OneMAX Problem

1. Fuctions:

*def create\_ind(ind\_len)* ⇒ creates individual based with input of individual length

*def create\_pop(pop\_size,indiv\_len)* ⇒ creates population with input of population size and individual size

*def fitness(individual)* ⇒ evaluates and returns fitness value of individual

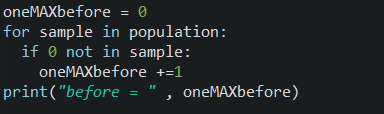
Fitness value is the number of “1” that individual contains. For example, I=[0,0,0,1,0,0,0,1,0,0] has fitness value of 2. As OneMAX tries to create individuals with only “1”s, we give higher fitness values to individuals who have higher number of “1”s, in order to give them higher chance while selecting parents to apply operations.

*def one\_pt\_cross(p1, p2)* ⇒ performs crossover on two parent individuals and returns two new child individual. It is one point crossover, which randomly chooses crossover point over range of length of individual, then applies crossover operation from this point. One of the new children will have parts before this point, another one will have parts after this one.

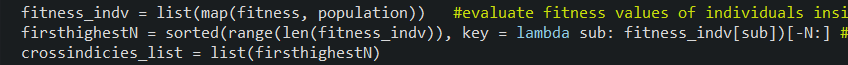
def crossover(pop, cross, cx\_prob) ⇒ performs crossover based on probability. Links to other crossover function. We perform crossover operation with probability which was defined earlier.

1. Procedures

First we create initial population by *create\_pop* function.Then we check how any individuals inside initial populations contains only “1”. We store number of these individuals to *oneMAXbefore*. We do this, in order to compare this number with number which will be generated after evaluation has been carried out:



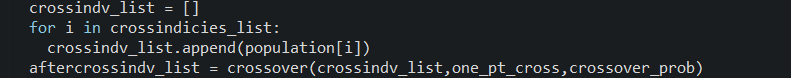
Then our evaluation starts with for loop*(for G in range (50):)*. Inside loop, first we evaluate fitness values of individuals inside population and then put them inside list named *fitness\_indv*. Then we sort this list by selecting indices of first highest N numbers, and assigning it to *firsthighestN*.Then we make list named *crossindicies\_list* out of *firsthighestN*. Remember that, inside this list, there will be stored indice numbers that indicates that inside population array, individuals having those indice numbers, will have the highest fitness values. And obviously, those individuals will be used for crossover operation, therefore, we named list as *crossindicies\_list:*

**

==================================================

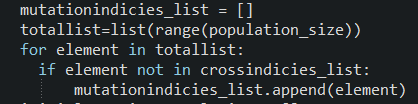
Then we create list named *crossindv\_list,* which will be used to store individuals with highest N(0.6\*population\_size) fitness values. We use for loop, to take indices by traversing *crossindicies\_list,* then using those indices we find individuals from population and insert those individuals to *crossindv\_list*.

Then we apply crossover operation to this list and have new list named *aftercrossindv\_list:*

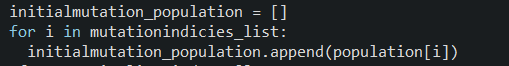


==================================================

Then we create new list to store indices of individuals for mutation operation. Note that, mutation operation will be applied to all those individuals who did not participate in crossover operation. So, number of individuals having mutation operation will be ***population\_size minus N = 0.4\*population\_size.*** And those indices are found and stored by for loop:

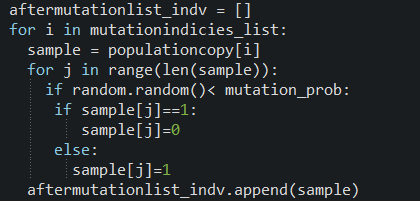


==================================================

Then we use for loop, to take indices by traversing *mutationindicies\_list,* then using those indices we find individuals from population and insert those individuals to list named *initialmutation\_population*: 

==================================================

Then, all the individuals inside *initialmutation\_population* is mutated with some probability and then stored to new list called *aftermutationlist\_indv:*

**

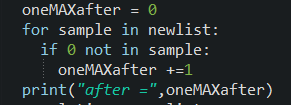
==================================================

Then we create joining *aftercrossindv\_list* and *aftermutationlist\_indv* will give us new population of individuals after single generation iteration, which is named *newlist:*



==================================================

Then we calculate to see how many individuals are oneMAX after iteration:



==================================================

Then we assign population to newlist, to get prepared for next generation:

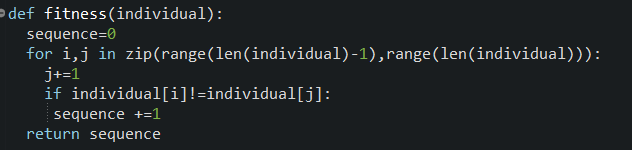


*Conclusion: After each generation, we print oneMAXafter value, which tells us how many individuals are oneMAX. Algorithm works fine, I guess.Note that our algorithm converges to number* ***60****, after enough generations, in our case 100 Generation, with crossover\_prob=0.8, mutation\_prob=0.1.* ***However****, If we set 50 Generation and crossover\_prob = 0.4, we will see that # of oneMAX* ***will not exceed 50.***

***Alternating 1s and 0s***

1. Functions

All the functions will be same as in oneMAX problem, except fitness function. Our fitness function will assign fitness value to individual based on number of alternating “1”s and “0”s:



1. Procedures

All the procedures are also same as in oneMAX problem. There might be very tiny changes.

*Conclusion: I tried different crossover, mutation probabilities and population and generation size. But could not find stable result or convergance of algorithm, however in all cases found individuals with alternating 1s and 0s. But number of these individuals vary in each run.*