0/1 Knapsack Using Genetic Algorithm

Problem Statement

In a Knapsack problem you are given a sack which can hold a maximum weight W along with N items each with a weight and a value. The goal is to select a set of items from given object such that their total weight is less than or equal to the knapsack weight W and at the same time the total value is maximized. There is only one quantity of each object.

Implementation Summary

To solve the problem using Genetic Algorithm we have represented a chromosome with a String made up of 1s and 0s, where each character in the string is considered as a gene. 1/0 represents whether an item is present in the sack or not, respectively. Item class has a weight and value. Individual class holds a chromosome and the total weight and total value of the Individual. The total weight and total value of an Individual are the phenotypes which are depent upon the genes the Individual inherited. The fitness of each Individual is measured by comparing its total weight with the maximum weight(W) the knapsack can hold. An Individual is considered fit only if its total weight is less than or equal to W. At each generation, the fitness of each Individual is checked and unfit Individuals are culled. The fit Individuals are carried on to the next generation. The culling process can be performed by parallel processing or on a single thread, which is specified before the execution start via a parameterized constructor. New Individual for next generation are then bred by mutating fit Individuals from current generation and/or by crossover between two fit Individual from current generation. The program first tries to keep the next generation diverse by adding new Individuals only if no Individual with same chromosome is already present in the population. If it fails to find unique Individual then it fills up the population with random crossover without checks. The fittest Individuals amongst the fit Individual from each generation is found and stored. The iteration of next generation stops when there is no progress detected between last three generations. This is being achieved by comparing the mean values of last the generations. If the difference between each pair of last three values is less than the delta then it is assumed no further progress can be made and the iteration of making next generation stops. At this point, the fittest Individuals from each generation are compared and the one with maximum total value is chosen as the solution.

Execution Steps:

Step 1: Input Knapsack capacity, number of items along with their values and weights, and the population size.

Step 2: Generate Individual with random chromosome

Step 3: Calculate the fitness of each Individual and cull the unfit ones.
If no Individual is fit in current generation go to step 2
else, sort the fit Individuals based on their value and make a note of the fittest amongst them.

Step 4: Check if any progress has been made since last three generations
If yes, continue to Step 5
If no, go to step 7

Step 5: Perform mutation and crossover between the fit Individuals for next generation

Step 6: Go to step 3

Step 7: Sort the best Individuals from each generation based on their value.
The one with highest value is our solution

Findings/Observations

Case 1:

Delta: 0.01
Knapsack capacity: 4000
Number of Items: 100
Varying population size:

Population Size	**Number of generations**	**Solution**	
		Value	**Weight**
10	**19**	**3346**	**3354**
50	**17**	**3580**	**3320**
100	**20**	**3743**	**3257**
200	**15**	**3759**	**3941**
500	**38**	**4728**	**3772**
1000	**41**	**4590**	**3710**
2000	**41**	**4719**	**3681**
5000	**40**	**4760**	**3940**
10000	**43**	**4763**	**3937**

Observation: As the population size increased, the weight increases and is closer to the desired value of 4000. Also, the number of generations increased with increase in population size.

Case 2:

Delta: 0.01
Number of Items: 100
Population size: 1000
Varying knapsack capacity

Knapsack Capacity	Number of generations	Solution	
		Value	Weight
10	267	490	10
50	2475	955	45
150	206	1564	136
350	293	2275	325
750	118	3053	747
1500	57	3053	1447
3000	19	4025	2975
6000	34	4538	3762

Observation: Keeping the population size fixed at 1000 and varying the knapsack capacity, the number of generations overall decreases.

Case 3:

Delta: 0.001
Knapsack capacity: 5050
Number of Items: 100
Varying population size:

Population Size	Number of generations	Solution	
		Value	Weight
10	193	4795	4295
50	34	3854	3519
100	81	4687	4302
200	112	5050	5050
500	99	5049	4950
1000	107	5049	4950
2000	84	5047	4851
5000	84	5049	4950
10000	84	5050	5050

Observation: Comparing these results with case 1 with lower delta, the weight of the solution is closer to the optimal solution. But the generations are much more than in case 1.

