# Genetic Algorithm: The Knapsack Problem

*Pruthvi Raj Muddapuram, Vineeth Kashyap Kosiganti*

*Final Draft: April 15, 2018*

---

## Problem

We have created a Genetic Algorithm to solve the Knapsack problem. The problem states: Consider a thief breaks into a home to rob and he carries an old knapsack with limited capacity. Now the thief should figure out what items to carry along with him depending on the item's weight and value. He could either choose tiny but valuable items like jewelry or he could go for items which have less value but are heavy like chairs and tables.

## Implementation

Genetic Code: Each gene is an array of integers representing the items which are placed into the knapsack.

Gene Expression: Each item has a particular weight and value. The item weights are stored in an ArrayList.

Fitness Function: As we are looking for a combination of items with the highest value, the weights and the values of each member in the population are taken into account. We compare the total weight of each member, W(m), and compare it with the knapsack's capacity, W(k). If W(k) >= ∑W(m) then the fitness, F(g) equals the sum of values of the members, V(m) of the population i.e. F(g) = ∑V(m). If W(k) < ∑W(m), i.e. if the sum of weights of members is more than the capacity of the knapsack, fitness is taken as 0 for that population.

Crossing Over: In the process of creating the next generation, we take two members of the parent population with a given crossover probability. A random function is used to pick random spots from both the parent genes to perform the crossover and as a result, the new breed is formed. Again, the same steps are repeated to obtain the fitness of all such new breeds and the best fitness is taken into consideration for the next generation.

Example: Suppose that we take 5 items into consideration, item1 with weight as 1 lb, item2 with weight as 2 lb and so on. Let's take a Knapsack with a capacity of 10 lb. Now we build a population with a different combination of these five items and randomly select two parent arrays from the population. Say,

Parent 1 = 1 1 1 0 0 (items 1, 2, 3)

Parent 2 = 0 0 0 0 1 (item 5)

Now we generate a random integer between 0 and 4 (no. of items - 1) and name it crossover point. Let's take two crossover points as 1, 2 for Children 1 and 2 respectively.

Now, we build the child gene array from both the parents by taking elements from indexes 0 to crossover point from Parent 1 and crossover point + 1 to the end from Parent 2 i.e.,

Child 1 = 1 1 0 0 1.

Child 2 = 1 1 1 0 1.

Now the weight of Child 1 and 2 are 8 lb and 11 lb respectively. So according to the fitness function, as the Child 1 is heavier than the Knapsack, it is culled and only Child 2 is carried forward to the next generation. The fitness of the Child 2 is the sum of the values of the elements in it.

After the initial generation we move on to make further generations and breed population and again calculate the fitness of that generation's new population and get the **best FITNESS** value and store it in an ArrayList and sort it using a normal **sort** and we repeat this process till the maximum number of generations are reached or if "mean fitness value" is repeated 3 times consecutively, whichever comes first.

Results:

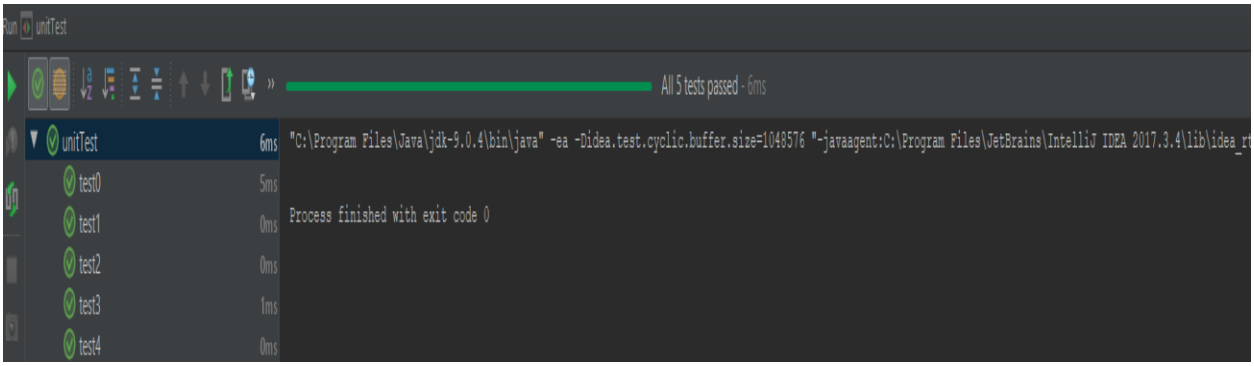
```
run:
Enter the number of items:
5
Enter the value of item 1:
1
Enter the weight of item 1:
1
Enter the value of item 2:
2
Enter the weight of item 2:
2
Enter the value of item 3:
3
Enter the weight of item 3:
3
Enter the value of item 4:
4
Enter the weight of item 4:
4
Enter the value of item 5:
5
Enter the weight of item 5:
5
Enter the knapsack capacity:
9
Enter the population size:
7
Enter the maximum number of generations:
6
Enter the crossover probability:
0.5
Enter the mutation probability:
0.2
```

