# MODUL PRAKTIKUM KECERDASAN BUATAN

## PERTEMUAN 7 – ALGORITMA GENETIKA

Tools : Jupyter notebook
Bahasa Pemrograman : Python

Dalam praktikum kali ini, kita akan belajar tentang bagaimana cara menerapkan algoritma genetika menggunakan Python.

### Algoritma Genetika

Algoritma genetika adalah algoritma pencarian heuristik adaptif yang termasuk dalam algoritma evolusioner. Algoritma genetika didasarkan pada gagasan seleksi alam dan genetika. Algoritma ini diperkenalkan oleh John Holland. Algoritma ini biasanya digunakan untuk menghasilkan solusi berkualitas tinggi untuk masalah pengoptimalan dan masalah pencarian.

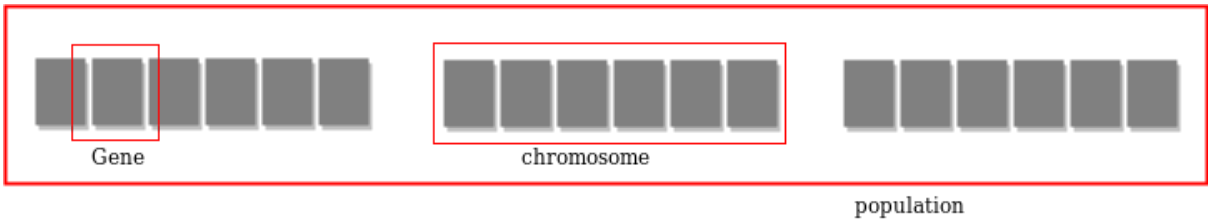
Algoritma genetika mensimulasikan proses seleksi alam yang berarti spesies yang dapat beradaptasi dengan perubahan lingkungannya mampu bertahan dan bereproduksi serta melanjutkan ke generasi berikutnya. Dengan kata sederhana, mereka mensimulasikan "*survival of the fittest*" di antara individu generasi berturut-turut untuk memecahkan masalah. Setiap generasi terdiri dari populasi individu dan setiap individu mewakili suatu titik dalam ruang pencarian dan solusi yang mungkin. Setiap individu direpresentasikan sebagai string dari character/integer/float/bit.

Algoritma genetika didasarkan pada analogi berikut:

1. Individu dalam populasi bersaing untuk mendapatkan sumber daya dan pasangan
2. Individu-individu yang sukses (paling kuat) kemudian disilangkan untuk menghasilkan lebih banyak keturunan daripada yang lain
3. Gen dari orang tua (*parent*) yang “paling kuat” menyebar sepanjang generasi, terkadang orang tua menghasilkan keturunan yang lebih baik daripada salah satu orang tuanya
4. Setiap generasi secara berturut-turut lebih cocok untuk lingkungan mereka.

### Populasi, Kromosom, Gen

**Populasi** adalah himpunan bagian dari semua solusi yang mungkin untuk masalah yang diberikan. Dengan cara lain, kita dapat mengatakan bahwa **populasi** adalah sekumpulan **kromosom**. **Kromoso**m adalah salah satu solusi untuk masalah saat ini. Dan setiap **kromoso**m adalah satu set **gen**. Dengan kata lain, kita dapat menggambarkan kromosom sebagai string. Jadi, kita dapat mengatakan bahwa populasi adalah kumpulan dari beberapa string. Dan setiap karakter dari string adalah gen. Adapun gambaran dari populasui, kromososm, dan gen dapat dilihat pada Gambar 7.1.




Gambar 7.1. Gambaran dari populasi, kromosom, dan gen (https://www.geeksforgeeks.org/genetic-algorithms/)

Untuk memulai algoritma genetika, pertama-tama kita perlu menginisialisasi populasi. Kita dapat menginisialisasi populasi dengan dua cara, yaitu: inisialisasi secara acak (random) atau inisialisasi secara heuristik.

## Fungsi Fitness

Setelah menginisialisasi populasi, kita perlu menghitung nilai *fitness* dari kromosom tersebut. Nilai *fitness* diberikan kepada setiap individu yang menunjukkan kemampuan individu (kromosom) untuk "berkompetisi". Individu yang memiliki nilai *fitness* optimal (atau mendekati optimal) itulah yang dicari.

