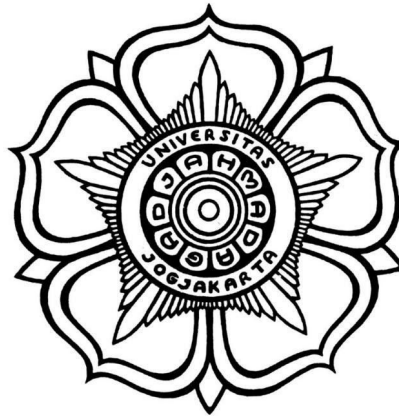


MACHINE LEARNING AND COMPUTATIONAL INTELLIGENCE

Nutrient Solution For Juvenile Stages In Hydroponics Using Genetic Algorithm



BY :

NADHIFA SOFIA

(19/448721/PPA/05804)

MASTER OF COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE AND ELECTRONICS

FACULTY OF MATHEMATICS AND NATURAL SCIENCES

UNIVERSITAS GADJAH MADA

2020

introduction ,related work, problem formulation, architecture, implementation, result

1. Introduction

Nutrition is very important in the growth and development of plants. It is also necessary for plant metabolism and their external supply. There are several stages of plant growth, and each growth stage needs different nutrients. One of the plant growth stages is Juvenile. Juvenile stages are vital stages for plant growth that do need a specific nutrient on it. One nutrient is called AB mix. AB mix consists of several contents such as Kalium, Phosphor, and Nitrogen. The amount of water that composes AB mix also has a great effect for plant growth. Thus, the right amount of composition of AB mix is essential for plant growth. As we know, nutrients in plants are an important thing as each growth stage needs different nutrients. The amount of nutrients can be given to plants, but the amount of nutrients that are suitable to plant needs are known, consequently the growth of plants in the juvenile stage is not optimal. Juvenile stage is a vital stage, so we compute the AB mix to produce the most efficient nutrient by using genetic algorithms.

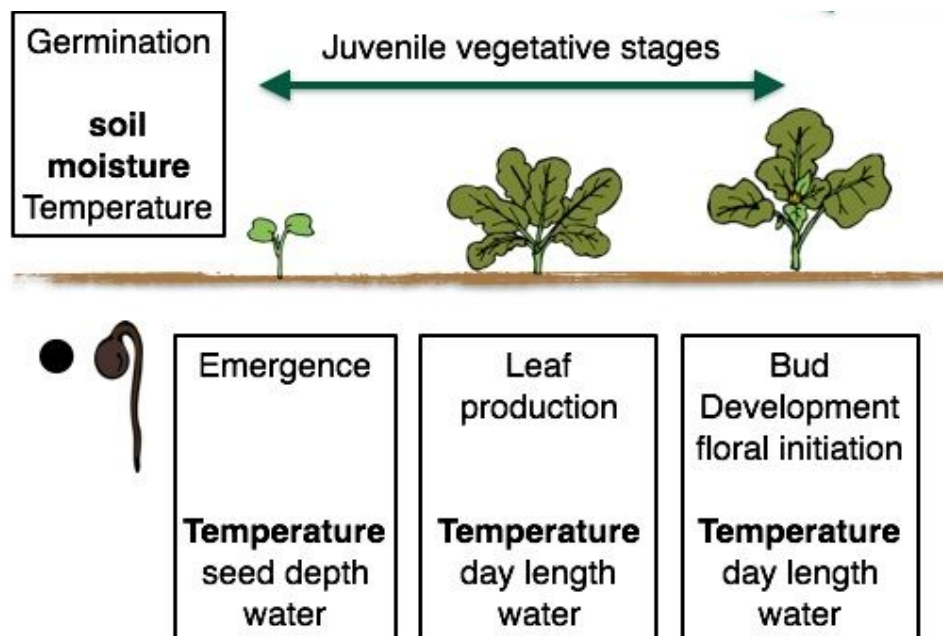


Figure 1. Juvenile vegetative stages

2. Problem Formulation

Problem:

The amount of Nutrients can be given to plants, but the amount of nutrients that are suitable to plant needs are known, consequently the growth of plants in the juvenile stage is not optimal.

Purpose:

Designing a system to determine the right nutrients in the juvenile stage using Genetic Algorithm.

Benefit:

Plants can grow well in the juvenile stage.

Problem Scope:

- The research is just for juvenile stage in 2 weeks after planting
- Plant that will be used is pepper (*Annum Capsicum L.*)
- Nutrient that will be analysis is Macronutrient (Nitrogen, Phosphor, Kalium)
- The plant characteristics that will be analysis is plant high, the number of leaf, and Wet weight
- The number of chromosomes are 10.

