JOB SCHEDULING WITH GENETIC ALGORITHM

A Paper
Submitted to the Graduate Faculty
Of the
North Dakota State University
Of Agriculture and Applied Science

By

Debarshi Barat

In Partial Fulfillment of the Requirements
For the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

January 2013

Fargo, North Dakota

North Dakota State University

Graduate School

Title

Job Scheduling using Genetic Algortihm

By

Debarshi Barat

The Supervisory Committee certifies that this ***disquisition*** compiles with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Simone Ludwig

Advisor

Dr. Kendall Nygard

Dr. Saeed Salim

Dr. Sanku Mallik

Approved by Department Chair:

04/ 02/2013
Date

Dr. Brian Slator
Signature

ABSTRACT

In this paper, we have used a Genetic Algorithm (GA) approach for providing a solution to the Job Scheduling Problem (JSP) of placing 5000 jobs on 806 machines. The GA starts off with a randomly generated population of 100 chromosomes, each of which represents a random placement of jobs on machines. The population then goes through the process of reproduction, crossover and mutation to create a new population for the next generation until a predefined number of generations are reached. Since the performance of a GA depends on the parameters like population size, crossover rate and mutation rate, a series of experiments were conducted in order to identify the best parameter combination to achieve good solutions to the JSP by balancing makespan with the running time. We found that a crossover rate of 0.3, a mutation rate of 0.15 and a population size of 100 yield the best results.

ACKNOWLEDGEMENTS

I would like to thank Dr. Simone Ludwig for her continuous support throughout my research work. Without her guidance it would not have been possible for me to come this far. I would also like to thank Dr. Kendall Nygard, Dr. Saeed Salem and Dr. Sanku Mallik for agreeing to serve on my graduate supervisory committee. Special thanks to Dr. Kendall Nygard and the Department of Computer Science for giving me the opportunity to pursue my higher studies at North Dakota State University. I would take this opportunity of thanking Mr. Anubrata Dutta without whose support and help it would have been difficult for me to complete my research work.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

1. INTRODUCTION

Optimization problems help us to find the optimum solution among a set of solutions for example the highest yield or the lowest cost, etc. There are different types of Optimization problems and different problems have different solutions to them. In this paper we look into one such optimization problem: Job Scheduling.

1.1. Optimization

Before the optimization process can be started, all problems to be optimized should be formulated as a system with its status controlled by a few input variables and its performance specified by a well-defined objective function or the fitness function, which can be denoted as *f*. The goal of optimization is to find the best value for each variable in order to achieve satisfactory performance. The variables required by the fitness function are referred to as the input or the decision variables. A particular setting of the input variables, *position,* denoted by x, $x \in \mathbb{R}^D$, where D is the total number of input variables can also be referred to as setting or decision vector. The variables could have simple constraints or could have complex ones. The set of all feasible results is called the *search space* or the *function space*. Subsets of the search space form neighborhoods. The result evaluated by the objective function to give a certain position is called the *objective value* or the fitness value.

The maximum value reached by an objective function is called the *maxima*, similarly the lowest value arrived at by the objective function is the *minima*. Both maxima and minima could be referred to as the *optimum* for that objective function. The optimum within a neighborhood is called *local optimum* while the *global optimum is* the optimum among all local optima.

Figure 1.1 shows the unimodal objective function $f(x) = x^2 + 0.5$ with a minimum 0.5 at x = 0. Both a maximum and a minimum can be called an optimum. A local optimum is an optimum within its neighborhood. Figure 1.2 shows the multimodal objective function f(x) = 10 sin(x) + x and three of its local minima. The global optimum is the optimum of all local optima; i.e., it is the optimum in the complete search space. Figure 1.2 shows a global minimum of roughly −18 located close to x = −8 in the search space [−10, 10].

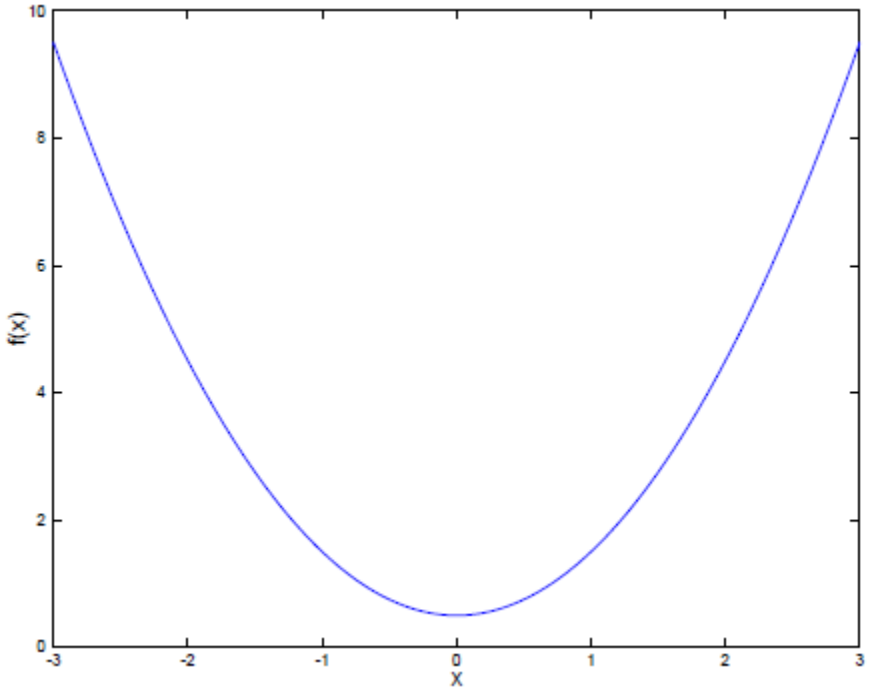

Figure 1.1: Unimodal Objective Function

A maximization problem is an optimization problem for which the position (i.e. input variables) with the highest objective value is to be found. A minimization problem is an optimization problem for which the position with the lowest objective value is to be found. A maximization problem can be converted into a minimization problem by negating the objective function.

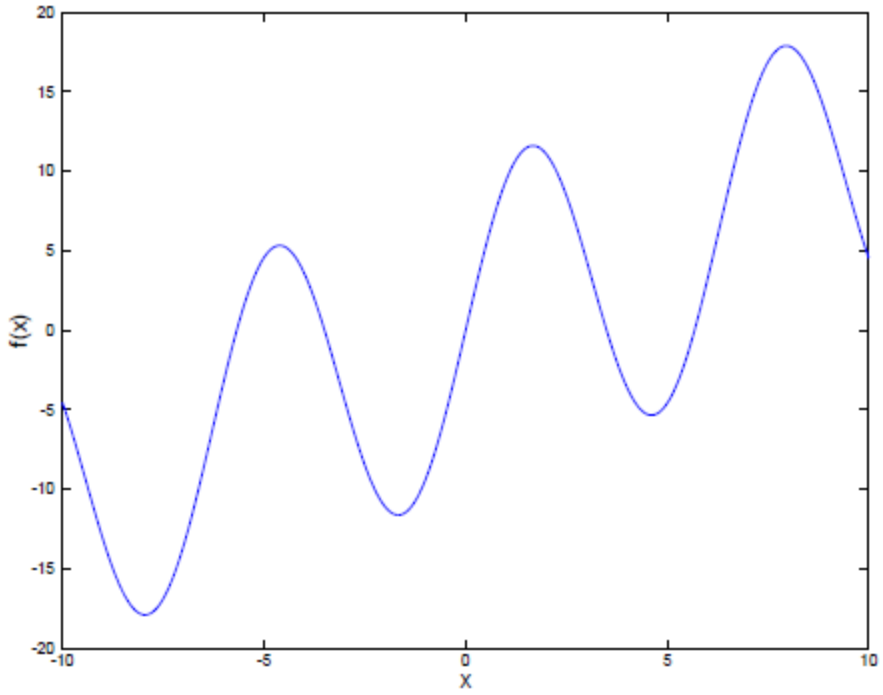

Figure 1.2: Multi-Modal Objective Function

1.2. Examples of Optimization Problems

Objective functions often attempt to model real entities. Creating an objective function that behaves like the real entity can be a challenging task on its own. Simplified descriptions of real world optimization problems include:

- minimize the output of certain chemical species by finding an optimal reaction temperature and pressure given a certain catalyst [1, 2]
- minimize the building cost of a car, ship, engine, or notebook without violating quality constraints [3, 4]
- minimize the air resistance of a car body [5]
- maximize the efficiency of a fuel cell [6]
- minimize the difference between a simulation and experimental measurements [7]
- maximize the volume of a structure given a certain amount of building material [8]

- minimize the length of a route that visits certain points at least once [9]
- maximize the potential yield or minimize the risk of a portfolio [10]
- minimize the operating cost of a fresh water system without violating constraints such as
- minimal amount of stored water [11]
- minimize the difference between power generation and demand for scheduling a hydroelectric power station [12, 13,14]

1.3. Job Scheduling Problem

One of the most famous global optimization problems is that of scheduling and among them one of the most famous is the Job Scheduling Problem (JSP) or Job Shop Scheduling Problem (JSSP), which is to schedule a set of n jobs on a set of m machines such that we can minimize the total time taken from the first job being scheduled to the execution of the last job. Operations of the same job cannot be processed concurrently and each job must be processed on any machine exactly once. A pictorial representation of 9 jobs being placed on 3 machines has been provided in Figure 1.3; where M1, M2 and M3 represent machines and J1-J9 represent the nine different jobs.

