International Refereed Journal of Engineering and Science (IRJES)
ISSN (Online) 2319-183X, (Print) 2319-1821
Volume 3, Issue 7 (July 2014), PP.39-49

Enhancing Performance of Genetic Algorithm for Static Job-Shop Scheduling Problems

K.Luchoomun, P.Auckloo, B.Sonah
Dept. of Computer Science, University of Mauritius, Mauritius

Abstract:- The Job Shop scheduling problem is one of the hardest combinatorial optimisation problems in various industrial environments. It consists of several jobs each with one or more operations, to be allocated to a set of machines with a view to minimise the makespan. In this paper, we begin by describing the Job Shop scheduling problem and the evolutionary algorithm used to solve the machine allocation problem. We then, propose additional genetic operators (random selection of chromosomes to initialise the population, measuring the initial population fitness to identify good chromosomes, cloning to improve survival probability) to enhance the existing genetic algorithm in terms of getting a shorter makespan in less computational time. Finally, we investigate the effect of population size, crossover rate and mutation rate on the makespan for both the existing genetic algorithm and the enhanced genetic algorithm. Our result shows that the enhanced genetic algorithm produce the optimal makespan in a shorter genetic evolution than existing genetic algorithm.

Keywords:- Evolutionary Algorithm (EA), Genetic Algorithm (GA), Job Shop Scheduling Problem (JSSP), Non-deterministic polynomial time (NP)

I. INTRODUCTION

Job Shop Scheduling (JSS) is an important activity in several industrial environment, especially in manufacturing planning and production line system. The growing complexity of operations and product-mix (total number of product lines that a manufacturer offers) have increased the problem complexity of the Job Shop. Over the last decade, research in scheduling has witnessed a steady increase in importance of machine allocation and jobs scheduling sequence in industrial productivity and performance.[1], [2]

The classical Job Shop scheduling problem is one of the most complicated scheduling problems with no accurate algorithms that yield the minimal makespan. Job Shop scheduling problem falls under two categories. In a flexible Job Shop[3],[4] scheduling problem (FJSSP) an operation can be processed by any machine from a given task set whereas in a static Job Shop scheduling (SJSSP) operations of job are processed according to a predetermined machine sequence.

In past research, several analytical techniques have been devised to solve the complex machine scheduling problem in the hope of finding an optimal or near-optimal schedule. The analytical techniques[1],[4] consist of linear programming, branch and bound and, heuristic approaches like neighborhood methods and priority rules. However, they fail to solve large job-shop problems where the number of machines, jobs and operations grow in number. For example, branch and bound technique becomes less reliable as the number of machines exceed 3.

In recent times, most of the studies have turned to new techniques[1],[4] such as simulation, artificial intelligence and genetic algorithm. These methods have proved a significant step forward in solving large job-shop scheduling problem with less computational effort and more optimal schedules.

In this paper, we focus on the static Job Shop scheduling problem where the machine sequence of the operations has already been determined. The following assumptions are considered:

1. There exists a finite set of *n* jobs, each job consisting of a set of operations with deterministic processing times and following a machine sequence. All jobs are synchronous, that is, they are released at time t = 0.
2. There exists a finite set of *m* machines where each machine can handle at most one operation at a given point in time till completion without pre-emption.
3. An operation of a job can only start provided the previous operation of the same job has completed.
4. Context switch overhead and migration cost are ignored.

In this paper, we bring additional genetic operators to enhance the existing genetic algorithm for the static Job Shop scheduling problem. The additional genetic operators consist of (a) creating the initial population with randomly selected chromosomes with their fitness value in view of getting good chromosomes

(b) measure the chance of survival of chromosomes and (c) cloning to take good components from the population to the next generation. We also, show through our experimental evaluation how the above enhancements yield a speedup in getting the optimal schedule in less computational time. Finally, we show the effect of population size, crossover rate and mutation rate on the makespan.

II. LITERATURE REVIEW

2.1 Job Shop Scheduling Problem

During the last decades, a large number of researches including a comprehensive study of scheduling problem have been developed to find a solution to JSSP. In the mid-fifties around 1954, Johnson introduced the scheduling problem for the first time, in the firm of a paper and in the forthcoming years, several studies have been developed to discuss the Job Shop problem [1],[5]. In general, JSSP is NP-hard and is probably one of the most computationally intractable combinatorial problems considered so far. As a matter of fact, one 10 × 10 (10 machines, 10 jobs) problem formulated by Muth and Thompson [6] remained unsolved for over 20 years. NP-hard refers to a class of problems where there is no efficient procedure for exactly finding shortest schedules for the Job Shop scheduling problem.

In general, the static Job Shop scheduling problem consists of *n* jobs and *m* machines where each job consists of one or more operations to execute on a predefined sequence of machines with known processing times. The *n* × *m* static Job Shop scheduling problem, can be described by a set of *n* jobs $\{ J_j \}_{1 \le j \le n}$ which is to be processed on a set of *m* machines $\{ M_k \}_{1 \le k \le m}$. A job J_j consists of a set of operations $\{ O_i \}_{1 \le i \le p}$, where operation O_{ij} represents the operation i of job j. P_{ij}^{k} represents the processing time of operation *i* of job *j* on machine *k* and T_{ij}^{k} represents the *starting time* of operation O_{ij} performed on machine M_k. A *schedule* represents the allocation of the operations of the jobs on the different machines according to the predetermined machine sequence. The time at which all the operations complete is called the *makespan*. A smaller makespan represents a better response time and processor utilisation.
Table 1 shows an example of a 3×3 static Job Shop scheduling problem.