3. Architecture

The system itself is separated into 2 parts; preprocessing and the main genetic algorithm processing. The goal of our problem is to find the best composition of Kalium, Nitrogen, Phospor, and Water to optimize plant height, the number of leaves, and Wet weight. So, firstly we need to find a formula of plant height, the number of leaves, and Wet weight based on data that we have. In this case we use linear regression to find the formula or equation of plant height, the number of leaves, and wet weight. Thus, the preprocessing phase in this case is to find the formula of plant height, the number of leaves, and Wet weight using Linear Regression method. The implementation function to find the formula or equation of plant height, the number of leaves, and wet weight can be seen in the flowchart below.

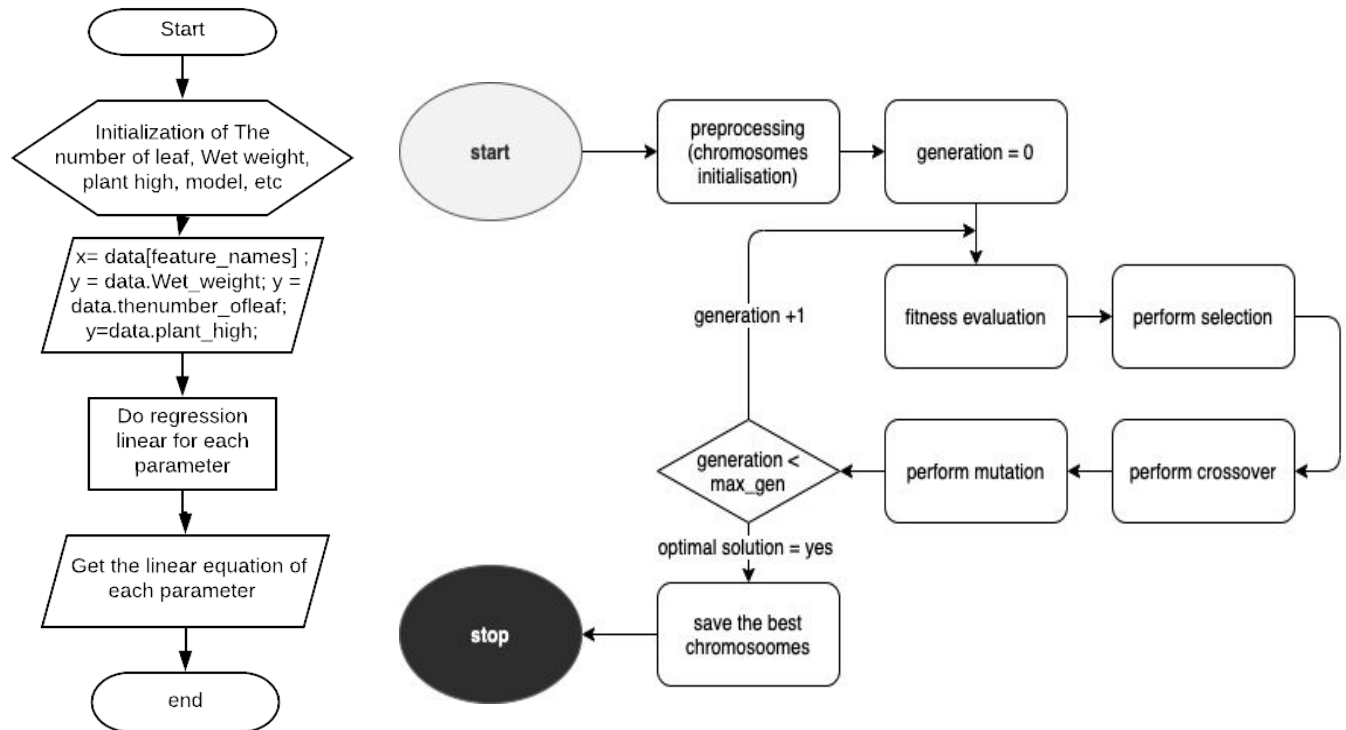


Figure 2. Flowchart for the system, left: preprocessing, right: the whole architecture

4. Implementation

Firstly, we should define the success criteria for seedling; by way of example the number of leaves are more than 4, the plant height is more than 15cm, and the wet weight of the plant is more than 8gr.

```

def beratbasahModel(data, feature_names):
    X=data[feature_names]
    y1=data.Berat_Basah
    model = LinearRegression().fit(X, y1)
    model=sm.OLS(y1,X).fit()
    X=sm.add_constant(X)
    model=sm.OLS(y1,X).fit()
    lin_model = np.around(model.params, decimals=3)
    return (lin_model)