M1			M2			M3		
J5	J7	J4	J1	J9	J2	J3	J6	J8

Figure 1.3: Example of A Chromosome Representation

Besides being an NP-hard problem, this has been one of the most difficult combinatorial problems to compute, so it has drawn a lot of research attention because of its theoretical, computational and empirical significance since it was introduced. JSP being such a complex problem, exact techniques such as branch and bound and dynamic programming only apply to small scale problems. However, for large scale problems, the results from these techniques sometimes become really unpredictable and require a lot of time to compute. Heuristic methods

such as dispatching priority rules, shifting bottleneck [36] approach and Lagrangian relaxation [38] are alternatives to large scale problems. With the emergence of new techniques from the field of artificial intelligence, much attention has been devoted to meta-heuristics. One main class of meta-heuristics is the construction and improvement heuristic, such as tabu search [15-17] and simulated annealing [18, 19]. Another main class of meta-heuristic is the population based heuristic. Successful examples of population based algorithms include genetic algorithm (GA) [20-22], particle swarm optimization (PSO) [23, 24], artificial immune system and their hybrids [25-27], and so on. The problem depends on the size, so schedulers are usually satisfied with an acceptance result that is not far from the actual optimum result. One of the search techniques that have been in use in the industry is the Genetic Algorithm (GA).

GA starts with a set of solutions (represented by chromosomes) called population. Chromosomes from the population are taken and used to form a new population. This is motivated by the desire, that the new population will be better than the old one in terms of a fitness criterion. Solutions which a GA starts with a set of solutions (represented by chromosomes) called population.re selected to form new solutions (offsprings) are selected according to their fitness - the more suitable they are the more chances they have to reproduce.

2. RELATED WORK

The most common method used to solve Job Shop Scheduling problem (JSSP) is by material requirement planning (MRP). However, MRP is mostly used as a planning tool and is hardly used for detailed level scheduling [28]. Scheduling is done in many companies by personnel with the help of Gantt charts and databases [29, 30, 28], often resorting to simple dispatching methods to solve immediate problems. This can result in chaos making the prediction of completion dates difficult and the work in progress (WIP) inventory increase.

Many dispatching rules have been implemented based on processing times, criticality of operations and due-dates and resource utilization [31]. The "critical ratio", which calculates the ratio of the remaining processing-time over the time to the due date, is very popular in job scheduling [29]. Some more complicated approaches take into account some combination of the above factors, like the Viviers algorithm [32] incorporates three priorities in the Shortest Processing Time (SPT) rule. Here, each job has a priority and an index equal to the processing time. Jobs having high priority have low index and are processed first. There are also many approaches which use dispatching rules or heuristics for scheduling [33, 34, 35]. It is generally very difficult to evaluate the performance of these methods because of the problem size. Moreover, these methods are not good at accommodating minor changes in jobs/resource availability as the whole schedule has to be redone.

As JSP is such a complex problem, techniques like Branch and Bound [12] and Dynamic Programming [19], which provide exact results, do not help when the problem set is large since these algorithms do not scale. Carlier and Pinson [37] provided 'for the first time' a solution involving 10 jobs and 10 machines. Though it is not a big problem according to industry

standards, the optimal solution required 17982s of CPU time on a PRIME 2655 computer. Moreover the solution might not be optimal once a new job arrives to get scheduled.

In an attempt to bridge the gap between heuristic and optimization approaches, Adams et al. [36] developed a heuristic based on the optimally solving single machine sequencing problems. A criterion for business of the machine was made up and the job sequence for the busiest machine was devised. The next busiest job sequence was then developed and then fed into the previously solved machine problem by "local re-optimization". However, schedule evaluation could only be achieved through selective enumeration.

The Lagrangian relaxation technique has recently been in use for scheduling problems. The method decomposes a problem into a number of smaller parts which makes it easier to solve. Fisher et al. [38] used a fixed lower bound for Lagrangian relaxation to get a more efficient enumeration method for job shop scheduling.

2.1. Chromosome Representation of JSSP

A schedule could be represented in two ways: *indirect* and *direct*. The chromosome contains an encoded schedule in indirect representations. It is transformed into a feasible schedule using a schedule builder. Indirect representations range from traditional binary representation [40] to domain specific knowledge representation.

In direct representation, the production schedule is directly represented by the chromosome. Nakano and Yamada [41] describe many ways for dealing with direct representation. They proposed a conventional genetic algorithm for solving the job shop scheduling problem. They represented the individuals using binary and job-based representation, and applied two-point crossover and bit-flip mutation. The phenotype of the individuals is the job sequence for each machine. Their systems, although good for small scale problems, loose

their utility when the problem size increases. Crafti [42] proposed a genetic tree based approach to solving the JSP problem. This includes crossover between donor tree and a receiver tree. Varying the mutation and crossover operators he reported good average solutions. Cardon et al. [43] proposed an integrated multi-agent and genetic algorithm approach to this problem and later implemented that idea.

2.2. Types of Feasible Schedules in JSSP

There are four classes of feasible schedules: inadmissible, semi-active, active and non-delay. Inadmissible schedule contains too much idle time. Operations could be shifted forward until no idle time exists to better the scheduling quality.

Semi Active schedules have no excess idle time; however they could also be improved by moving forward some operations without delaying others.

Active schedules have no idle time and none of the operations can be finished without delaying other operations. The active schedule guarantees optimal scheduling.

Non-delay schedules are active schedules which minimize the machine idle time. If a machine can be processed then it is not kept idle.

Two methods are applied to generate feasible schedules. The Giffler and Thompson method based on active scheduling and the Non-Delay (ND) algorithm which produces non-delay schedules.

2.3. The Giffler and Thompson Based Algorithms

The Giffler and Thompson method has been used in many JSSP implementations. Lin Goodman and Pinch [44,45], described the representation and the crossover operators used in previous Giffler-Thompson (GT) based GA approaches for JSSP. The offsprings are converted to active schedules to guarantee feasibility with the help of GT based algorithms.

Lin Goodman and Pinch developed two operators: the THX crossover and THX mutation. The temporal relations present in the schedule are transmitted by these operators. Hart and Ross had their own “Heuristic Combination Method” (HCM) [46] which uses an implicit representation where each gene in the chromosome contains a heuristic that performs the decision choice at each step of the schedule generation process.

2.4. Hybrid Methods and Local Search

JSSP being a difficult problem cannot be efficiently solved by a single process so there are many hybrid processes. Jain and Meeran [47] reviewed in detail Tabu Search, Genetic Algorithm and Simulated Annealing techniques and they produce hybrid solutions. Jain, Rangaswamy and Meeran [48] described and compared in detail in JSSP neighborhood models and move evaluation strategies.

3. APPROACH

3.1. Genetic Algorithm

GA uses earlier information to exploit the best solutions from the previous results, called generations, along with some random crossover and mutation to explore new regions in the solution space. In general GA uses three steps- selection, crossover and mutation. Selection based on the fitness (makespan in our case) is the source of exploitation, and crossover and mutation helps us to promote exploration. A generation of a GA contains a population of individuals, each of which corresponds to a possible solution in the search space. Each individual in the population is evaluated with a fitness function to produce a value which indicates the goodness of a solution. Selection helps in bringing forward certain members from the population to apply crossover and mutation on. Crossover takes pairs of individuals and uses parts of each to produce new individuals. Random mutations swap parts of an individual to prevent the GA from getting caught in a local minimum.