Table 1: 3x3 Static JSSP [6]

Jobs	Machine (processing time)		
1	O_{11} = 1(3)	O_{21} = 2(3)	O_{31} = 3(3)
2	O_{12} = 1(2)	O_{22} = 3(3)	O_{32} = 2(4)
3	O_{13} = 2(3)	O_{23} = 1(2)	O_{33} = 3(1)

For example, the entry O_{23} = 1(2) means operation 2 of Job3 is scheduled to run on machine 1 with execution time 2.

2.2 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are basically used for searching procedure based on the mechanisms of natural selection and population genetics for solving complex problems by mimicking the processes of Darwinian evolution. These algorithms have been applied by many users in different areas of engineering, computer science and operations research. Current evolutionary approaches include evolutionary programming, evolutionary strategies, genetic algorithms and genetic programming. In genetic programming, the solutions are in the form of computer programs, and their fitness is determined by their ability to solve a computational problem. Another approach of EA is the evolution strategy, which works with vectors of real numbers as representation of solutions, and usually uses self-adaptive mutation rates. Other related technique includes the Bees Algorithm, which is based on foraging behaviour of honey bees. The bees algorithm has been applied in many applications such as routing and scheduling. Evolutionary algorithms can outperform conventional optimisation methods when applied to difficult real-world problems. [3],[8]
In the following section, we describe the fundamental features of the genetic algorithm (GA).

2.3 Genetic Algorithm

In the industrial engineering world, in particular the manufacturing systems, many optimisation problems are very complex in nature and quite hard to solve by conventional optimisation techniques. Since 1960, an interest in imitating living beings based on the mechanism of natural selection and natural genetics has emerged to solve such kinds of hard optimisation problems. The usual form of genetic algorithm was described by Goldberg and differs from conventional search techniques [8].

Genetic algorithm starts with an initial set of random solutions called ***population***. Each individual in the population is called a ***chromosome*** which represents a solution to the problem and which is encoded as a string of symbols, usually as a binary bit string. The chromosomes evolve through successive iterations, called ***generations***. During each generation, the chromosomes are evaluated, using measures of ***fitness***. Creating the next generation, new chromosomes, called ***offspring***, are formed by (a) merging 2 chromosomes from current generation using a ***crossover operator*** and (b) modifying a chromosome using a ***mutation operator*** [7]. Among the chromosomes of the current generation, and based on their fitness, some will be rejected and some will be retained to form the ***new generation*** together with their offspring. With evolution, the algorithm converges to the best chromosomes, which represent the solutions converging to the optimal one. This evolution is described in Figure 1. [8]