```

```
def jumlahdaunModel(data, feature_names):
    X=data[feature_names]
    y1=data.Jumlah_Daun
    model = LinearRegression().fit(X, y1)
    model=sm.OLS(y1,X).fit()
    X=sm.add_constant(X)
    model=sm.OLS(y1,X).fit()
    lin_model = np.around(model.params, decimals=3)
    return (lin_model)

def tinggitanamanModel(data, feature_names):
    X=data[feature_names]
    y1=data.Tinggi_Tanaman
    model = LinearRegression().fit(X, y1)
    model=sm.OLS(y1,X).fit()
    X=sm.add_constant(X)
    model=sm.OLS(y1,X).fit()
    lin_model = np.around(model.params, decimals=3)
    return (lin_model)

data = pd.read_excel('data/data_dummy.xlsx', sheet_name='Sheet2')
feature_names=['Kalium', 'Nitrogen', 'Phosphor', 'Water']

berat_model = beratbasahModel(data, feature_names)
berat_model = np.array(berat_model)

daun_model = jumlahdaunModel(data, feature_names)
daun_model = np.array(daun_model)

tinggi_model = tinggitanamanModel(data, feature_names)
tinggi_model = np.array(tinggi_model)
```

The result model will be stored in model.csv, then used in genetic algorithm process.

```

row_list = [
    ["Model", "Kalium", "Nitrogen", "Phosphor", "Water", "Constant" ],
    ["tinggi_tanaman", tinggi_model[1], tinggi_model[2],
    tinggi_model[3], tinggi_model[4], tinggi_model[0]],
    ["jumlah_daun", daun_model[1], daun_model[2],
    daun_model[3], daun_model[4], daun_model[0]],
    ["berat_basah", berat_model[1], berat_model[2],
    berat_model[3], berat_model[4], berat_model[0]]
]
with open('data/model.csv', 'w', newline='') as file:
    writer = csv.writer(file, quoting=csv.QUOTE_NONNUMERIC,
                        delimiter=',')
    writer.writerows(row_list)

```

The result model will be like :

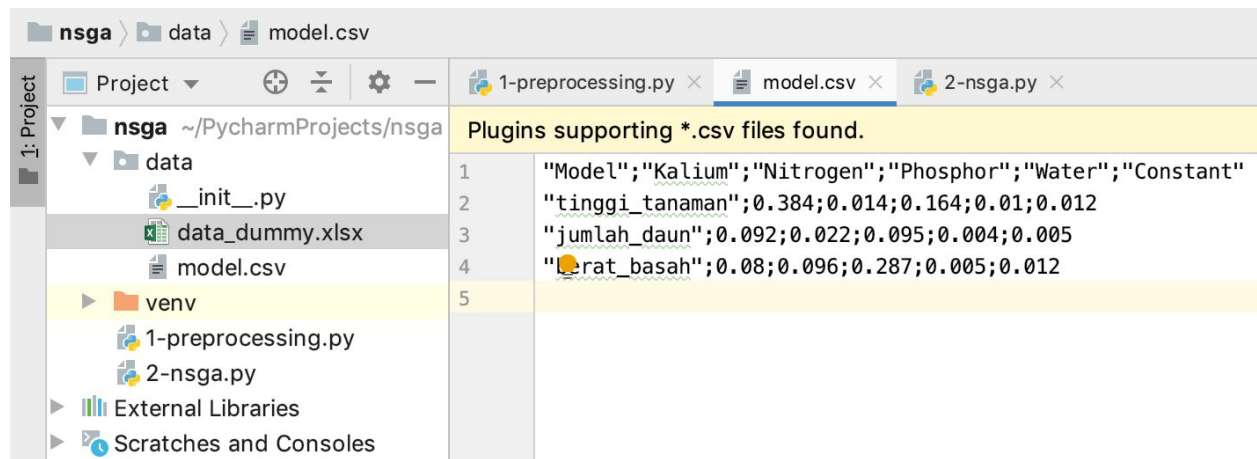


Figure 3. The result from preprocessing data

Model	Kalium	Nitrogen	Phospor	Air	Constant
tinggi_tanaman	0.384	0.014	0.164	0.01	0.012
jumlah_daun	0.092	0.022	0.095	0.004	0.005
berat_basah	0.08	0.096	0.287	0.005	0.012

