GA with Priority Rules for Solving Job-Shop Scheduling Problems

GA with Priority Rules for Solving Job-Shop Scheduling Problems

S. M. Kamrul Hasan, *Student Member, IEEE*, Ruhul Sarker, *Member, IEEE*, and David Cornforth

***Abstract*— The Job-Shop Scheduling Problem (JSSP) is considered as one of the difficult combinatorial optimization problems and treated as a member of NP-complete problem class. In this paper, we consider JSSPs with an objective of minimizing makespan while satisfying a number of hard constraints. First, we develop a genetic algorithm (GA) based approach for solving JSSPs. We then introduce a number of priority rules such as partial reordering, gap reduction and restricted swapping to improve the performance of the GA. We run the GA incorporating these rules in a number of different ways. We solve 40 benchmark problems and compared their results with that of a number of well-known algorithms. We obtain optimal solutions for 27 problems, and the overall performance of our algorithms is quite encouraging.**

***Index Terms*— Job-Shop Scheduling, Makespan, Genetic Algorithm, Heuristics.**

I. Introduction

THE job-shop scheduling problem (JSSP) is a common problem in the manufacturing industry. A classical JSSP is a combination of N jobs and M machines. Each job consists of a set of operations that has to be processed on a set of known machines, and has a known processing time. A schedule is a complete set of operations, required by a job, to be performed on different machines, in a given order. In addition, the process may need to satisfy other constraints. The total time between the starting of the first operation and the ending of the last operation is termed as the *makespan*. Makespan minimisation is widely used as an objective in solving JSSPs [1-7]. A feasible schedule contains no conflicts such as (i) no more than one operation of any job can be executed simultaneously and (ii) no machine can process more than one operation at the same time. The schedules are generated on the basis of predefined sequence of machines and the given order of job operations.

The JSSP is widely acknowledged as one of the most difficult NP-complete problems [8-10] which is also well-known for its practical applications in many manufacturing industry. Over the last few decades, a good number of algorithms have been developed for solving JSSPs. However, no single algorithm is capable of solving all kinds of JSSPs optimally (or near optimally) within a reasonable time limit. Thus, there is scope to analyze the difficulties of JSSPs as well as to design algorithms that may be able to solve most of the standard problems.

In early stages, Akers and Friedman [11] and Giffler and Thompson [12] explored only a subset of the alternative solutions to suggest acceptable schedules. Although such an approach was computationally expensive but it could solve the problems much quicker than a human can do at that time. Later, the Branch-and-Bound (B&B) was widely popular for solving JSSPs which uses the concept of omitting a subset of solutions those are out of the bounds [13-15]. Among them, Carlier and Pinson [15] solved a 10×10 JSSP optimally for the first time which was proposed in 1963 by [16]. They considered the $N\times M$ JSSP as M one-machine problem and evaluated the best preemptive solution for each machine. The algorithm relaxes the constraints in all other machines except the one under consideration. The concept of converting M machines problem to one-machine problem is also used by Emmons [17] and Carlier [18]. As the complexity of this algorithm is directly dependent on the number of machines, it is not computationally cheaper for large scale problems.

Although the above algorithms can achieve optimality or near optimality, they are computationally expensive, even out of reach for large problems with the current computation power. For this reason, numerous heuristic and meta-heuristic approaches are proposed in last few decades. These approaches do not guarantee optimality, but provide good quality solutions within a reasonable period of time. Examples for such approaches applied to JSSPs are GA [3, 4, 6, 19-21], Tabu Search (TS) [22-24], Shifting Bottleneck (SB) [1, 25], and Greedy Randomized Adaptive Search Procedure (GRASP) [2], simulated annealing (SA) [26, 27].

In this research, we examine the performance of a traditional genetic algorithm (TGA) for solving JSSPs. Each individual represents a particular complete schedule and is specified with a binary chromosome. After reproduction, the genotype to phenotype mapping employed may result in an infeasible individual, which is then repaired to be feasible. The phenotype representation of the problem is a matrix of $M\times N$ integer numbers where each row represents the sequence of operations in a given machine. A binary genotype facilitates simple crossover and mutation techniques. Moreover, the representation makes the repairing process easier [4-6].

Manuscript received December 13, 2007. This work was supported by the University of New South Wales at the Australian Defence Force Academy in the form of a Postgraduate Research Scholarship.

S. M. Kamrul Hasan is with the University of New South Wales at the Australian Defence Force Academy, Canberra, ACT 2600 Australia (phone: +61 2 62688180; fax: +61 2 62688581; e-mail: kamrul@adfa.edu.au).

Dr. Ruhul Sarker, Senior Lecturer with the School of ITEE, University of New South Wales at the Australian Defence Force Academy, Canberra, ACT 2600 Australia (e-mail: r.sarker@adfa.edu.au).

Dr. David Cornforth, Senior Lecturer with the School of ITEE, University of New South Wales at the Australian Defence Force Academy, Canberra, ACT 2600 Australia (e-mail: d.cornforth@adfa.edu.au).

After analyzing the traditional GA solutions, we realize that solutions can be further improved by applying simple rules or local search. Here, we introduce several priority rules such as partial reordering (PR), gap reduction (GR) and restricted swapping (RS) to improve the performance of the traditional GA. These rules can be considered as local search as well in addition to genetic operators applied. The result of a rule will be accepted if and only if it improves the solution. The details of the priority rules are discussed in a later section. This work implements a GA incorporating different combinations of these priority rules. For ease of explanation, in this paper, we identify the combinations PR with GA, GR with GA, GR with RS and GA as PR-GA, GR-GA and GR-RS-GA respectively. To test the performance of our proposed algorithms, we solve 40 benchmark problems as reported in Lawrence [28]. The priority rules that we propose improve the performance of traditional a GA for solving JSSPs. Among the priority rules, GR-RS-GA is the best performing algorithm. It obtained the optimal solution for 27 out of 40 test problems. The overall performance of GR-RS-GA is better than many key JSSP algorithms appearing in the literature. The current version of our algorithms is much refined from our earlier version. The earlier version of the algorithms with experimental results of fewer test problems can be found in Hasan *et al.* [29, 30].

