Modified Genetic Algorithm for Job-Shop Scheduling: A Gap-Utilization Technique

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

***Abstract*—The Job-Shop Scheduling Problem (JSSP) is one of the most critical combinatorial optimization problems. The objective of JSSP in this research is to minimize the makespan. In this paper, we propose two Genetic Algorithm (GA) based approaches for solving JSSP. Firstly, we design a simple heuristic to reduce the completion time of jobs on the bottleneck machines that we call the reducing bottleneck technique (RBT). This heuristic was implemented in conjunction with a GA. Secondly; we propose to fill any possible gaps left in the simple GA solutions by the tasks that are scheduled later. We call this process the gap-utilization technique (GUT). With GUT, we also apply a swapping technique that deals only with the bottleneck job. We study 35 test problems with known solutions, using the existing GA and our proposed two algorithms. We obtain optimal solutions for 23 problems, and the solutions are very close for the rest.**

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

I. Introduction

THE job-shop scheduling is a well-known combinatorial problem. A classical JSSP is a combination of N jobs comprising M operations; where M is the number of machines. This problem considers that–

- 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 can not 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.
- There is a pre-defined processing time for each operation O_{ji} in a particular machine m which is T_{jim}, where j and i are the index of job and operation respectively.
- There should be a pre-defined order of machines for each job that must have to be maintained.

Manuscript received March 26, 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).

The problem can be defined mathematically as one type of minimization problem. Let $\boldsymbol{N}=\{1,2,\ldots,N\}$ be the set of jobs, and $\boldsymbol{M}=\{1,2,\ldots,M\}$ the set of machines. Consider $\boldsymbol{T}=(t_1,t_2,\ldots,t_M,t_{M+1},\ldots,t_{M\times N})$ the set of processing time for all operations and $\boldsymbol{C}=(C_1,C_2,\ldots,C_M,C_{M+1},\ldots,C_{N\times M})$ the vector of the finishing time of all operations. The aim is to minimize the finishing time of the last operation $\boldsymbol{C_{max}}$ that is called the makespan. Mathematically we can define the objective function like –

$$\text{Minimize} \quad T_{N\times M}(C_{max}) \tag{1}$$

$$C_{j-1} \le C_j - t_j \text{ where } j=2, 3,\ldots, N\times M \tag{2}$$

Equation (2) reflects the fact that an operation can not be started until the predecessor operations are completed.

Within the previous few decades, much effort has been made to solve the JSSP. Some approaches are somewhat effective for small scale problems. But research is still going on to develop a generalized algorithm that will be independent of problem scale.

In the very early stage, Giffler and Thompson [1] proposed an exact algorithm for production scheduling. Their algorithm generates a particular set of all possible schedules which they termed active schedules, and picked the best as the solution. They later showed that every optimal schedule is an active optimal schedule. Though it was computationally expensive, it could solve the problem quicker than a human can do.

The JSSP can be formulated as an integer programming model, and the corresponding model can be solved using an enumeration method with a branch and bound technique. However, solving large scale JSSP using the branch and bound technique is beyond the capacity of current computational power. Brucker *et al.* [2], D'Ariano *et al.* [3] and Peter *et al.* [4] proposed this technique for solving JSSP. Among them, Carlier and Pinson [5] was successful in achieving an impressive solution of a 10×10 problem that was proposed by Fisher and Thompson [6] in 1963.

In the recent past, GA was used successfully to solve combinatorial optimization problems [7-11]. Over the last decade, a growing number of metaheurstic algorithms have already been proposed to solve complex problems. This includes simulated annealing [12] and tabu search [13, 14]. Cardon *et al.* [15] and Vacher *et al.* [16]worked to integrate a multi-agent system with a genetic algorithm. Later, they applied their approach to job-shop scheduling [17, 18]. Adams *et al.* [19] proposed a shifting bottleneck metaheuristic approach for solving JSSP. Their approach

proposes a conversion of multi-machine problem to an equivalent one-machine problem to obtain optimality.

Beatrice and Mario [20] implemented a deadlock-free strategy for solving JSSP. Later they hybridized the strategy with GA and introduced a simple local search technique (LSGA). They also developed tabu search and integrated tabu search with GA. They showed that this approach performs better that LSGA.

