0-1 Knapsack with Genetic Algorithm

By
Mustafa SARITEMUR
GitHub:
https://github.com/MustafaSarit/School-Projects/blob/master/Knapsack.java

Description of Code

- This program is writen with java programming language.
- Program have a main class that calls other classes to calculate result.
- First it ask user for an input text file and it have to be like:

First line: price of items separated with space.	10 20 30 40 50
Second line: weight of items separated with space.	10 20 30 40 50
Third line: capacity of the knapsack.	10

- Other inputs are in integer format so user have to enter integers.

Description of Code

- After taking inputs it start with reading the text file and write its content to the matching variables or arrays.
- Then it create random initial population chromosomes(size taken as an input from user via console) but it checks if chromosome's total weight is equal or lesser than capacity.
- Chromosome will look like this: (length depends on the input file)

- While creating population it also save every chromosomes's fitness score
  - Fitness score is the total price of the chromosome.

Description of Code

- After Creating initial population it keep the fittest chromosome in an array.
- Then it start to create new population. User decide how many population will be created.
- It chose two parent from the population with roulette-wheel selection. And create two, one or none chromosomes. If both children fits two of them will join new population if one then one join or both of them don’t fit both will be deleted.
- Children can be created in two ways:
  - There will be cross-over. If yes after cross-over it checks if there will be mutation. Crossover and mutation probability taken from user as console inputs.
  - There will be no cross-over and both parents will be copyed to the new children.

Description of Code

- It keeps creating new children until size of new generation is equal to one less then old generation's size because fittest of the old generation automatically copy to new generation. (Elitism)
- It keeps creating new populations until it create enough generations. Amount of new generations are given by user with console input.
- After all of that, it keeps all fittest chromosomes of the all generations in an array and read this aray to find the best solution.
- Output will be like:

```
10  20  50  30  90  60  120 40  250 100 160 130 150 Prices
1   1   1   0   0   1   1   1   1   0   0   0   1   Total Price: 700 Weight: 45 Knapsack Capacity: 48
5   8   5   2   3   4   6   3   12  7   9   10  2   Weights
```

Performance Analysis

```
Enter Size of population: 10
Enter Number of generation(Bigger number means optimal result but will take long time): 100
Enter Crossover probability(Between 0-100): 40
Enter Mutation probability(Between 0-100): 30
Choose Crossover point between 1-12: 6
10  20  50  30  90  60  120 40  250 100 160 130 150 Prices
0   0   0   1   1   0   0   1   1   1   1   1   0   Total Price: 800 Weight: 46 Knapsack Capacity: 48
5   8   5   2   3   4   6   3   12  7   9   10  2   Weights
Process takes 5 ms
```

With small populations and few generations result will not be optimal but program will finish in a short time.

```
Enter Size of population: 1000
Enter Number of generation(Bigger number means optimal result but will take long time): 100000
Enter Crossover probability(Between 0-100): 40
Enter Mutation probability(Between 0-100): 30
Choose Crossover point between 1-12: 6
10  20  50  30  90  60  120 40  250 100 160 130 150 Prices
0   0   0   1   1   1   1   0   1   0   1   1   1   Total Price: 990 Weight: 48 Knapsack Capacity: 48
5   8   5   2   3   4   6   3   12  7   9   10  2   Weights
Process takes 86400 ms
```

With big populations and great number of generations result will be optimal but may take long time