The paper is organized as follows. After the introduction, a brief outline of a standard job-shop scheduling problem is given. A short review on the traditional genetic algorithm to solve JSSPs is provided in Section III. Section IV introduces new priority rules for improving the performance of traditional GA. Section V presents the proposed algorithms and implementation aspects. Section VI shows the experimental results and necessary statistical analysis to measure the performance of the algorithms. Finally, the conclusions and future research direction are presented.

II. Problem Definition

The standard job-shop scheduling problem makes the following assumptions:

- Each job consists of a finite number of operations.
- The processing time for each operation in a particular machine is defined.
- There is a pre-defined sequence of operations that has to be maintained to complete each job.
- Delivery times of the products are undefined.
- There is no setup cost or tardy cost.
- A machine can process only one job at a time.
- Each job visits each machine only once.
- No machine can deal with more than one type of task.
- The system cannot be interrupted until each operation of each job is finished.
- No machine can halt a job and start another job before finishing the previous one.
- Each and every machine has full efficiency.

The objective of the problem is the minimization of the maximum time taken to complete each and every operation while satisfying the machining constraints and required operational sequence of each job.

III. Job-Shop Scheduling with Genetic Algorithm

In this paper, we consider the minimization of makespan as the objective of JSSPs. According to the problem definition, the sequence of machine used (those are also sequence of operations) by each job is given. In this case, if we know either the starting or finishing time of each operation, we can calculate the makespan for each job and generate the whole schedule. In JSSPs, the main problem is to find the sequence of jobs to be operated on each machine that minimizes the overall makespan.

Chromosome Representation

In solving JSSPs using GAs, the chromosome of each individual usually comprises the schedule. Chromosomes can be represented by binary, integer or real numbers. Some popular representations for solving JSSP are: operation based, job based, preference-list based, priority-rule based, and job pair-relation based representations [31]. We select the job pair-relation based genotype representation due to the flexibility of applying genetic operators. The same representation is also used by some other authors [4-6, 32, 33]. In this representation, a chromosome is symbolized by a binary string, where each bit stands for the order of a job pair (u,v) for a particular machine m. For a chromosome C_p;

$$C_{p,m,u,v} = \begin{cases} 1 & \text{if the job } j_u \text{ leads the job } j_v \\ 0 & \text{Otherwise} \end{cases} \quad (1)$$

A value of 1 means that for the individual p, the job u must lead the job v in machine m. The job having the maximum number of 1s is the highest priority job for that machine. The length of each chromosome is;

$$l = M \times (N-1) \times N/2 \quad (2)$$

where N is the number of jobs, M is the number of machines and the length l is the number of pairs formed by considering any two jobs. This binary string acts as the genotype of individuals. It is possible to construct a phenotype which is the job sequence for each machine. The construction is described in Table I. This representation is helpful if the conventional crossover and mutation techniques are used. The heuristic operators or priority rules discussed in later sections are applied to the constructed phenotype.

Reproduction operators used are the simple two-point crossover and mutation. The crossover points are selected randomly. After applying the operators, we perform repairing techniques (i.e. local and global harmonization operations) to make the solution feasible, as these operators may produce an infeasible solution [4-6].

Local Harmonization

This is the technique of constructing the phenotype (sequence of operations for each machine) from the binary genotype. *M* tables are formed from a chromosome of length *l* as of Equation (2). Table I shows the way to construct the phenotype from the genotype by applying local harmonization.

Table I.A represents the binary chromosome (for a 3 jobs and 3 machines problem) where each bit represents the preference of one job with respect to another job in the corresponding machine. The first block on the left shows the relationship between job 1 and job 2. The machines are listed as m_1 m_3 m_2, because that is the order of machines for job 1, as given in Table 1.C. Table I.B.1, I.B.2 and I.B.3 represent the job pair based relationship in machine m_1, m_2 and m_3 respectively which are mapped from the chromosome, mentioned in Equation (1). In Table I.B.1, the '1' in cell j_1-j_2 indicates that job j_1 will appear before job j_2 in machine m_1. Similarly, the '0' in cell j_1-j_3 indicates that job j_1 will not appear before job j_3 in machine m_1. In the same Table I.B, the column of *S* represents the priority of each job which is the row sum of all 1s for the job presented in each row. A higher number represents a higher priority because it is leading all other jobs. So for machine m_1, job j_3 has the highest priority. If more than one job has an equal priority in a given machine, a repairing technique modifies the order of these jobs to introduce different priorities.

Consider a situation where the order of jobs for a given machine is j_1-j_2, j_2-j_3 and j_3-j_1. This will provide *S*=1 for all jobs in that machine. By swapping the content of cells j_1-j_3 and j_3-j_1, it provides *S*=2, 1 and 0 for jobs j_1, j_2 and j_3 respectively.

