1. Solution Approach (Genetic Algorithms)

The genetic algorithm workflow consists of following main steps:
(a) Defining/Initializing the problem instance (e.g. users, servers, server capacity, constraints etc.)
(b) Defining the objective function to be maximized/minimized. Also known as Fitness in GA lingo
(c) Individual (aka Chromosome aka solution) representation & population initialization
(d) Selection Strategy– Based on the fitness of individuals
(e) Crossover Strategy – How to do reproduction of parents for next generation
(f) Mutation Strategy – How to do some random changes in an individual/chromosome

Explanation of Solution (Assume num_users = N, num_servers = M)

(1.a) Defining/Initializing the problem instance

- set whatever users/servers etc. you want on **lines 5 – 23 in configuration_users_servers.py**
- whatever number of users & servers you set, select or set appropriate parameters for genetic algorithm computation in the if-else logic on **line 29/39/49 in configuration_users_servers.py**
- constraint (eqn 1.3 in the pdf) is defined on **line 69 in UserServerResourceAllocation.py**

(1.b) Objective/Fitness function (to judge a solution/individual's quality)

The objective function is given in eqn (1.1) in the provided pdf. But there are two constraints also. So to get an optimal solution we define a hybrid objective function:

- defined as eqn (1.1) – penalty*constraint_violations
- this means that when we don't have any constraint violations we'll not be penalized & our hybrid_objective will be maximum of true objective
- the constraint eqn 1.3 is a hard constraint which can not be violated
- the constraint 1.2 is automatically followed because the way we represent an individual/solution each user is by default assigned only one server or no server. So there is no violation here. But it's preferable to allocate a user some/any server rather than not allocating him at all. So we define it as a soft constraint with soft penalty etc.

(more comments/illustration included in line 27 in the definition of function getHybridObjectiveFunction in the UserResourceAllocation.py file)

(1.c) Individual (aka Chromosome) representation & population initialization

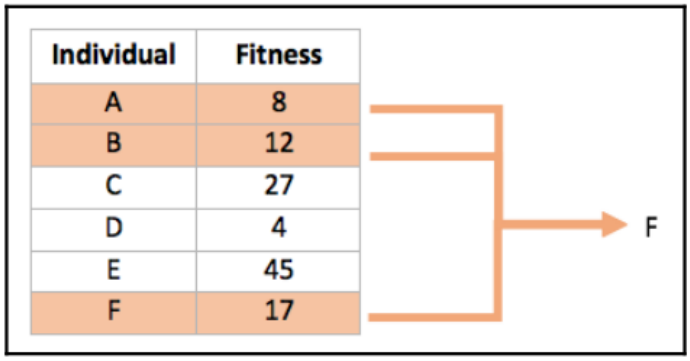
I represent a tentative solution (aka individual or chromosome):

- a list of length N
- each element of the list can take any value between 0 to M.
- each element of the list means which server was allocated to that index (or user). Last number M just means no allocation, other numbers from 0 to M-1 are represent the M servers

(more comments/illustration included in line 46-62 in the main.py file)

(1.d) Selection Strategy (line 75 in main.py)

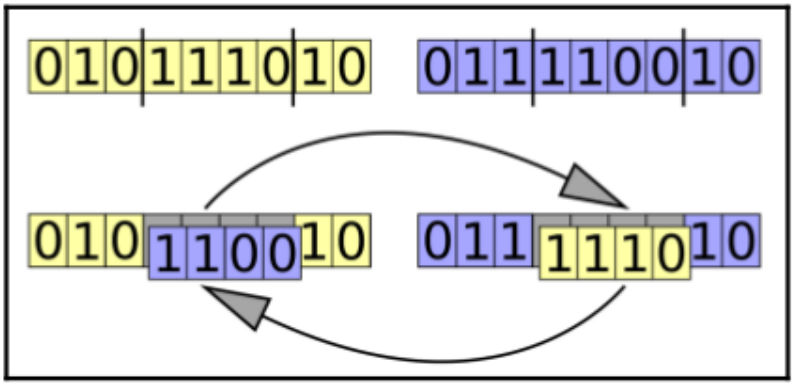
I use a tournament selection strategy of size 3



Tournament selection example with a tournament size of three

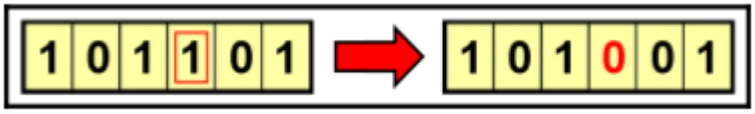
(1.e) Crossover Strategy (line 76 in main.py)

I use a two point crossover strategy. We select any two points in each parent & reproduce by interchanging the parts between them



(1.f) Mutation Strategy (line 77 in main.py)

Just as in a binary mutation (below pic) we interchange between integers 0 to 1 & vice versa,



I use a mutation such that any integer between 0-M can be changed to any other between 0-M as above

2. Parameter Selection Strategy

Most of the parameters come by testing & seeing what gives better solution. Generally experienced modelers use their previous experience. **There is no exact/exclusive thumb rule for parameter selection & it'll vary from problem to problem & modeler to modeler experience.**

I am including some comments about what the parameters signify & how they are chosen.