Table 1. The result for fitness function

Fitness Function

Our goal is to find the best composition of Kalium, Nitrogen, Phospor, and Water to optimize plant height, the number of leaves, and wet weight. From our data preprocessing, we find linear equations of objective criteria (plant height, the number of leaves, and Wet weight) as

well as coefficients of each variable (Kalium, Nitrogen, Phospor, and Water). Thus we can write objective fitness function as

$$Plant\ Height = 0.384k + 0.014n + 0.164p + 0,01w + 0.012$$

$$Number\ of\ Leaf = 0.178k + 0.059n + 0.183p + 0,007w + 0.01$$

$$Wet\ Weight = 0.08k + 0.096n + 0.287p + 0,005w + 0.012$$

Best condition is a plant that has the biggest difference values of objective criteria (plant height, the number of leaves, and Wet weight) after being given ABmix composition (Kalium, Nitrogen, Phospor, and Water) with a plant that has no ABmix. Plant that has no ABmix has a objective criteria value :

- Plant height : 5.65
- The number of leaf : 8
- Wet weight : 0.375

Thus, fitness function of our problem can be written as follow

$$Fitness = (Plant\ Height - 5.65) + (Number\ of\ Leaf - 8) + (Wet\ Weight - 0.375)$$

```
def getFitness(chromosome):  
    # Computation of fitness function  
    objective_stems = (model[0,1]*chromosome[:,0]) +  
(model[0,2]*chromosome[:,1]) + (model[0,3]*chromosome[:,2]) +  
(model[0,4]*chromosome[:,3]) + model[0,5]  
    objective_leaf = (model[1,1]*chromosome[:,0]) +  
(model[1,2]*chromosome[:,1]) + (model[1,3]*chromosome[:,2]) +  
(model[1,4]*chromosome[:,3]) + model[1,5]  
    objective_weight = (model[2,1]*chromosome[:,0]) +  
(model[2,2]*chromosome[:,1]) + (model[2,3]*chromosome[:,2]) +  
(model[2,4]*chromosome[:,3]) + model[2,5]  
    fitness = (objective_stems - 5.65) + (objective_leaf - 8) +  
(objective_weight - 0.375)  
    return(fitness)
```

Chromosome Initialization

We initiate a population of chromosomes with 10. Each chromosome has 4 genes that consist of random integer variables within range 1-20 for Kalium, Nitrogen, and Phosphor; and random variables within range 500-1500 for water. Implementation of this process can be seen in figure below.

```
# Initializing n = Number of chromosome
n = 10
# Initialization of chromosomes
# chromosome
chromosome_abmix = np.random.randint(1,21 , (n,3))
chromosome_water = np.random.randint(500,1501 , (n,1))
chromosome = np.append(chromosome_abmix, chromosome_water, axis=1)
print("First Chromosomes : \n",chromosome)
```

```
Terminal: Local x +
(base) Nadhifas-MacBook-Pro:nsga nadhifasofia$ python 2-nsga.py
First Chromosomes :
[[ 10   5  17 704]
 [  6   4   5 904]
 [ 19  16  20 1213]
 [  7  13   5 862]
 [ 20  17   3 1261]
 [  6   5   2 575]
 [ 17  12   5 1497]
 [ 10  20  18 682]
 [  1   6   3 689]
 [ 19   7   2 842]]
Total Fitness : 151.415
```

Figure 4. Chromosomes initialization

Selection

For selection, we use the Roulette Wheel as a selection process. Steps of this process can be written as follow :

- Calculate fitness probability of each chromosome.
- Calculate cumulative sum of fitness probability.
- Generate random value between 0-1 as much as number of chromosome
- Compare each random value with cumulative probability of each individual fitness.

- Select the individual if the random value is in range of its cumulative probability and select the chromosome of those indexes as a chromosome in the new pop for the next gen, as it is a higher value.

