

Job Shop Scheduling using Genetic Algorithm

Course: Program Structure and Algorithm

Team Number: 528

Team Members:

Akshay Mahesh Bhusare

Pratik Hemant Patil

Rishabh Bharat Jain

I. Introduction

Job Shop Scheduling is an NP-Complete scheduling problem where a combination of predefined set of jobs and machines are to be deduced. There are 'j' number of jobs to be run on 'm' number of machines in 'o' number of operations. The combination of jobs and machines run in an operation with the execution sequence is to be found which will result in minimum time to complete the entire process.

They cannot be formulated as a linear programming and no simple rules or algorithms yield to optimal solutions in a short time. In this project we used Genetic Algorithm to deal with this problem of job shop scheduling.

II. Population

Population of our project is made up of various candidates which representing a set of jobs and machines sequences being run. Each candidate includes a list of Chromosomes formed in a random order. The chromosome is formed by a combination of the operation number, job number and machine number. This combination represents the **Genotype** and the sequence of their arrangement i.e. the order in which they will be processed represents the **Phenotype**.

As mentioned earlier, the genotype representation is made up of a combination of operation, job and machine number. For example, '020511' is a genotype/chromosome in this project, the first 2 digits represent the operation number, the next 2 digits represent the job number and the last 2 digits represent the machine number. This sequence indicates that operation 02 will have job 05 running on Machine 11. Multiple tasks of a job running on a machine are clubbed together to form an operation. Each operation is to be run in sequence representing the operations being performed in a factory. This population represents a single generation.

Number of jobs, machines, operations as well as candidates in a population can be set from a configuration file "Constants.java" in the Model package of the project.

III. Constraints

This project is based on the assumptions of following constraints:

1. *Operations should be executed in sequence:*
All the available operations will be executed in a sequential manner.
2. *Every job should run on every machine at least once in every candidate:*
Each candidate has a combination of jobs and machine running in an operation. The candidate is formed in a manner that every job must be paired with every available machine at least once.
3. *If a job is running in the preceding operation, the same job cannot be started in the next operation:*
If Job 1 is still being executed in Operation 1, it cannot be started in the next operation i.e. a job cannot be run parallelly on any of the machines in any operation.
4. *An operation will be processed by the machine only if it is idle:*
The next operation will start on a machine only if the machine is idle or else, the operation will wait till the machine is idle
5. *No two jobs will be simultaneously processed by a machine:*
A machine cannot simultaneously run multiple jobs. The machine will start executing the next job only after completing execution of the previous job.
6. *A job cannot be simultaneously processed on multiple machines:*
If a job is being executed by any machine, the same job cannot be executed on any other machine.

- Machine sequence

Operation	O1	O2	O3
Job 1	M2	M3	M1
Job 2	M1	M3	M2
Job 3	M3	M2	M1

Figure 1: Machine Sequence of Job Execution

The above figure gives an idea of a candidate representing the job sequence being run on machines in each operation.

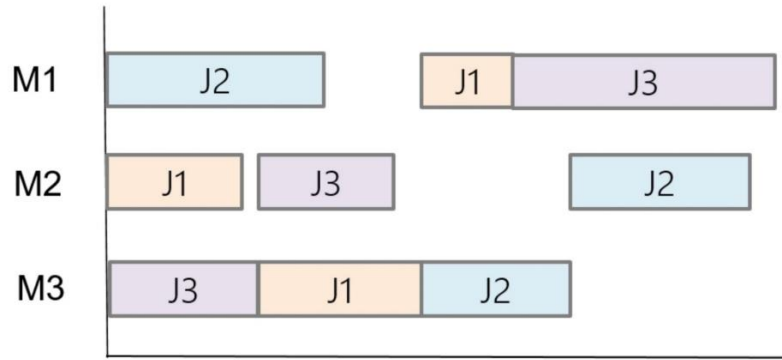


Figure 2: Job execution sequence in an Operation

Figure 1 represents the execution of jobs on different machines in an operation as per schedule in the chromosome. Here, Job2 is run on Machine1 and simultaneously Job1 and Job3 are running on Machine2 and Machine3 respectively. After the initial sequence is completed, we can see that Machine2 completes the job and is available, so the next job scheduled to run on machine2 can start its execution. But, as we can see, the next job is Job3 which is still being run on Machine3 and hence Machine2 must wait till execution of Job3 on the other machine is completed. All the other jobs are processed in a similar manner obeying all the above-mentioned constraints.