3.2. Creation of First Generation

In our problem, we are using the datasets that have been taken from the MetaCentrum workload log provided by the Czech National Grid Infrastructure [50]. This dataset has 5000 jobs which need to be executed by 806 machines. The term makespan refers to the cumulative time to complete all the operations on all machines. It is the time taken from scheduling the first job submitted until the completion of the last job. The objective of the problem is to find a valid schedule that yields the minimum makespan.

Each machine has 6-7 jobs placed on them. The total running time for each machine is then calculated. Since all the machines are running in parallel, the time taken by the machine which runs the longest signifies the total makespan for the entire chromosome. Similar

chromosomes are then created and the makespan for each chromosome is calculated. Once the makespans for all the chromosomes are calculated the least makespan among the chromosome returns the best makespan for the generation while the average of all the chromosome makespan returns the average for that particular generation. Figures 3.1 and 3.2 represent two chromosomes which have three machines and nine jobs placed on the three machines.

M1			M2			M3		
J6	J4	J2	J1	J7	J8	J9	J3	J5

Figure 3.1: Chromosome 1

M1			M2			M3		
J8	J5	J3	J2	J9	J4	J6	J1	J7

Figure 3.2: Chromosome 2

3.3. Selection of Parents for Crossover

Once the makespan is calculated for the different chromosomes, tournament selection is done to filter out those chromosomes which have better makespan values (in this case lesser makespan value) and these chromosomes are then selected to undergo Crossover and Mutation. In this problem the tournament size has been taken to be two. Two chromosomes are randomly chosen from the population and their makespan values are compared, whichever chromosome has a lesser makespan value is deemed the winner. After the parents have been chosen, crossover is applied on them.

3.4. Crossover

Crossover is a genetic operator that combines two parent chromosomes to produce new offspring chromosomes. The idea behind crossover is that the new chromosomes may be better than both of the parents if it takes the best characteristics from each of the parents.

Crossovers can be performed in multiple ways which are briefly discussed below:

3.4.1. One-Point Crossover

The crossover operator randomly selects a crossover point within a chromosome and interchanges the two parent chromosomes at this point to produce two new offspring.

Consider the following 2 parents which have been selected for crossover. The "|" symbol indicates the randomly chosen crossover point.

Parent1: 11001|010

Parent2: 00100|111

After interchanging the parent chromosomes at the crossover point, the following offspring are produced:

Offspring1: 11001|111

Offspring2: 00100|010

3.4.2. Two-Point Crossover

A crossover operator that randomly selects two crossover points within a chromosome then interchanges the two parent chromosomes between these points to produce two new offspring.

Consider the following 2 parents, which have been selected for crossover. The "|" symbols indicate the randomly chosen crossover points.

Parent 1: 110|010|10

Parent 2: 001|001|11

After interchanging the parent chromosomes between the crossover points, the following offspring are produced:

Offspring1: 110|001|10

Offspring2: 001|010|11

3.4.3. Uniform

A crossover operator that decides (with some probability – know as the mixing ratio) which parent will contribute each of the gene values in the offspring chromosomes. This allows the parent chromosomes to be mixed at the gene level rather than the segment level (as with one and two point crossover). For some problems, this additional flexibility outweighs the disadvantage of destroying building blocks.

Consider the following 2 parents, which have been selected for crossover:

Parent 1: 11001010

Parent 2: 00100111

If the mixing ratio is 0.5, approximately half of the genes in the offspring will come from Parent 1 and the other half will come from Parent 2. Below is a possible set of offspring after uniform crossover:

Offspring 1: $1_1 0_2 1_2 0_1 0_2 0_1 1_1 1_2$

Offspring 2: $0_2 1_1 0_1 0_2 1_1 1_2 1_2 0_1$

In our problem we have decided to use one-point crossover where after constructing the parent pool, two chromosomes are selected at random from there and crossover is applied. The crossover point is selected randomly as indicated by the arrow below. The two parent chromosomes are interchanged at this point to give rise to two new off-springs. A scaled-down example of the crossover process is described below in Figure 3.3.

In Figure 3.4, the asterisks show duplicate jobs being placed due to crossover. Since this is not possible in real life, the duplicates have been randomly replaced by unplaced jobs resulting in the following chromosomes as shown in Figure 3.5.

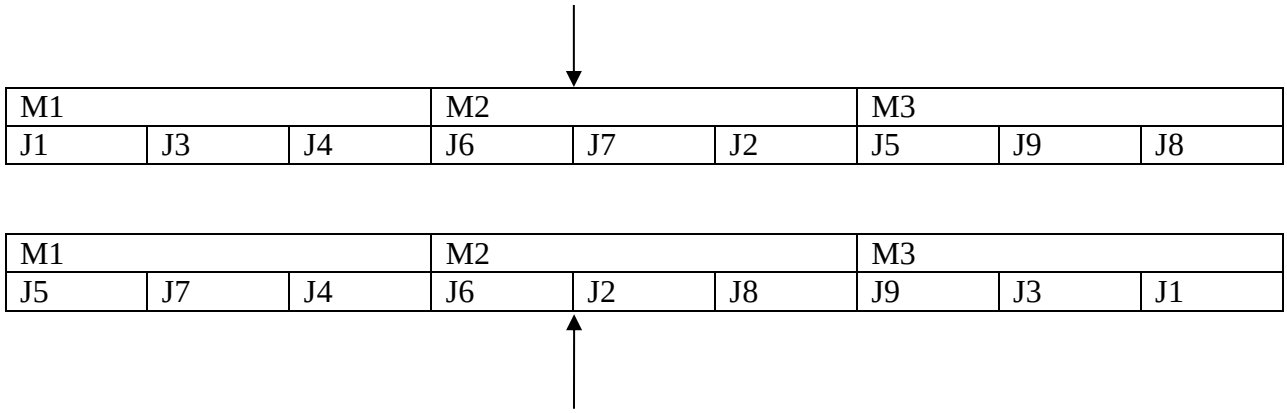

Figure 3.3: Chromosome Encoding before Crossover

After Crossover:

M1			M2			M3		
J1	J3	J4	J6	J2	J8	J9	J3	J1
							*	*

M1			M2			M3		
J5	J7	J4	J6	J7	J2	J5	J9	J8
				*		*		

Figure 3.4: Chromosome Encoding after Crossover

M1			M2			M3		
J1	J3	J4	J6	J2	J8	J9	J5	J7

M1			M2			M3		
J5	J7	J4	J6	J1	J2	J3	J9	J8

Figure 3.5: Both Chromosomes after Removal of Duplicates

Once the duplicates have been removed after crossover, two offspring are produced for the new generation and the makespan for each machine and hence for each chromosome is again calculated**.**

Crossover rate: The crossover rate controls the capability of GAs in exploiting a located hill to reach the local optima. The higher the crossover rate, the quicker the exploitation proceeds. A crossover rate that is too large would disrupt individuals faster than they could be exploited. Typically, the crossover rate has values between 0.3-0.7

3.5. Mutation

Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible. Mutation is an important part of the genetic search as help helps to prevent the population from stagnating at any local optima. Mutation occurs during evolution according to a user-definable mutation probability. This probability should usually be set fairly low (0.01 is a good first choice). If it is set to high, the search will turn into a primitive random search. Mutation can be performed in various ways some of which are described briefly below:

- **Flip Bit:** A mutation operator that simply inverts the value of the chosen gene (0 goes to 1 and 1 goes to 0). This mutation operator can only be used for binary genes.
- **Boundary:** A mutation operator that replaces the value of the chosen gene with either the upper or lower bound for that gene (chosen randomly). This mutation operator can only be used for integer and float genes.
- **Non-Uniform:** A mutation operator that increases the probability that the amount of the mutation will be close to 0 as the generation number increases. This mutation operator keeps the population from stagnating in the early stages of the evolution then allows the genetic algorithm to fine tune the solution in the later stages of evolution. This mutation operator can only be used for integer and float genes.
- **Uniform:** A mutation operator that replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene. This mutation operator can only be used for integer and float genes.