TABLE I
DERIVATION PHENOTYPE FROM THE BINARY GENOTYPE AND PRE-DEFINED SEQUENCES

1	0	1	0	0	1	1	0	0
m_1	m_3	m_2	m_1	m_3	m_2	m_2	m_1	m_3
j_1–j_2			j_1–j_3			j_2–j_3		

I.A

	j_1	j_2	j_3	*S*
j_1	*	1	0	*1*
j_2	0	*	0	*0*
j_3	1	1	*	*2*

I.B.1 – m_1

	j_1	j_2	j_3	*S*
j_1	*	1	1	*2*
j_2	0	*	1	*1*
j_3	0	0	*	*0*

I.B.2 – m_2

	j_1	j_2	j_3	*S*
j_1	*	0	0	*0*
j_2	1	*	0	*1*
j_3	1	1	*	*2*

I.B.3 – m_3

	j_{o1}	j_{o2}	j_{o3}
j_1	m_1	m_3	m_2
j_2	m_2	m_1	m_3
j_3	m_1	m_3	m_2

I.C

	m_{t1}	m_{t2}	m_{t3}
m_1	j_3	j_1	j_2
m_2	j_1	j_2	j_3
m_3	j_3	j_2	j_1

I.D

Table I.C shows the pre-defined operational sequence of each job. In this table, j_{o1}, j_{o2} and j_{o3} represent the first, second and third operation for a given job. According to the priorities found from I.B, the Table I.D is generated which is the phenotype or schedule. For example, the sequence of m_1 is j_3 j_1 j_2, because in I.B.1, j_3 is the highest priority and j_2 is the lowest priority job. In Table I.D, the top row (m_{t1}, m_{t2} and m_{t3}) represents the first, second and third task on a given machine. For example, as of Table I.D, the first task in machine m_1 is to process the first task of job j_3.

Global Harmonization

For an $N \times M$ static job-shop scheduling problem, there will be $(N!)^M$ possible solutions. Only a small percentage of these solutions are feasible. Global harmonization is a repairing technique for changing infeasible solutions to feasible. Suppose job j_3 requires its first, second and third operation to be processed on machine m_3, m_2 and m_1 respectively, and job j_1 requires machine m_1, m_3 and m_2 respectively. Further assume that an individual solution (chromosome) indicates that j_3 is scheduled on machine m_1 first to process its first operation and then job j_1. Such a schedule is infeasible as it violates the defined sequence of operations for job j_3. In this case, the swap of places between job j_1 with job j_3 on machine m_1 would allow job j_1 to have its first operation on m_1 as required and it may provide an opportunity for job j_3 to visit m_3 and m_2 before visiting m_1 as per its order. Usually, the process identifies the violations sequentially and performs the swap one by one until the entire schedule is feasible. In this case, there is a possibility that some swaps performed earlier in the process are required to swap back to its original position to make the entire schedule feasible.

The technique is useful not only for the binary representations, but also for the job-based or operation based representation. Further details on the use of global harmonization with GAs for solving JSSPs can be found in [4-6]. In our proposed algorithm, we consider multiple repairing to narrow down the deadlock frequency. As soon as the deadlock occurs, the algorithm identifies at most one operation from each job that can be scheduled immediately. Starting from the first operation, the algorithm identifies the corresponding machine of the operation and swaps the tasks in that machine so that at least the selected task disallows deadlock for the next time. For *N* jobs, the risk of getting into deadlock will be removed for at least *N* operations.

After performing global harmonization, we get a population of feasible solutions. We calculate the makespan of all the feasible individuals and rank them based on their fitness values. We then apply genetic operators to generate the next population. We continue this process until satisfying the stopping criteria.

IV. PRIORITY RULES AND JSSPS

As reported in the literature, different priority rules are imposed in conjunction with GAs to improve the JSSP solution. Dorndorf and Pesch proposed twelve different priority rules for achieving better solutions for JSSPs [21]. However they suggested choosing only one of these rules

while evaluating the chromosome. They also applied the popular shifting bottleneck heuristic proposed by [1] for solving JSSP. This heuristic ingeniously divides the scheduling problem into a set of single machine optimization and re-optimization problems. It selects a machine identified as a bottleneck one by one. After the addition of a new machine, all previously established sequences are re-optimized. However these algorithms were implemented while evaluating the individuals in GA and generating the complete schedule.

In this section, we introduce three new priority rules. We propose to use these rules after the fitness evaluation as the process requires analyzing the individual solutions from the preceding generation. The rules are briefly discussed below.

Partial Reordering (PR)

In the first rule, we identify the machine m_k which is the deciding factor for makespan in phenotype p and the last job j_k that is to be processed by the machine m_k. The Machine m_k can be termed as the bottleneck machine in the chromosome under consideration.

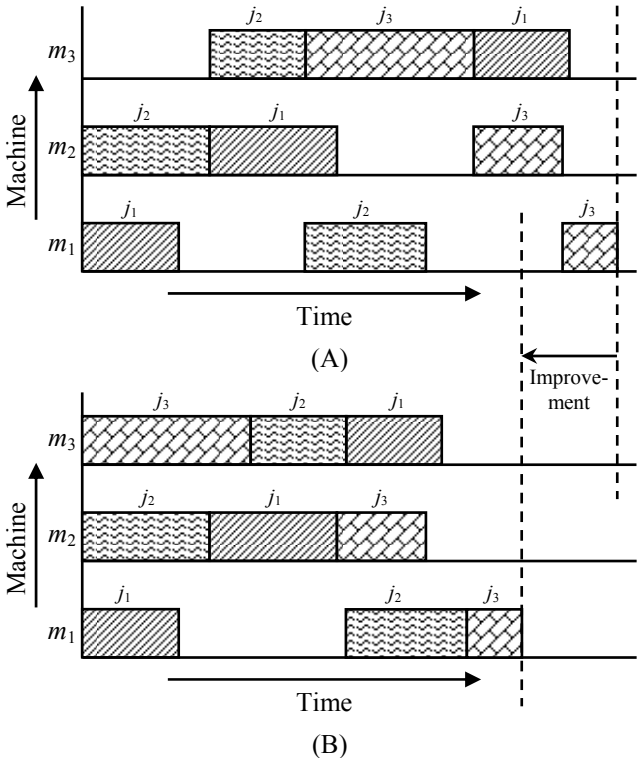

Fig. 1. Gantt chart of the solution (A) before applying the partial reordering (B) after applying partial reordering and reevaluation.