IV. Expression

- Expression is a crucial step in genetic algorithm where every candidate is processed based on predefined expression.
- In job shop scheduling, the maximum time required to complete all the operations is calculated in expression step.
- Time required to complete every job (J1, J2, J3,....., Jn) is different on different machine (M1,M2,M3,.....Mn), this is available in "TimeArray". This is a Job * Machine array where every entry shows the time required for i^{th} job to run on j^{th} machine.

Following is the short representation of TimeArray:

	MACHINE 0	MACHINE 1	MACHINE 2	MACHINE 3	MACHINE 4
JOB0	900.00	410.00	710.00	110.00	650.00
JOB1	170.00	690.00	680.00	320.00	510.00
JOB2	920.00	620.00	190.00	520.00	1,000.00
JOB3	480.00	250.00	350.00	950.00	710.00
JOB4	130.00	110.00	600.00	200.00	520.00

- Time required by every candidate is calculated as the time where all the jobs are completed and all machines are free. This time is stored as "Tmax"
- The highest "Tmax" among all candidates in the generation is considered as "Tmax" of the generation.

V. Fitness

The fitness of a candidate is determined the minimum time it takes to execute the list of chromosomes i.e. the schedule of running a job on a machine in an operation.

Fitness calculation is based on TMax (time taken for the whole schedule of jobs in a candidate to be completed). The greater the TMax , the less will be the fitness of the candidate.

Hence, fitness is determined by the inversion of TMax value,

$$f = \frac{1}{TMax}$$

VI. Crossover

Once fitness for all the candidates is calculated, we select the top 30% of the candidates based on the fitness value. These set of fittest candidates are moved to the next generation. The selection of fittest candidates can be changed from the configuration file.

These candidates now form the initial population of the next generation. The rest of the population in this generation is formed by recombining the chromosomes of the previous generation candidates. This provides diversity in population.

