Mathiyazhagan, Hema (001813583)

Kathiresan, Kiran Kumar (001427809)

Program Structures and Algorithms-INFO6250

19 April 2019

GENETIC ALGORITHMS-KNAPSACK PROBLEM

TABLE OF CONTENTS

Objective

We are implementing Knapsack problem to solve a typing game in which people get scored for words they type within the maximum time(75 seconds in our case) based on the length of the word they type.

Our objective is to find the best possible set of words within the time to get the maximum points using Genetic Algorithm and comparing the result with other students who took the test.

Words	Points	Time
Computer	8	5
Mode	4	2
Matrix	6	3
Water	5	4

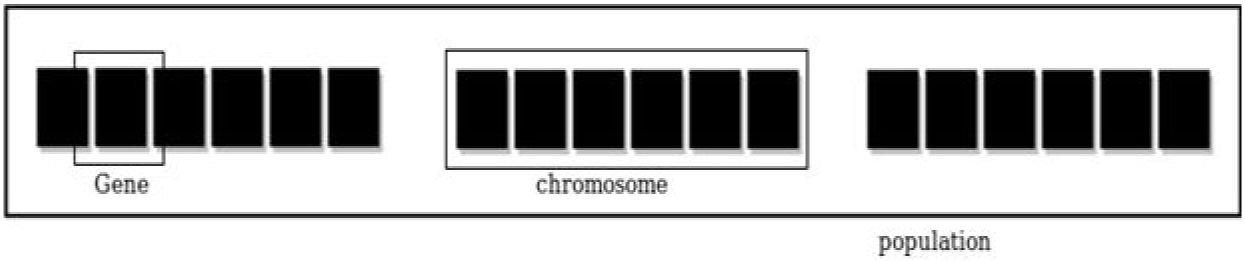
Introduction:

Genetic Algorithm:

Genetic Algorithms are adaptive heuristic search algorithms that belong to larger part of evolutionary algorithms. Genetic algorithm is based on natural selection and Genetics. Genetic algorithms are intelligent exploitation of random search supported by historical data and the process of natural selection to direct the search into a better place in the solution space. They are commonly used for finding better solutions for optimization problems and search problems.

In simple words genetic algorithms are generally based on the idea of survival of the fittest among the individuals who are in the consecutive generation. Each generation consist of group of individuals commonly referred to as the population and each member of the population refers to a point in the search space and possible solution. Each individual is represented as a string of character/integer/float/bits. Each string(if an individual is represented as a string) is analogous to a chromosome.

A fitness score is given to each chromosome which shows the ability of an individual to compete. Individuals having optimal fitness score(or near optimal) are sought.



Operators of Genetic Algorithm:

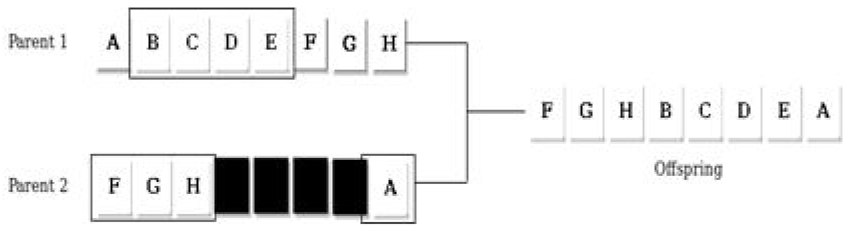
Once the initial generation is created, algorithm evolves the generation using following operators:

1)Selection Operators:

Individual with good fitness score is selected and allowed to pass on his gene to the upcoming generation(based on idea of survival of fittest).

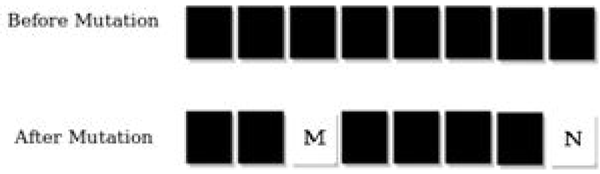
2)CrossOver Operators:

This represents mating between individuals. Two individuals who are selected using selection operator and the genes for crossover are chosen randomly. Then the genes selected are exchanged thus creating a new individual with new genes.