For our case, we are not adding a new chromosome. We just replace chromosomes after new chromosomes as a result of each process. Thus, the number of the population always remains 10. Implementation of this process can be seen in figure below

```
def selection(fitness, chromosome):  
    # Calculating the total of fitness function  
    total = fitness.sum()  
    print("Total Fitness :", total)  
  
    # Calculating Probability for each chromosome  
    prob = fitness/total  
    print("Probability :\n", prob)  
  
    # SELECTION  
    # Selection using Roulette Wheel  
  
    # Calculating Cumulative Probability  
    cum_sum = np.cumsum(prob)  
    print("Cumulative Sum :\n", cum_sum)  
  
    # Generating Random Numbers in the range 0-1  
    Ran_nums = np.random.random((chromosome.shape[0]))  
    print("Random Numbers \n:", Ran_nums)  
  
    # Making a new matrix of chromosome for calculation purpose  
    chromosome_2 = np.zeros((chromosome.shape[0], 4))  
    #print(chromosome_2)  
  
    for i in range(Ran_nums.shape[0]):  
        for j in range(chromosome.shape[0]):  
            if Ran_nums[i] < cum_sum[j]:  
                chromosome_2[i, :] = chromosome[j, :]  
                break  
  
    chromosome = chromosome_2.astype(int)  
    return(chromosome)
```

```

Terminal: Local × +
Probability :
[0.09520853 0.05969026 0.20995278 0.06513225 0.16125879 0.01228412
0.18437407 0.10690486 0.01196051 0.09323383]
Cumulative Sum :
[0.09520853 0.15489879 0.36485157 0.42998382 0.59124261 0.60352673
0.7879008 0.89480567 0.90676617 1.          ]
Random Numbers
: [0.51446902 0.17345437 0.20100205 0.54653962 0.05746823 0.89240289
0.47388211 0.51089337 0.76845477 0.98564604]

```

For instance, as we can see, the random number in index 2 (0.173) lies between cumulative probability in index 4 and 5. Then we select chromosome 5 then replace chromosome 2. Apply this process for all populations. Thus, new population will become

```

Chromosomes after selection :
[[ 20  17  3 1261]
 [ 19  16 20 1213]
 [ 19  16 20 1213]
 [ 20  17  3 1261]
 [ 10   5 17  704]
 [ 10  20 18  682]
 [ 20  17  3 1261]
 [ 20  17  3 1261]
 [ 17  12  5 1497]
 [ 19   7  2  842]]
2: Favorites
★

```

Figure 5. Chromosomes generated after selection

Crossover

For crossover, we use 1-cut point crossover. It chooses one position in chromosomes randomly. Child 1 consists of a head chromosome of parent 1 and tail of parent 2. Child 2 consists of a head chromosome of parent 2 and tail of parent 1. In order to keep the number of chromosomes remaining 10, in our case we modified crossover just to produce one child (Child 1 : head chromosomes of parent 1 and tail of parent 2) and directly replace the first parent. Detail step can be written as follow :

- Randomly generated random values as much as the number of chromosomes.
- From each generated random then compare with our Crossover rate. In our case we use crossover rate 0.25
- If random value is less than crossover rate, then select the chromosome in same index as generated random to be selected parents that will be perform crossover