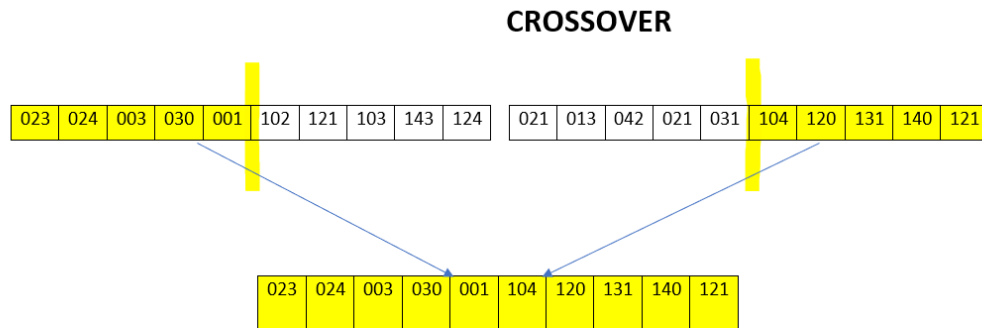


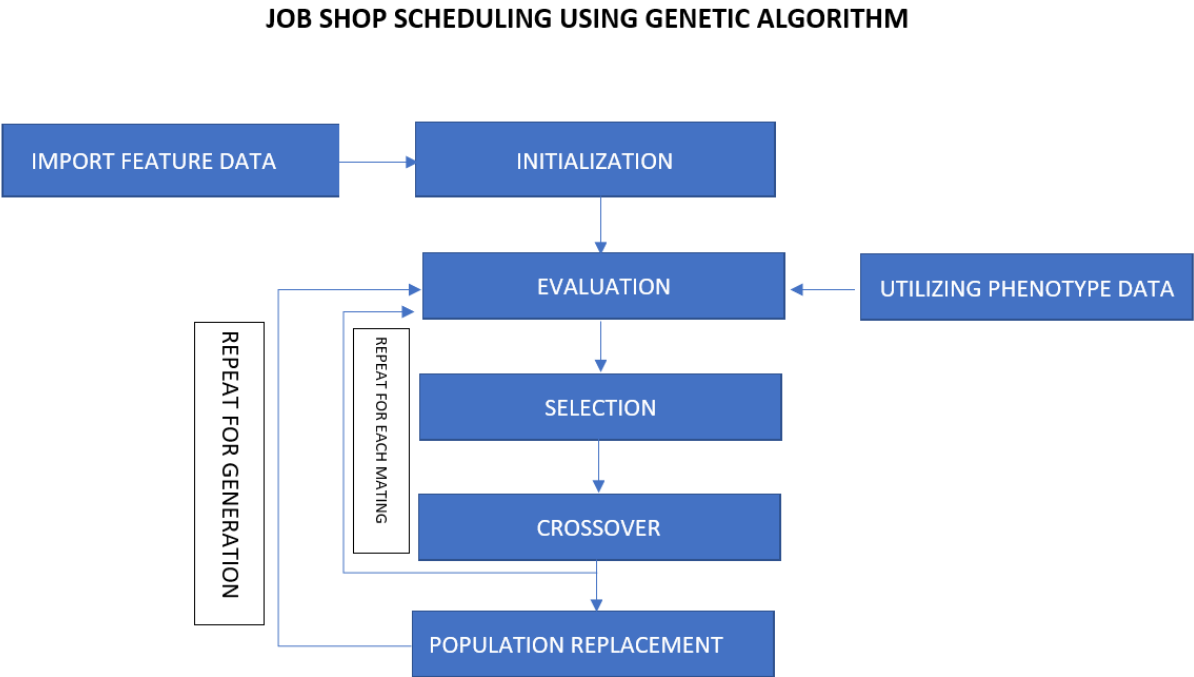
Figure 3: Crossover

The previous generation candidates are selected at random for crossover, these are called 'Parents'. After selecting two parents from the chromosome pool of previous generation, they are merged together to form 2 new candidates which are added to the new generation. The merge process takes place by selecting the first half of Parent1 (i.e. the chromosome list of parent1 is divided equally and then the first half is selected) and second half of Parent2 to form Child.

Each generation is further used to create a new generation, until a point where the fitness value of a lot of the consecutive generations is similar. This will be the exit point of our project indicating that the job scheduling has reached its ultimate level.

VII. Diagrammatic Representation

Following is the diagrammatic representation of the overall Genetic Algorithm implemented in this project.