3)Mutation Operator:

Due to crossover,if there is a change in upcoming generations which are desired in the upcoming offsprings then it is said to be mutated.



The summary of the algorithm:

1)Initialize the populations in a random manner.

2)Determine the fitness of population.

3)Until the desired characters shows up:

a)Select parents from population.

b)Crossover and generate the new population.

c)Perform mutation on new population.

d)Calculate fitness of new population.

Implementation:

Genetic Code & Expression:

Each base is a binary value that represents whether the word is typed or not.

0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1

Above string represent scores of individuals in a population when 1 being the score above average and 0 being the one below average.

Fitness Function:

In our case we take the time taken by each individual and compare it with the maximum time(75 seconds) and return the total points taken within the maximum time.

0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1

Selection:

In this process we calculate the probability of each and every chromosome getting selected and store the pattern and probability value in a hash map.

Probability value of each chromosome is calculated by fitness of that chromosome divided by the total fitness of the population

From the fitness function we obtain chromosomes are deemed fit and we obtain the total fitness value.

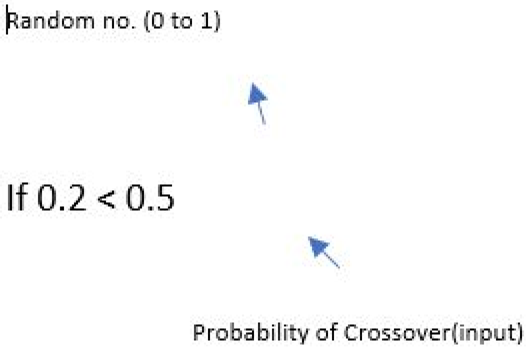
Sorting function:

We sort the hashmap according to the probability values using our function which creates separate list of keys and values. These values are then sorted and these values along with corresponding keys from previous hash map is stored in a new linked hash map which is then sorted.

We calculate the cumulative probability of each chromosome from the least fitness value to the highest fitness value. Using random generator we generate the values between 0 to 1 and the chromosome with the cumulative probability value which is greater than the random value is added to the new population.

Crossover:

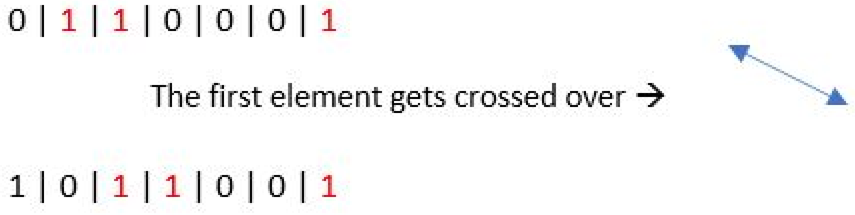
In this process we generate a random number and compare it with the probability of crossover that we input with the main function and swap the genes that satisfy the criteria.



Random no. (0 to 1)

If 0.2 < 0.5

Probability of Crossover(input)



0 | 1 | 1 | 0 | 0 | 0 | 1

The first element gets crossed over →

1 | 0 | 1 | 1 | 0 | 0 | 1

Mutation:

For mutation to be implemented,we are just flipping the genes from ‘0’ to ‘1’ and vice versa.

