Project by:

Rahul Verma (20175183)

Dishant Mewada (17079349)

(0,1,2) Knapsack Problem

Problem – Given a set of objects (each having its own weight and value), find the objects with the maximum value.

Limitations – Sum of weight of all objects not exceeding 1000. Random allocation of values of objects in between 1-10 and weighs in between 1-100.

Constraints – Given a list of potential solutions, object is randomly selected once, twice or not at all.

Objective – Maximizing the function.

Methodology used – DEAP framework using simple Genetic Algorithms with Tournament and Roulette Wheel selections.

There are various ways of addressing this problem, like dynamic programming (it is time consuming process), but we chose genetic algorithm as a search algorithm to find solutions to this problem.

Before we dive deeper, let us try to understand the entire genetic lifecycle:

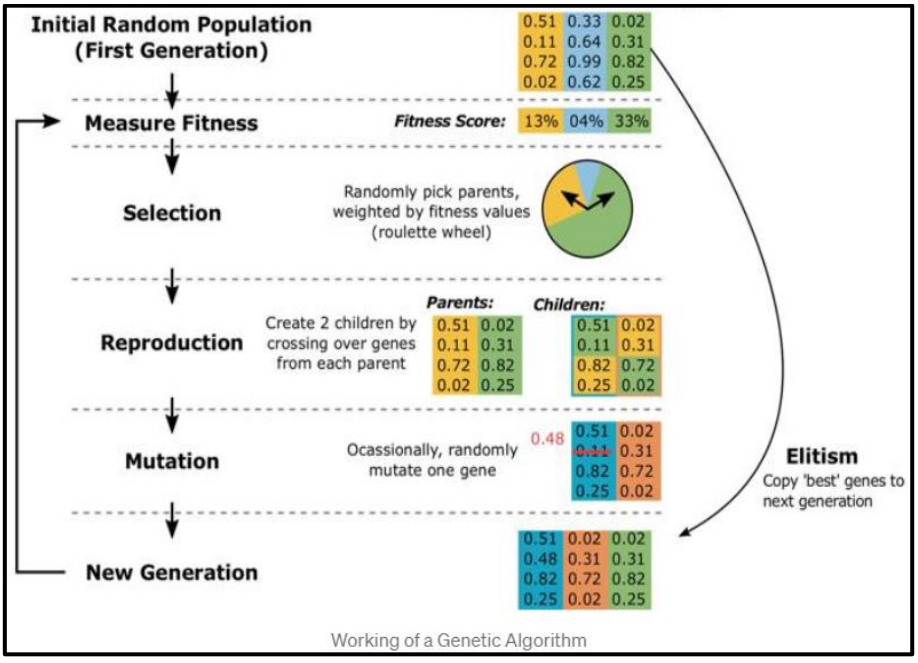


Fig: Life-cycle of genetic algorithm

Initialization – The population representing our solution is initialized in binary format. There are various ways to represent a population (also known as Schemata), however due to simplicity of our problem, we chose binary representation.

Fitness Score – It is the score of every individual measuring how fit an individual is. This function operates at the genetic level. This follows the philosophy of 'survival of the fittest'.

Selection – This is the process of selecting individuals to undergo mating, there are few selection methods to choose from, depending upon the complexity / problem types.

Reproduction – Individuals that had undergone mating produce off-springs, sharing gene from each parent A and B.

Mutation – Flipping the value of one gene on probability basis to generate a completely new set of genes. This is done to introduce variety.

Elitism – Optional, but used to propagate fittest individuals to next level.

This cycle continues until the we reach one of the limits specified, one we normally use, is number of generations.

Problem Specific Constants

Representing this problem using Genetic Algorithm's framework, where terminologies represented are as follows:

- **INDIVIDUAL_LENGTH = 100 –** This is the length of the object, which is 100 in this case.
- **POPULATION_SIZE = 500 –** 500 such individuals required for search operation, this is a good and recommended number for experiment.
- **P_CROSSOVER = 0.9 –** Crossover is a process of combining genes from two parent to create an offspring. It can be done either using one point or multi-point.
- **P_MUTATION = 0.1** – Mutation refers to the process of flipping of bits to create a different type of gene to have a variety in population. It is recommended to use a very small value for mutation.
- **MAX_GENERATIONS = 100** – Could be thought of number of iterations. Like POPULATION_SIZE, this is a good number to experiment with.
- **NBR_ITEMS = 100 –** This is the number of items.
- **MAX_WEIGHT = 1000 –** Constraint as defined in 'Constraints' section in the beginning.
- **RANDOM_SEED = 42 –** Random seed is used to deliver same results in case of multiple runs, thus having a control over stochasticity of genetic algorithms.

