Q1 Genetic Algorithms

In [1]:

```
import numpy as np
import pandas as pd
from math import*

def fitnessfunc(x):
    return -x**2 + 8*x + 15

sol = np.linspace(0,15,num=16,endpoint=True)
sol = sol.astype(int)
data = pd.read_csv('/home/nikhil/Documents/Berkeley Courses/Machine-Learning/hw3/encodeA.csv')
encodA = np.zeros((len(sol),4))
for i in range(0,len(sol)):
    encodA[i,:] = np.array(data.iloc[i,1:5])

data = pd.read_csv('/home/nikhil/Documents/Berkeley Courses/Machine-Learning/hw3/encodeB.csv')
encodB = np.zeros((len(sol),4))
for i in range(0,len(sol)):
    encodB[i,:] = np.array(data.iloc[i,1:5])
ans1a = []
encodA = encodA.astype(int)
encodB = encodB.astype(int)

for i in range(0,len(sol)):
    if fitnessfunc(sol[i]) > 27:
        ans1a = ans1a + [sol[i]]
print('good solutions:' + str(ans1a))

print('good solutions from Encoding A')
for i in range(len(ans1a)):
    print(encodA[ans1a[i],:])

print('good solutions from Encoding B')
for i in range(len(ans1a)):
    print(encodB[ans1a[i],:])
```

```
good solutions:[3, 4, 5]
good solutions from Encoding A
[1 0 0 0]
[0 0 1 0]
[0 0 0 1]
good solutions from Encoding B
[1 1 0 1]
[1 0 1 1]
[1 1 1 1]
```

Q1a) Schema for 'good solutions' via Encoding A is [_,0,,]. Length of which is 0 and order is 1. Schema for 'good solutions' via Encoding B is [1,,_*,1]. Length of which is 2 and order is 2. Since we want these 'good solutions' to grow more, according to Holland's logic we should use schema with low order and length hence I will use encoding A for applying GA.

Q1b)

In [2]:
```python
x = np.array([10,1,15,6,0,9])
fitness = fitnessfunc(x)
for i in range(0,len(x)):
    print('x=' + str(x[i])+ ', ' + str(encodA[int(x[i]),:].astype(int
)) + ', fitness = ' + str(fitness[i]))
```

```
x=10, [0 1 0 1], fitness = -5
x=1, [0 0 1 1], fitness = 22
x=15, [1 1 1 1], fitness = -90
x=6, [0 0 0 0], fitness = 27
x=0, [1 0 1 1], fitness = 15
x=9, [1 1 0 0], fitness = 6
```

In [3]:
```python
# Creating pairs
candidates = np.zeros((len(x),4))
sort_idx = np.argsort(fitness,axis = None)

sort_x = np.zeros_like(x)
for i in range(0,len(x)):
    candidates[i,:] = encodA[x[sort_idx[i]],:]
    sort_x[i] = x[sort_idx[i]]
print("sorted candidates")
print(candidates)
print(sort_x)

fitness = np.sort(fitness)
```

```
sorted candidates
[[1. 1. 1. 1.]
 [0. 1. 0. 1.]
 [1. 1. 0. 0.]
 [1. 0. 1. 1.]
 [0. 0. 1. 1.]
 [0. 0. 0. 0.]]
[15 10  9  0  1  6]
```

In [4]:
```
#Q1c

def fitness_pop(function,x):
    return np.sum(function(x))

def crossover12(a,b):
    temp_a = np.array([a[0],b[1],b[2],b[3]])
    temp_b = np.array([b[0],a[1],a[2],a[3]])
    return temp_a,temp_b


new_members = np.zeros_like(candidates)
population = np.copy(candidates)

for i in range(0,len(candidates)):
    new_members[i],new_members[-1-i] = crossover12(candidates[i],cand
idates[-1-i])
# print(new_members)

# inversion function decodes the binary encoding

def inversion(a,ref):
    for i in range(0,len(ref)):
        if np.array_equal(a,ref[i,:]) == True:
            return int(i)

def is_member(a,b):
    for i in range(len(b)):
        if np.array_equal(a,b[i,:]) == True:
            return True
        else:
            return False


def is_new(a,pop):
    for j in range(len(a)):
        pop = np.vstack((pop,a[j]))
        if is_member(a[j],pop) == False:
            print(str(a[j]) + " is a new member with fitness=" + str(
fitnessfunc(inversion(a[j],encodA))))
        else:
            print(str(a[j]) + " is already a member")
    return pop

population = is_new(new_members,population)
# print("new population")
# print(population)

print('total fitness before crossover')
print(np.sum(fitness))

fitness = np.zeros(len(population))
for i in range(len(population)):
    fitness[i] = fitnessfunc(inversion(population[i],encodA))

print('total fitness after crossover')
```

```python
print(np.sum(fitness))

print('fittest element or best soluion')
print(str(population[np.argsort(fitness)[-1]]) + ',' + str(inversion(
population[np.argsort(fitness)[-1]],encodA)))
```

```
[1. 0. 0. 0.] is a new member with fitness=30
[0. 0. 1. 1.] is a new member with fitness=22
[1. 0. 1. 1.] is a new member with fitness=15
[1. 1. 0. 0.] is a new member with fitness=6
[0. 1. 0. 1.] is a new member with fitness=-5
[0. 1. 1. 1.] is a new member with fitness=-33
total fitness before crossover
-25
total fitness after crossover
10.0
fittest element or best soluion
[1. 0. 0. 0.],3
```