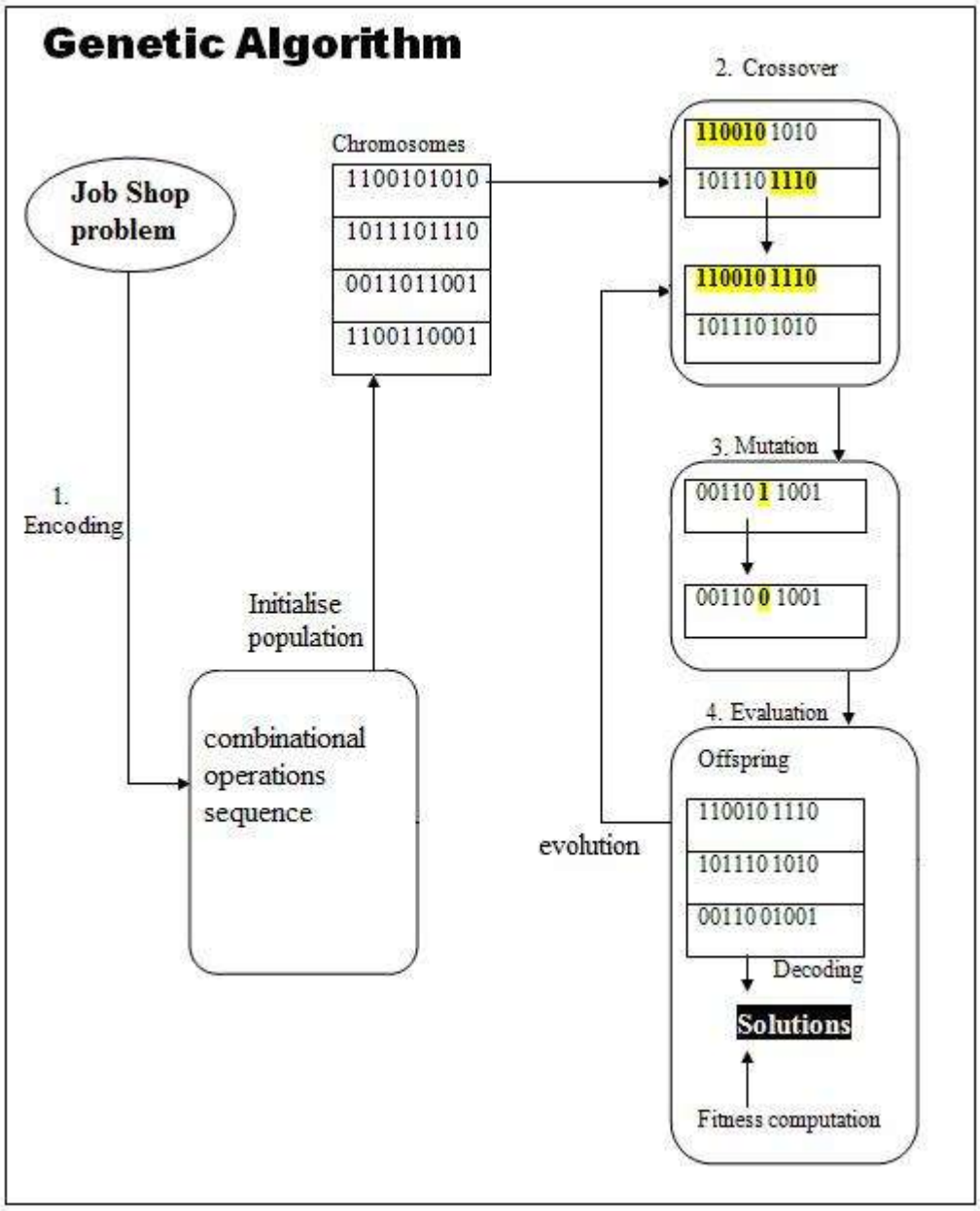

Fig. 1: Structure of Genetic Algorithm

In reality, there exist only four kinds of operations in genetic algorithm, namely (1) encoding of chromosomes using combinational technique, (2) crossover, (3) mutation and (4) evaluation of fitness. [8]

Encoding of chromosomes

A direct approach representation has been considered for the encoding of chromosomes. In the direct approach, a schedule (the solution of JSSP) is encoded into a chromosome, and genetic algorithm is used to

evolve those chromosomes to determine a better schedule. Gen, Tsujimura, and Kubota devised an operation-based representation which consists of encoding a schedule as a sequence of operations, each operation represented by a gene. They name all operations for a job with the same symbol and then interpret them according to the order of occurrence in the given chromosome. Let us consider an example: (3×3) static JSSP as shown in Table 1.

The length of the chromosome is determined by the total number of operations in the (3×3) Static JSSP, which is 9. Suppose a chromosome is given as [3 2 2 1 1 2 3 1 3]. where each element represents a gene or fundamentally an operation. We notice that there are three occurrence of digit 1, 2 and 3. This means that Job1, Job2 and Job3 have 3 operations each. The table below illustrates an example of a chromosome representation.

Gene	0	1	2	3	4	5	6	7	8
Chromosome	3	2	2	1	1	2	3	1	3

The order of arrangement of the genes is important, as it represents the sequence in which the operations are assigned to respective machines. Consider gene 6. Gene 6 belongs to Job3 and corresponds to its second operation as it occurs second in the gene string.

Crossover

Crossover is the main genetic operator that operates on two chromosomes at a time by swapping one or more genes among them, generating two new chromosomes, that is, their offsprings.[4],[9] Figure 2 shows two new chromosomes as a result of a crossover between chromosome 1 and chromosome 2 starting at a cut-point and ending at another. The crossover can involve a random subset of genes between the cut-points. For example in the diagram below, a crossover of 3 genes occurred between Chromosome 1 and Chromosome 2.

Chromosome 1	11011	0**10**11	10110
Chromosome 2	11011	**100**001	11110
Offspring 1	11011	0**100**11	10110
Offspring 2	11011	**010**001	11110

Cut-points

Fig. 2: Crossover

The *crossover rate*, denoted by C_R, is defined as the ratio of the number of offspring produced in each generation to the population size P. The parameter is very important as it determines the crossover count C_c, that is, the number of offspring produced in a crossover process, which is given by $\lfloor P \times C_R \rfloor$. For example if the population size is 10 and the crossover rate is 0.2, two new offsprings will emerge from a crossover.

Mutation

Mutation is an operator which produces random changes in a particular chromosomes. [4],[9],[10] The mutation operator serves a crucial role in the genetic algorithm, by re-introducing a new gene that was lost during the evolution of the chromosome. For example in Figure 3, the mutation process has re-introduced the good gene '1' at fourth position and gene '0' at last position.

Previous generation: **Chromosome (good)**	110**1**111000011110
Current generation: **Offspring**	110**0**111000011111
Next generation: **Mutate**	110**1**111000011110

Fig. 3: Mutation

The *mutation rate*, denoted be M_R , is a parameter that controls the number of genes to be mutated, that is, the number of new genes to be introduced in the population for trial. This number is denoted mutated count M_c and is given by $\lfloor C_c \times M_R \rfloor$. The positions of genes to be mutated are chosen at random.

Evaluation of fitness

A fitness evaluation function is used to measure the fitness of a chromosome. Since a chromosome corresponds to a schedule with its makespan value, the latter is used to determine the fitness of the former. The lower the makespan, the more fit the chromosome.

III. ENHANCEMENT FEATURES

Unlike other search and optimisation techniques, genetic algorithm *promises convergence but not optimality.* This implies the choice when to stop a genetic algorithm is not well defined. The genetic algorithm process will stop when generations have gone by with no better chromosome identified. Since there is no guarantee of optimality, successive run of genetic algorithm will provide different chromosome with varying fitness measures. This is one of drawback in simple genetic algorithm.