Binato *et al.* [21] proposed another metaheuristic method called the greedy randomized adaptive search procedure (GRASP) for JSSP. GRASP is an iterative method which is the combination of construction and local search. Dorndorf and Pesch [11] discussed several priority rules in their work and developed a priority rule-based genetic algorithm (P-GA) for solving JSSP. Later, they implemented two shifting bottleneck based GAs with two different parameter sets.

Della Croce *et al.* [10] solved JSSP by using GA, and introduced an encoding technique based on a preference rule. They proposed a lookahead simulation method for the encoding. They obtained very competitive results for 11 test problems.

Our work starts with the implementation of an existing simple genetic algorithm (SGA) for solving the classical JSSP and flow-shop problems proposed by Nakano & Yamada [22], and Yamada [23]. They proposed the use of binary genotype even though JSSP is an ordering problem. They introduced a repairing technique, as most of the genotypes produced by the one-point or two-point crossover and bit-flip mutation are illegal. On the other hand, the phenotype is a matrix of $M \times N$ integer numbers where each row represents the sequence of jobs in a given machine. We apply both genotype and phenotype representations to characterize the individuals. A binary genotype is effective for the simple crossover and mutation techniques. But the modifications we propose are based on the phenotype. That is why we use both genotype and phenotype as the representation of the solution.

In this paper, we introduce two new modifications to an existing GA for improving the solution. Firstly, we propose a simple re-ordering rule that identifies the job that takes the longest time to complete the schedule, which we call the bottleneck job. The rule is to assign the first operation of that job to the required machine as a priority task and then reschedule other jobs as necessary. We call this approach the reducing bottleneck technique (RBT).

In the second approach, we propose a gap-utilization technique (GUT). We track any gaps left in the schedule obtained from GA solution and assign possible tasks in those gaps that are scheduled later. In this approach, we also perform swapping between the operations of the bottleneck job and immediate predecessor operation of the corresponding operation. To test the performance of our proposed modifications, we solve 35 benchmark problems designed by Lawrence [24]. Both of our algorithms provide better results as compared to the simple GA discussed above and other algorithms reported in the literature.

The paper is organized in eight different sections. After the introduction, Section II introduces the integration of JSSP with GA. Sections III, IV and VI contain the three newly introduced ideas by us. Section VI discusses about the flow of implementing the algorithms along with the necessary parameters. Section VII gives the experimental results and necessary statistical analysis to measure the performance of the algorithms. We conclude the paper with a brief summary and future direction in Section VIII. Finally, the Appendix contains details of the algorithms that we implement.

II. Job-Shop Scheduling with Genetic Algorithm

In this paper, our objective is to solve JSSP by minimizing the makespan. GA is a widely accepted tool for finding the optimal or near optimal solution within a reasonable amount of time. In most cases, it improves the quality of the solution if the appropriate genetic operators are applied with appropriate parameters.

According to the problem definition, the sequence of machines for each job is given. So the operation sequences of each job and either starting or finishing time of each operation is enough to evaluate the whole schedule. In the case of JSSP, the chromosome of each individual comprises the schedule. It mainly focuses on the sequence of operations for each machine.

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, job pair-relation based representations etc. [25]. We select the job pair-relation based representation for genotype as of [22, 23] due to the flexibility of applying genetic operators.

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;

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

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

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

where N stands for the number of jobs, M for the number of machines and the length L is the number of pairs formed by a job with any other job [22]. 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.

Job pair-based representation is helpful if the conventional crossover and mutation techniques are used. We use this representation for the flexibility of applying simple

reproduction operators. We also use the constructed phenotype as the chromosome to apply some other heuristic operators those are discussed later.

In this research, we construct the phenotype directly from the binary string i.e. the chromosome, and perform the two-point crossover and mutation. After applying the operators, we perform the local and global harmonization operations to make the solution feasible as these reproduction operators may produce an infeasible solution [22].