[*Note: Optimal Solution was found in each Case 1-3 without any constraint violation but with different parameters]

[Case 1: 20 Users, 3 Servers] [Case 2: 1000 Users, 7 Servers] [Case 3: 10000 Users, 100 Servers]

	Comments	Case 1	Case 2	Case 3
POPULATION_SIZE	the optimal solution/individual is going to come from the previous generation of POPULATION_SIZE number of parents. **So more population means better chances of finding an optimal/fittest offspring from that population.** Generally something between 100-500 works for easy/hard problems. **Use more for big/hard problems.** The more the better, but more will take more time. So time/optimality trade-off	100	**500**	**500**
HALL_OF_FAME_SIZE	This basically means how many best individuals to keep as it is without crossover/reproduction in next generation. **This helps is retaining/remembering some good individuals for a longer time.** Generally neither too low nor too high is good & something in range 20-50 works good for population size of 100-500. Needs to be tried & seen	20	30	30
MAX_GENERATIONS	**number of iterations/loops GA will run to find the optimal solution. The more the better but generally around 100-500 works for easy/hard (small/big) problems.** Try & see	50	100	100
TOURNAMENT_SIZE	This means how to select from current most fit individuals for next generation. **TOURNAMENT_SIZE individuals are selected based on objective/fitness value & out of that one is randomly selected. This is repeated POPULATION_SIZE times.** Needed as explained in fig of sec 1.d (selection strategy). Generally 2-5 needs to be tried but **most of the time 2 or 3 works.**	3	3	3
P_CROSSOVER	**Consecutive pairs of individuals form last generation of size POPULATION_SIZE are selected with probability P_CROSSOVER for crossover/reproduction.** Generally .8 or .9 works but any value between 0-1 can be tried. **But the higher the better & preferably around .9-.95** Needed as in Sec 1.e	.9	.9	.9
P_MUTATION	Probability of mutating an individual. **This helps in exploring new solution .** But sometimes too high value can slow down convergence as we will be changing even good individuals too much. Generally .1, .2, .3 works but sometimes might need higher values. **Try higher values like .4, .5 to explore more/new solutions if not getting optimality** Needed as in Sec 1.f	.2	.2	**.5**

IND_PB	**Probability with which each attribute of the individual (chosen for mutation) is mutated.** Generally most common value is 1/length(individual) but something between .01-.1 might also work.	.01	.01	.01
HARD_CONSTRAINT_PENALTY	Used to guide algorithm go in right direction so that no constraints of this sort are violated. **Generally arbitrary high numbers are chosen/tried.** e.g. eqn 1.3 in our case shouldn't be violated. **Increase till no/less violation.**	1000	10000	**10000**
SOFT_CONSTRAINT_PENALTY	Used to guide algorithm go in right direction so that no constraints of this sort are also violated but they can be violated without making solution infeasible. **Generally some number <= HARD_CONSTRAINT_PENALTY is chosen/tried.** e.g. eqn 1.2 in our case can be 0 (it'd be <=1, so it's not violation but but its better to not have 0) which means that user wasn't allocated any server. **Increase till no/less violation.**	100	1000	**1000**

3. Running Code & Analyzing Results on Some Test Cases

Case 1 [20 Users, 3 Servers] [code running time < 5 seconds]

parameters – line 28-37 configuration_users_servers.py

[Optimality Found, No Constraint Violation]

```
============= Summary - Problem/Solution ===============

Number Of Users (20), Number Of Servers (3)

HURRAY !!!!!!!!!!!!!!!!!
Optimal objective function found & No Constraint Violation

Optimal Objective Function Value (Eqn 1.1) =  36829.40031421199

Number Of Soft Constraint Violations (eqn 1.2 - Users allocation limit) = 0

Number Of Hard Constraint Violations (eqn 1.3 - Server Capacity) = 0

Saving Server To User Allocation Scheme to Folder ../output/20Users_3Servers/

Saving User To Server Allocation Scheme to Folder ../output/20Users_3Servers/

============= End Summary ==================
```

Case 2 [1000 Users, 7 Servers] [code running time 130 seconds]

parameters – line 38-47 configuration_users_servers.py

[Optimality Found, No Constraint Violation]

I increased the POPULATION_SIZE & also increased the Penalty compared to Case 1 as I was not getting optimal value with parameters of Case 1

```
============= Summary - Problem/Solution ===============

Number Of Users (1000), Number Of Servers (7)

HURRAY !!!!!!!!!!!!!!!!!
Optimal objective function found & No Constraint Violation

Optimal Objective Function Value (Eqn 1.1) =  2044645.0319289993

Number Of Soft Constraint Violations (eqn 1.2 - Users allocation limit) = 0

Number Of Hard Constraint Violations (eqn 1.3 - Server Capacity) = 0

Saving Server To User Allocation Scheme to Folder ../output/1000Users_7Servers/

Saving User To Server Allocation Scheme to Folder ../output/1000Users_7Servers/

============= End Summary ==================
```

Case 3 [10000 Users, 100 Servers] [code running time 1260 seconds ~ 21 Minutes]

parameters – line 48 - 57 configuration_users_servers.py

[Optimality Found, No Constraint Violation]

I increased the P_MUTATION from .2 used previously in Case 1,2 to .5

```
============= Summary - Problem/Solution ===============

Number Of Users (10000), Number Of Servers (100)

HURRAY !!!!!!!!!!!!!!!!!
Optimal objective function found & No Constraint Violation

Optimal Objective Function Value (Eqn 1.1) =  20083932.494745582

Number Of Soft Constraint Violations (eqn 1.2 - Users allocation limit) = 0

Number Of Hard Constraint Violations (eqn 1.3 - Server Capacity) = 0

Saving Server To User Allocation Scheme to Folder ../output/10000Users_100Servers/

Saving User To Server Allocation Scheme to Folder ../output/10000Users_100Servers/

============= End Summary ==================
```