In this section, we propose to hybridize the genetic algorithm with heuristic methods to improve the identification of good chromosomes for the initial population. To achieve this, the first step is to randomly choose chromosomes to build up the initial population. Then we measure the performance (schedule) of each chromosome in terms of its makespan value. This makespan value helps determine the regions where good chromosomes can be found in the chromosomes' pool. Thus, this technique is more efficient than the conventional combinatorial method, since it narrows the search space for good chromosomes when building the initial population. Starting initially with good chromosomes yields new good offsprings. Consider a chromosome length of 4, with two 0's and two 1's. The number of different combinatorial possibilities of chromosomes is given in the pool as in Figure 4.

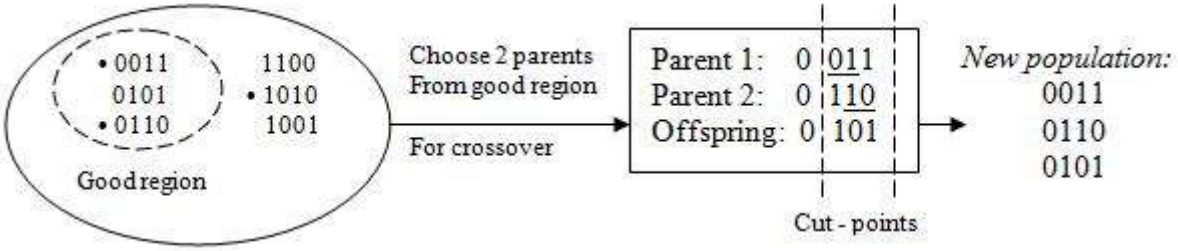

Fig. 4: Pool of chromosomes

The number of chromosomes present in the pool is given by 4! / (2! ×2!) = 6. Three random chromosomes (marked by •) are selected in the initial population. A chromosome will consist of random arrangement of operations. After the initial population has been populated, the fitness (makespan) of the each chromosome needs to be evaluated as shown below. Let us consider the same example of (3×3) Static JSSP from Table 1. The machine route and execution time are known beforehand.

$$\text{MachineSeqMatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{pmatrix}, \text{ExecutionMatrix} = \begin{pmatrix} 3 & 3 & 3 \\ 2 & 3 & 4 \\ 3 & 2 & 1 \end{pmatrix}$$

Now, consider a chromosome which has been randomly populated.

Gene	0	1	2	3	4	5	6	7	8
Chromosome	**3**	**2**	**2**	**1**	**1**	**2**	**3**	**1**	**3**

Its fitness, that is, the makespan is calculated as follows:

```
For each gene i
        Identify Job number j and operation number O
        Machine allocation = Machine SeqMatrix[j][O]
        Execution time = Execution Matrix[j][O]
```

After obtaining the makespan value of each chromosome from the initial population, a pair of good chromosomes is selected for crossover. For example, in Figure 4 two chromosomes from the good region are chosen for producing the offsprings which yields to a better offspring found inside the good region itself.

As the population evolves in the process of crossover and mutation with new offspring, we propose a new heuristic to measure which chromosomes should survive from the current generation to the next generation. Consider a population of 5 chromosomes A, B, C, D and E with their two offsprings O1 and O2 having arbitrary makespan 33, 25, 27, 28, 23, 30, 23 respectively. Notice that chromosome E has the lowest makespan and therefore is the fittest chromosome in this generation. To determine the chance that a chromosome survives to the next generation, we need to transform the fitness of the population to a distribution.

Fitness transformation refers to the application of a standard mathematical distribution transformation function where the fitness of each chromosome *i* in the population of size P is calculated as a survival probability. Let D_i represents the fitness distribution value of each chromosome in P, and is given by the reciprocal of G_i, where the latter is the difference between the makespan L_i of chromosome *i* and the minimum makespan L_{min} of the population. Let D_g refers to the chromosome with the largest distribution value and D_h to the chromosome with the second largest distribution value. Let *l* refers to the chromosome having the least makespan in P. The survival probability V_i in P is given by

$$\Pr ob(V_i) = \begin{cases} 2 * D_q - D_h / S, & i = l \\ D_i / S, & i \neq l \end{cases}$$

where S is given by $S = \sum_{i=1}^{P} D_i + m(2 * D_g - D_h)$

where *m* stands for the frequency of chromosomes having least makespan.

The probabilities of the chromosomes are given in the following table.

Population	**Makespan**	**D_i**	**Prob(V_i)**	**Selected to survive for next population**
A	33	1/(33-23) =0.1	0.037	**Rejected**
B	25	1/(25-23) =0.5	0.186	**Yes**
C	27	1/(27-23) =0.25	0.093	**Yes**
D	28	1/(28-23) =0.2	0.074	**Yes**
E	23 (min)	1/(23-23) =0	0.278	**Rejected**
O1	30	1/(30-23) =0.143	0.053	**Rejected**
O2	23 (min)	1/(23-23) =0	0.278	**Yes and Cloned**
		∑ (D_i) = 1.193	**∑ = 1**	

The higher the survival probability, the more likelihood a chromosome will pass on to next the generation. From the above table, chromosomes B, C, D and O2 have survived as they have the highest survival probabilities. Notice that our technique clones the fittest chromosome as it stands out among the rest. Having clones of the fittest chromosome increases the probability of producing good offsprings as a result of crossover and mutation.