Then we find the machine (say m') required by the first operation of job j_k. The re-ordering rule then suggests that the first operation of job j_k must be the first task on machine m' if it is not the case as scheduled. If we move the job j_k from its current l^{th} position to the 1^{st} position, we may need to push some other jobs currently scheduled on machine m' to the right. In addition, it may provide an opportunity to shift some jobs to the left on other machines. The overall process helps to reduce the makespan for some chromosomes.

A simple example of the re-ordering process is shown in Fig. 1.A, where the makespan is the completion time of job j_3 on machine m_1, so m_1 is the bottleneck machine. Here, job j_3 requires machine m_3 for its first operation. If we move j_3 from its current position to the first operation of machine m_3, it is necessary to shift job j_2 to the right for a feasible schedule on machine m_3. These changes create an opportunity to move job j_1 on m_3, j_3 on m_2 and j_3 on m_1 to the left without violating the operational sequences. As shown in Fig. 1(B), the resulting chromosome is able to improve its makespan. The change of makespan is indicated by the dotted line.

Gap Reduction (GR)

After each generation, the generated phenotype usually leaves some gaps between the jobs. Sometimes, these gaps are necessary to satisfy the precedence constraints. However, in some cases, a gap could be removed or reduced by placing a job from the right side of the gap. For a given machine, this is like swapping between a gap from left and a job from right of a schedule. In addition, a gap may be removed or reduced by simply moving a job to its adjacent gap at the left. The process would help to develop a compact schedule from the left and continue up to the last job for each machine. Of course, it must ensure no conflict or infeasibility before accepting the move.

The rule is to identify the gaps in each machine and candidate jobs which can be placed in those gaps without violating the constraints and not increasing the makespan. The same process is carried out for any possible shift of jobs to the left of the schedule. The gap reduction rule, with swapping between gap and job, is explained using a simple example.

A simple instance of a schedule is shown in Fig. 2(A). In the phenotype p, j_1 follows j_2 in machine m_2, however, job j_1 can be placed before j_2, as shown in Fig. 2(B), due to the presence of an unused gap before j_2. A swap between this gap and job j_1 would allow the processing of j_1 on m_2 earlier than the time shown in Fig. 2(A). This swapping of j_1 on m_2 creates an opportunity to move this job to the left on machine m_1 (see Fig. 2(C)). Finally, j_3 on m_2 can also be moved to the left which ultimately reduces the makespan as shown in Fig. 2(D).

Restricted Swapping (RS)

For a given machine, the restricted swapping rule allows a swap between the adjacent jobs if and only if the resulting schedule is feasible. The process is carried out only for the job which takes the longest time for completion.

Suppose job j' takes the longest time for completion as the phenotype p. The algorithm starts from the last operation of j' in p and checks with the immediate predecessor operation whether these two are swappable or not. The necessary conditions for swapping are; none of the operations can start before finishing time of the immediate predecessor operation of that corresponding job.

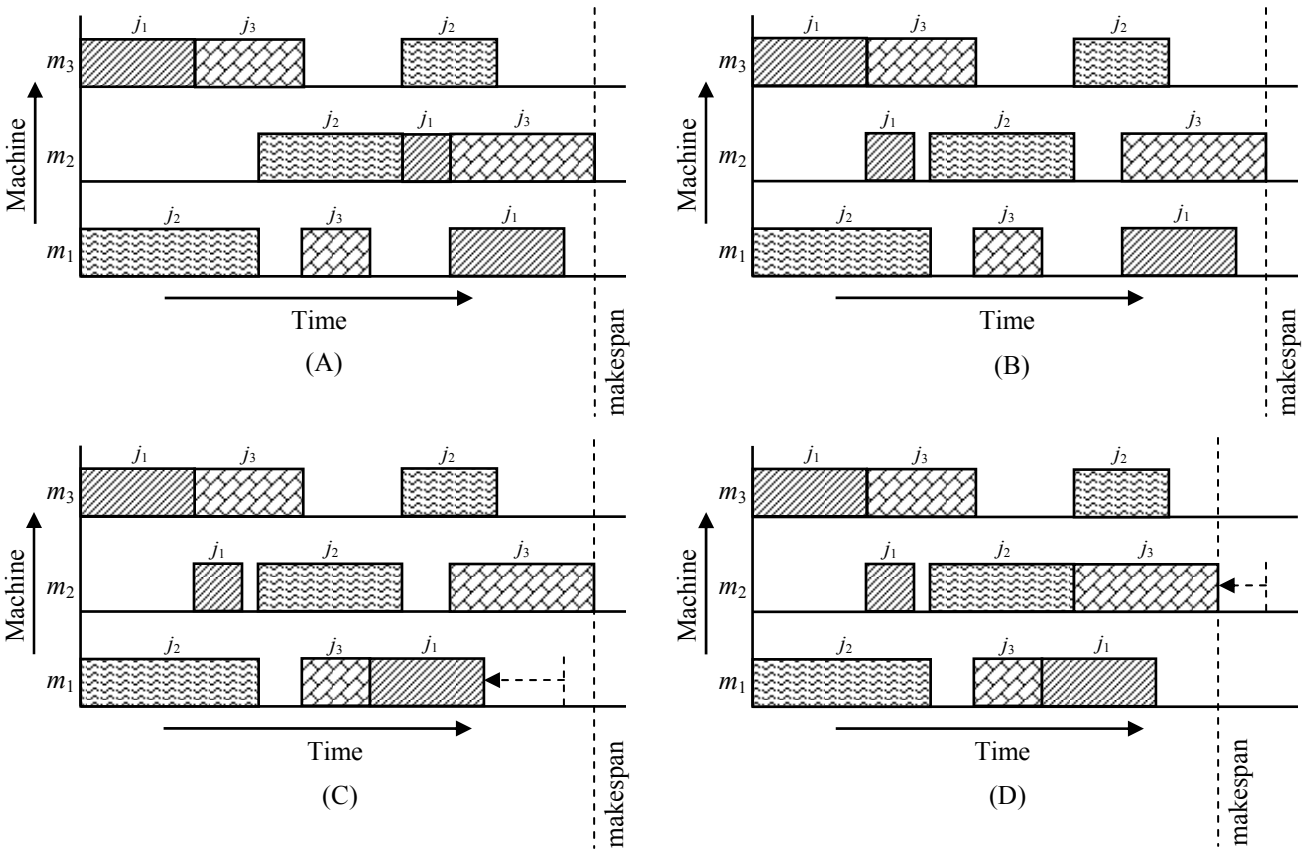

Fig. 2. Two steps of a partial Gantt chart while building the schedule from the phenotype for a 3×3 job-shop scheduling problem. The *x* axis represents the execution time and the *y* axis represents the machines.

