PROGRAM STRUCTURES AND ALGORITHM

INFO 6205

KNAPSACK PROBLEM USING GENETIC ALGORITHM

Professor: Robin Hillyard

By

Ajjunesh Raju

Pooja Narasimhan

Team 18

PROBLEM STATEMENT:

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.[1]

We have optimized Knapsack Problem using Genetic Algorithm.

APPROACH:

A genetic algorithm (GA) uses natural selection process that is similar to biological evalution to optimize constained and unconstrained problems. Population of each solution is modified and children for the next generation by their parents. Optimal solution achieved over successive generations.

Initial Setup:

- The number of items is set to 10. The value of weight and volume are assigned to each item.
- The capcity of the sack is set to 50 and volume is set to 2500.
- The crossover probabity and mutation probability is set to 0.5 and 0.03 respectively.
- The following steps are repeated for 100 generation until the stoping criteria is met.

Step 1: Generating the population

The chromosomes are generated based on the based on the population size. Each bit in the chromosome is assigned 0 or 1 untill the population length is reached. Here we have chosen population size as 10.

Step 2: Breeding

Two chromosomes are randomly chosen and according to the probability crossover and mutation functions are performed.

The population breeding is done *parallely* using Java 8 Parallel Stream.

Step 3: Crossover

If the probability is less the 0.5 then crossover functionality is performed according to the crossover point.

Example:

Chromosome 1: 01 | 10 | 01 |10 | 0 | 1

Crossover: 01 | 10 | 01 | 10 | 0 | 0

Chromosome 2: 10 | 10 | 10 | 10 | 1 | 0

Step 4: Mutation

If the probability is less than 0.03 then the mutation functionality is performed according to the mutation point. If there are two consecutive 1 bits then 0 bit is add inbetween them or If there are two consecutive 0 bits then 1 bit is add inbetween them

Example: if mutation point = 3, Based on the chromosome values the one of the following mutation occurs.

Case 1: Chromosome: 01 | 00 | 01 01 01

Mutation: 01 | 0 1 0 | 01 01 01

Case 2: Chromosome: 01 | 00 | 01 01 01

Mutation: 01 | 1 0 0 | 01 01 01

Case 3: Chromosome: 01 | 11 | 01 01 01

Mutation: 01 | 1 0 1 | 01 01 01

Case 4: Chromosome: 01 | 11 | 01 01 01

Mutation: 01 | 0 1 1 | 01 01 01

Step 4: Checking Fitness

Fitness is checked for each new population. For Fitness we are taking two traits, volume and weight. If the bit in the chromosome is 1 then the weight and volume of the item in that particular index is added to the total capacity and total volume respectively.If the total capacity and volume is less than or equal to the sack capcity and sack volume then that fitness value is returned. The population with the highest fitness rate is selected as the best in that generation.

The fitness value of population is stored in a Map and sorted by the fitness value.

Step 5: Stopping criteria

If the mean fitness vale of 4 consecutive generation is same then the algorithm is stopped. If generations 11, 12, and 13 has the same average fitness value 25.6, the algorithm will stop.

OUTPUT:

The number of items: 10
the weight of item 1: 13
the volume of item 1: 767.7449040428301
the weight of item 2: 9
the volume of item 2: 353.2233699130104
the weight of item 3: 9
the volume of item 3: 521.8429977167773
the weight of item 4: 1
the volume of item 4: 3.6332248513128773
the weight of item 5: 22
the volume of item 5: 443.0928335217951
the weight of item 6: 24
the volume of item 6: 1160.0583941921864
the weight of item 7: 6
the volume of item 7: 159.16023819837162
the weight of item 8: 14
the volume of item 8: 1370.263553052531
the weight of item 9: 12
the volume of item 9: 747.8418654759108
the weight of item 10: 16
the volume of item 10: 1409.9138059170418
The knapsack capacity: 50
The knapsack volume: 3000.0
The population size: 10
The maximum number of generations: 100
The crossover probability: 0.5
The mutation probability: 0.03

Generation 1:

Population:
0 -- 0101000111
1 -- 1011101110
2 -- 1100111111
3 -- 1010000110
4 -- 0100001110
5 -- 1110001100
6 -- 1111111000

7 -- 1000100001
8 -- 1000010000
9 -- 1011111000

Fitness Score:
----

0 - 0.0
1 - 0.0
2 - 0.0
3 - 0.0
5 - 0.0
6 - 0.0
7 - 0.0
9 - 0.0
8 - 7.0
4 - 26.0

Best solution:0100001110
Average fitness: 3.3
Best Fitness Score: 26.0

Generation 2:
*****
Population:

0 - 0100001110
1 - 0100001110
2 - 0100001110
3 - 0100001110
4 - 0100001110
5 - 0100001110
6 - 0100001110
7 - 0100001110
8 - 0100001110
9 - 0100001110

Fitness Score:
----

0 - 26.0
1 - 26.0

2 - 26.0
3 - 26.0
4 - 26.0
5 - 26.0
6 - 26.0
7 - 26.0
8 - 26.0
9 - 26.0

Best solution 2: 0100001110
Average fitness: 26.0
Best Fitness: 2: 26.0
Number of times Crossover Occured: 1
Number of times Cloning Occured: 4
Mutation did not occur

Generation 3:
*****
Population:

0 - 0100001110
1 - 0100001110
2 - 0100001110
3 - 0100001110
4 - 0100001110
5 - 0100001110
6 - 0100001110
7 - 0100001110
8 - 0100001110
9 - 0100001110

Fitness Score:
----

0 - 26.0
1 - 26.0
2 - 26.0
3 - 26.0
4 - 26.0
5 - 26.0
6 - 26.0
7 - 26.0

8 - 26.0
9 - 26.0

Best solution 3: 0100001110
Average fitness: 26.0
Best Fitness: 3: 26.0
Number of times Crossover Occured: 2
Number of times Cloning Occured: 3
Mutation did not occur

Generation 4:
*****
Population:

0 - 0100001110
1 - 0100001110
2 - 0100001110
3 - 0100001110
4 - 0100001110
5 - 0100001110
6 - 0100001110
7 - 0100001110
8 - 0100001110
9 - 0100001110

Fitness Score:
----

0 - 26.0
1 - 26.0
2 - 26.0
3 - 26.0
4 - 26.0
5 - 26.0
6 - 26.0
7 - 26.0
8 - 26.0
9 - 26.0

Best solution 4: 0100001110
Average fitness: 26.0
Best Fitness: 4: 26.0

Number of times Crossover Occured: 3
Number of times Cloning Occured: 2
Mutation did not occur

Generation 5:
*****
Population:

0 - 0100001110
1 - 0100001110
2 - 0100001110
3 - 0100001110
4 - 0100001110
5 - 0100001110
6 - 0100001110
7 - 0100001110
8 - 0100001110
9 - 0100001110

Fitness Score:
----

0 - 26.0
1 - 26.0
2 - 26.0
3 - 26.0
4 - 26.0
5 - 26.0
6 - 26.0
7 - 26.0
8 - 26.0
9 - 26.0

Best solution 5: 0100001110
Average fitness: 26.0
Best Fitness: 5: 26.0
Number of times Crossover Occured: 3
Number of times Cloning Occured: 2
Mutation did not occur

Generation 6:
*****

Population:

0 - 0100001110
1 - 0100001110
2 - 0100001110
3 - 0100001110
4 - 0100001110
5 - 0100001110
6 - 0100001110
7 - 0100001110
8 - 0100001110
9 - 0100001110