Notice that chromosome E has been rejected and chromosome O2 has been accepted, since chromosome E is an old member of the population whereas chromosome O2 is a newly born member. The rationale is to give a newly born member a higher chance to evolve than an old member which has already evolved long enough through generations. The newly born chromosome O2, has high probability of producing better makespan than the old chromosome E. Moreover, cloning is limited to only a few, so as to give other members of the selected population the chance to evolve.

The whole process of the enhanced genetic algorithm is summarised in the Figure 5.

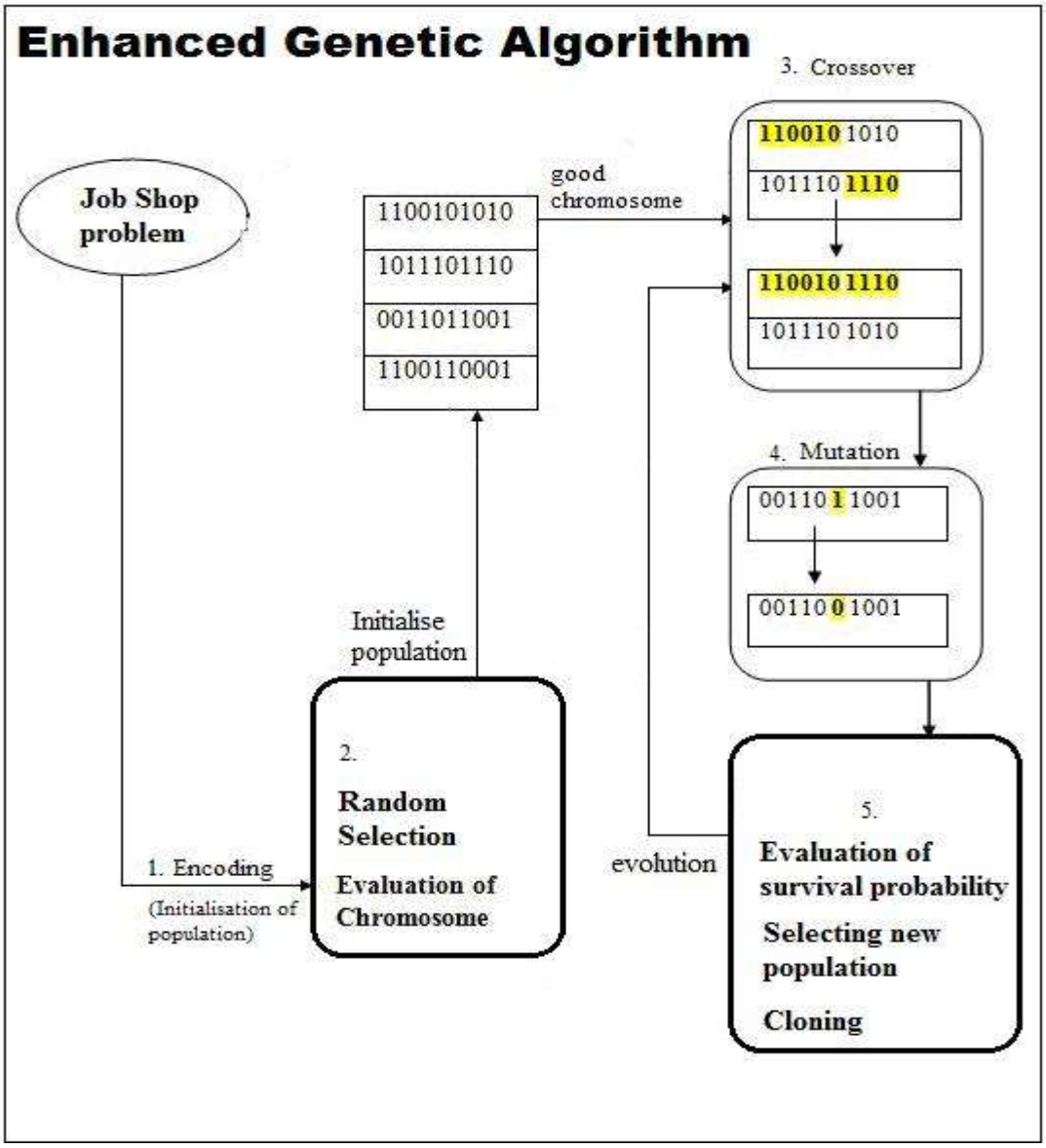

Fig. 5: Structure of Enhanced Genetic Algorithm

IV. EXPERIMENTAL EVALUATION

This section reports on the experiments made to compare the performance of the enhanced genetic algorithm and the existing genetic algorithm. The static Job Shop scheduling problem chosen arbitrarily for our experiments is a 5×3 one, consisting of 5 Jobs, 3 Machines and a maximum of 4 operations:

Table 2: 5x3 Static JSSP

Jobs	Machine Sequence(Execution Time)			
Job0	M0(3)	M2(3)	M1(2)	M0(2)
Job1	M1(3)	M0(2)	M2(1)	
Job2	M2(4)	M0(2)	M1(3)	M0(1)
Job3	M2(2)	M1(2)	M0(3)	M1(3)
Job4	M1(2)	M2(2)		

Genetic parameters play an important role in producing optimal solution for different static job-shop scheduling problems. These parameters are:

1. Initial population size which represents the number of chromosomes
2. Crossover rate, refers to the probability of crossover occurring between two genes cut-points.