And both operations have to be finished before starting the immediate successive operations of the corresponding jobs. Interestingly, the algorithm does not collapse the feasibility of the solution. It may change the makespan if any of the operations are the last operation of the corresponding machine. But it will give an alternate solution which may improve the fitness of the solution in successive generations, when the phenotype will be rescheduled. The process also allows swapping between two randomly selected individuals. This is done for few individuals only. As the complexity of the algorithm is simply an order of N, it does not affect the overall computational complexity that much.

V. Implementation

As we initially implement the TGA, we generate a set of random individuals. Each individual is mainly represented by a binary chromosome. We use the job-pair relation based representation and report the effectiveness of the representation. We use simple one point crossover and bit flip mutation as reproduction operators. We carry out a set of experiments with different crossover and mutation rates to analyze the robustness of the algorithm.

We set the population size to 2500 and the number of generations to 1000. In JSSP, the feasible space is small compared to the solution space, so a large population gives a higher probability of generating feasible solutions.

As we use the GR technique as a part of evaluation, this is applied to every individual. On the other hand, we apply PR and RS to only 5% of individuals randomly selected from the population in every generation.

After successful implementation of TGA, we introduce the priority rules, as discussed in the last section, to TGA as follows:

- Partial reordering rule with TGA (PR-GA)
- Gap reduction rule with TGA (GR-GA) and
- Gap reduction and restricted swapping rule with TGA (GR-RS-GA)

For ease of explanation, we describe the steps of GR-RS-GA below.

Let R_c and R_m be the two-point crossover and bit-mutation probability respectively. $P(t)$ is the set of current individuals at time t and $P'(t)$ is the evaluated set of individuals at time t.

1. Initialize $P(t)$ as a random population $P(t=0)$ of size $|P(t)|$, where each random individual is a bit string of length l.
2. Repeat
   A. Set $t:=t+1$
   B. Evaluate $P'(t)$ from $P(t\text{-}1)$ by the following steps;
      i. Decode each individual p by using the job-based decoding with the local harmonization and global harmonization methods to repair illegal bit strings.
      ii. Generate the complete schedule with starting and

ending time of each operation by applying the gap reduction (GR) rule and calculate the objective function f of p.

iii. Rank the individuals according to the fitness values from higher to lower fitness value.

iv. Apply elitism; i.e. preserve the solution having the best fitness value in the current generation so that it can survive at least up to the next generation.

C. Apply restricted swapping rule (RS) on the randomly selected 5% of the individuals.

D. Go to Step 3 if the stopping criteria are met.

E. Modify $P'(t)$ using the following steps;

i. Select the current individual p from $P'(t)$ and select a random number R between 0 and 1.

ii. If $R \leq R_c$ then

a. Select randomly one individual p_1 from the top 15% of the population and two individuals from the rest. Play a tournament between the last two and choose the winner individual w. Apply two-point crossover between p_1 and w; generate p_1' and w'.

b. Else if $R > R_c$ and $R \leq (R_c + R_m)$ then randomly select one individual from $P(t)$ and apply bit-flip mutation.

c. Else continue.

[End of Step ii If]

iii. Reassign the $P(t)$ by $P'(t)$ to initialize the new generation preserving the best solution as elite.

[End of Step 2 Loop]

3. Save the best solutions among all of the feasible solutions.

[End of Algorithm]

Sometimes the action of genetic operators may take the good individuals away from the optimal. In this case, the elitism ensures the survival of the best individuals [34, 35]. We apply elitism in each generation to preserve the best solution found so far and also to inherit the elite individuals more than the rest.

During the crossover operation, we use the tournament selection that chooses one individual from the elite class of the individual (i.e. the top 15%) and two individuals from the rest. This selection then plays a tournament between the last two and performs crossover between the winner and elite one. We use two-point crossover and a bit-flip mutation technique. We rank the individuals on the basis of the fitness value. A high selection pressure on the better individuals may contribute to premature convergence. If the elite class or most of them have the same solution, then their offspring will be quite similar after some generations. In such case, a higher mutation rate would help to diversify the population.

To test the performance of the our proposed algorithms, we solve 40 benchmark problems designed by Lawrence [28] and compare with several existing algorithms. The problems range from 10×5 to 30×10 and 15×15 where $N \times M$ represents N jobs and M machines.

VI. Experimental Results

The results for the benchmark problems are obtained by executing the algorithms on a personal computer. Results are tabulated in Tables II, III and IV. Table II compares the performance of our four algorithms (TGA, PR-GA, GR-GA, and GR-RS-GA) in terms of the % average relative deviation (ARD) from the best result published in the literature, the standard of % relative deviation (SDRD), and the fitness evaluation which is the average number of generations to achieve the best solution multiplied by the population size. It also includes the number of problems where the algorithms found optimal solution.