Case 4:

Delta: 0.001
Number of Items: 100
Population size: 1000
Varying knapsack capacity:

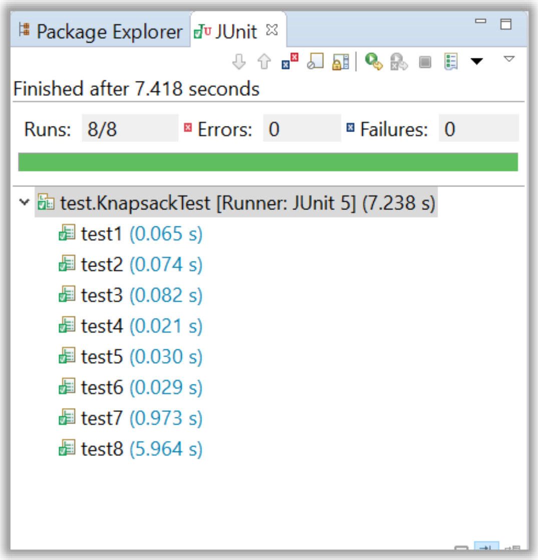
Knapsack Capacity	Number of generations	Solution	
		Value	Weight
10	297	394	10
50	569	864	45
150	817	1480	136
350	697	2200	325
750	716	3097	741
1500	513	3969	1485
3000	448	4748	2928
6000	107	5049	4950

Observation: With lower delta the number of generations are higher but the solution is even closer to the supplied knapsack capacity

Conclusion:
The population size and the delta play a key role in finding the optimum solution but genetic algorithm does not guarantee to find the optimum solution. This can be seen from above cases. Thus, it is important not to have a small population size compared to the number of items. But at the same time increasing the population size after a threshold value did not make much difference.

The parallel execution takes lesser generations to come to a solution and most of the time, the solution obtained is better than what is achieved from single threaded. Parallel was faster in execution time as compared to single threaded which is as expected but it comes at the cost of memory. The memory required for each iteration is N times more (N being the population size) since a copy of the list is maintained made every generation.

Screenshots:

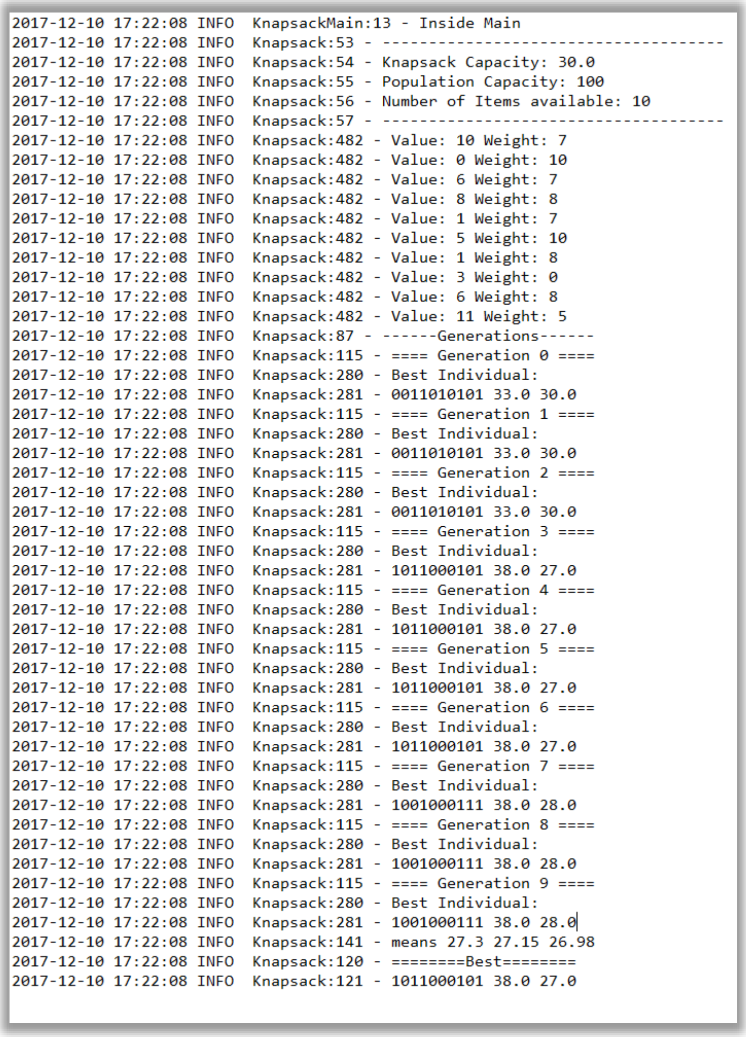


```
Finished after 7.418 seconds

Runs:  8/8    Errors:  0    Failures:  0

test.KnapsackTest [Runner: JUnit 5] (7.238 s)
    test1 (0.065 s)
    test2 (0.074 s)
    test3 (0.082 s)
    test4 (0.021 s)
    test5 (0.030 s)
    test6 (0.029 s)
    test7 (0.973 s)
    test8 (5.964 s)
```