The function IS_NEW tells whether there a newly generated member is already a part of population or not. Here we see that there are a bunch of new members in population pool. The fitness of population has increase from -25 to 10. The best solution is now x=3, [1,0,0,0] with fitness = 30

In [5]:
```python
#Q1d

def mutate(a,k):
    for i in range(len(a)):
#         r = round(np.random.rand(1))
#         a[i,k] = r
        if int(a[i,k]) == 1:
            a[i,k]=0
        else:
            a[i,k] = 1
    return a

new_members = mutate(population,2)

population= is_new(new_members,population)

print('total fitness before crossover')
print(np.sum(fitness))

fitness = np.zeros(len(population))
for i in range(len(population)):
    fitness[i] = fitnessfunc(inversion(population[i],encodA))

print('total fitness after crossover')
print(np.sum(fitness))

print('fittest element or best soluion')
print(str(population[np.argsort(fitness)[-1]]) + ',' + str(inversion(
population[np.argsort(fitness)[-1]],encodA)))
```

```
[1. 1. 0. 1.] is already a member
[0. 1. 1. 1.] is a new member with fitness=-33
[1. 1. 1. 0.] is a new member with fitness=-69
[1. 0. 0. 1.] is a new member with fitness=27
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 1. 0.] is a new member with fitness=31
[1. 0. 1. 0.] is a new member with fitness=22
[0. 0. 0. 1.] is a new member with fitness=30
[1. 0. 0. 1.] is a new member with fitness=27
[1. 1. 1. 0.] is a new member with fitness=-69
[0. 1. 1. 1.] is a new member with fitness=-33
[0. 1. 0. 1.] is a new member with fitness=-5
total fitness before crossover
10.0
total fitness after crossover
-184.0
fittest element or best soluion
[0. 0. 1. 0.],4
```

Here we again see that there are a bunch of new members. The fitness of population has decreased from 10 to -134. The best solution is now x=4, [0,0,1,0] with fitness = 31

In [6]:

```
#Q1e
population = np.delete(population,np.argsort(fitness)[0],axis = 0)
fitness= np.delete(fitness,np.argsort(fitness)[0],axis=0)
population = np.vstack((population,population[np.argsort(fitness)[-1
]]))
fitness = np.zeros(len(population))
for i in range(len(population)):
    fitness[i] = fitnessfunc(inversion(population[i],encodA))

print('least fit element deleted and most fit element cloned and put
 at the bottom')
print(population)
```

```
least fit element deleted and most fit element cloned and put at the
bottom
[[1. 1. 0. 1.]
 [0. 1. 1. 1.]
 [1. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [1. 0. 1. 0.]
 [0. 0. 0. 1.]
 [1. 0. 0. 1.]
 [1. 1. 1. 0.]
 [0. 1. 1. 1.]
 [0. 1. 0. 1.]
 [1. 1. 0. 1.]
 [0. 1. 1. 1.]
 [1. 1. 1. 0.]
 [1. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [1. 0. 1. 0.]
 [0. 0. 0. 1.]
 [1. 0. 0. 1.]
 [1. 1. 1. 0.]
 [0. 1. 1. 1.]
 [0. 1. 0. 1.]
 [0. 0. 1. 0.]]
```