TABLE II
Comparing our Four Algorithms

No. of Problems	Algorithm	Optimal Found	ARD (%)	SDRD (%)	Fitness Eval. (10^3)
40 (la01–la40)	TGA	15	3.591	4.165	664.90
	PR-GA	16	3.503	4.192	660.86
	GR-GA	23	1.360	2.250	356.41
	GR-RS-GA	27	0.968	1.656	388.58

From Table II, it is clear that the performance of GR-GA is better than both PR-GA and TGA. The addition of RS to GR-GA, which is known as GR-RS-GA, has clearly enhanced the performance of the algorithm in terms of both ARD and SDRD. Though GR-RS-GA takes few more fitness evaluation but gives better fitness value. It is due to the fact that RS changes some properties in the solutions which help to generate a new solution. As more new solutions are generated, the algorithm needs more fitness evaluation.

Out of 40 test problems, GR-RS-GA obtained the best known solution for 27 problems, whereas GR-GA obtained the best known solution for 23 problems. GR-RS-GA produced better solutions (although non-optimal) for 10 problems, and both algorithms obtained a non-optimal equal solution for 3 problems. Details of the results are found in Table III. The rate of improvement made by PR and RS in each generation is usually lower than that of GR. PR considers only the bottleneck job. However, GR is applied to all individuals. PR makes good improvement at the initial generations and has insignificant effect at later stages. The process of GR eventually makes most changes performed by PR over some (or many) generations. As a result, the inclusion of PR with GR does not help to improve the performance of the algorithm. Both PR and RS are applied to only 5% of the individuals. Here, the role of RS is to increase the diversity like mutation. The increase in the rate of PR and RS does not provide any benefit either in term of quality of solution or computational time. For this reason, we have not presented other possible variants such as PR-RS-GA and GR-RS-PR-GA.

TABLE III
COMPARISON OF THE % RELATIVE DEVIATIONS WITH THE RESULTS FOUND IN LITERATURE

Problem	TGA	PR-GA	GR-GA	GR-RS-GA	Aarts *et al.*		Ombuki and Ventresca	Dorndorf & Pesch			Croce *et al.*	Binato*et al.*	Adams *et al.*	
					GLS1	GLS2		PGA	SBGA1	SBGA2			SB I	SB II
la01	0.15	0.15	0.00	0.00	0.00	0.00	–	0.00	0.00	–	0.00	0.00	0.00	–
la02	0.00	0.00	0.00	0.00	1.98	0.61	–	3.97	1.68	–	1.68	0.00	9.92	2.14
la03	3.35	3.35	0.00	0.00	2.68	2.01	–	3.85	1.17	–	11.56	1.17	4.36	1.34
la04	2.71	2.71	0.00	0.00	1.53	0.68	–	5.08	0.00	–	–	0.00	1.19	0.51
la05	0.00	0.00	0.00	0.00	0.00	0.00	–	0.00	0.00	–	–	0.00	0.00	–
la06	0.00	0.00	0.00	0.00	0.00	0.00	–	0.00	0.00	–	0.00	0.00	0.00	–
la07	0.00	0.00	0.00	0.00	0.00	0.00	–	0.00	0.00	–	–	0.00	0.00	–
la08	0.00	0.00	0.00	0.00	0.00	0.00	–	0.00	0.00	–	–	0.00	0.58	0.00
la09	0.00	0.00	0.00	0.00	0.00	0.00	–	0.00	0.00	–	–	0.00	0.00	–
la10	0.00	0.00	0.00	0.00	0.00	0.00	–	0.00	0.00	–	–	0.00	0.10	–
la11	0.00	0.00	0.00	0.00	0.00	0.00	–	0.00	0.00	–	0.00	0.00	0.00	–
la12	0.00	0.00	0.00	0.00	0.00	0.00	–	0.00	0.00	–	–	0.00	0.00	–
la13	0.00	0.00	0.00	0.00	0.00	0.00	–	0.00	0.00	–	–	0.00	0.00	–
la14	0.00	0.00	0.00	0.00	0.00	0.00	–	0.00	0.00	–	–	0.00	0.00	–
la15	0.00	0.00	0.00	0.00	0.00	0.00	–	2.49	0.00	–	–	0.00	0.00	–
la16	5.19	5.19	0.11	0.00	3.39	3.39	1.48	6.67	1.69	1.69	3.60	0.11	8.04	3.49
la17	1.28	0.13	0.00	0.00	0.89	0.89	1.02	3.19	0.38	0.00	–	0.00	1.53	0.38
la18	1.53	1.53	1.53	0.00	0.94	1.18	1.06	8.02	0.00	0.00	–	0.00	5.07	1.30
la19	5.70	6.41	0.95	0.00	2.49	2.02	2.14	4.51	2.49	0.71	–	0.00	3.92	2.14
la20	7.21	7.21	0.55	0.55	1.22	1.55	0.55	2.88	1.00	0.89	–	0.55	2.44	1.33
la21	4.97	4.21	4.11	3.15	3.63	3.73	6.50	8.89	2.68	2.68	4.88	4.30	12.05	3.63
la22	6.36	6.26	3.88	3.56	2.91	1.83	6.69	7.66	0.86	0.97	–	3.56	12.19	1.83
la23	1.07	1.07	0.00	0.00	0.00	0.00	0.29	3.88	0.00	0.00	–	0.00	2.81	0.00
la24	5.24	5.45	4.71	2.57	3.74	4.92	10.37	8.45	2.67	2.35	–	4.60	6.95	4.39
la25	10.24	10.24	1.43	1.43	3.99	3.38	7.16	3.79	3.17	3.07	–	5.22	7.27	4.09
la26	6.32	6.98	0.16	0.00	1.81	1.48	7.31	4.93	0.08	0.00	1.07	4.35	7.06	0.49
la27	7.53	7.53	5.10	4.13	5.91	5.26	9.31	11.58	3.00	2.75	–	6.88	7.29	4.53
la28	6.66	6.25	2.22	1.64	5.35	4.03	7.89	9.13	1.97	2.06	–	6.33	3.29	2.80
la29	9.33	9.51	6.91	5.53	11.50	8.90	13.31	15.47	4.06	4.58	–	11.75	11.84	7.09
la30	1.62	0.59	0.00	0.00	3.47	2.29	7.08	4.13	0.00	0.00	–	0.96	3.54	0.00
la31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	–
la32	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	–	0.00	0.00	–
la33	0.00	0.00	0.00	0.00	0.00	0.00	1.51	0.00	0.00	0.00	–	0.00	0.00	–
la34	1.57	1.10	0.00	0.00	0.93	0.52	3.66	0.00	0.00	0.00	–	1.86	0.00	–
la35	0.53	0.00	0.00	0.00	0.32	0.11	3.71	0.00	0.00	0.00	–	0.00	0.00	–
la36	9.46	9.54	3.86	3.08	4.42	3.39	7.10	8.28	3.86	3.86	2.92	5.21	6.55	2.92
la37	11.74	10.52	4.94	3.22	3.72	3.79	8.59	7.23	6.23	3.51	–	4.29	6.30	1.86
la38	14.30	14.30	9.03	5.85	7.44	7.27	13.88	8.36	4.60	3.76	–	5.94	7.02	4.93
la39	8.52	8.84	2.43	1.54	3.73	3.73	12.81	9.57	3.97	3.57	–	4.62	7.14	3.24
la40	11.05	11.05	2.45	2.45	4.17	3.11	8.27	8.10	4.26	2.45	–	3.03	8.51	3.85