Evolution:

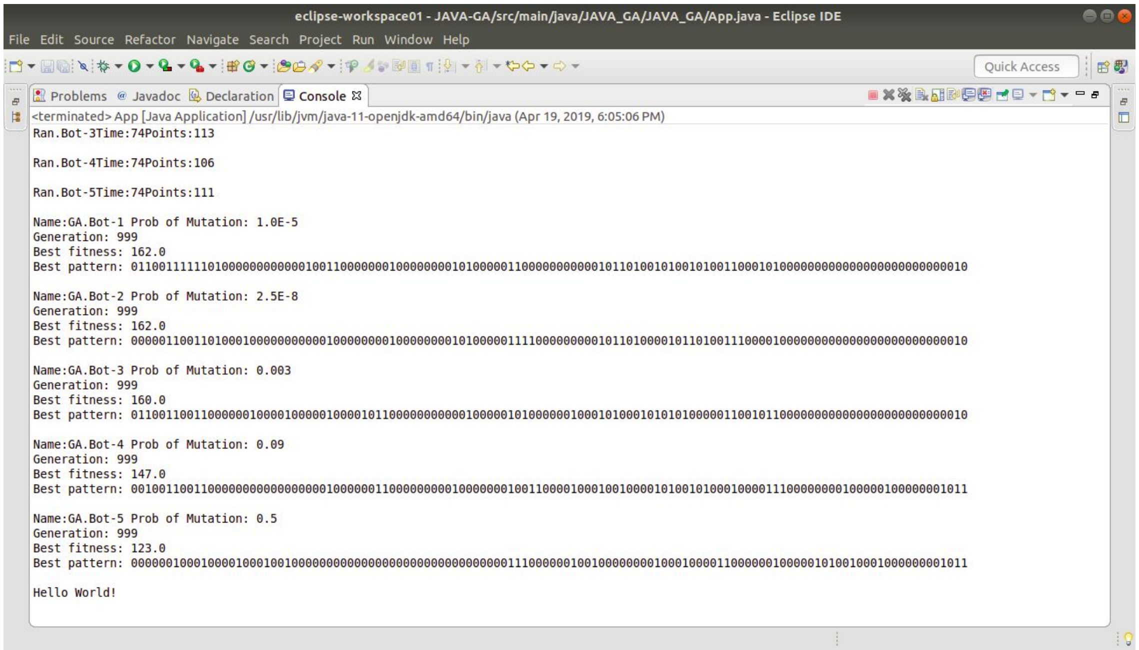
We create an array of string called population with size called as population size. For each element in the population we are generating a random string.population array of strings with each array being a chromosome and each character of array being a gene with binary values randomly.

Parallel Processing:

We implemented parallel processing while adding population patterns to calculate selected probability function. Adding population patterns to the array forms the bottleneck of calculate selected probability function. We used IntStream() to achieve parallel processing in the function. Ranging from 0 to length of the array,parallel() returns an equivalent stream which obeys parallelism.

Result:

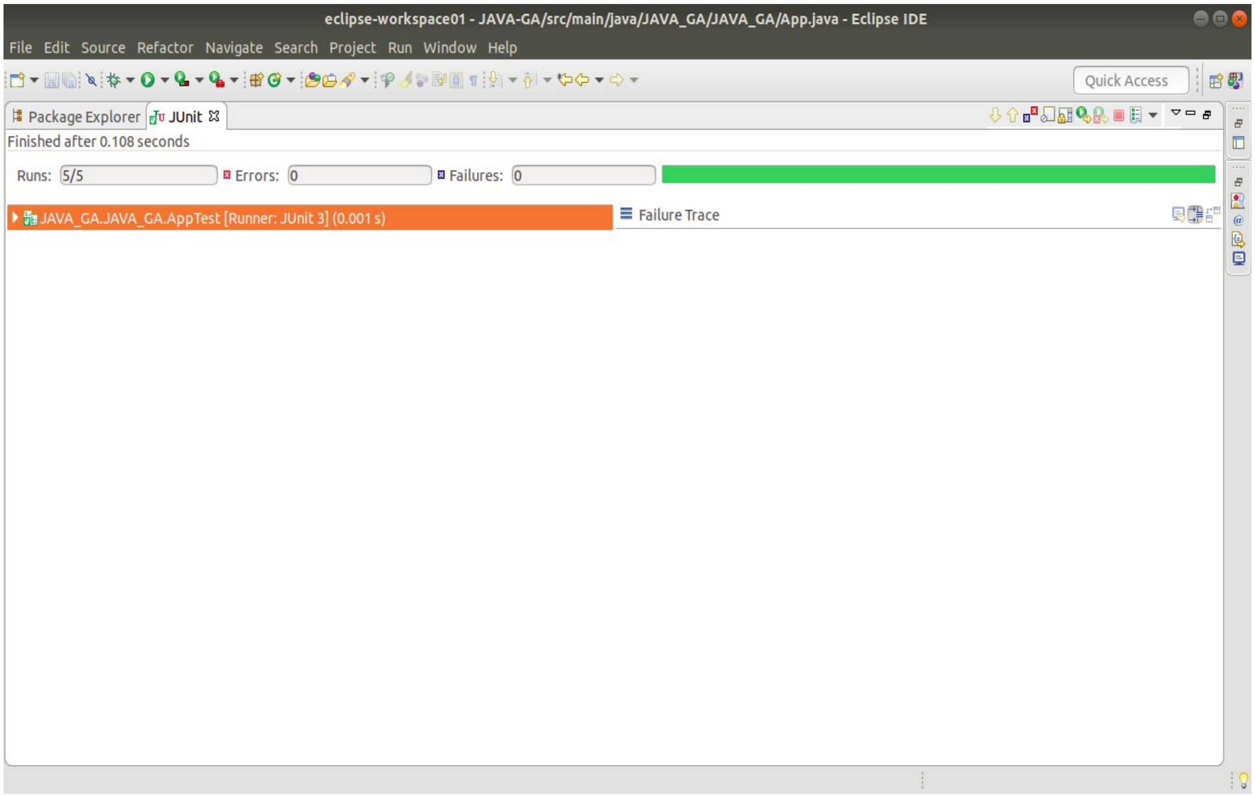
Output screenshot:


eclipse-workspace01 - JAVA-GA/src/main/java/JAVA_GA/JAVA_GA/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Problems
@ Javadoc
Declaration
Console

```
<terminated> App [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Apr 19, 2019, 6:05:06 PM)
Ran.Bot-3Time:74Points:113

Ran.Bot-4Time:74Points:106

Ran.Bot-5Time:74Points:111

Name:GA.Bot-1 Prob of Mutation: 1.0E-5
Generation: 999
Best fitness: 162.0
Best pattern: 0110011111101000000000000001001100000000100000000101000001100000000000010110100101001010011000101000000000000000000000000000010

Name:GA.Bot-2 Prob of Mutation: 2.5E-8
Generation: 999
Best fitness: 162.0
Best pattern: 00000110011010001000000000000010000000001000000000101000001111000000000010110100001011010011100001000000000000000000000000000010

Name:GA.Bot-3 Prob of Mutation: 0.003
Generation: 999
Best fitness: 160.0
Best pattern: 011001100110000001000010000010000101100000000000010000010100000001000101000101010100000110010110000000000000000000000000000010

Name:GA.Bot-4 Prob of Mutation: 0.09
Generation: 999
Best fitness: 147.0
Best pattern: 0010011001100000000000000000010000000110000000001000000010011000010001001000010100101000100001110000000001000001000000001011

Name:GA.Bot-5 Prob of Mutation: 0.5
Generation: 999
Best fitness: 123.0
Best pattern: 000000100010000100010010000000000000000000000000000000001110000000100100000000010001000011000000100000101001000100000000001011

Hello World!
```

We ran multiple instances of genetic Algorithm varying probability of mutation and obtain the best fitness value(maximum points) and pattern(words typed) in that generation. Ran-Bot-1,Ran-Bot-2,Ran-Bot-3,Ran-Bot-4,Ran-Bot-5 are simulations in which words are chosen randomly within time frame without genetic algorithm to compare the result with the bot which attempted the problem with the genetic algorithm.