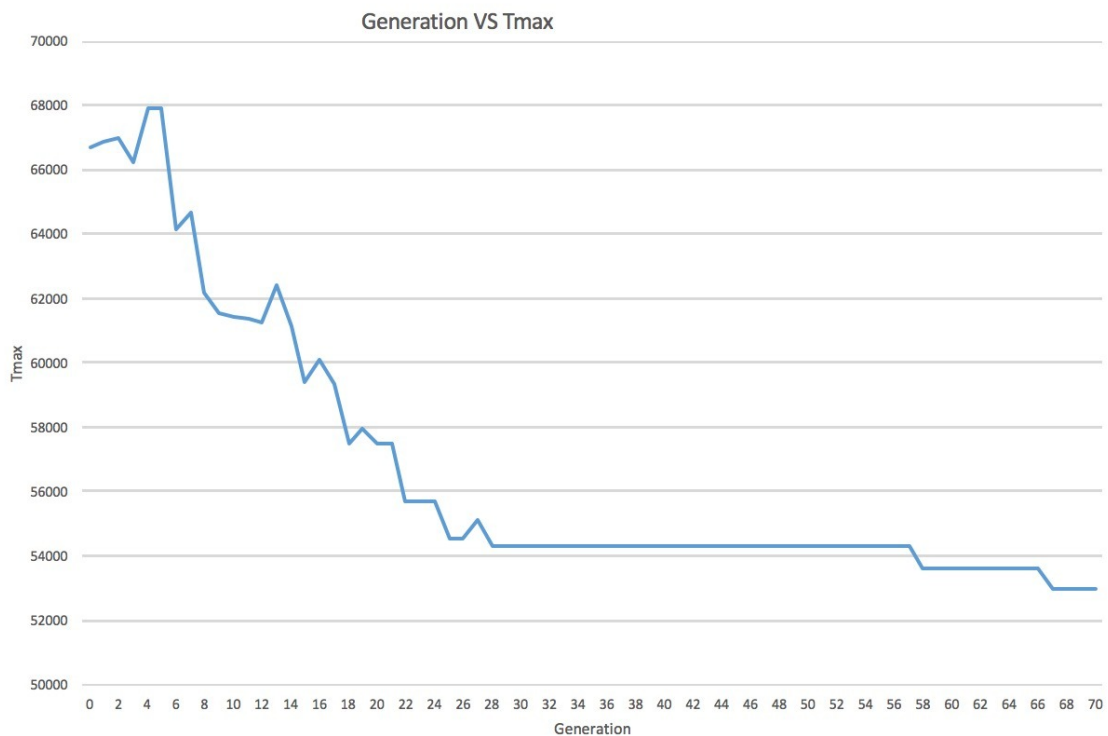


VIII. Results

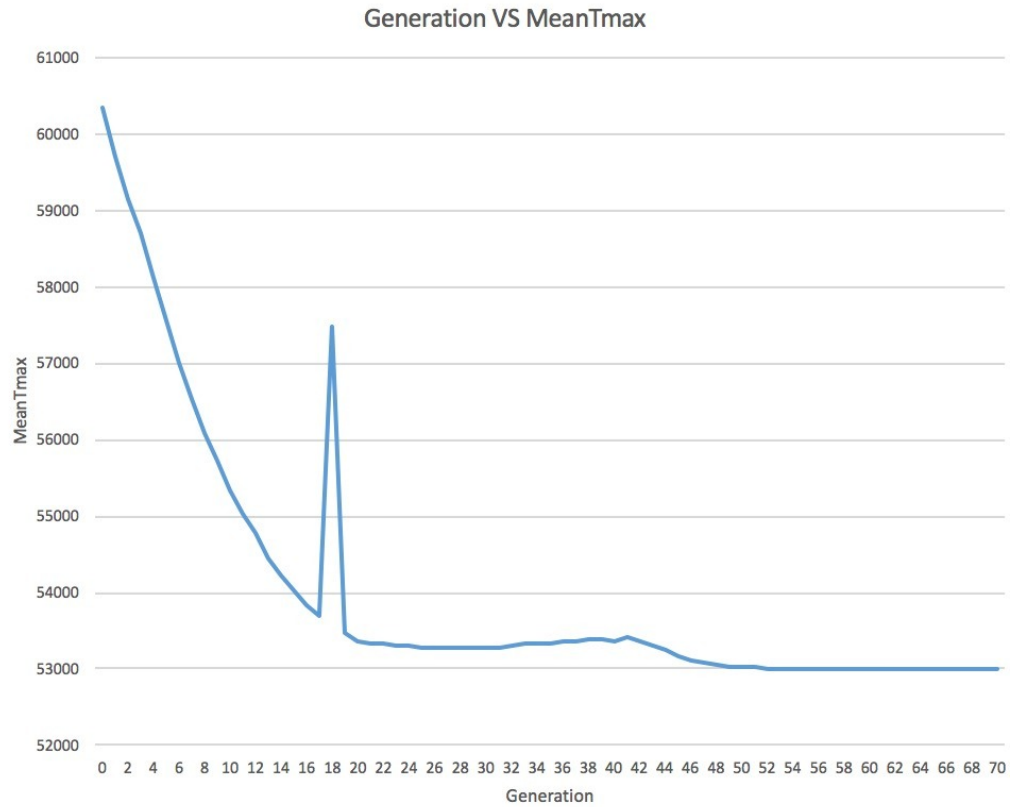
Configuration for results:

Population Size (Candidates in 1 generation)	1000
Number of Operations	40
Number of Jobs	60
Number of Machines	40
Crossover factor (Fittest Selection in %)	30

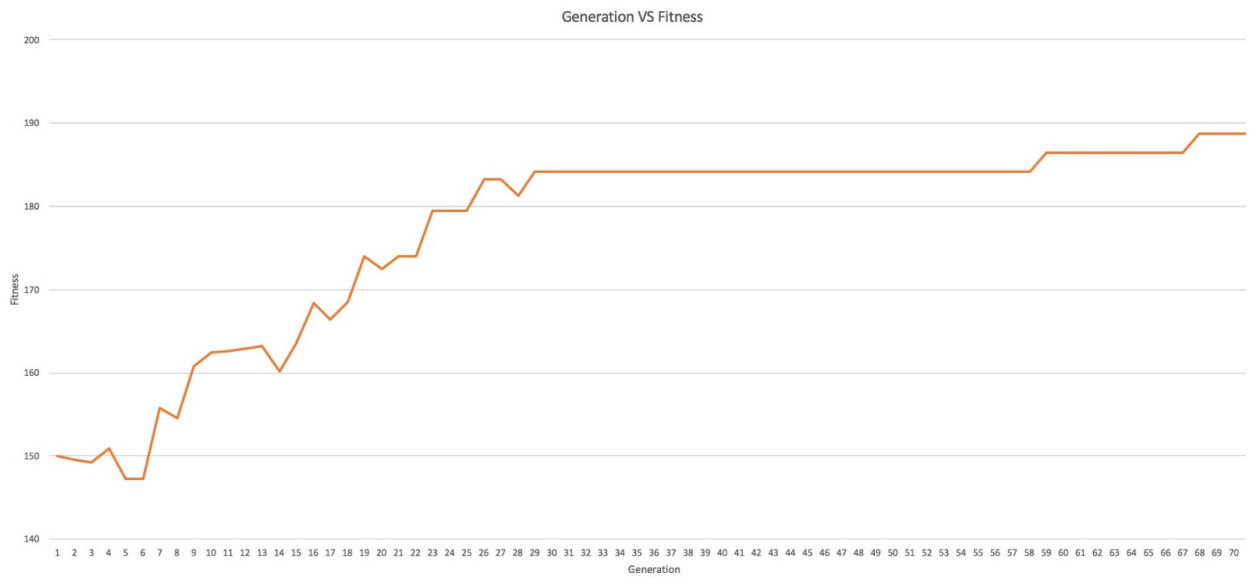
Graphical Representation for Genetic Algorithm implementation:



The above graph represents Generation vs TMax of the fittest candidate. Here we can see as the generation increases, the TMax decreases i.e. the performance or the time required to complete the entire task reduces.