A. Local Harmonization

This is the technique of constructing the phenotype i.e. the sequence of operations for each machine; from binary genotype [22]. *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
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		

1.A

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

1.B.I – m_1

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

1.B.II – m_2

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

1.B.III – 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

1.C.I

	m_1	m_2	m_3
j_1	1	3	2
j_2	2	1	3
j_3	1	3	2

1.C.II

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

1.D

Table I.A represents the binary chromosome where each bit represents the preference of one job with respect to another job in the corresponding machine. The first bit is 1, which means that job j_1 will appear before job j_2 in machine m_1. Table 1.B.I, 1.B.II and 1.B.III represent the job pair based relationship in machine m_1, m_2 and m_3 respectively, mentioned in Equation (1). In 1.B, *S* represents the priority of each job which is the sum of all 1s for the corresponding job. 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 the same priority, Nakano & Yamada [22], and Yamada [23] proposed a technique to repair the corresponding bits so that each job has different priorities in the corresponding machine. The 1.C.I and 1.C.II represents the pre-defined operational sequence of each job and order of a job in corresponding machine respectively. For example, j_1m_3=2 in 1.C.II because the second operation of j_1 will visit m_3. In Table 1.C.I and 1.D, jj_{ok} and m_im_{ok} represents the k^{th} operation of i^{th} job and i^{th} machine respectively. According to the priorities found from 1.B, the 1.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 1.B.I, j_3 has the highest priority and so on.

B. Global Harmonization

Global harmonization is a repairing technique which forces the solutions to be feasible. The situation when each machine is waiting for such an operation that is not ready to be scheduled; is called deadlock. In such situations, Nakano & Yamada [22], and Yamada [23] proposed to find out the nearest schedulable operation with respect to the current operations and put that operation at the front of the remaining unscheduled operations of the corresponding machine. In our algorithm, we move the schedulable operations of each job to the current position to avoid more traversing. It reduces the amount of repairs required.

Finally, we evaluate the individuals using the makespan as the performance measurement. Mathematically –

$$f = \min\left\{\max\left(\sum_{j=1}^{N}\left(T(j,m)+B_p(m,j)\right)\right)\right\} \qquad (3)$$

where m varies from 1 to M, $T(j,m)$ is the execution time for job j in machine m and $B_p(m,j)$ is the j^{th} idle time in machine m of the individual p.

III. REDUCING BOTTLENECK TECHNIQUE

As reported in the literature, different priority rules are used to improve the JSSP solution.

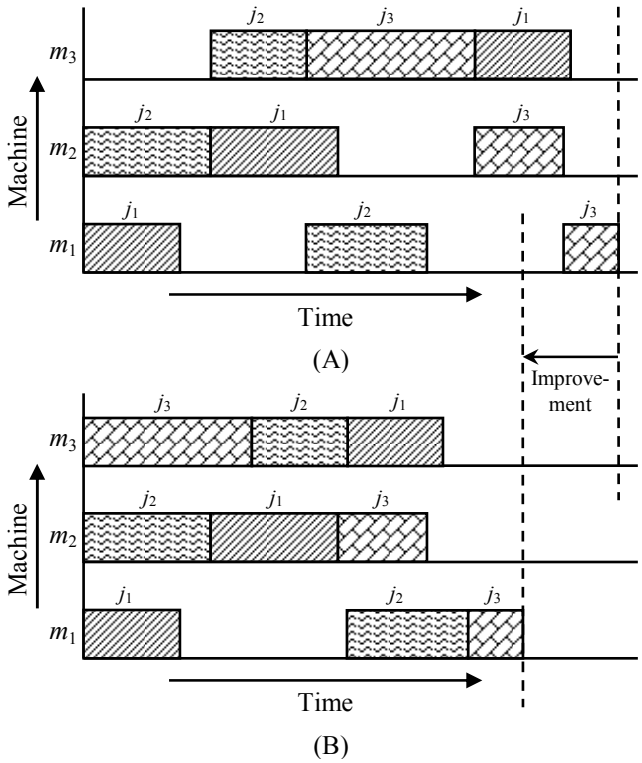

Fig. 1. Gantt chart of the solution (A) before applying the reducing bottleneck (B) after applying reducing bottleneck and reevaluation.