Implementation of this process can be seen in figure below

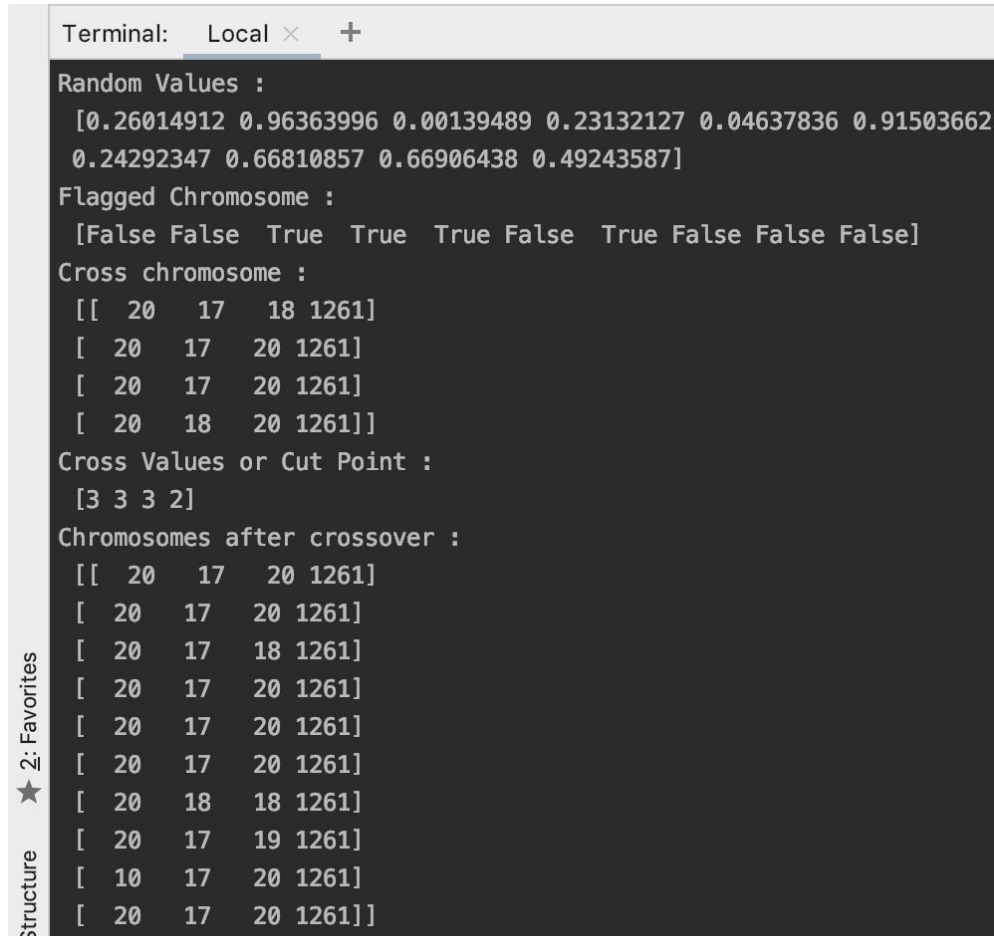
```
def crossover(chromosome):  
    # CROSSOVER  
    R = np.random.random((chromosome.shape[0]))  
    print("Random Values :\n",R)  
  
    # Crossover Rate  
    pc = 0.25  
    flag = R < pc  
    print("Flagged Chromosome : \n",flag)  
  
    # Determining the cross chromosomes  
    cross_chromosome = chromosome[(i == True) for i in flag]  
    print("Cross chromosome :\n",cross_chromosome)  
    len_cross_chrom = len(cross_chromosome)  
  
    # Calculating cross values  
    cross_values = np.random.randint(1,4,len_cross_chrom)  
    print("Cross Values or Cut Point :\n",cross_values)  
  
    cpy_chromosome = np.zeros(cross_chromosome.shape)  
  
    # Performing Cross-Over  
  
    # Copying the chromosome values for calculations  
    for i in range(cross_chromosome.shape[0]):  
        cpy_chromosome[i , :] = cross_chromosome[i , :]  
  
    if len_cross_chrom == 1:  
        cross_chromosome = cross_chromosome  
    else :  
        for i in range(len_cross_chrom):  
            c_val = cross_values[i]  
            if i == len_cross_chrom - 1 :  
                cross_chromosome[i , c_val:] = cpy_chromosome[0 , c_val:]  
            else :  
                cross_chromosome[i , c_val:] = cpy_chromosome[i+1 , c_val:]  
  
    #print("Crossovered Chromosome :",cross_chromosome)  
  
    index_chromosome = 0
```

```

index_newchromosome = 0
for i in flag :
    if i == True :
        chromosome[index_chromosome, :] =
cross_chromosome[index_newchromosome, :]
        index_newchromosome = index_newchromosome + 1
        index_chromosome = index_chromosome + 1

return(chromosome)

```



```

Terminal: Local x +
Random Values :
[0.26014912 0.96363996 0.00139489 0.23132127 0.04637836 0.91503662
0.24292347 0.66810857 0.66906438 0.49243587]
Flagged Chromosome :
[False False True True True False True False False False]
Cross chromosome :
[[ 20 17 18 1261]
 [ 20 17 20 1261]
 [ 20 17 20 1261]
 [ 20 18 20 1261]]
Cross Values or Cut Point :
[3 3 3 2]
Chromosomes after crossover :
[[ 20 17 20 1261]
 [ 20 17 20 1261]
 [ 20 17 18 1261]
 [ 20 17 20 1261]
 [ 20 17 20 1261]
 [ 20 17 20 1261]
 [ 20 18 18 1261]
 [ 20 17 19 1261]
 [ 10 17 20 1261]
 [ 20 17 20 1261]]

```

Figure 6. Chromosomes after crossover

As we can see, the chromosomes that are selected as parents are chromosome[1] and chromosome[5]. And the cut point is 3 for the first crossover and 2 for the last crossover.

Mutation

For mutation, we use uniform mutation. we select a random gene from our chromosome, let's say x_i and assign a uniform random value to it. Let x_i be within the range $[a_i, b_i]$ then we assign $U(a_i, b_i)$ to x_i . $U(a_i, b_i)$ denotes a uniform random number from within the range $[a_i, b_i]$.