Fitness Function

```
def MaxFitness(individual):
    return sum(individual),  # return a tuple
```

Fitness function is quite simple, it takes an individual and returns the sum of all the values.

Knapsack Function

```
def knapsack(individual):
    weight = 0.0
    value = 0.0
    for i in range(NBR_ITEMS):
        value += items[i][0]*individual[i]
        weight += items[i][1]*individual[i]
    if len(individual) > NBR_ITEMS or weight > MAX_WEIGHT:
        return (value-(weight-MAX_WEIGHT)*2),weight
    else:
        return (value), weight
```

Knapsack function takes an individual as an argument and then we have weight and value initialized as 0. The variable 'i' loops through the length of the items, which is 100 and updates the weight and value in each case. We then have a condition, where if length of an individual is greater than 100 or weight of an object is more than 1000, then the individual is penalized, the penalty function is the difference of mean of squared error. In the end, the values and weights are returned. Here, penalty function is really important to control the behavior of individual values.

Observation 1: Tournament Selection

```
# create an operator that randomly returns 0,1, or 2:
toolbox.register("zeroOrOneOrTwo", random.randint, 1 , 2)
```

Random Operator Generator: This is the function used to randomly assign values (0,1 or 2) to our individuals.

Tournament Selection Toolbox:

```
# Tournament selection with tournament size of 3:
toolbox.register("select", tools.selTournament, tournsize=4)
```

Tournament selection is a selection mechanism in genetic algorithm, where two individuals (parents) are chosen (to create an offspring) and they go into a tournament, the individual with the highest fitness value wins the tournament and then the next pair is chosen to undergo a tournament and this happens until the number of offspring generated replaces the original number of population. We have used tournament size of 4 in this case, recommended range is between 3-8. Below is a visual representation of Tournament process for a better understanding:

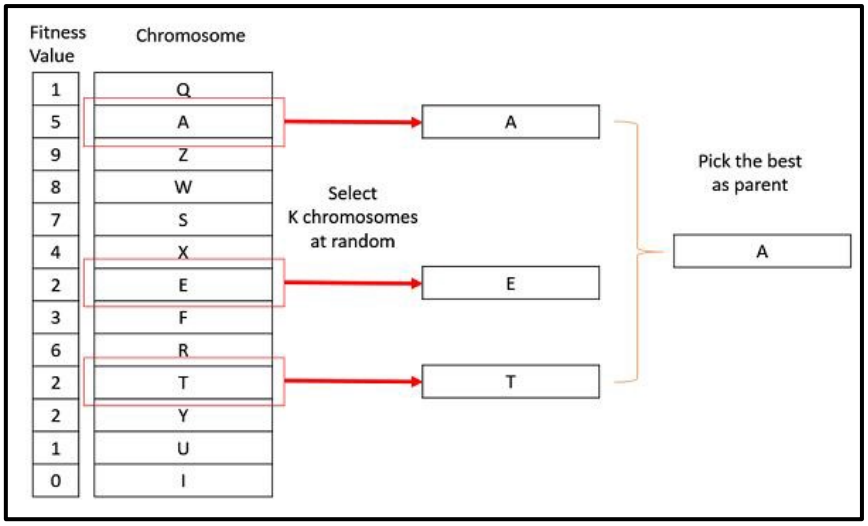
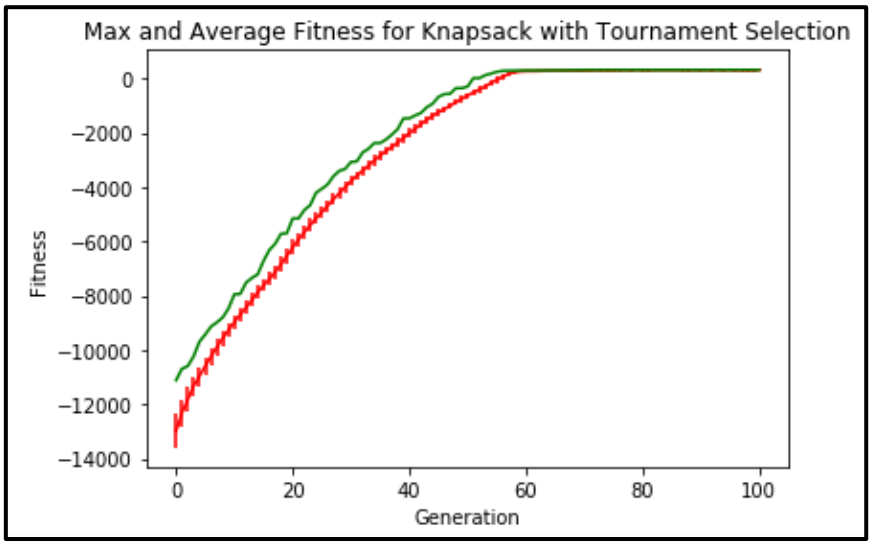