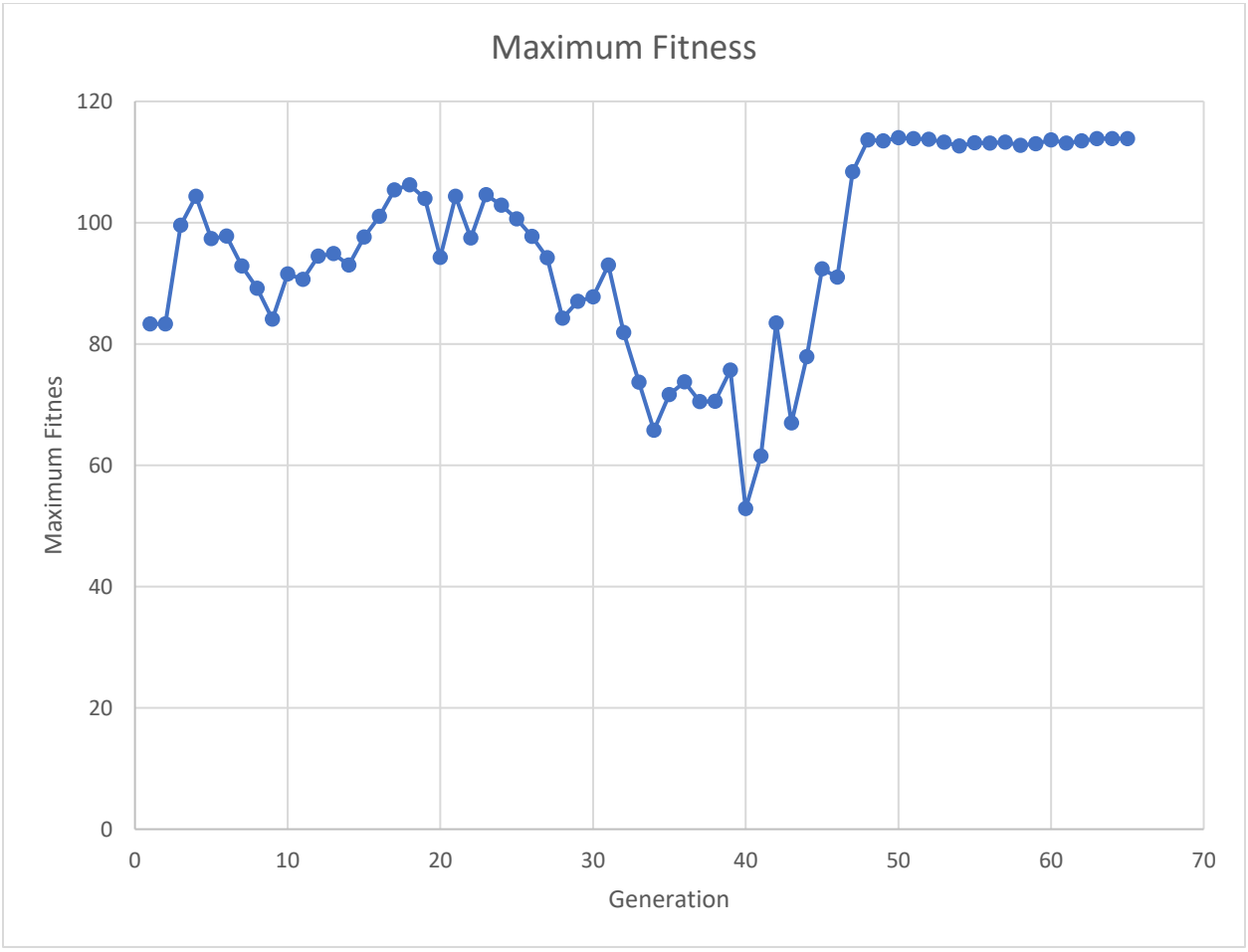
```
Initial Generation:
====================
Population:
1 - 11100
2 - 00111
3 - 00100
4 - 11010
5 - 11000
6 - 01101
7 - 11011

Fitness:
1 - 6.0
2 - 0.0
3 - 3.0
4 - 7.0
5 - 3.0
6 - 0.0
7 - 0.0

Best solution of initial generation: 11010
Mean fitness of initial generation: 2.7142857142857144
Fitness score of best solution of initial generation: 7.0
```

```
Generation 2:
=============
Population:
1 - 11010
2 - 11100
3 - 11010
4 - 11010
5 - 11010
6 - 11100
7 - 11010

Fitness:
1 - 6.0
2 - 0.0
3 - 3.0
4 - 7.0
5 - 3.0
6 - 0.0
7 - 0.0

Best solution of generation 2: 11010
Mean fitness of generation: 2.7142857142857144
Fitness score of best solution of generation 2: 7.0
Clonning occurred 1 times
Crossover occurred 2 times
Newly evolved number in Generation 2 is 3
Discarded from the generation 4
Mutation did not occur
```

```
Generation 6:
==============
Population:
1 - 11010
2 - 11010
3 - 11010
4 - 11010
5 - 11010
6 - 11010
7 - 11010

Fitness:
1 - 7.0
2 - 7.0
3 - 7.0
4 - 7.0
5 - 7.0
6 - 7.0
7 - 7.0

Best solution of generation 6: 11010
Mean fitness of generation: 7.0
Fitness score of best solution of generation 6: 7.0
Clonning occurred 3 times