Algoritma genetika mempertahankan populasi dan individu (kromosom) beserta dengan nilai *fitness* mereka. Individu (kromosom) yang memiliki nilai *fitness* lebih baik diberi lebih banyak kesempatan untuk bereproduksi daripada yang lain. Individu (kromosom) dengan nilai *fitness* yang lebih baik dipilih untuk disilangkan sehingga menghasilkan keturunan yang lebih baik dengan menggabungkan kromosom orang tua. Ukuran populasi bersifat statis sehingga ruangan harus dibuat untuk pendatang baru. Jadi, beberapa individu akan mati dan digantikan oleh pendatang baru yang pada akhirnya menciptakan generasi baru ketika semua kesempatan untuk persilangan dari populasi lama habis. Diharapkan dari generasi ke generasi solusi yang lebih baik akan diperoleh sementara yang paling tidak cocok akan mati.

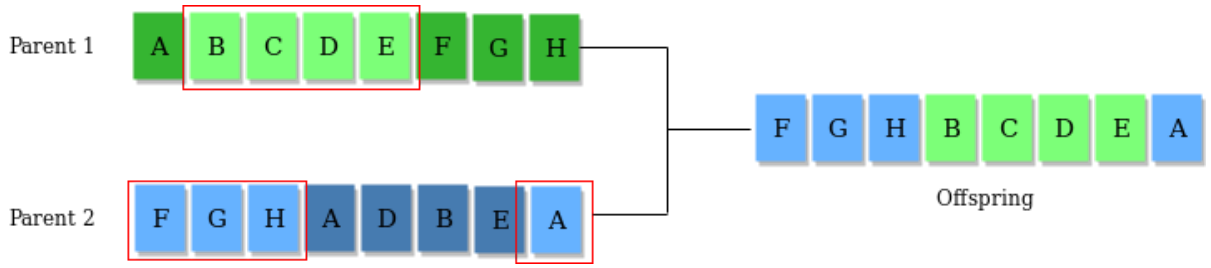
Setiap generasi baru rata-rata memiliki lebih banyak "gen yang lebih baik" daripada individu (kromosom/ solusi) generasi sebelumnya. Dengan demikian setiap generasi baru memiliki "solusi parsial" yang lebih baik daripada generasi sebelumnya. Setelah keturunan yang dihasilkan tidak memiliki perbedaan yang signifikan dari keturunan yang dihasilkan oleh populasi sebelumnya, maka populasi tersebut dinyatakan konvergen.

## Pemilihan Induk (*Parent Selection*)

Pemilihan induk dilakukan dengan menggunakan nilai *fitness* dari kromosom yang dihitung dengan fungsi *fitness*. Berdasarkan nilai *fitness* tersebut, kita perlu memilih pasangan kromosom dengan nilai *fitness* tertinggi.

## Persilangan (*Crossover*)

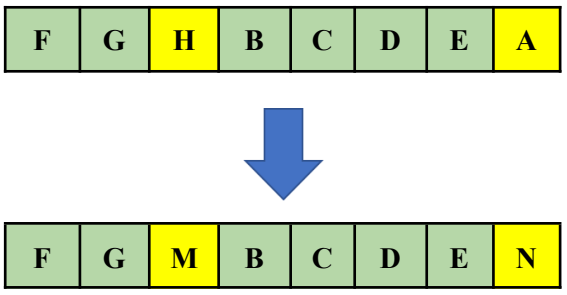
Pemilihan induk dilakukan dengan menggunakan nilai *fitness* dari kromosom yang dihitung dengan fungsi *fitness*. Berdasarkan nilai *fitness* tersebut, kita perlu memilih pasangan kromosom dengan nilai *fitness* tertinggi. Gambaran dari proses persilangan dapat dilihat pada Gambar 7.2.