Dorndorf and Pesch [11] proposed 12 different priority rules to achieve good solutions. However they suggested choosing only one of these rules while evaluating the

chromosome. They also applied shifting bottleneck heuristic proposed by Adams *et al.* [19] for solving JSSP.

But these algorithms were implemented while evaluating the individuals in GA. It acts as a step of evaluating i.e. generating the complete feasible schedule; the individuals. On the other hand, we propose to apply our heuristics after completing the evaluation as the proposed re-ordering, gap-utilization and swapping techniques require scheduled solution.

In this process, we identify the bottleneck machine m'' first in phenotype p. Then we determine the corresponding job of the last operation of m'' which the longest job in p. Assume j' requires machine m' for its first operation. The ordering technique then moves the operation of job j' in machine m' from its current k^{th} position to the 1^{st} position. Algorithm I and II in Appendix describe the process of the ordering.

Fig 1 (A) shows that j_3 is the longest job of the given solution. The algorithm puts the first operation of that job in the first position of the corresponding machine which is shown in Fig 1 (B). The change of makespan after reordering can be observed from Fig. 1. as indicated by the dotted line.

IV. Gap-Utilization Technique

The repairing technique proposed by Nakano & Yamada [22], and Yamada [23] ensures the feasibility of individual solutions that may not produce quality objective value. By applying simple rules, it is possible to improve the quality of solution.

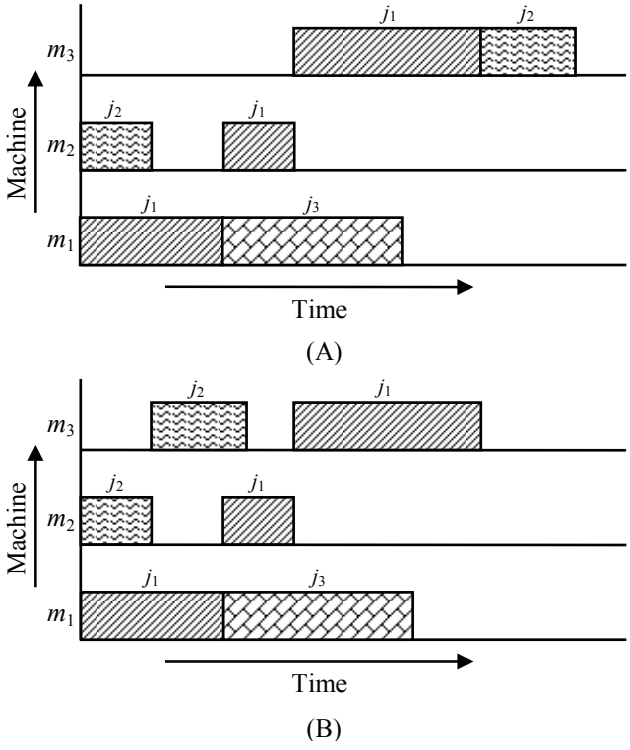

Fig. 2. Two steps of a partial Gantt chart while building the schedule from the phenotype. The *x* axis represents the execution time and the *y* axis represents the machines.

We explain the technique using Fig. 2. In the phenotype p, j_2 follows j_1 in machine m_3, but it is possible to place j_2 before j_1 as shown in Fig. 2. (A). It is due to the fact that there is an unused gap. It allows starting the operation of j_2 on m_3 earlier which is clear from Fig. 2. (B).

Table II reflects the effect of the newly introduced gap-utilization technique. The shaded cells represent the operations that are already been scheduled. j_{ok} and m_{ok} represent the k^{th} operation of job j and machine m respectively.

TABLE II
Status of the Phenotypes when Gap-Utilization Technique is Applied

	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

(A)

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

(B)

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

(C)

V. Adjusting Bottleneck Technique

After the implementation of GUT, we swap each operation that belongs to the bottleneck job with the immediate predecessor of the corresponding operation if and only if the both the operations do not conflict with any other operation.

Suppose j' is the bottleneck job in 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 the immediate predecessor operation of that corresponding job, 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 detail of this algorithm is mentioned in Algorithm IV of Appendix.

We choose some of the individuals randomly and apply swapping. As the complexity of the algorithm is simply an order of n, it does not affect the overall computational complexity that much.