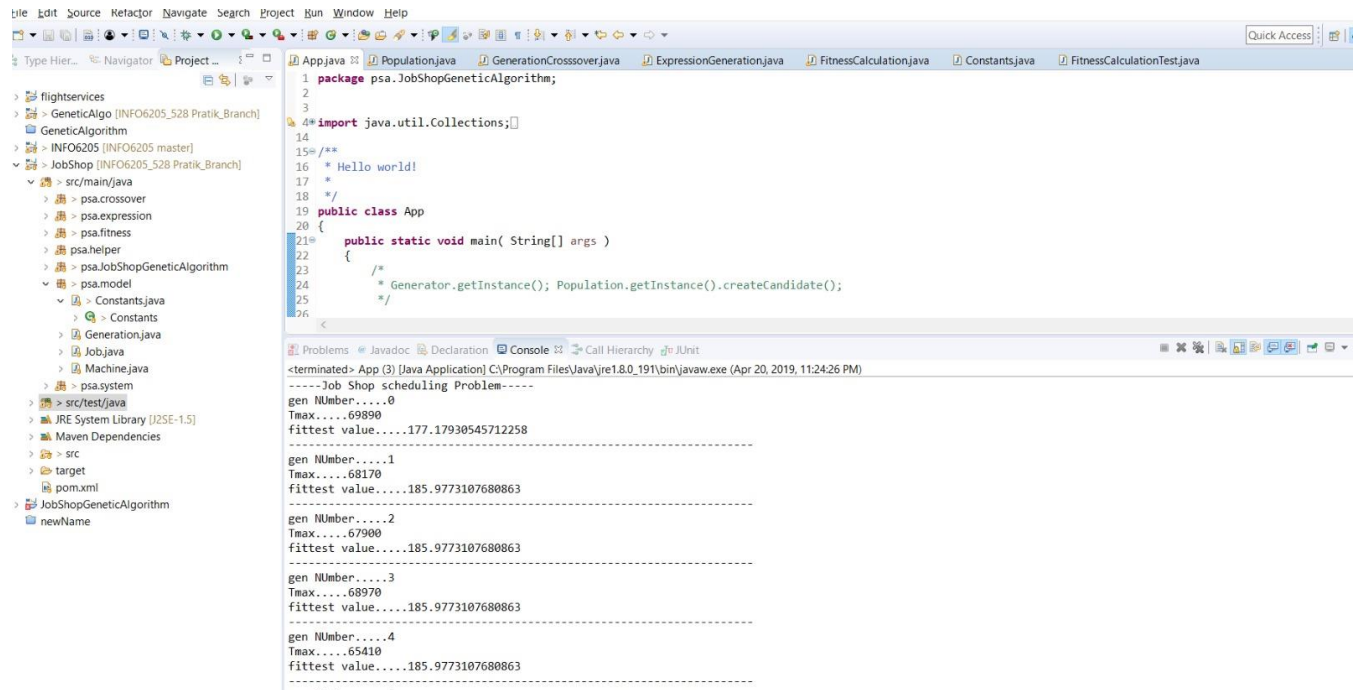


The above graph represents Generation vs Mean value of TMax of all candidates in a generation.



The above graph represents Generation vs Fitness value of the fittest candidate in each generation.

Code Run:



```
1 package psa.JobShopGeneticAlgorithm;
2
3
4 import java.util.Collections;
5
6
7
8
9
10
11
12
13
14
15 /**
16  * Hello world!
17  */
18
19 public class App
20 {
21     public static void main( String[] args )
22     {
23         /*
24          * Generator.getInstance(); Population.getInstance().createCandidate();
25          */
26     }
27 }
```

Problems Javadoc Declaration Console Call Hierarchy JUnit

<terminated> App (3) [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (Apr 20, 2019, 11:24:26 PM)