Detail step can be written as follow :

- Calculate total number of gen
- Calculate total mutation. It is calculated based on the mutation rate set. We set mutation rate 0.1. So if total number of gen is 40 then total mutation will be $= 0.1 * 40 = 4$
- Select which gen will performed mutation through random method
- Generated new value that will replace old value of selected mutation gen
- Replace old value of selected gen with new value

Implementation of this process can be seen in figure below.

```
def mutation(chromosome):
    #MUTATION
    # Calculating the total no. of generations
    a ,b = chromosome.shape[0] ,chromosome.shape[1]
    total_gen = a*b
    print("Total Gen :",total_gen)

    #mutation rate = pm
    pm = 0.1
    no_of_mutations = int(np.round(pm * total_gen))
    print("Total Mutations : " ,no_of_mutations)

    # Calculating the Generation number for choose gen mutation
    gen_num = np.random.randint(0,total_gen, no_of_mutations)
    print("Gen for Mutation : " , gen_num)

    for i in range(no_of_mutations):
        a = gen_num[i]
        row = a//4
        #print("Chromosome :", row)
        col = a%4
        #print("Gen :", col)
        print("Mutation in Gen : %d of Chromosome : %d " %(col,row))
        if (col==3) :
            # Generating a random number which can replace the selected
            chromosome to be mutated
            Replacing_num = np.random.randint(500,1501)
            print("Replaced value : " , Replacing_num)
        else :
            # Generating a random number which can replace the selected
            chromosome to be mutated
```

```

    Replacing_num = np.random.randint(1,21)
    print("Replaced value : " , Replacing_num)
    chromosome[row , col] = Replacing_num

#print(" Chromosomes After Mutation : " , chromosome)
return(chromosome)

```

```

Terminal: Local x +
Total Gen : 40
Total Mutations : 4
Gen for Mutation : [ 1  5  8 38]
Mutation in Gen : 1 of Chromosome : 0
Replaced value : 7
Mutation in Gen : 1 of Chromosome : 1
Replaced value : 5
Mutation in Gen : 0 of Chromosome : 2
Replaced value : 15
Mutation in Gen : 2 of Chromosome : 9
Replaced value : 17
Chromosomes after mutation :
[[ 20  7  20 1261]
 [ 20  5  20 1261]
 [ 15 17  18 1261]
 [ 20 17  20 1261]
 [ 20 17  20 1261]
 [ 20 17  20 1261]
 [ 20 18  18 1261]
 [ 20 17  19 1261]
 [ 10 17  20 1261]
 [ 20 17  17 1261]]

```

Figure 7. Chromosomes after mutation

For our case, there are a total 40 genes, thus total mutation of gene is 4. Selected gen that will perform mutation is [1 5 8 38]. This gen will be update into new value. For instance In gen 1, its located in chromosome 0 gen 1, current value is 1 and will be update with 7. Do this process for all selected gen.