Table III compares the best results published in literature with that found by our four algorithms (TGA, PR-GA, GR-GA, and GR-RS-GA) and some other key results as published in literature. Table III starts with the column of problem instances. The following columns show the ARD of other algorithms found in literature. Each pair of columns next to the best known fitness represents the best fitness and relative deviation in % from the best. We consider our four algorithms (TGA, PR-GA, GR-GA, and GR-RS-GA), local search GA [20, 36], shifting-bottleneck GA [21], simple GA [3], GRASP [2] and shifting bottleneck heuristic [1].

In most of the test problems, our proposed GR-RS-GA performs better than other algorithms. In terms of average relative deviation (ARD) and standard deviation of the relative deviations (SDRD), our algorithm outperforms all other algorithms reported in this paper.

As different authors used different number of problems, we make a comparison based on only those problems the authors considered. For example, as Ombuki and Ventresca [20] used 25 problems out of 40, we calculate ARD and SDRD based on only those particular problems to perform a close measurement. The result is tabulated in Table IV.

TABLE IV
COMPARISON OF THE % DEVIATIONS FOR THE DIFFERENT NUMBER OF PROBLEMS AUTHORS CONSIDERED

No. of Problems	Test Problems	Authors	Algorithm	ARD (%)	SDRD (%)
40	la01 – la40	Our Proposed	GR-RS-GA	0.9680	1.6559
		Aarts *et al.*	GLS1	2.0540	2.5279
			GLS2	1.7518	2.1974
		Dorndorf & Pesch	PGA	4.0028	4.0947
			SBGA (40)	1.2455	1.7216
		Binato *et al.*	-	1.8683	2.7817
		Adams *et al.*	SB I	3.6740	3.9804
25	la16 – la40	Our Proposed	GR-RS-GA	1.5488	1.8759
		Ombuki and Ventresca	-	5.6676	4.3804
		Dorndorf & Pesch	SBGA (60)	1.5560	1.5756
24	Selected (see Table III)	Our Proposed	GR-RS-GA	1.6810	1.8562
		Adams *et al.*	SB II	2.4283	1.8472
9	Selected (see Table III)	Our Proposed	GR-RS-GA	0.6231	1.3137
		Croce *et al.*	-	2.5710	3.6025

Result in the Table IV shows that GA-GR-RS performing better than any other algorithms considered even if we consider only a subset of the problems other authors considered. Though in some cases, SDRD is not as good as other authors, but it is competitive and better in terms of ARD.

Finally, we can summarize from the experimental result that PR-GA performs better than traditional GA. GR-GA makes a big change in the quality of solutions and fitness evaluation. The RS rule helps to improve the solution quality of GR-GA which is outperforming.

VII. CONCLUSION

The JSSP is a very well known member of the combinatorial optimization problem class. A considerable amount of work has already been done to improve algorithms to give a steady and optimal output. Some algorithms are eminent for special case problems. But still no algorithm guarantees optimality. We observe from numerous experiments that GA is able to produce good quality solutions for smaller problems within a reasonable

period of time. Integration of some other techniques significantly improves the solution quality. Still our algorithms do not ensure optimality. But it gives a competitive result within a reasonable period of time. In future work we plan to work on the diversity of the solutions in solution space and apply the ideas to the flexible job-shop scheduling problems. Moreover, we would like to consider large scale problems and real world problems to justify the performance of our algorithms.

REFERENCES

[1] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management Science,* vol. 34, pp. 391-401, 1988.

[2] S. Binato, W. Hery, D. Loewenstern, and M. Resende, *A GRASP for Job Shop Scheduling*: Kluwer Academic Publishers, 2000.

[3] F. D. Croce, R. Tadei, and G. Volta, "A genetic algorithm for the job shop problem," *Computers & Operations Research,* vol. 22, pp. 15-24, 1995.

[4] R. Nakano and T. Yamada, "Conventional genetic algorithm for job shop problems," in *Fourth Int. Conf. on Genetic Algorithms*, Morgan Kaufmann, San Mateo, California, 1991, pp. 474-479.

[5] T. Yamada, "Studies on Metaheuristics for Jobshop and Flowshop Scheduling Problems," in *Department of Applied Mathematics and Physics*. Doctor of Informatics Kyoto, Japan: Kyoto University, 2003, p. 120.

[6] T. Yamada and R. Nakano, "Genetic algorithms for job-shop scheduling problems," in *Modern Heuristic for Decision Support*, UNICOM seminar, London, 1997, pp. 67-81.