- **Gaussian:** A mutation operator that adds a unit Gaussian distributed random value to the chosen gene. The new gene value is clipped if it falls outside of the user-specified lower or upper bounds for that gene. This mutation operator can only be used for integer and float genes.

In our method we have randomly chosen a gene (job) from each chromosome and flipped the job placement among the chromosome. For example in Figure 3.6, we have randomly chosen job J9 in Chromosome 1 and job J6 in Chromosome 2 and flipped their positions among the chromosomes.

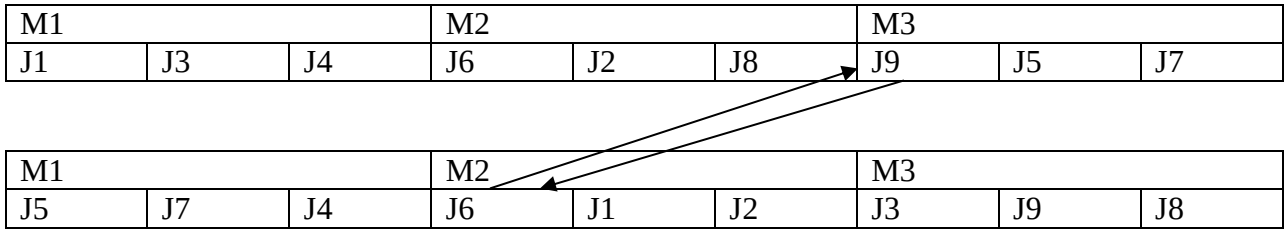

Figure 3.6: Chromosome Encoding before Mutation

M1			M2 *			M3		
J1	J3	J4	J6	J2	J8	J6	J5	J7

M1			M2			M3 *		
J5	J7	J4	J9	J1	J2	J3	J9	J8

Figure 3.7: Chromosome Encoding after Mutation

In Figure 3.7, the asterisks show that there are duplicate jobs being placed on two machines. This duplication is removed by randomly choosing one of the duplicates and replacing it by an unplaced job. Figure 3.8 represents the chromosomes after the duplicate jobs have been removed.

Mutation Rate: Mutation occurs during evolution according to a user-definable mutation probability. In this case the mutation rate has been taken in the range 0.025-0.15.

M1			M2			M3		
J1	J3	J4	J9	J2	J8	J6	J5	J7

M1			M2			M3		
J5	J7	J4	J9	J1	J2	J3	J6	J8

Figure 3.8: Both Chromosomes after Removal of Duplicates

4. IMPLEMENTATION DETAILS

The application developed as a part of this project is a java application consisting of different classes. We will briefly discuss those classes later in this chapter. Inputs required for this application are the number of iterations, tournament size, gridlet size (total number of jobs to be placed), crossover rate and mutation rate. All these inputs are given as parameters at the beginning of the algorithm run.

4.1. RunGA Class

The main class RunGA calls all the other classes in the application. The variables- the number of iterations, tournament size, gridlet size, crossover rate and mutation rate all are assigned values to run the algorithm. This class also helps in plotting the average makespan and best makespan values over the number of iterations in a 2D graph. Table 4.1, shows the variables initialized in this class.

Table 4.1: Variable initialization of RunGA class

VARIABLE	TYPE	PURPOSE
populationSize	Int	Defines the number of chromosomes in any particular generation
tournamentsSize	Int	Defines the tournament size used to select the best chromosomes for crossover. Here the size is 2.
gridLetSize	Int	Total number of jobs that need to be placed
crossoverProb	Double	The crossover rate.
mutationProb	Double	The mutation rate
maxNumOfGenerations	Int	The total number of iterations or generations for which the Genetic Algorithm runs

4.2. Machine, Job, Cluster and DataLoader Classes

Machine.java and Job.java help in getting the description of the machines and the jobs from the data set respectively. There are 806 machines which are further divided into clusters using the cluster.java class. There are 5000 jobs in total. However, depending on the gridlet size parameter as entered we get the description of the jobs accordingly. DataLoader.java class helps in loading the jobs on the machine.

4.3. Selector Class

The Selector.java has a few important methods namely – `selectMachine()`, `selectJob()`, `createChormosome()`,`allocateJobs()` and `createJob()`.

`selectMachine()`: This method selects the machines randomly among the 800 machines.

`selectJob()`: This method selects the jobs randomly among the gridlet size mentioned.

`createChormosome()`: This method sets the machines and allocates the jobs on these machines to create a chromosome.

`allocateJobs()`: This method allocates the jobs on the machines. A map between the machine and the list of jobs to be placed on each machine is created. The jobs are equally distributed among the machines.

4.4. Chromosome Class

Some of the more important methods in this class are `crossover()`, `calculateFitness()`, `clearDuplicates()` and `mutation()`.

- `crossover()`: This is the method responsible for execution of the crossover after the best chromosomes have been selected using tournament selection. The one point crossover as has been discussed earlier has been executed here.
- `calculateFitness()`: This method calculates the makespan values after crossover or mutation has been applied.

- `clearDuplicates()`: This method removes all the duplicate jobs that might have been placed on more than one machine as a result of crossover or mutation. After crossover or mutation the duplicates among the jobs are randomly chosen and randomly replaced by un-placed jobs.
- `mutation()`: This method helps in implementation of mutation as discussed earlier.

4.5. GeneticAlgorithm Class

Among the important methods in this class are `createInitialPopulation()`, `startGeneration()` and `tournamentSelection()`.

- `createInitialPopulation()`: This method helps in creating the first set of chromosomes which would then be subjected to tournament selection and then crossover and mutation.
- `startGeneration()`: This method is always called to initiate the crossover and mutation once the tournament selection for a particular generation has been done.
- `tournamentSelection()`: This method helps in randomly selecting 2 chromosomes from the population and then choosing the best among them with respect to the one having a lower makespan value.

5. EXPERIMENTS AND RESULTS

This section describes various experiments that are performed by changing the parameters, i.e. crossover rate, the mutation rate and the population size. Each experiment has been done 10 times each, and the average makespan, the best makespan and the running times of each experiment are noted and compared to the results we got in the related experiments.

5.1. Experiment 1: Varying the Crossover Rate

As discussed earlier we have done a one-point crossover between the parents selected from the tournament selection. The crossover rates have been varied between 0.3 to 0.7. The mutation rate has been kept fixed at 0.025. The population size is 100, and the number of generations is 250. A total of 5000 jobs were placed on 806 machines during each experiment.

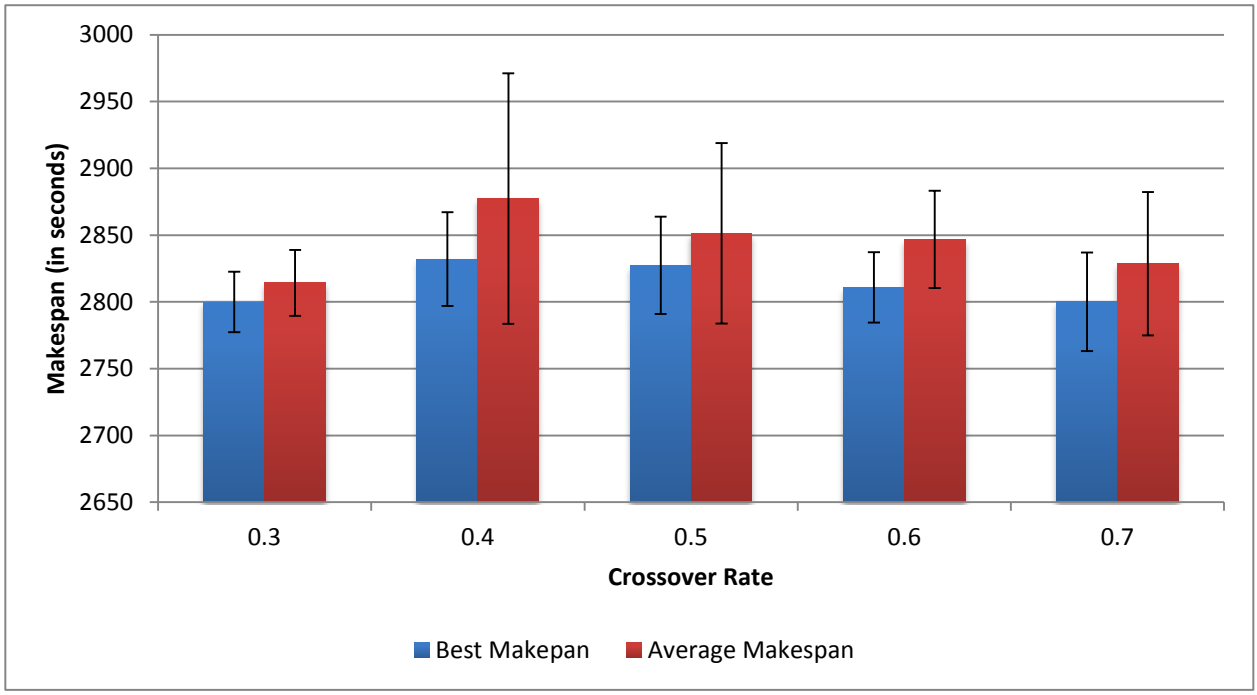

Figure 5.1: Makespan versus Crossover Rate

Figure 5.1 displays the result distribution in a bar chart diagram together with standard deviation bars. As can be seen from the figure, a crossover rate of 0.3 provides the best result of best and average makespan values of 2777.02 seconds respectively.