Fig: Tournament Selection

Using tournament selection, we were able to find objects with a total value of 321 and combined weight not exceeding 1000. Below graph shows, how average value of function was increasing from generation to generation.



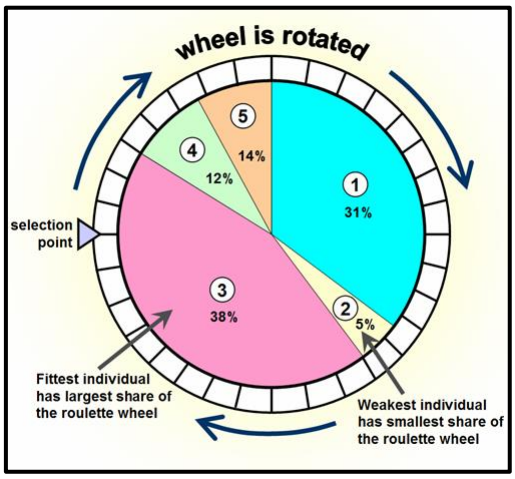
In the below image, the best ever individual found can be seen along with the fitness value and weight.

```
XXBest Ever Individual =  [1, 1, 0, 0, 2, 1, 2, 1, 0, 2, 1, 0, 0, 0, 0, 0, 0,
1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 2,
0, 0, 1, 0, 0, 0, 1, 0, 0, 2, 0, 1, 2, 0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2, 1, 2, 0, 1, 0, 2,
0, 2, 0, 1, 0]
XXFitness:  0   (321.0, 1000.0)
```

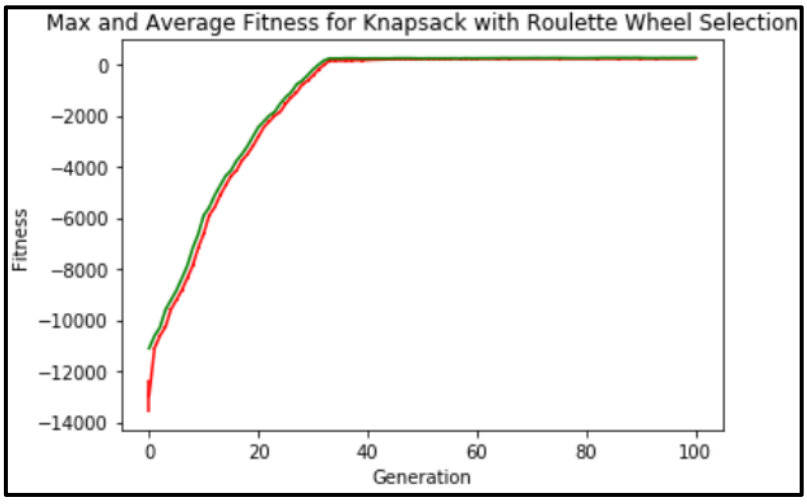
Observation 2: Roulette Wheel Selection

Using the same parameters as above, we tried Roulette Wheel Selection, which is another way of selecting individuals (parents) to create an offspring. The individuals are placed on an artificial roulette wheel and are then selected probabilistically. This model is tend to be bias, as the individual with the highest fitness value is always selected as a parent to create an offspring, since that individual shares a big portion of wheel, the probability of selection is quite obvious.

Below is a visual representation of Roulette Wheel for a better understanding:



Using Roulette Wheel Selection, we were able to find objects with a total value of 264 and combined weight not exceeding 1000, but maximum weight we found was = 965. Below graph shows, how average value of function was increasing from generation to generation.