Figure 1: All test cases passed successfully



```
2017-12-10 17:22:08 INFO  KnapsackMain:13 - Inside Main
2017-12-10 17:22:08 INFO  Knapsack:53 - -------------------------------------
2017-12-10 17:22:08 INFO  Knapsack:54 - Knapsack Capacity: 30.0
2017-12-10 17:22:08 INFO  Knapsack:55 - Population Capacity: 100
2017-12-10 17:22:08 INFO  Knapsack:56 - Number of Items available: 10
2017-12-10 17:22:08 INFO  Knapsack:57 - -------------------------------------
2017-12-10 17:22:08 INFO  Knapsack:482 - Value: 10 Weight: 7
2017-12-10 17:22:08 INFO  Knapsack:482 - Value: 0 Weight: 10
2017-12-10 17:22:08 INFO  Knapsack:482 - Value: 6 Weight: 7
2017-12-10 17:22:08 INFO  Knapsack:482 - Value: 8 Weight: 8
2017-12-10 17:22:08 INFO  Knapsack:482 - Value: 1 Weight: 7
2017-12-10 17:22:08 INFO  Knapsack:482 - Value: 5 Weight: 10
2017-12-10 17:22:08 INFO  Knapsack:482 - Value: 1 Weight: 8
2017-12-10 17:22:08 INFO  Knapsack:482 - Value: 3 Weight: 0
2017-12-10 17:22:08 INFO  Knapsack:482 - Value: 6 Weight: 8
2017-12-10 17:22:08 INFO  Knapsack:482 - Value: 11 Weight: 5
2017-12-10 17:22:08 INFO  Knapsack:87 - ------Generations------
2017-12-10 17:22:08 INFO  Knapsack:115 - ==== Generation 0 ====
2017-12-10 17:22:08 INFO  Knapsack:280 - Best Individual:
2017-12-10 17:22:08 INFO  Knapsack:281 - 0011010101 33.0 30.0
2017-12-10 17:22:08 INFO  Knapsack:115 - ==== Generation 1 ====
2017-12-10 17:22:08 INFO  Knapsack:280 - Best Individual:
2017-12-10 17:22:08 INFO  Knapsack:281 - 0011010101 33.0 30.0
2017-12-10 17:22:08 INFO  Knapsack:115 - ==== Generation 2 ====
2017-12-10 17:22:08 INFO  Knapsack:280 - Best Individual:
2017-12-10 17:22:08 INFO  Knapsack:281 - 0011010101 33.0 30.0
2017-12-10 17:22:08 INFO  Knapsack:115 - ==== Generation 3 ====
2017-12-10 17:22:08 INFO  Knapsack:280 - Best Individual:
2017-12-10 17:22:08 INFO  Knapsack:281 - 1011000101 38.0 27.0
2017-12-10 17:22:08 INFO  Knapsack:115 - ==== Generation 4 ====
2017-12-10 17:22:08 INFO  Knapsack:280 - Best Individual:
2017-12-10 17:22:08 INFO  Knapsack:281 - 1011000101 38.0 27.0
2017-12-10 17:22:08 INFO  Knapsack:115 - ==== Generation 5 ====
2017-12-10 17:22:08 INFO  Knapsack:280 - Best Individual:
2017-12-10 17:22:08 INFO  Knapsack:281 - 1011000101 38.0 27.0
2017-12-10 17:22:08 INFO  Knapsack:115 - ==== Generation 6 ====
2017-12-10 17:22:08 INFO  Knapsack:280 - Best Individual:
2017-12-10 17:22:08 INFO  Knapsack:281 - 1011000101 38.0 27.0
2017-12-10 17:22:08 INFO  Knapsack:115 - ==== Generation 7 ====
2017-12-10 17:22:08 INFO  Knapsack:280 - Best Individual:
2017-12-10 17:22:08 INFO  Knapsack:281 - 1001000111 38.0 28.0
2017-12-10 17:22:08 INFO  Knapsack:115 - ==== Generation 8 ====
2017-12-10 17:22:08 INFO  Knapsack:280 - Best Individual:
2017-12-10 17:22:08 INFO  Knapsack:281 - 1001000111 38.0 28.0
2017-12-10 17:22:08 INFO  Knapsack:115 - ==== Generation 9 ====
2017-12-10 17:22:08 INFO  Knapsack:280 - Best Individual:
2017-12-10 17:22:08 INFO  Knapsack:281 - 1001000111 38.0 28.0
2017-12-10 17:22:08 INFO  Knapsack:141 - means 27.3 27.15 26.98
2017-12-10 17:22:08 INFO  Knapsack:120 - ========Best========
2017-12-10 17:22:08 INFO  Knapsack:121 - 1011000101 38.0 27.0
```

Figure 2: Output on run