A running time analysis has also been done for each data point. As is represented in Figure 5.2, the average running time when the crossover rate is 0.3 is minimum, and the running time for this value of crossover also has the second lowest standard deviation.

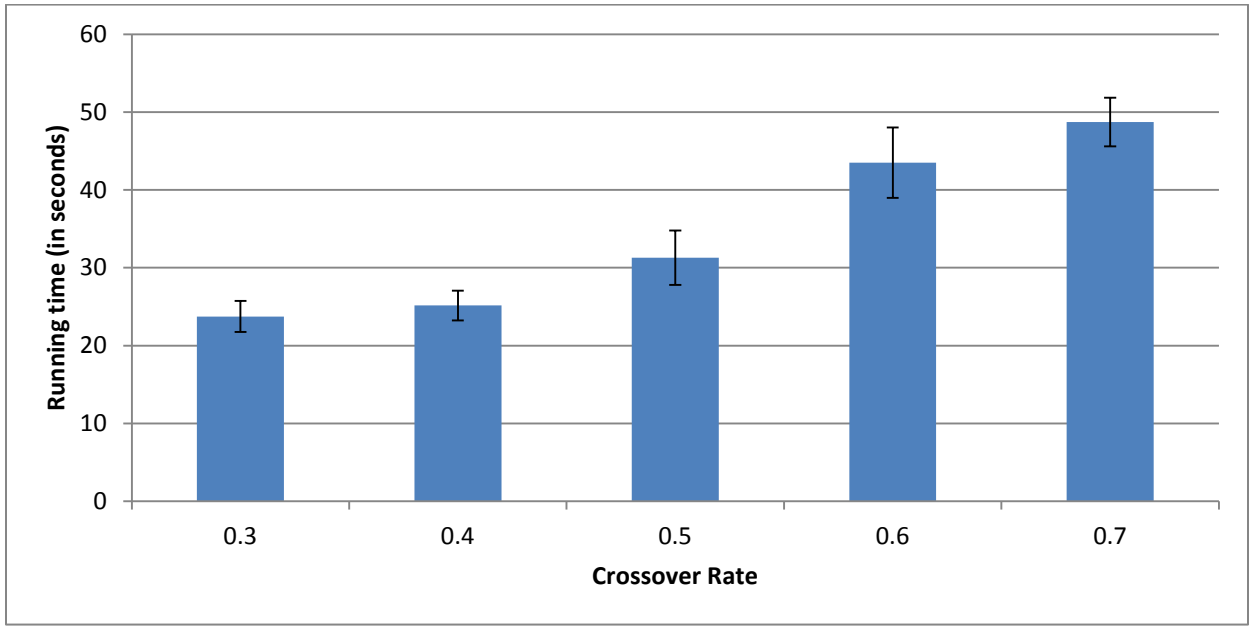

Figure 5.2: Running Time versus Crossover Rate

Since we see that a crossover rate of 0.3 provides the best results we maintain the crossover rate to be 0.3 in our future experiments in a bid to obtain optimum results.

5.2. Experiment 2: Varying the Mutation Rate

Next we change the mutation rate within the range of 0.025 to 0.15, keeping the crossover rate at 0.3, and the population size at 100.

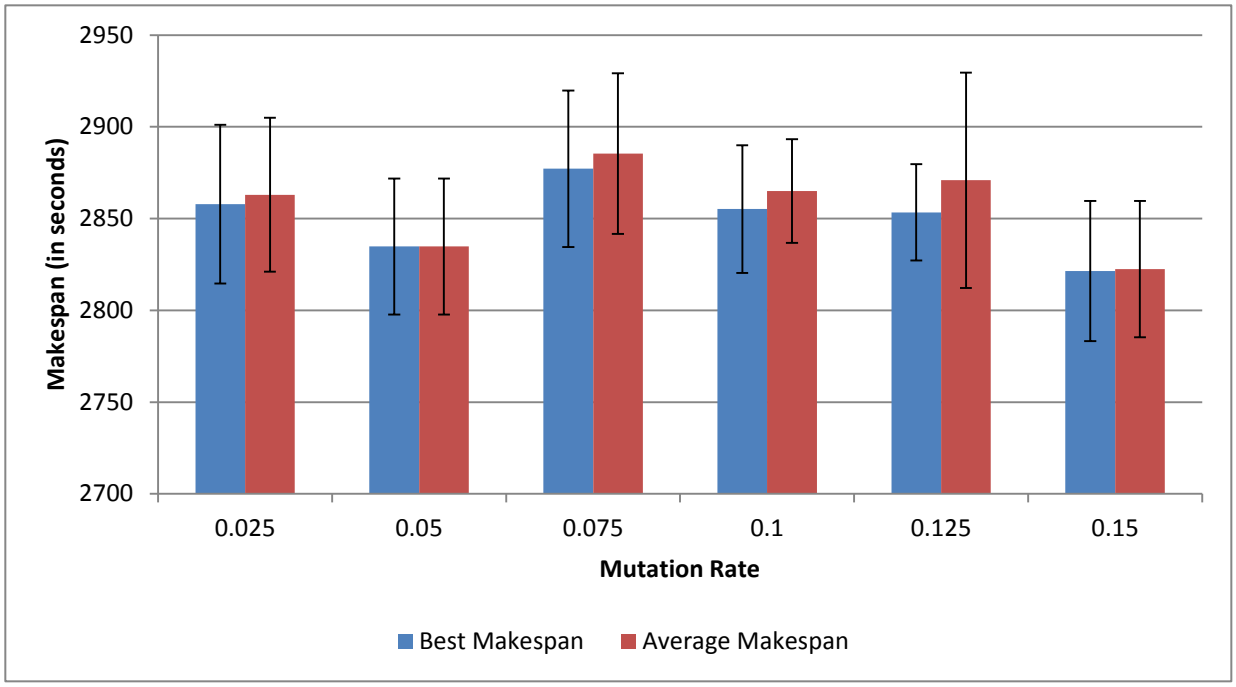

Figure 5.3: Makespan versus Mutation Rate

We observe from Figure 5.3, that having a mutation rate of 0.15, we obtain the lowest makespan / best makespan of 2800.5 seconds, and the average makespan of 2802.9 seconds.

However, the running time analysis shows that the average running time for a mutation rate of 0.15 is the highest. In addition, the standard deviation of the running times for the 10 experiments is the highest among the comparable values, as shown in Figure 5.4.

Hence, in this situation we had to do a tradeoff between the makespan values and the running time of the algorithm. Since the makespan values that we have got for a mutation rate of 0.15 are significantly lower than the other makespan values, we choose the mutation rate to be 0.15 for the remaining experiments.

5.3. Experiment 3: Varying the Population Size

Finally, we change the population size i.e. the number of chromosomes in the population and observe the changes in the best makespan and the average makespan values, keeping the crossover rate at 0.3 and the mutation rate at 0.15.