[7] W. Wang and P. Brunn, "An Effective Genetic Algorithm for Job Shop Scheduling," *Proceedings of the Institution of Mechanical Engineers -- Part B -- Engineering Manufacture,* vol. 214, pp. 293-300, 2000.

[8] M. R. Garey, D. S. Johnson, and R. Sethi, "The Complexity of Flowshop and Jobshop Scheduling," *Mathematics of Operations Research,* vol. 1, pp. 117-129, 1976.

[9] J. K. Lenstra and A. H. G. Rinnooy Kan, "Computational complexity of discrete optimization problems." vol. 4 Rotterdam: Annals of Discrete Mathematics, 1979, pp. 121-140.

[10] M. R. Garey and D. S. Johnson, *Computers and intractability : a guide to the theory of NP-completeness*. San Francisco: W. H. Freeman, 1979.

[11] S. B. J. Akers and J. Friedman, "A Non-Numerical Approach to Production Scheduling Problems," *Journal of the Operations Research Society of America,* vol. 3, pp. 429-442, Novmber 1955 1955.

[12] B. Giffler and G. L. Thompson, "Algorithms for Solving Production-Scheduling Problems," *Operations Research,* vol. 8, pp. 487-503, 1960.

[13] S. Ashour and S. R. Hiremath, "A branch-and-bound approach to the job-shop scheduling problem," *International Journal of Production Research,* vol. 11, pp. 47-58, 1973.

[14] P. Brucker, B. Jurisch, and B. Sievers, "A branch and bound algorithm for the job-shop scheduling problem," *Discrete Applied Mathematics,* vol. 49, pp. 107-127, 1994.

[15] J. Carlier and E. Pinson, "An Algorithm for Solving The Job-Shop Problem," *Management Science,* vol. 35, pp. 164-176, Feb 1989 1989.

[16] J. F. Muth and G. L. Thompson, *Industrial scheduling*. Englewood Cliffs, N.J.,: Prentice-Hall, 1963.

[17] H. Emmons, "One-Machine Sequencing to Minimize Certain Functions of Job Tardiness," *Operations Research,* vol. 17, pp. 701-715, 1969.

[18] J. Carlier, "The one-machine sequencing problem," *European Journal of Operational Research,* vol. 11, pp. 42-47, 1982.

[19] J. E. Biegel and J. J. Davern, "Genetic algorithms and job shop scheduling," *Computers & Industrial Engineering,* vol. 19, pp. 81-91, 1990.

[20] B. M. Ombuki and M. Ventresca, "Local Search Genetic Algorithms for the Job Shop Scheduling Problem," *Applied Intelligence,* vol. 21, pp. 99-109, 2004.

[21] U. Dorndorf and E. Pesch, "Evolution based learning in a job shop scheduling environment," *Computers & Operations Research,* vol. 22, pp. 25-40, 1995.

[22] M. Dell'Amico and M. Trubian, "Applying tabu search to the job-shop scheduling problem," *Annals of Operations Research,* vol. 41, pp. 231-252, 1993.

[23] J. W. Barnes and J. B. Chambers, "Solving the job shop scheduling problem with tabu search." vol. 27: Taylor & Francis, 1995, pp. 257 - 263.

[24] S. G. Ponnambalam, P. Aravindan, and S. V. Rajesh, "A Tabu Search Algorithm for Job Shop Scheduling," *The International Journal of Advanced Manufacturing Technology,* vol. 16, pp. 765-771, 2000.

[25] S. Dauzere-Peres and J. B. Lasserre, "A modified shifting bottleneck procedure for job-shop scheduling," *International Journal of Production Research,* vol. 31, pp. 923-932, 1993.

[26] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra, "Job Shop Scheduling by Simulated Annealing," *Operations Research,* vol. 40, pp. 113-125, 1992.

[27] S. G. Ponnambalam, N. Jawahar, and P. Aravindan, "A simulated annealing algorithm for job shop scheduling," *Production Planning and Control,* vol. 10, pp. 767-777, 1999.

[28] S. Lawrence, "Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques," Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania 1984.

[29] S. M. K. Hasan, R. Sarker, and D. Cornforth, "Modified Genetic Algorithm for Job-Shop Scheduling: A Gap-Utilization Technique," in *Evolutionary Computation, IEEE Congress on*, Singapore, 2007, pp. 3804-3811.

[30] S. M. K. Hasan, R. Sarker, and D. Cornforth, "Hybrid Genetic Algorithm for Solving Job-Shop Scheduling Problem," in *Computer and Information Science, 6th IEEE/ACIS International Conference on*, Melbourne, Australia, 2007, pp. 519-524.

[31] S. G. Ponnambalam, P. Aravindan, and P. S. Rao, "Comparative Evaluation of Genetic Algorithms for Job-shop Scheduling," *Production Planning & Control,* vol. 12, pp. 560-674, 2001.

[32] J. Paredis, "Handbook of Evolutionary Computation," in *Parallel Problem Solving from Nature 2* Brussels, Belgium: Institute of Physics Publishing and Oxford University Press, 1992.

[33] J. Paredis, T. Back, D. Fogel, and Z. Michalewicz, "Exploiting constraints as background knowledge for evolutionary algorithms," in *Handbook of Evolutionary Computation*: Institute, 1997, pp. G1.2:1-6.

[34] H. Ishibuchi and T. Murata, "A multi-objective genetic local search algorithm and its application to flowshop scheduling," *Systems, Man and Cybernetics, Part C, IEEE Transactions on,* vol. 28, pp. 392-403, 1998.

[35] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Reading, Mass: Addison-Wesley Pub. Co, 1989.

[36] E. H. L. Aarts, P. J. M. Van Laarhoven, J. K. Lenstra, and N. L. J. Ulder, "A Computational Study of Local Search Algorithms for Job Shop Scheduling," *ORSA Journal on Computing,* vol. 6, pp. 118-125, Spring 1994.