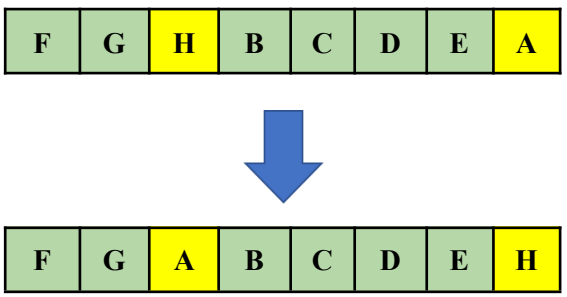

Gambar 7.2. Gambaran dari proses persilangan (https://www.geeksforgeeks.org/genetic-algorithms/)

## Mutasi (*Mutation*)

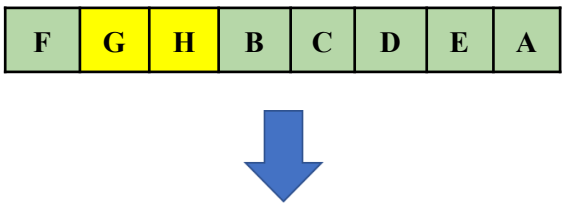
Mutasi membawa keragaman pada populasi. Ada berbagai jenis mutasi seperti mutasi Bit Flip, mutasi Swap, mutasi Inversi, dll. Adapun proses mutasi dapat dilihat pada Gambar 7.3.




(a)




(b)



| F | H | G | B | C | D | E | A |
|---|---|---|---|---|---|---|---|

(c)

Gambar 7.3. Proses mutasi (a) Mutasi Bit Flip, (b) Mutasi Swap, dan (c) Mutasi Inversi

## Penerapan Algoritma Genetika

Pada sesi kali ini, kita akan menerapkan algoritma genetika menggunakan Python. Pada sesi ini, diberikan sebuah string target dengan tujuan untuk menghasilkan string target mulai dari string acak dengan panjang yang sama sehingga analogi berikut dibuat:

a. Karakter A-Z, a-z, 0-9, dan simbol khusus lainnya dianggap sebagai gen
b. Sebuah string yang dihasilkan oleh karakter ini dianggap sebagai kromosom/solusi/Individu
c. Nilai *fitness* adalah jumlah karakter yang berbeda dari karakter dalam string target pada indeks tertentu. Jadi individu yang memiliki nilai fitness lebih rendah diberikan preferensi lebih.

Langkah-langkah yang perlu dilakukan antara lain:

1. Inisialisasi populasi secara acak (*random*)
2. Menentukan populasi dari *fitness*
3. Selama belum konvergen, maka perlu dilakukan tahapan berikut:
   a. Seleksi induk dari populasi
   b. Melakukan persilangan untuk menghasilkan generasi baru
   c. Melakukan mutasi pada populasi baru
   d. Menghitung nilai *fitness* untuk populasi baru

### Langkah 1: Inisialisasi populasi secara acak (random)

```
# Python3 program to create target string, starting from
# random string using Genetic Algorithm

import random

# Number of individuals in each generation
POPULATION_SIZE = 100

# Valid genes
GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP
QRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}'''

# Target string to be generated
TARGET = "I love Artificial Intelligence"
```

Pada percobaan kali ini, target hasil yang diharapkan adalah kalimat " I Love Artificial Intelligence". Adapun populasi yang diberikan terdiri atas 100 individu (kromosom/ solusi) dimana setiap individu mengandung gen A-Z, a-z, 0-9.

### Langkah 3 : Mendefinisikan kelas individu untuk melakukan seleksi, persilangan, mutasi, dan menghitung nilai *fitness*

```
class Individual(object):
    '''
    Class representing individual in population
    '''
    def __init__(self, chromosome):
        self.chromosome = chromosome
        self.fitness = self.cal_fitness()

    @classmethod
    def mutated_genes(self):
        '''
        create random genes for mutation
        '''
        global GENES
        gene = random.choice(GENES)
        return gene
```

```
    @classmethod
    def create_gnome(self):
        '''
        create chromosome or string of genes
        '''
        global TARGET
        gnome_len = len(TARGET)
        return [self.mutated_genes() for _ in range(gnome_len)]
```

```
    def mate(self, par2):
        '''
        Perform mating and produce new offspring
        '''

        # chromosome for offspring
        child_chromosome = []
        for gp1, gp2 in zip(self.chromosome, par2.chromosome):

            # random probability
            prob = random.random()

            # if prob is less than 0.45, insert gene
            # from parent 1
            if prob < 0.45:
                child_chromosome.append(gp1)

            # if prob is between 0.45 and 0.90, insert
            # gene from parent 2
            elif prob < 0.90:
                child_chromosome.append(gp2)

            # otherwise insert random gene(mutate),
            # for maintaining diversity
            else:
                child_chromosome.append(self.mutated_genes())

        # create new Individual(offspring) using
        # generated chromosome for offspring
        return Individual(child_chromosome)
```

```
    def cal_fitness(self):
        '''
        Calculate fitness score, it is the number of
        characters in string which differ from target
        string.
        '''
        global TARGET
        fitness = 0
        for gs, gt in zip(self.chromosome, TARGET):
            if gs != gt: fitness+= 1
        return fitness
```