3. Mutation rate, refers to the probability of mutation to take place

The first experiment consists of demonstrating the effect of population size on makespan with the crossover rate of 0.6 and mutation rate of 0.4.

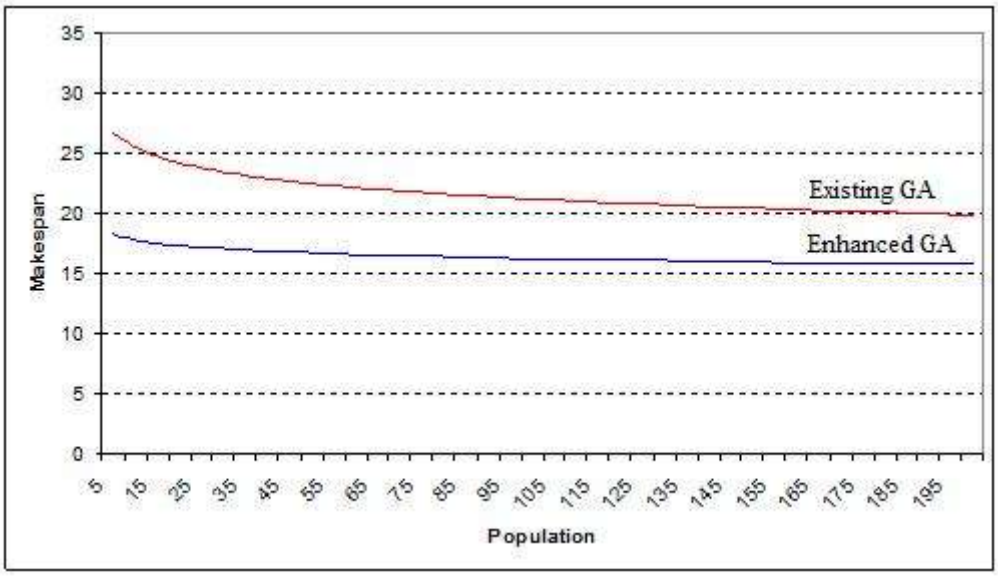

Fig. 6: Effect on Population size on Makespan

In figure 6, we observe that for both the Existing GA and the Enhanced GA, the makespan improves with increasing population size until a point of plateau. A larger population means a larger variety of chromosomes from which there exists a higher probability of obtaining good offsprings after mating.

Moreover, we notice that Enhanced GA reaches the point of plateau with a smaller population size than the Existing GA. The reason is because in Enhanced GA, good chromosomes have been identified from the very beginning to form the initial population. Moreover, during the evolution outstanding chromosomes were cloned so as to increase the probability of passing good genes to the offsprings. These explain why the Enhanced GA results in better makespan than the Existing GA in a shorter evolution.

The second experiment shows the effect of crossover rate on makespan with an initial population size of 10 and a mutation rate fixed at 0.4.

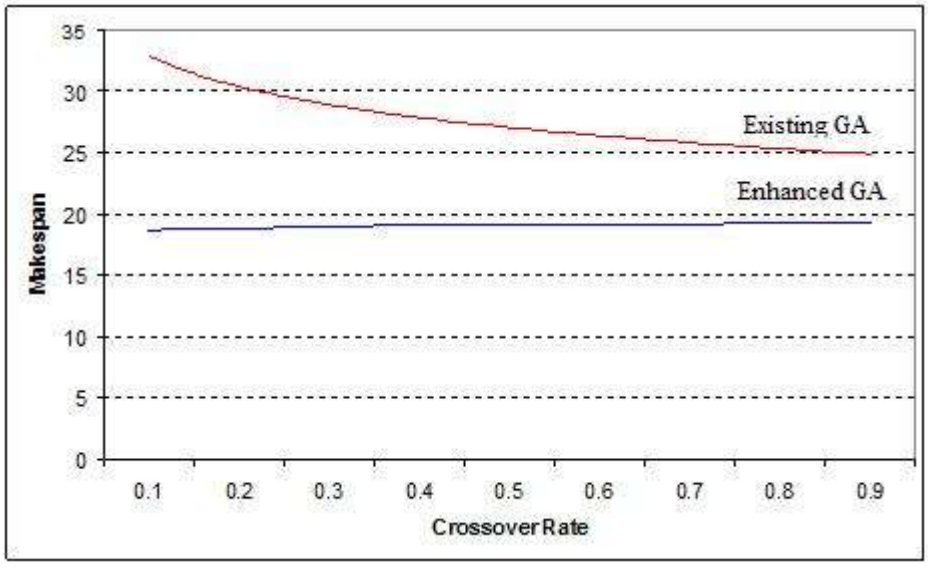

Fig. 7: Effect of Crossover Rate on Makespan

In Figure 7, with low crossover rates (0.1 to 0.5), the Existing GA does not perform well in finding optimal makespan, because a low crossover rate corresponds to a low number of offsprings and subsequently to a low level of evolution. The more the crossover rate, the more the offsprings, and the more the gene evolution.

On the other hand, Enhanced GA performs well with even low crossover rate as the initial population itself already consists of good chromosomes (parents) resulting in offsprings inheriting good genes from their parents. However, when the crossover rate is high, we notice a small degradation of makespan. With a high crossover rate means a high number of offsprings to be born from a high number of crossover instances between parents. A high level of crossover among parents of a generation may result in some of the offsprings losing resemblance to their parents.