-----Job Shop scheduling Problem-----

gen Number.....0
Tmax.....69890
fittest value.....177.17930545712258

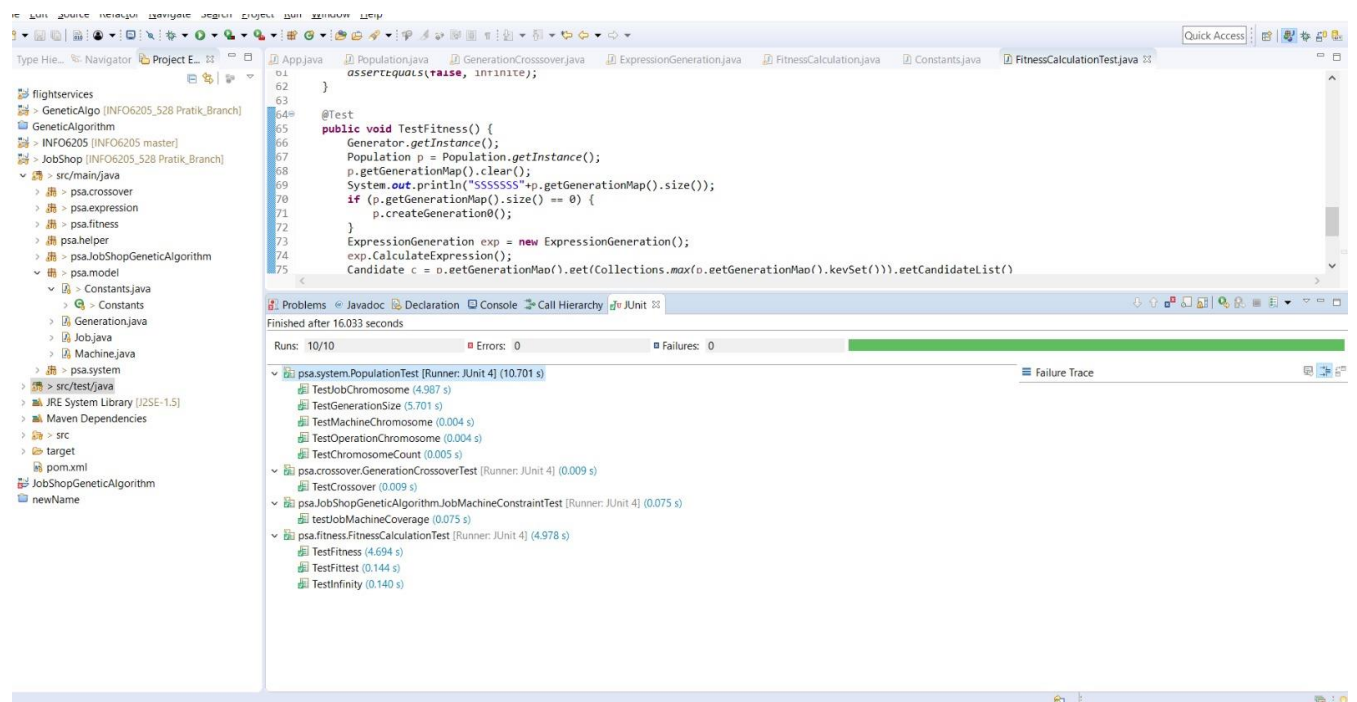
gen Number.....1
Tmax.....68170
fittest value.....185.9773107680863

gen Number.....2
Tmax.....67900
fittest value.....185.9773107680863

gen Number.....3
Tmax.....68970
fittest value.....185.9773107680863

gen Number.....4
Tmax.....65410
fittest value.....185.9773107680863

Test Cases Run:



```
1 package psa.JobShopGeneticAlgorithm;
2
3
4 import java.util.Collections;
5
6
7
8
9
10
11
12
13
14
15 /**
16  * Hello world!
17  */
18
19 public class App
20 {
21     public static void main( String[] args )
22     {
23         /*
24          * Generator.getInstance(); Population.getInstance().createCandidate();
25          */
26     }
27 }
```

Problems Javadoc Declaration Console Call Hierarchy JUnit

Finished after 16.033 seconds

Runs: 10/10 Errors: 0 Failures: 0

psa.system.PopulationTest [Runner: JUnit 4] (10.701 s)

- TestJobChromosome (4.987 s)
- TestGenerationSize (5.701 s)
- TestMachineChromosome (0.004 s)
- TestOperationChromosome (0.004 s)
- TestChromosomeCount (0.005 s)

psa.crossover.GenerationCrossoverTest [Runner: JUnit 4] (0.009 s)

- TestCrossover (0.009 s)

psa.JobShopGeneticAlgorithm.JobMachineConstraintTest [Runner: JUnit 4] (0.075 s)

- testJobMachineCoverage (0.075 s)

psa.fitness.FitnessCalculationTest [Runner: JUnit 4] (4.978 s)

- TestFitness (4.694 s)
- TestFittest (0.144 s)
- TestInfinity (0.140 s)

IX. Conclusion:

Application of genetic algorithm gives a good result most of the time. Although GA takes plenty of time to provide a good result, it provides a flexible framework for evolutionary computation and it can handle varieties of objective function and constraint. Job Shop scheduling can be done using Genetic Algorithm and the result shows a stability in the implementation after Generation 70. Thus, the ultimate point where the perfect scheduling of jobs on machines is achieved after 70 generations.

X. GitHub Link:

https://github.com/bhusareakshay/INFO6205_528.git