho, F., & Willianto, T. (2020). *Optimasi dengan Genetic Algorithm Menggunakan Python*. School of Computer Science, Binus University. Retrieved 26 March 2021, from https://socs.binus.ac.id/2020/07/29/optimasi-dengan-genetic-algorithm-menggunakan-python/

Suyanto, S. (2014). *Artificial Intelligence: Searching-Reasoning-Planning-Learning (Revisi Kedua)*. Penerbit Informatika.

Shiffman, D. (2017). *9.8: Genetic Algorithm: Improved Pool Selection - The Nature of Code* [Video]. Retrieved 14 March 2021, from https://www.youtube.com/watch?v=ETphJASzYes&list=PLRqwX-V7Uu6bJM3VgzjNV5YxVxUwzALHV&index=8&t=928s&ab_channel=TheCodingTrain.

Shiffman, D. (2016). *Playlist 9: Genetic Algorithms - The Nature of Code* [Video]. Retrieved 14 March 2021, from https://youtube.com/playlist?list=PLRqwX-V7Uu6bJM3VgzjNV5YxVxUwzALHV.

Shiffman, D. (2012). *The nature of code*. [D. Shiffman]. https://natureofcode.com/book/

6. VIDEO PRESENTASI

Ryan Abdurohman : https://youtu.be/byxD2Hci9Dc
Azriel Naufal Aulia : https://youtu.be/yFKBSa2TXI0
Muhammad Faiz Abdurrahman Djauhar : https://youtu.be/iJj5gYFiWxk