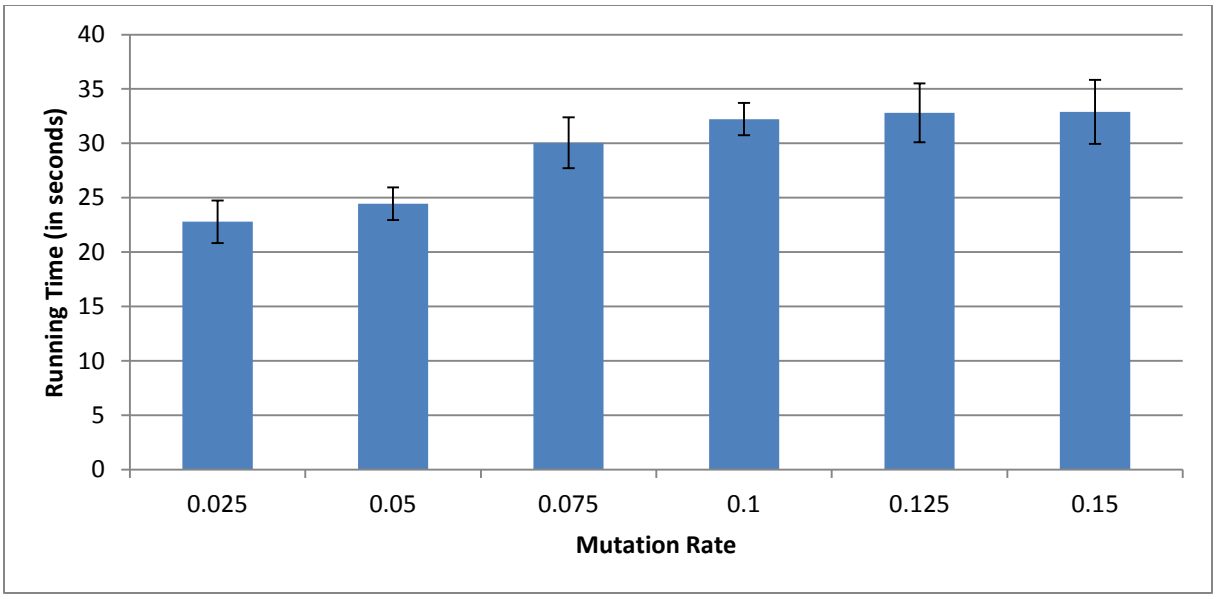

Figure 5.4: Running Time versus Mutation Rate

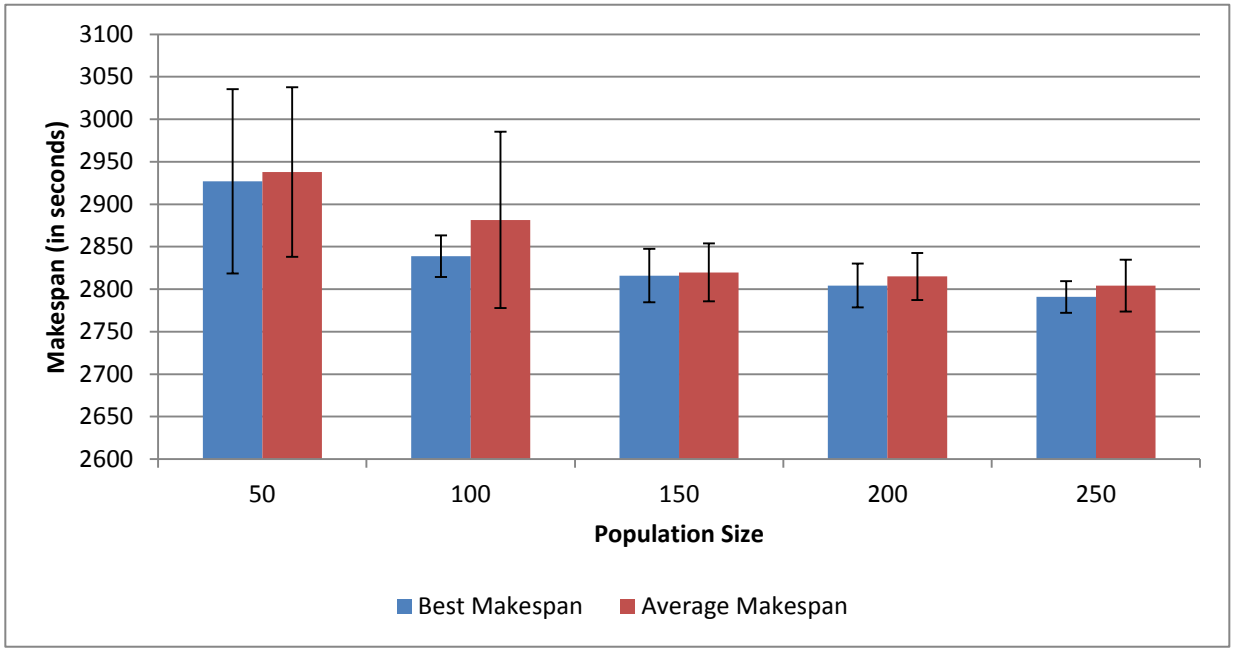

Figure 5.5: Makespan versus Population Size

Figure 5.5 shows that for a population size of 250, we obtain the best makespan and average makespan values.

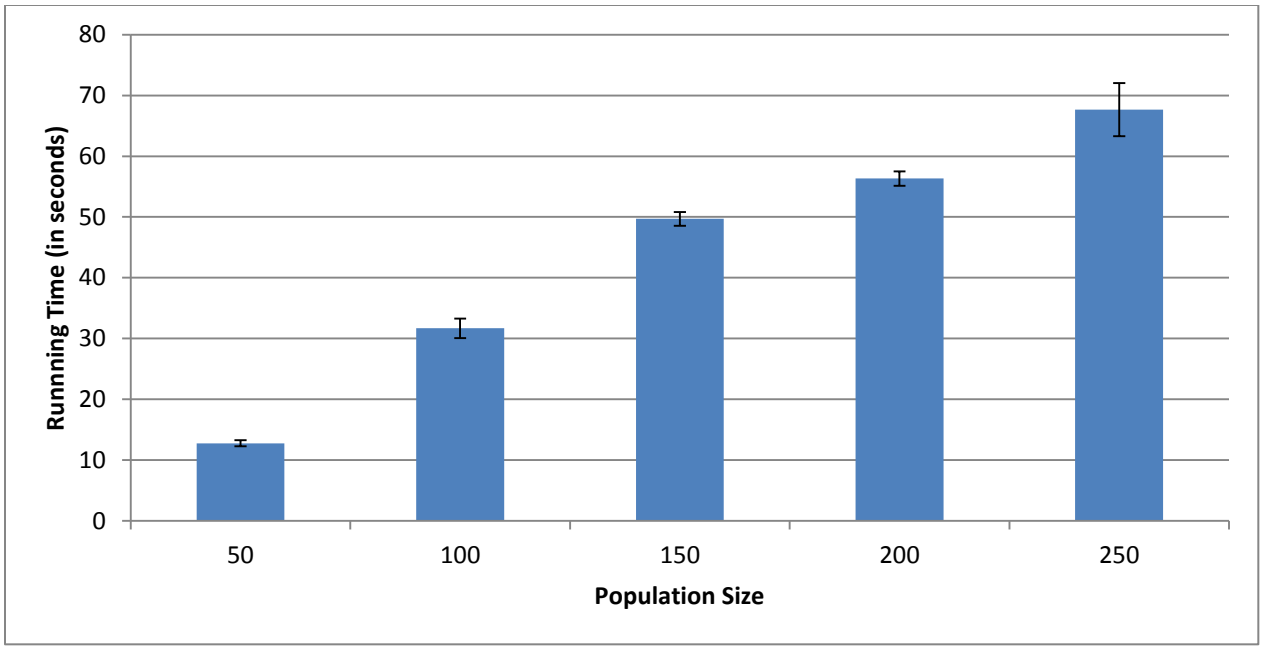

Figure 5.6: Running Time versus Population Size

The running time analysis is presented in Figure 5.6. In spite of getting much better makespan values for a population size of 250, we choose the population size to be 100 as the running time for a population size of 250 is almost double than that of a population size of 100.

Finally keeping the crossover rate, the mutation rate, and the population size fixed at values that we decided upon above (crossover rate of 0.3, mutation rate of 0.15, and population size of 100), we investigate the makespan and running time when the gridlet size i.e. the number of jobs from 1000 to 5000, is varied.