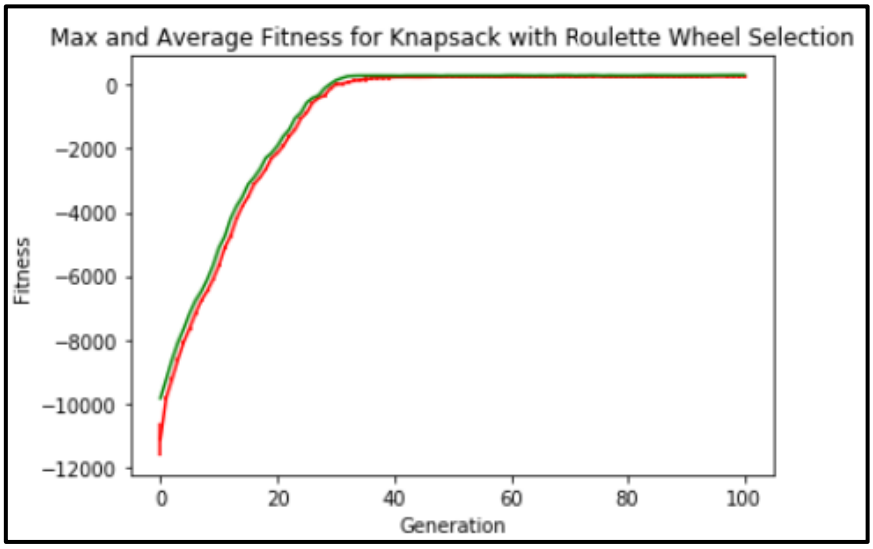
In the below image, the best ever individual found can be seen along with the fitness value and weight.

```
XXBest Ever Individual =  [1, 1, 2, 0, 2, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
2, 0, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
1, 0, 1, 0, 2, 1, 0, 0, 0, 1, 0, 1, 2, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 2, 1, 1, 0, 0, 0, 1,
0, 2, 0, 0, 1]
XXFitness:  0   (264.0, 965.0)
```

Conclusion: When compared the results of both the experiments, the following table gives an insight about what was delivered by each experiment.

Selection Method	Tournament	Roulette Wheel
Value	321	264
Weight	1000	965

We were not convinced with the output produced by Roulette Wheel Selection, as there is a quite good gap (between the value ranges produced by each experiment), we decided to fine tune the model and tried to get as close as we can (maximize the function). This was achieved by changing the random seed from 42 (used in last experiment) to 40 only for Roulette Wheel Selection and we ended up with the following conclusions:



In the below image, the best ever individual found can be seen along with the fitness value and weight and finally in the table in the end, we compare all the values achieved so far. We were able to maximize the fitness function and see improvements of over 38 and 27 for value and weight respectively.

```
XXBest Ever Individual =  [0, 0, 0, 2, 1, 1, 0, 2, 0, 1, 0, 2, 0, 0, 1, 2, 0,
0, 0, 0, 0, 1, 1, 0, 0, 0, 2, 0, 1, 0, 2, 1, 0, 0, 0, 2, 0, 2, 0, 0, 0, 1, 0,
2, 0, 1, 0, 2, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 2, 0, 0, 0, 1, 0, 0, 1, 0, 0,
0, 0, 1, 1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
0, 0, 0, 0, 1]
XXFitness:  0   (302.0, 992.0)
```

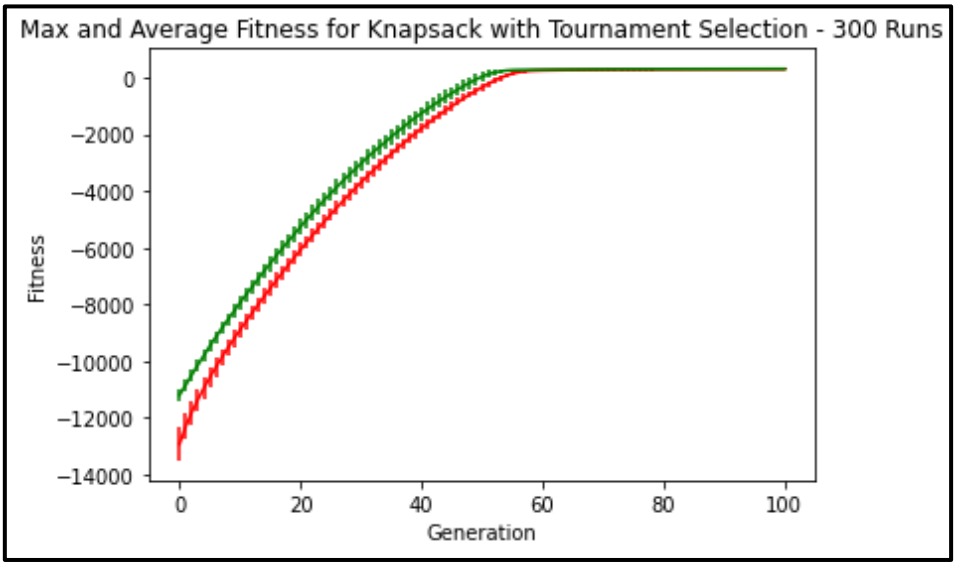
Selection Method	Tournament	Roulette Wheel
Value	321	302 **(Δ = 38)**
Weight	1000	992 **(Δ = 27)**