Crossover occurred 0 times
Newly evolved number in Generation 6 is 3
Discarded from the generation 4
Mutation did not occur

Optimal list of items to include in knapsack:
1 2 4 Best generation is 0
Best fitness is 7.0
BUILD SUCCESSFUL (total time: 19 seconds)
```

```
Discarded from the generation 4
Mutation did not occur

Generation 6:
=============
Population:
1 - 11010
2 - 11010
3 - 11010
4 - 11010
5 - 11010
6 - 11010
7 - 11010

Fitness:
1 - 7.0
2 - 7.0
3 - 7.0
4 - 7.0
5 - 7.0
6 - 7.0
7 - 7.0

Best solution of generation 6: 11010
Mean fitness of generation: 7.0
Fitness score of best solution of generation 6: 7.0
Clonning occurred 3 times
Crossover occurred 0 times
Newly evolved number in Generation 6 is 3
Discarded from the generation 4
Mutation did not occur

Optimal list of items to include in knapsack:
1 2 4 Best generation is 0
Best fitness is 7.0
```


unitTest
All 5 tests passed - 6ms
unitTest 6ms
test0 5ms
test1 0ms
test2 0ms
test3 1ms
test4 0ms
"C:\Program Files\Java\jdk-9.0.4\bin\java" -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2017.3.4\lib\idea_r
Process finished with exit code 0


Maximum Fitness
Maximum Fitnes
Generation
0
20
40
60
80
100
120
0
10
20
30
40
50
60
70