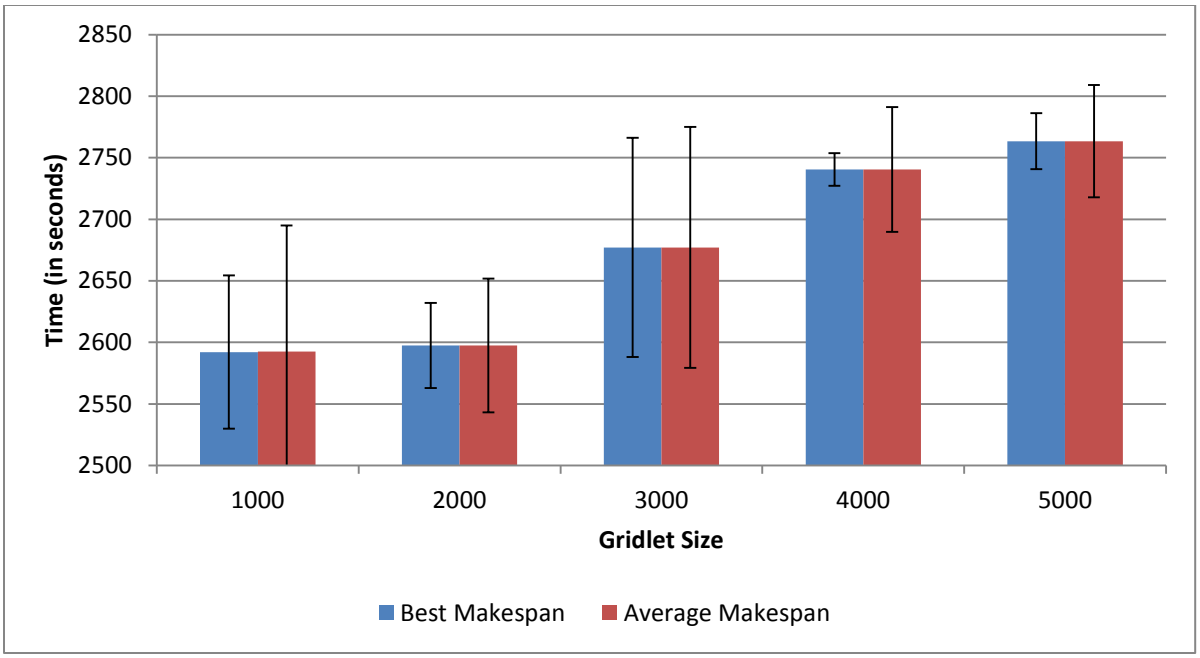

Figure 5.7: Makespan versus Gridlet size

As expected, Figure 5.7 shows that in order to place 5000 jobs onto the different machines, the makespan values are expectedly higher than those when the number of jobs is smaller. The average makespan and the best makespan for placing 5000 jobs on 806 machines with a crossover rate of 0.3, a mutation rate of 0.15, and a population size of 100 is 2850 seconds for both.

Looking at the makespan in terms of increasing number of generations, as shown in Figure 5.8, it is observed that the best makespan is found after 25 generations.

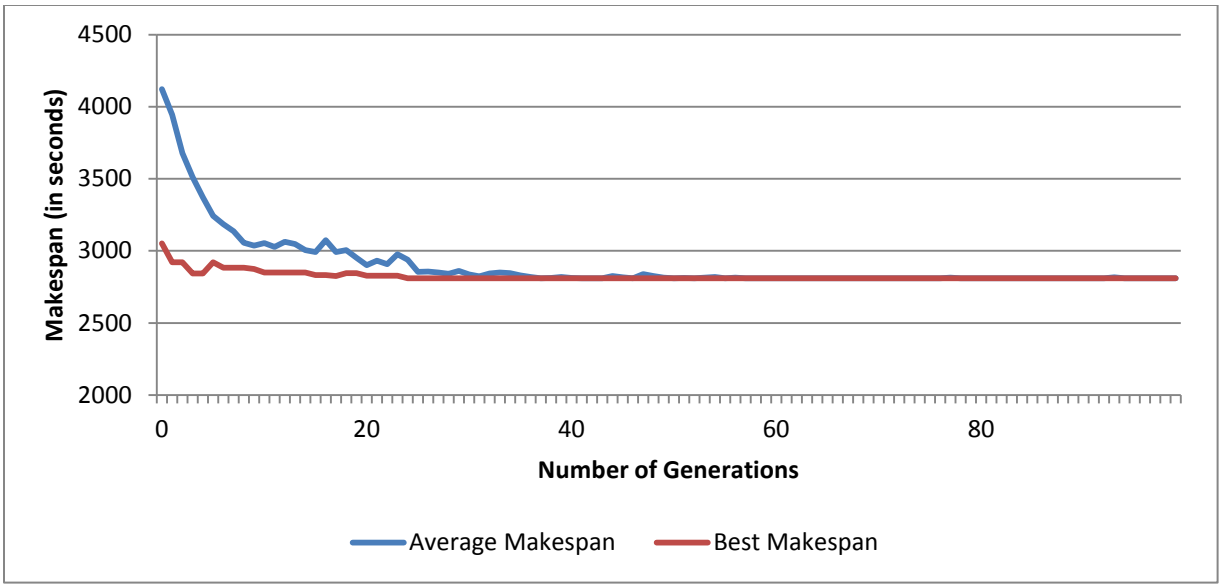

Figure 5.8: Makespan versus Number of Generations

6. CONCLUSION AND FUTURE WORK

In this paper, we used a Genetic Algorithm to minimize the makespan of placing 5000 jobs on 806 machines. In order to achieve that, first we randomly placed the 5000 jobs on the machines in the form of a chromosome and generated 100 of these chromosomes. Then, we calculated the makespan for each chromosome and selected the best chromosomes using tournament selection. Afterwards, we performed crossover and mutation with specific parameters and replaced the newly generated chromosomes with the previous ones in the population. This process was repeated for a predefined number of generations.

We conducted experiments by varying the different parameter values such as crossover rate, mutation rate, and population size in order to analyze the effect of the parameter values on the outcome of the optimization, and specifically on the makespan. Our results show that though the value of the resulting makespan may vary from one experiment to the other, over a series of 10 experiments and taking the average values of the outcome each time, the best combination of the parameter values that we have arrived at when balancing the makespan with the running time is: crossover rate of 0.3, a mutation rate of 0.15, and a population size of 100. These values obtained the same average and best makespan value of 2850 seconds.

However, the dataset chosen for our research work has constant execution times on all the machines and the capacity of each of the 806 machines has been assumed to be exactly the same. This might not always be true in real life scenarios since the capacity of each machine to run particular jobs may be different from each other. Also, different jobs might have different running times for different machines. This needs to be further investigated.

REFERENCES

[1] M. Holena. 2008. “Genetic algorithms for the optimization of catalysts in chemical engineering”. *Proceedings of the World Congress on Engineering and Computer Science.*

[2] R. Luus and B. Bojkov. 2009. “Global optimization of the bifunctional catalyst problem”. The Canadian Journal of Chemical Engineering.