## Langkah 2-3: Membuat fungsi utama untuk menentukan populasi dan *fitness* serta menerapkan langkah 3 hingga konvergen

```
# Driver code
def main():
    global POPULATION_SIZE

    #current generation
    generation = 1

    found = False
    population = []

    # create initial population
    for _ in range(POPULATION_SIZE):
                gnome = Individual.create_gnome()
                population.append(Individual(gnome))
```

```
    while not found:

        # sort the population in increasing order of fitness score
        population = sorted(population, key = lambda x:x.fitness)

        # if the individual having lowest fitness score ie.
        # 0 then we know that we have reached to the target
        # and break the loop
        if population[0].fitness <= 0:
            found = True
            break

        # Otherwise generate new offsprings for new generation
        new_generation = []

        # Perform Elitism, that mean 10% of fittest population
        # goes to the next generation
        s = int((10*POPULATION_SIZE)/100)
        new_generation.extend(population[:s])
```

```
        # From 50% of fittest population, Individuals
        # will mate to produce offspring
        s = int((90*POPULATION_SIZE)/100)
        for _ in range(s):
            parent1 = random.choice(population[:50])
            parent2 = random.choice(population[:50])
            child = parent1.mate(parent2)
            new_generation.append(child)

        population = new_generation

        print("Generation: {}\tString: {}\tFitness: {}".\
              format(generation,"".join(population[0].chromosome),population[0].fitness))

        generation += 1


    print("Generation: {}\tString: {}\tFitness: {}".\
          format(generation,
          "".join(population[0].chromosome),
          population[0].fitness))

if __name__ == '__main__':
    main()
```

*Output*

```
Generation: 1   String: IivgHV:A8,i%9p"W9(LvD}WetnoHw
        Fitness: 27
Generation: 2   String: I[vgHV A8,i[9VEW
(L6D]W?in
,w(     Fitness: 25
Generation: 3   String: I-(gnV A8tiu9pEW9(LvD]Weiq
Nw(     Fitness: 24
Generation: 4   String: I-(gnV A8tiu9pEW9(LvD]Weiq
Nw(     Fitness: 24
Generation: 5   String: u 3;nV:AZtif9p"V0 3pnFtkin Q}(  Fitness: 23
Generation: 6   String: u 3;nV:AZtif9p"V0 3pnFtkin Q}(  Fitness: 23
Generation: 7   String: I vI g AZtuf9p"-0 3pnFi?id Q}e  Fitness: 21
Generation: 8   String: I vI g AZtuf9p"-0 3pnFi?id Q}e  Fitness: 21
Generation: 9   String: I vI g AZtuf9p"-0 3pnFi?id Q}e  Fitness: 21
Generation: 10  String: I ldhi A8tit$p"N
 ]6?ylGi%Yn{v   Fitness: 19
Generation: 11  String: I ldhi A8tit$p"N
 ]6?ylGi%Yn{v   Fitness: 19
```

- 
- 
- 

```
Generation: 3965        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3966        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3967        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3968        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3969        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3970        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3971        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3972        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3973        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3974        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3975        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3976        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3977        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3978        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3979        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3980        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3981        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3982        String: I love ArtifYcial Intelligence  Fitness: 1
Generation: 3983        String: I love Artificial Intelligence  Fitness: 0
```

**Latihan: Uji Coba Menerapkan Algoritma Genetika untuk Optimasi**

Misalkan terdapat sebuah persamaan, $f(x) = -(x^2) + 10$ . Dibutuhkan sebuah solusi agar persamaan tersebut memiliki nilai maksimum dan batasannya adalah $0 \leq x \leq 31$. Temukan solusi terbaik menggunakan algoritma genetika!

**Referensi:**


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

https://towardsdatascience.com/introduction-to-genetic-algorithm-and-python-implementation-for-function-optimization-fd36bad58277

https://machinelearningmastery.com/simple-genetic-algorithm-from-scratch-in-python/

https://towardsdatascience.com/genetic-algorithm-implementation-in-python-5ab67bb124a6