In the third experiment, we observe the effect of mutation rate on makespan with fixed population size of 10 and crossover rate of 0.6.

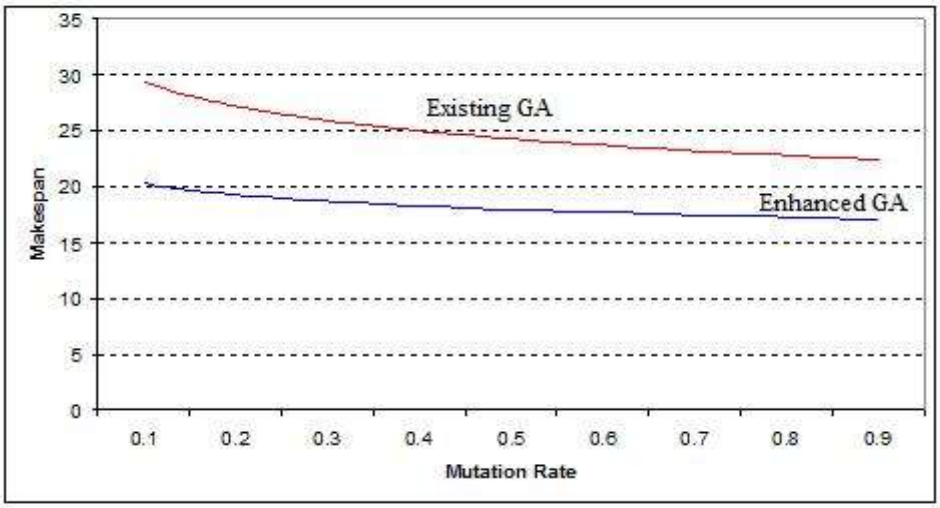

Fig. 8: Mutation Rate and other GA parameters fixed

Mutation rate means that the probability of part of genes of offspring's chromosome will be mutated. If mutation rate is zero, an exact copy of offspring is produced, resulting in a nil evolution. The higher the mutation rate, the higher the variation in the chromosome and the higher the chance of evolution of the chromosome.

In Figure 8, as explained earlier, given the population evolves from good chromosomes from the very start in the Enhanced GA, the latter outperforms the Existing GA.

From the above three experiments, the Existing GA performs well with a population size of 160 or above, crossover and mutation rates of 0.8 or above, whereas the Enhanced GA, with a population size of 70 or above, crossover rate of 0.4 or below and mutation rate of 0.6 or above. In the final experiment we demonstrate the evolution of the makespan over a few generations for both algorithms with their respective genetic parameters.

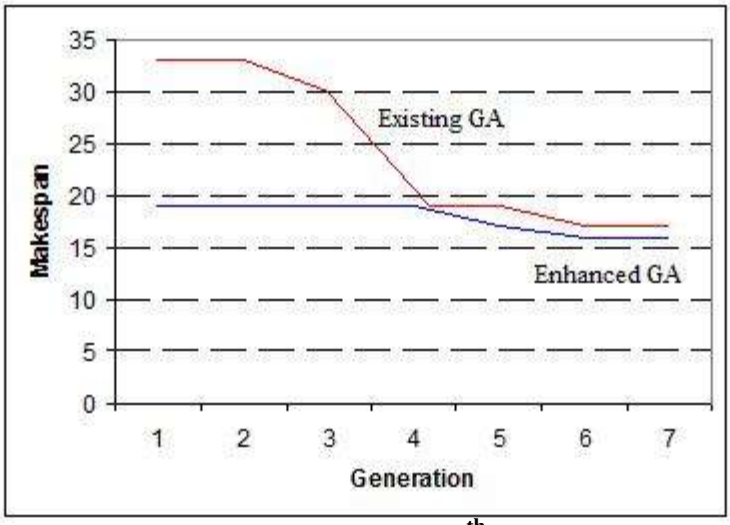

Fig. 9: Optimal Makespan in nth generation

In Figure 9, we observe that the Existing GA is costly, more time consuming, as it takes much time to evolve with a large population size. Moreover, the Existing GA consumes more resources (large population size, high crossover and mutation rates), as a result for the makespan value to evolve through generations. The optimal makespan 17 is obtained after six generations. On the other hand, the Enhanced GA is able to achieve the makespan 19 in the very first generation itself and even better makespan 16 after six generations.

Finally, Figure 10 and Figure 11 show the job schedule and machine allocation for the static Job Shop scheduling problem in Table 2, when the Existing GA and Enhanced GA are used. Notice the schedules obtained from Existing GA and the Enhanced GA have makespan of 17 and 16 respectively.