In [7]:

```
#two point crossover by pairing fittest with least fit

def crossover23(a,b):
    temp_a = np.array([a[0],b[1],b[2],a[3]])
    temp_b = np.array([b[0],a[1],a[2],b[3]])
    return temp_a,temp_b


# first we need to sort the array as per fitness
new_members = np.zeros_like(population)
sort_idx = np.argsort(fitness)
for i in range(len(population)):
    new_members[i],new_members[-1-i] = crossover23(population[sort_id
x[i]],population[sort_idx[-1-i]])

population = is_new(new_members,population)
# print("new population")
# print(population)

print('total fitness before crossover')
print(np.sum(fitness))

fitness = np.zeros(len(population))
for i in range(len(population)):
    fitness[i] = fitnessfunc(inversion(population[i],encodA))

print('total fitness after crossover')
print(np.sum(fitness))

print('fittest element or best soluion')
print(str(population[np.argsort(fitness)[-1]]) + ',' + str(inversion(
population[np.argsort(fitness)[-1]],encodA)))
```

```
[1. 0. 1. 0.] is a new member with fitness=22
[1. 0. 1. 0.] is a new member with fitness=22
[1. 0. 1. 0.] is a new member with fitness=22
[1. 0. 0. 1.] is a new member with fitness=27
[1. 0. 0. 1.] is a new member with fitness=27
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[1. 0. 1. 0.] is a new member with fitness=22
[1. 0. 1. 0.] is a new member with fitness=22
[1. 1. 0. 1.] is already a member
[1. 1. 0. 1.] is already a member
[1. 1. 1. 1.] is a new member with fitness=-90
[1. 1. 1. 1.] is a new member with fitness=-90
[0. 1. 1. 1.] is a new member with fitness=-33
[0. 1. 1. 1.] is a new member with fitness=-33
[0. 1. 0. 1.] is a new member with fitness=-5
[0. 1. 0. 1.] is a new member with fitness=-5
[0. 1. 1. 0.] is a new member with fitness=-18
[0. 1. 1. 0.] is a new member with fitness=-18
[0. 1. 1. 0.] is a new member with fitness=-18
total fitness before crossover
-84.0
total fitness after crossover
-150.0
fittest element or best soluion
[0. 0. 1. 0.],4
```

Here we again see that there are a bunch of new members. The fitness of population has decreased from -84 to -150. The best solution is still x=4, [0,0,1,0] with fitness = 31

In [8]:
```
#Q1f
population = np.delete(population,np.argsort(fitness)[0],axis = 0)
fitness= np.delete(fitness,np.argsort(fitness)[0],axis=0)
population = np.vstack((population,population[np.argsort(fitness)[-1
]]))
fitness = np.zeros(len(population))
for i in range(len(population)):
    fitness[i] = fitnessfunc(inversion(population[i],encodA))

print('least fit element deleted and most fit element cloned and put
 at the bottom')
print(population)
```

```
least fit element deleted and most fit element cloned and put at the
bottom
[[1. 1. 0. 1.]
 [0. 1. 1. 1.]
 [1. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [1. 0. 1. 0.]
 [0. 0. 0. 1.]
 [1. 0. 0. 1.]
 [1. 1. 1. 0.]
 [0. 1. 1. 1.]
 [0. 1. 0. 1.]
 [1. 1. 0. 1.]
 [0. 1. 1. 1.]
 [1. 1. 1. 0.]
 [1. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [1. 0. 1. 0.]
 [0. 0. 0. 1.]
 [1. 0. 0. 1.]
 [1. 1. 1. 0.]
 [0. 1. 1. 1.]
 [0. 1. 0. 1.]
 [0. 0. 1. 0.]
 [1. 0. 1. 0.]
 [1. 0. 1. 0.]
 [1. 0. 1. 0.]
 [1. 0. 0. 1.]
 [1. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]
 [1. 0. 1. 0.]
 [1. 0. 1. 0.]
 [1. 1. 0. 1.]
 [1. 1. 0. 1.]
 [1. 1. 1. 1.]
 [0. 1. 1. 1.]
 [0. 1. 1. 1.]
 [0. 1. 0. 1.]
 [0. 1. 0. 1.]
 [0. 1. 1. 0.]
 [0. 1. 1. 0.]
 [0. 1. 1. 0.]
 [0. 0. 1. 0.]]
```