[3] L. dos Santos Coelho and V.C. Mariani. 2006. “An efficient particle swarm optimization approach based on cultural algorithm applied to mechanical design”. *Evolutionary Computation, 2006. CEC 2006, pages 1099–1104.*

[4] Q. He and L. Wang. 2007. “An effective co-evolutionary particle swarm optimization for constrained engineering design problems”. *Engineering Applications of Artificial Intelligence, 20(1):89–99.*

[5] F. Muyl, L. Dumas, and V. Herbert. 2004. “Hybrid method for aerodynamic shape optimization in automotive industry”. *Computers and Fluids, 33(5-6):849–858.*

[6] Y. Choi and H. G. Stenger. 2005. “Kinetics, simulation and optimization of methanol steam reformer for fuel cell applications”. *Journal of Power Sources, 142(1-2):81–91.*

[7] F. Van den Bergh and A.P. Engelbrecht. 2000. “Cooperative learning in neural networks using particle swarm optimizers”. *South African Computer Journal, (26):84–90.*

[8] M.Y. Wang, X. Wang, and D. Guo. 2003. “A level set method for structural topology optimization”. *Computer Methods in Applied Mechanics and Engineering, 192(1-2):227–246.*

[9] M.T. Jonsson and T.P. Runarsson. 1996. “Optimizing the sailing route for fixed groundfish survey stations”. *International Council for the Exploration of the Sea.*

[10] L. Chan, J. Karceski, and J. Lakonishok. 1999. “On portfolio optimization: forecasting covariances and choosing the risk model”. *Review of Financial Studies, 12(5):2937–2974.*

[11] A.C. Zecchin, A.R. Simpson, H.R. Maier, and J.B. Nixon. 2005. “Parametric study for an ant algorithm applied to water distribution system optimization”. *IEEE Transactions on Evolutionary Computation, 9(2):175–191.*

[12] J. Chuanwen and E. Bompard. 2005. “A self-adaptive chaotic particle swarm algorithm for short term hydroelectric system scheduling in deregulated environment”. *Energy Conversion and Management, 46(17):2689–2696.*

[13] S. Liu and J. Wang. 2009. “An improved self-adaptive particle swarm optimization approach for short-term scheduling of hydro system”. *International Asia Conference on Informatics in Control, Automation and Robotics. CAR 2009, pages 334–338.*

[14] T. Niknam. 2010. “A new fuzzy adaptive hybrid particle swarm optimization algorithm for non-linear, non-smooth and non-convex economic dispatch problem”. *Applied Energy, 87(1):327–339.*

[15] C. Y. Zhang, P. G. Li, Y. Q. Rao. 2008. “A very fast TS/SA algorithm for the job shop scheduling problem”. *Computers & Operations Research, vol. 35, pp. 282-294.*

[16] E. Nowicki, C. Smutnicki. 1996. “A fast taboo search algorithm for the job shop scheduling problem”. *Management Science, vol. 41, no. 6, pp. 113-125.*

[17] A. M. Dell, M. Trubian. 1993. “Applying tabu-search to job shop scheduling problem”. *Annals of Operations Research, vol. 41, no. 3, pp. 231-252.*

[18] T. Y. Wang, K. B. Wu. 2000. “A revised simulated annealing algorithm for obtaining the minimum total tardiness in job shop scheduling problems”. *International Journal of Systems Science, vol. 31, no. 4, pp. 537-542.*

[19] M. Kolonko. 1999. “Some new results on simulated annealing applied to the job shop scheduling problem”. *European Journal of Operational Research, vol. 113, no. 1, pp. 123-136.*

[20] I. Moon, S. Lee, H. Bae. 2008. “Genetic algorithms for job shop scheduling problems with alternative routings”. *International Journal of Production Research, vol. 46, no. 10, pp. 2695-2705.*

[21] J. F. Goncalves, J. J. D. M. Mendes, M. G. C. Resende. 2005. “A hybrid genetic algorithm for the job shop scheduling problem”. *European Journal of Operational Research, vol. 167, no. 1, pp. 77-95.*

[22] C. Bierwirth, D. C. Mattfeld. 1999. “Production scheduling and rescheduling with genetic algorithms”. *Evolutionary Computation, vol. 7, no. 1, pp. 1-17.*

[23] B. Liu, L. Wang, Y. H. Jin. 2008 “An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers”. *Computers & Operations Research, vol. 35, no. 9, pp. 2791-2806.*

[24] M. F. Tasgetiren, Y. C. Liang, M. Sevkli. 2007. “A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem”. *European Journal of Operational Research, vol. 177, no. 3, pp. 1930-1947.*

[25] H. W. Ge, L. Sun, Y. C. Liang, F. Qian. 2008. “An effective PSO-and-AIS-based hybrid intelligent algorithm for job-shop scheduling”. *IEEE Transactions on System, Man and Cybernetics, Part A, Systems and Humans, vol. 38, no. 2, pp. 358-368.*

[26] C. A. C. Coello, D. C. Rivera, N. C. Cortes. 2003. “Use of an artificial immune system for job shop scheduling”. *Lecture Notes in Computer Science, vol. 2787, pp. 1-10.*

[27] J. H. Yang JH, L. Sun, H. P. Lee, Y. Qian, Y. C. Liang. 2008. “Clonal selection based memetic algorithm for job shop scheduling problems”. *Journal of Bionic Engineering, vol. 5, no. 2, pp. 111-119.*

[28] K. P. White Jr. 1990. "Advances in the theory and practice of scheduling”. *Advances in Industrial Systems, Control and Dynamic Systems, vol 37, CT Leondes, Ed. San Diego, CA: Academic, pp. 115-158.*

[29] S.C. Graves. 1981. “ A review of production scheduling”. *Operations Res. Vol. 18, pp. 841-852.*

[30] K. N. McKay. 1988. F. R. Safayeni and J.A. Buzacott, “Job-shop Scheduling theory: What is relevant?”. *Interfaces, vol. 18, pp 84-90.*

[31] J.H. Blackstone, D. T. Phillips and G. L. Hogg. 1982. “A state of the art survey dispatching rules for manufacturing job shop operations”. *Int. J Productions Res., vol. 20. Pg 27-45.*

[32] F. Viviers. 1983. “A decision support system for job shop scheduling”. European J *Operational Res., vol. 14, no. 1, pp 95-103.*

[33] M.S. Fox and S.F. Smith. 1984. “ISIS- A knowledge based system for factory scheduling”. *Expert Syst., vol 1, pp. 25-49.*

[34] A. Kuziak. 1990. “Intelligent Manufacturing Systems. Englewood Cliffs”. *NJ: Prentice-Hall.*

[35] P.S. Ow, S.F. Smith and R. Howie. 1988. “A cooperative scheduling systems”. *Proc. 2nd Int. Conf. Expert Syst. Leading Edge in Production Planning Contr.*

[36] J. Adams, E. Balas and D. Zawack. 1988. “ The shifting bottleneck procedure for job shop scheduling”. *Management Sci. vol 34, no. 3, pp. 391-401.*

[37] J. Carlier and E. Pinson. 1989. “An algorithm for solving the job-shop problem”. *Manag. Sci. vol 35, no. 2, pp. 164-176.*

[38] M. L. Fisher. 1973. “Optimal solution of scheduling problems using Lagrange multipliers, Part 1”. *Operations res. Vol 21, pp 1114-1127.*

[39] M. Held and R. M. Karp. 1962. “A dynamic programming approach to sequencing problems”. *Journal of the Society for Industrial and Applied Mathematics, vol. 10, pp 196-210.*

[40] R. Nakano, T. Yamada. 1991. “Conventional Genetic Algorithm for Job Shop Problems”. *International Conference Genetic Algorithm.*

[41] T. Yamada, and R. Nakano. 1992. “A Genetic Algorithm Applicable to Large scale Job-Shop Problems”. *In parallel problem solving from Nature 2 (PPSN 2).*

[42] D. Crafti. 2004. "A Job Shop Scheduler using a Genetic Tree Algorithm". *School of Computer Science and Software Engineering, vol. Ph.D. Melbourne, Australia: Monash University, Clayton Campus, p. 63.*

[43] A. Cardon, T. Galinho, and J.-P. Vacher. 2000. "Genetic algorithms using multi-objectives in a multi-agent system". *Robotics and Autonomous Systems, vol. 33, p. 179.*

[44] S. Lin, E. D. Goodman, W. F. Punch. 1997. “A genetic Algorithm Approach by Dynamic Job Scheduling Problems”. *International Conference Genetic Algorithm.*

[45] S. Lin, E. D. Goodman, W. F. Punch. 1997. “Investigating Parallel Algorithms on Job Shop Scheduling Problems”. Evolutionary programming Conference.

[46] E. Hart and P. Ross. 1988. *“c”. Parallel Problem Solving from Nature.*

[47] A. S. Jain, S. Meeran. 1998. “A state of the art review of Job Shop Scheduling techniques”. *Submitted to Journal of Heuristics.*

[48] A. S. Jain, B. Rangaswamy, S. Meeran, "Job Shop Neighbourhoods and Move Evaluation Strategies". *Journal of Scheduling, 1998.*

[49] S. Bagchi, S. Uckum, Y. Miyabe, K. Kawanura. 1991. "Exploring Problem-Specific recombination operators for Job Scheduling". *International conference on Genetic Algorithm.*

[50] Dalibor Klusácek, Hana Rudová. 2010. "Alea 2 – Job Scheduling Simulator". *Third International ICST Conference on Simulation Tools and Techniques.*