Final observations: Multiple Runs

So far, we have tried running this experiment only one time. In order to understand, if the results obtained previously were luck by chance or really there was some limitations, we ran the experiments with multiple number of times to build our confidence and convince ourselves. We are using same problem specific constants as before, but now we have one new parameter, number of runs **(N_RUBS)** = 300.

Observation 3: Tournament Selection with multiple runs

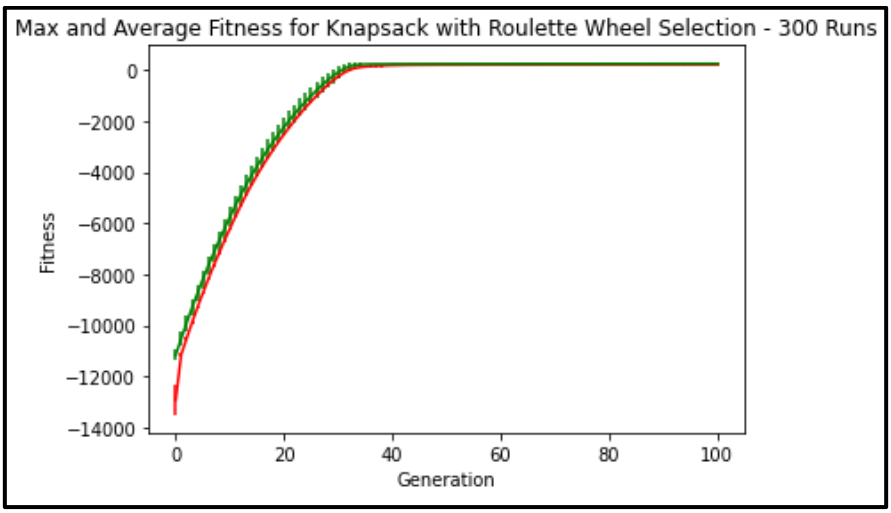
We ran the experiment using Tournament Selection with all the parameters being same as the last experiment.



Results: We were able to achieve a max value from all the runs as : 334 and overall weight as : 997.

Observation 4: Roulette Wheel Selection with multiple runs

We ran the experiment using Roulette Wheel Selection with Random Seed = 42 and Number of Runs = 300.



Results: We were able to achieve a max value from all the runs as : 319 and overall weight as : 990.

Selection Method (runs = 300)	Tournament	Roulette Wheel
Value	334	319
Weight	997	990

Final Conclusion: We started with a single run and were able to find perfect weight = 1000 in case of tournament selection along with a max value = 321. Later, we improvised Roulette Wheel Selection to maximize the function. Finally, we ran both the experiments under multiple runs which produced the best results so far when compared to each observations.

There are two key takeaways for us from these experiments:

1. **Tournament Selection was better than Roulette Wheel Selection : This could be due to always selection of fittest individual.**
2. **A minimum of 302 as a value was obtained and a maximum of 334 was obtained, however there was a trade-off – When we have max value, we don't have weight = 1000, it is some what in the range of 990-997, which may not be an best solution.**

Supporting codes:

Single Run Tournament	 20175183_17079349 _ECProject_Tournam
Single Run Roulette Wheel	 20175183_17079349 _ECProject_RouletteV
Multi-Run Tournament	 20175183_17079349 _ECEProject_Tournan
Multi-Run Roulette Wheel	 20175183_17079349 _ECEProject_Roulette