Update Generation and Final Result

Then the population will be updated and repeat the same process until n generation . For our case, let's say that the number of generations is 2 and the number of chromosomes remains 10. So after 2 generations, the best chromosome from the remaining 10 chromosomes that has best fitness will be selected. Since our case is search for best composition of Kalium, Nitrogen, Phosphor, and Water to optimize plant height, the number of leaf, and Wet weight , based on our chromosome that we have, solution is :

```
epoch = 0
# Initializing g = Number of generation
g = 5

while epoch < g :

    #Measure Fitness
    fitness = getFitness(chromosome)

    #Apply SELECTION on chromosome
    chromosome = selection(fitness,chromosome)

    print("Chromosomes after selection : \n",chromosome)

    chromosome = crossover(chromosome)

    print("Chromosomes after crossover : \n",chromosome)

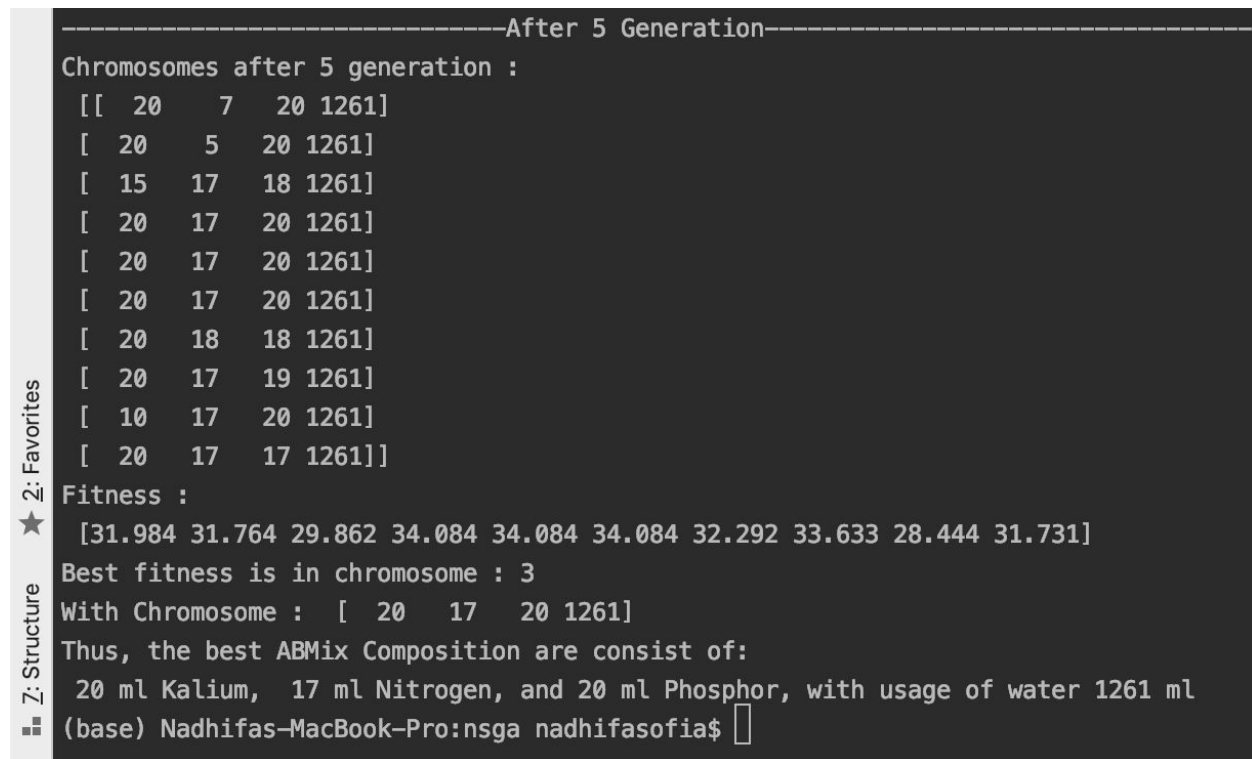
    chromosome = mutation(chromosome)

    print(" Chromosomes after mutation : \n" , chromosome)

    epoch = epoch + 1
    print("\n")

print("-----After %d
Generation-----" %g)
fitness = getFitness(chromosome)
print("Chromosomes after %d generation :\n"%g, chromosome)
print("Fitness : \n",fitness)
print("Best fitness is in chromosome : %d" %np.argmax(fitness))
print("With Chromosome : ", chromosome[np.argmax(fitness)])
#print("Thus, best ABMix Composition are consist of: %d ml Kalium, %d ml
Nitrogen, and %d ml Phosphor, with usage of water %d ml "
%(chromosome[0],chromosome[1],chromosome[2],chromosome[3]))

print("Thus, the best ABMix Composition are consist of: \n %d ml Kalium, %d
ml Nitrogen, and %d ml Phosphor, with usage of water %d ml "
%(chromosome[np.argmax(fitness)][0],chromosome[np.argmax(fitness)][1],chromoso
me[np.argmax(fitness)][2],chromosome[np.argmax(fitness)][3]))
```



```

-----After 5 Generation-----
Chromosomes after 5 generation :
[[ 20  7  20 1261]
 [ 20  5  20 1261]
 [ 15 17 18 1261]
 [ 20 17 20 1261]
 [ 20 17 20 1261]
 [ 20 17 20 1261]
 [ 20 18 18 1261]
 [ 20 17 19 1261]
 [ 10 17 20 1261]
 [ 20 17 17 1261]]
Fitness :
[31.984 31.764 29.862 34.084 34.084 34.084 32.292 33.633 28.444 31.731]
Best fitness is in chromosome : 3
With Chromosome : [ 20  17  20 1261]
Thus, the best ABMix Composition are consist of:
 20 ml Kalium, 17 ml Nitrogen, and 20 ml Phosphor, with usage of water 1261 ml
(base) Nadhifas-MacBook-Pro:nsga nadhifasofia$

```

Figure 8. The final result of GA process

After getting through the second generation. The best chromosome is [20 17 20 1261]. Means that in order to optimize plant height, the number of leaves, and Wet weight , the best composition of ABmix that can be used consists of **20 ml Kalium, 17 ml Nitrogen, and 20 ml Phosphor with usage of water 1261 ml.**

Code : <https://github.com/dhifaaans/nsga>