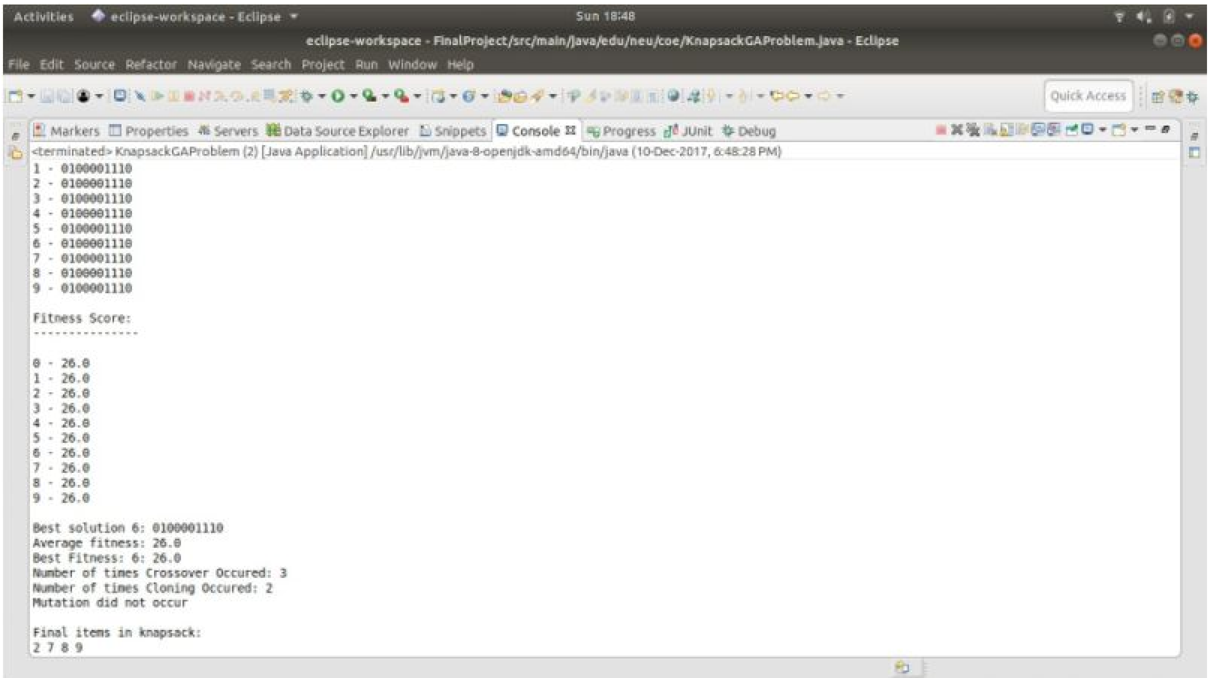
Fitness Score:
----

0 - 26.0
1 - 26.0
2 - 26.0
3 - 26.0
4 - 26.0
5 - 26.0
6 - 26.0
7 - 26.0
8 - 26.0
9 - 26.0

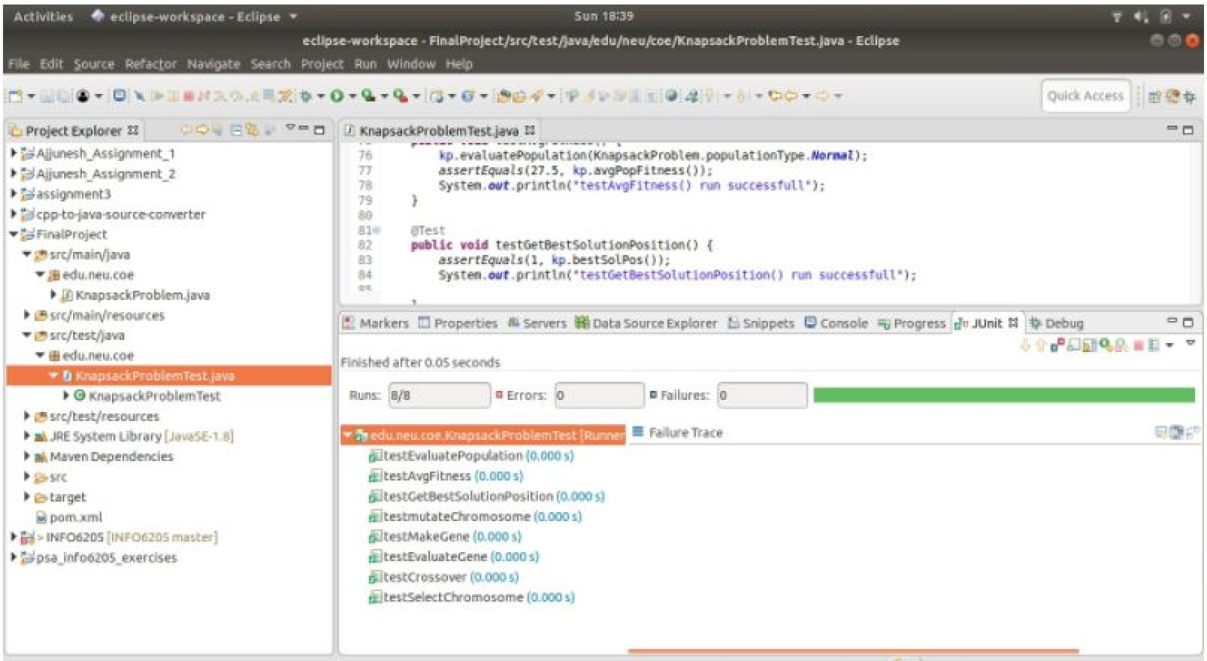
Best solution 6: 0100001110
Average fitness: 26.0
Best Fitness: 6: 26.0
Number of times Crossover Occured: 3
Number of times Cloning Occured: 2
Mutation did not occur

Final items in knapsack:

2 7 8 9



Test case:



CONCLUSION:

We have implemented genetic algorithm for finding good solution for knapsack problem. By implementing GA complexity of Knapsack problem has been reduced from exponential(n items) to logarithmic (By a factor of number of generations it takes to find a solution), which helps us to find optimal solutions for NP problems. Our project result shows that the implementation of a good selection method is very important for the good performance of a genetic algorithm.

REFERENCE:

[1]. https://en.wikipedia.org/wiki/Knapsack_problem