Test case output:

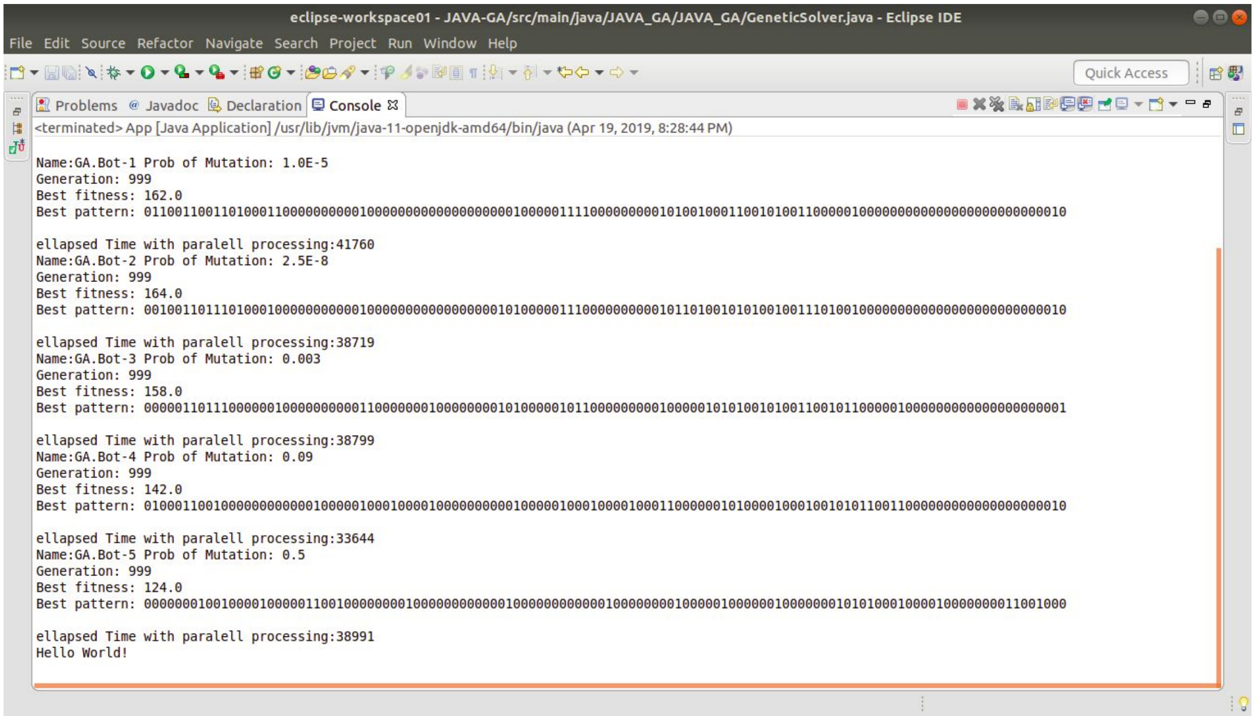

eclipse-workspace01 - JAVA-GA/src/main/java/JAVA_GA/JAVA_GA/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Package Explorer
JUnit
Failure Trace

```
Finished after 0.108 seconds
Runs: 5/5    Errors: 0    Failures: 0
JAVA_GA.JAVA_GA.AppTest [Runner: JUnit 3] (0.001 s)
```

Parallel computing output:

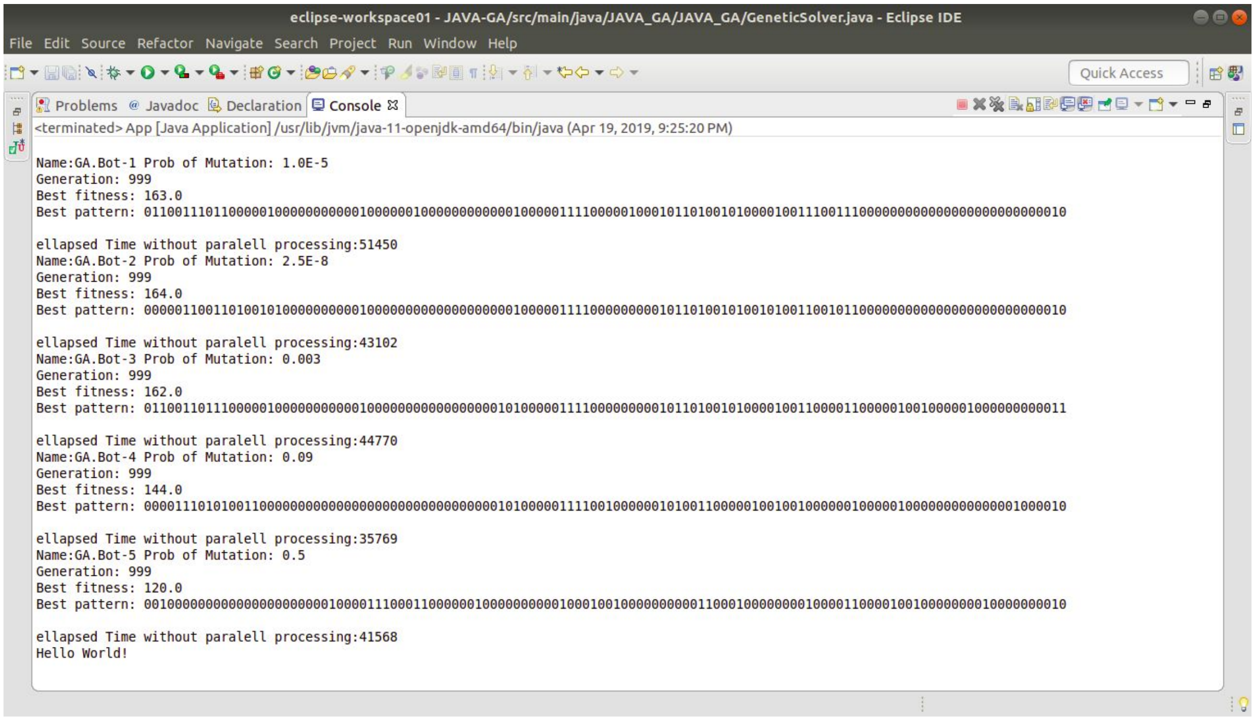
Ellapsed time with parallel computing is found to be lower than the one calculated without parallel computing.

Output with parallel computing:


eclipse-workspace01 - JAVA-GA/src/main/java/JAVA_GA/JAVA_GA/GeneticSolver.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Problems
Javadoc
Declaration
Console