In [9]:

```
# first we need to sort the array as per fitness

def crossover34(a,b):
    temp_a = np.array([b[0],b[1],b[2],a[3]])
    temp_b = np.array([a[0],a[1],a[2],b[3]])
    return temp_a,temp_b

new_members = np.zeros_like(population)
sort_idx = np.argsort(fitness)
for i in range(len(population)):
    new_members[i],new_members[-1-i] = crossover34(population[sort_id
x[i]],population[sort_idx[-1-i]])

population = is_new(new_members,population)
# print("new population")
# print(population)

print('total fitness before crossover')
print(np.sum(fitness))

fitness = np.zeros(len(population))
for i in range(len(population)):
    fitness[i] = fitnessfunc(inversion(population[i],encodA))

print('total fitness after crossover')
print(np.sum(fitness))

print('fittest element or best soluion')
print(str(population[np.argsort(fitness)[-1]]) + ',' + str(inversion(
population[np.argsort(fitness)[-1]],encodA)))
```

```
[0. 0. 1. 1.] is a new member with fitness=22
[0. 0. 1. 0.] is a new member with fitness=31
[0. 0. 1. 0.] is a new member with fitness=31
[0. 0. 1. 0.] is a new member with fitness=31
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[0. 0. 0. 1.] is a new member with fitness=30
[1. 0. 0. 0.] is a new member with fitness=30
[1. 0. 0. 0.] is a new member with fitness=30
[1. 0. 0. 0.] is a new member with fitness=30
[1. 0. 0. 1.] is a new member with fitness=27
[1. 0. 0. 1.] is a new member with fitness=27
[1. 0. 0. 1.] is a new member with fitness=27
[1. 0. 1. 1.] is a new member with fitness=15
[1. 0. 1. 0.] is a new member with fitness=22
[1. 0. 1. 0.] is a new member with fitness=22
[1. 0. 1. 0.] is a new member with fitness=22
[1. 0. 1. 0.] is a new member with fitness=22
[1. 0. 1. 0.] is a new member with fitness=22
[1. 0. 1. 0.] is a new member with fitness=22
[0. 1. 0. 0.] is a new member with fitness=15
[0. 1. 0. 1.] is a new member with fitness=-5
[0. 1. 0. 1.] is a new member with fitness=-5
[0. 1. 0. 1.] is a new member with fitness=-5
[0. 1. 1. 1.] is a new member with fitness=-33
[0. 1. 1. 1.] is a new member with fitness=-33
[0. 1. 1. 1.] is a new member with fitness=-33
[0. 1. 1. 1.] is a new member with fitness=-33
[0. 1. 1. 1.] is a new member with fitness=-33
[0. 1. 1. 1.] is a new member with fitness=-33
[0. 1. 1. 1.] is a new member with fitness=-33
[0. 1. 1. 1.] is a new member with fitness=-33
[0. 1. 1. 1.] is a new member with fitness=-33
[1. 1. 0. 1.] is already a member
[1. 1. 0. 1.] is already a member
[1. 1. 0. 1.] is already a member
[1. 1. 0. 1.] is already a member
[1. 1. 1. 0.] is a new member with fitness=-69
[1. 1. 1. 0.] is a new member with fitness=-69
[1. 1. 1. 0.] is a new member with fitness=-69
[1. 1. 1. 0.] is a new member with fitness=-69
total fitness before crossover
-29.0
total fitness after crossover
-69.0
fittest element or best soluion
[0. 0. 1. 0.],4
```

Here we again see that there are a bunch of new members. The fitness of population has decreased from -29 to -69. The best solution is still x=4, [0,0,1,0] with fitness = 31

Q1g) As such we did get the right answer for maxima, however, the fitness score of the population did not exactly increase even though we only used mutation once. So maybe we can find better encoding.

Q2 Artificial Neural Network

In [10]:

```
#Q1a) There are 8 weights, but only 4 are independent

w11 = np.random.rand()
w12 = np.random.rand()
w13 = np.random.rand()
w14 = np.random.rand()
w21 = np.random.rand()
w22 = np.random.rand()
w23 = np.random.rand()
w24 = np.random.rand()

w1 = np.array([w11,w12,w13,w14])
w2 = np.array([w21,w22,w23,w24])
print(w1)
print(w2)
```

```
[0.94108549 0.17270798 0.31909035 0.78678205]
[0.23892267 0.72514132 0.03577083 0.3381335 ]
```

In [11]:
```python
i1 = [-1,1]
i2= [-1,-1]
i3= [1,-1]

x11 = w11*i1[0] + w12*i2[0]
x12 = w13*i2[1] + w14*i3[1]

print(x11)
print(x12)

def activation(x):
    if tanh(x) >= 0:
        return 1
    else:
        return -1

y11 = activation(x11)
y12 = activation(x12)
print(y11)
print(y12)

x21 = w21*y11 + w23*y12
x22 = w22*y11 + w24*y12

y21 = activation(x21)
y22 = activation(x22)
y_output = [y21,y22]
print(y_output)

def sec_structure(y):
    if np.array_equal(y,[1,-1]) == True:
        print('Helix')
    if np.array_equal(y,[-1,1]) == True:
        print('beta sheet')
    if np.array_equal(y,[-1,-1]) == True:
        print('Coil')
    return 'calculation done'

print('secondary structure:')
print(sec_structure(y_output))
```

```
-1.11379346939
-1.10587239798
-1
-1
[-1, -1]
secondary structure:
Coil
calculation done
```

Q1c) we will use a mean square type definition to quanitfy error.

In [12]:
```
y21_obs = -1
y22_obs = -1

error_y21 = 0.5*(y21_obs- y21)**2
print('error in first output node ' + str(error_y21))

error_y22 = 0.5*(y22_obs- y22)**2
print('error in second output node ' + str(error_y22))

print('the errors for the hidden layer will come from back-propogation')
```

```
error in first output node 0.0
error in second output node 0.0
the errors for the hidden layer will come from back-propogation
```

Since error in output layer is zero, error has to be zero in hidden layers as well. The expressions for errors will follow in scanned pages that follow.