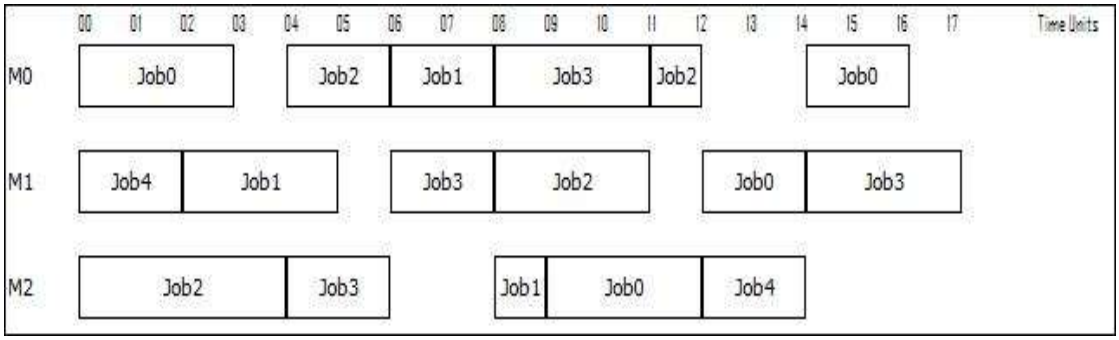

Fig. 10: Machine Allocation using Existing GA

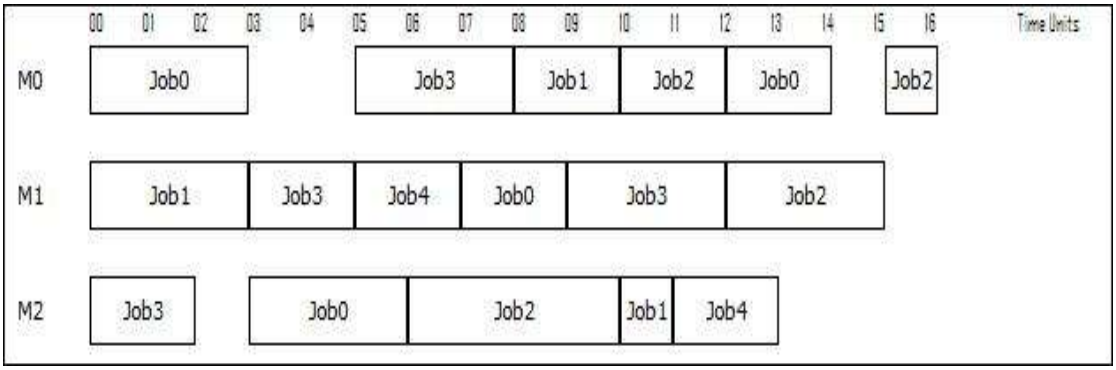

Fig. 11: Machine Allocation using Enhanced GA

V. CONCLUSION

In this paper new genetic operators have been proposed to enhance the performance of the existing genetic algorithm namely, the initialisation of a randomly selected population with its fitness to identify regions of good chromosomes before crossover and mutation are applied, and cloning to improve the probability of

survival of chromosomes to the next generation. Our results show that the Enhanced GA outperforms the Existing GA in terms of obtaining the optimal makespan with a shorter evolution. In the Enhanced GA, an initial population of good chromosomes have already been identified before the evolution starts. Moreover, during the process of evolution, the selection of chromosomes for the next generation was based on the computation of their survival probabilities. Cloning was also applied to increase the chance of passing good genes to the offsprings.

Proper selection of genetic parameters for an application of genetic algorithms is still an open issue. These parameters; population size, crossover rate and mutation rate are usually selected heuristically. There are no rules on the exact strategies to be adopted for different problems. In this paper, experiments of GA parameters was a challenging step for determining these controlling parameters properly, in order to improve the performance of the proposed genetic algorithm. Our results, where the effect of population size, crossover rate and mutation rate were investigated on the makespan, show that the Enhanced GA performs well with a lower population size, a lower crossover rate and a moderate mutation rate as compared to the Existing GA.

REFERENCES

[1]. Arisha, A., Young, P., El Baradie, M., 2001, Job Shop Scheduling Problem: an Overview (International Conference for Flexible Automation and Intelligent Manufacturing (FAIM 01), 1st July 2001, Dublin, Ireland. School of Marketing at ARROW@DIT, pp 682 – 693).

[2]. R.A. Mahdavinejad, A new approach to job shop-scheduling problem (School of Mechanical Engineering, Faculty of Engineering, University of Tehran. 2010).

[3]. Celia Gutiérrez, Overlap Algorithms in Flexible Job-shop Scheduling (Universidad Complutense de Madrid, Spain, International Journal of Artificial Intelligence and Interactive Multimedia, Vol. 2, Nº 6, Pg 1)

[4]. Khaled Mesghouni, Slim Hammadi and Pierre Borne, year. On Modeling Genetic Algorithms for Flexibile Job-Shop Scheduling Problem

[5]. Johnson, S. M, 1954. Optimal Two and Three-Stage Production Schedules with Setup Times Included (Naval Research Logistic Quarterly, Vol. 1, No. 1).

[6]. J. F. Muth and G. L. Thompson, Industrial Scheduling (Prentice-Hall, Englewood Cliffs, N. J, 1963).

[7]. A. M. S. Zalzala and P. J. Fleming,. Genetic Algorithms in Engineering System (The Institution of Electrical Engineers, London, United Kingdom, 1997).

[8]. Mitsuo Gen, . Genetic Algorithm and Engineering Design, Introduction – General Structure of Genetic Algortithm.

[9]. Wen-Yang Lin, Wen-Yuan Lee and Tzung-Pei Hong, Adapting Crossover and Mutation Rates in Genetic Algorithms (Dept of Information Management, I-Shou University, Journal of Information Science and Engineering, 2003).

[10]. Alan Piszcz, Terence Soule, A survey of Mutation Techniques in Genetic Programming (Dept. of Computer Science, University of Idaho, USA).