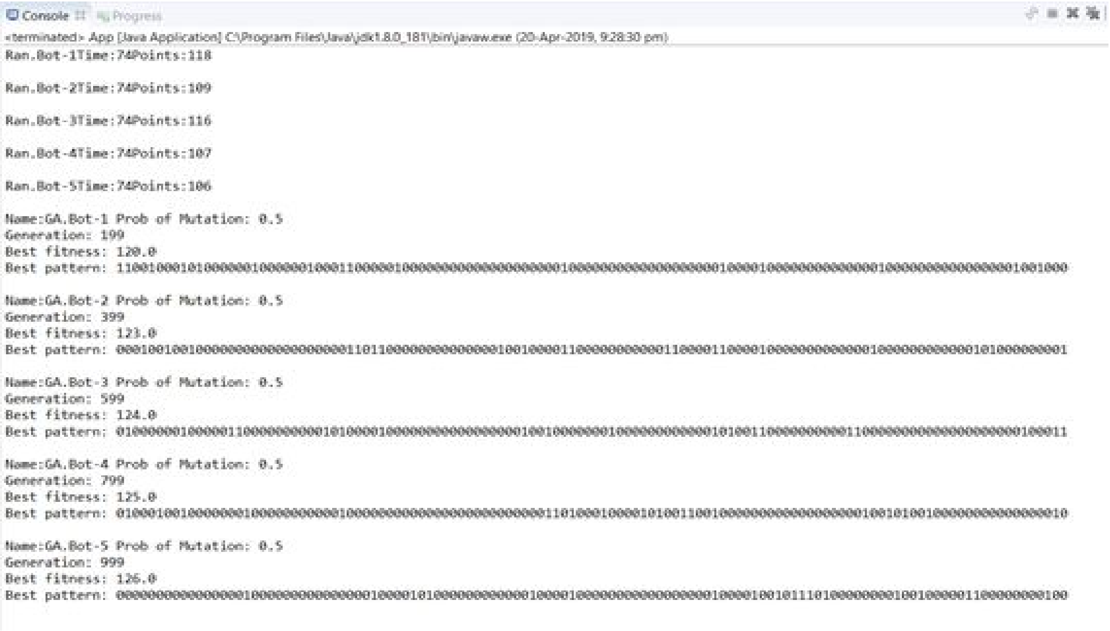
```
<terminated> App [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Apr 19, 2019, 8:28:44 PM)

Name:GA.Bot-1 Prob of Mutation: 1.0E-5
Generation: 999
Best fitness: 162.0
Best pattern: 011001100110100011000000000010000000000000000000100000111100000000001010010001100101001100000100000000000000000000000000010

ellapsed Time with paralell processing:41760
Name:GA.Bot-2 Prob of Mutation: 2.5E-8
Generation: 999
Best fitness: 164.0
Best pattern: 001001101110100010000000000010000000000000000001010000011100000000001011010010101001001110100100000000000000000000000000010

ellapsed Time with paralell processing:38719
Name:GA.Bot-3 Prob of Mutation: 0.003
Generation: 999
Best fitness: 158.0
Best pattern: 000001101110000001000000000011000000010000000010100000101100000000001000001010100101001100101100000100000000000000000000001

ellapsed Time with paralell processing:38799
Name:GA.Bot-4 Prob of Mutation: 0.09
Generation: 999
Best fitness: 142.0
Best pattern: 010001100100000000000001000001000100001000000000010000010001000010001100000010100001000100101011001100000000000000000000010

ellapsed Time with paralell processing:33644
Name:GA.Bot-5 Prob of Mutation: 0.5
Generation: 999
Best fitness: 124.0
Best pattern: 000000010010000100000110010000000001000000000000100000000000010000000010000010000000100000000101010001000010000000011001000

ellapsed Time with paralell processing:38991
Hello World!
```

Output without parallel computing:

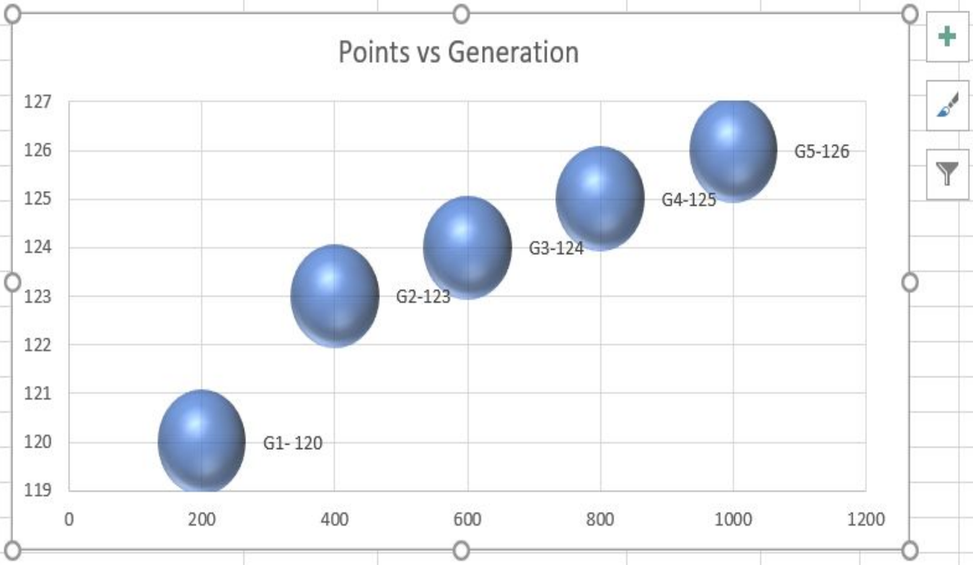

eclipse-workspace01 - JAVA-GA/src/main/java/JAVA_GA/JAVA_GA/GeneticSolver.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Problems
Javadoc
Declaration
Console

```
<terminated> App [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Apr 19, 2019, 9:25:20 PM)

Name:GA.Bot-1 Prob of Mutation: 1.0E-5
Generation: 999
Best fitness: 163.0
Best pattern: 011001110110000010000000000010000001000000000000100000111100000100010110100101000010011100111000000000000000000000000000010

ellapsed Time without paralell processing:51450
Name:GA.Bot-2 Prob of Mutation: 2.5E-8
Generation: 999
Best fitness: 164.0
Best pattern: 000001100110100101000000000010000000000000000000100000111100000000001011010010100101001100101100000000000000000000000000010

ellapsed Time without paralell processing:43102
Name:GA.Bot-3 Prob of Mutation: 0.003
Generation: 999
Best fitness: 162.0
Best pattern: 011001101110000010000000000010000000000000000001010000011110000000000101101001010000100110000110000010010000010000000000011

ellapsed Time without paralell processing:44770
Name:GA.Bot-4 Prob of Mutation: 0.09
Generation: 999
Best fitness: 144.0
Best pattern: 000011101010011000000000000000000000000000000001010000011110010000000101001100000100100100000001000001000000000000000001000010

ellapsed Time without paralell processing:35769
Name:GA.Bot-5 Prob of Mutation: 0.5
Generation: 999
Best fitness: 120.0
Best pattern: 001000000000000000000000010000111000110000000100000000000100010010000000000011000100000000010000110000100100000000010000000010

ellapsed Time without paralell processing:41568
Hello World!
```

PERFORMANCE GRAPH:



```
<terminated> App [Java Application] C:\Program Files\Java\jdk1.8.0_181\bin\javaw.exe (20-Apr-2019, 9:28:30 pm)
Ran.Bot-1Time:74Points:118

Ran.Bot-2Time:74Points:109

Ran.Bot-3Time:74Points:116

Ran.Bot-4Time:74Points:107

Ran.Bot-5Time:74Points:106

Name:GA.Bot-1 Prob of Mutation: 0.5
Generation: 199
Best fitness: 120.0
Best pattern: 110010001010000000100000010001100000010000000000000000000000100000000000000000000100001000000000000000010000000000000000001001000

Name:GA.Bot-2 Prob of Mutation: 0.5
Generation: 399
Best fitness: 123.0
Best pattern: 0001001001000000000000000000000110110000000000000000100100001100000000000011000011000010000000000000001000000000000001010000000001

Name:GA.Bot-3 Prob of Mutation: 0.5
Generation: 599
Best fitness: 124.0
Best pattern: 0100000010000011000000000001010000100000000000000000010010000000100000000000001010011000000000011000000000000000000000100011

Name:GA.Bot-4 Prob of Mutation: 0.5
Generation: 799
Best fitness: 125.0
Best pattern: 0100010010000000100000000000100000000000000000000000000001101000100001010011001000000000000000000001001010010000000000000000010

Name:GA.Bot-5 Prob of Mutation: 0.5
Generation: 999
Best fitness: 126.0
Best pattern: 0000000000000000100000000000000010000101000000000000100001000000000000000000010000100101110100000000100100000110000000000100
```



Conclusion :

In the genetic algorithm we can see that the values move towards the best possible combination after multiple generations but when we randomly select the solution the probability of getting the best combination is very low.