VI. Implementation

At first we implement the simple genetic algorithm (SGA) as proposed by Nakano & Yamada [22], and Yamada [23]. They proposed the job pair relation based representation in their work. We also use the phenotype as another form of representation, as all the modification that we propose directly operates on the phenotype. We apply simple genetic operators like two-point crossover and bit-flip mutation on the binary genotype, as in [22, 23].

Then we incorporate the reducing bottleneck technique (RBT) with SGA, which is described in Algorithm II of

Appendix. In the second approach, we implement our gap-utilization technique (GUT) with the SGA. Nakano & Yamada [22], and Yamada [23] proposed the simple evaluation technique. However our GUT does certain reordering on the phenotype before creating schedules. We also apply the adjusting bottleneck technique (ABT) after implementing GUT for possible further improvement. ABT is described in Algorithm IV in Appendix. For ease of explanation, the combined GUT, ABT and SGA approach is introduced as GUT algorithm in this paper.

A. Genetic Algorithm with Gap-Utilization Technique

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. K is the total number of individuals in each generation.

1. Initialize $P(t)$ as a random population $P(t=0)$ of size, where each random individual is a bit string of length L.
2. Repeat until some stopping criteria are met
   A. Ste t:=t+1
   B. Evaluate $P'(t)$ from $P(t$-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-utilization technique (GUT) 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 adjusting bottleneck technique (ABT) on some of the individuals selected in a random manner.
   D. 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 $R \le R_c$ then
         a. Select randomly one individual p_1 from the top 25% 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 $R \le (R_c + R_m)$ then randomly select one individual from $P(t)$ and apply bit-flip mutation.
         c. Else make the clone of p as p'.
         [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]

These steps represent only the GUT. In the case of RBT, we simply apply the reducing bottleneck technique instead of the step 2.C and apply simple technique while evaluating the individuals in step 2.B.ii. The algorithms are clearly depicted in two different flowcharts in Fig. 3.

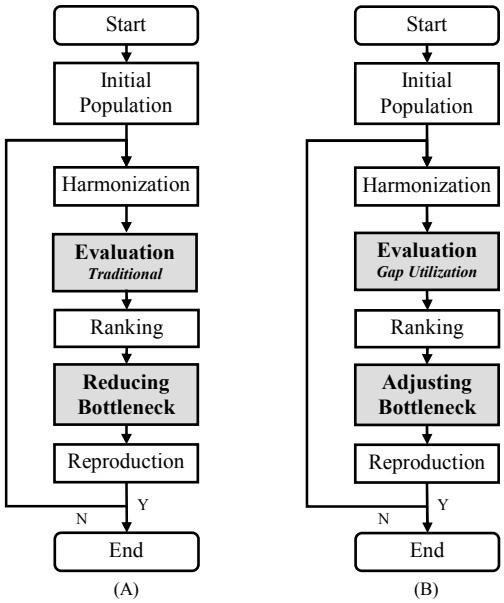

Fig 3. The flowchart (A) represents the first approach RBT and (B) represents the second approach GUT (here Y means the stopping criteria are met and N otherwise). The differences are indicated by the shaded areas.

Sometimes the genetic operators misguide the individuals and cause the solution to diverge from the optimal. The elitism technique is effective in this case to ensure the survival of the fittest [26]. We apply elitism in each generation to preserve the best solution so far found 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 one and two-point crossovers, and a bit-flip mutation technique. We also apply the ranking on the basis of the fitness value.

To test the performance of the our proposed algorithms, we solve 35 benchmark problems designed by Lawrence [24] and compare with several existing algorithms. The problems range from 10×5 to 30×10 where $n \times m$ represents n jobs and m machines.

VII. Experimental Result

The results for the benchmark algorithm are obtained by executing the algorithms on a personal computer. Results are tabulated in Table III, IV and V. Table III compares the best makespan found in our three approaches with that of other existing algorithms and the result found so far. All the results are based on 30 individual runs with different random seeds.

TABLE III
COMPARISON OF THE FITNESS VALUES WITH LITERATURE

Instances	Best Known Solution	SGA	RBT	GUT	Binato *et al.* (GRASP)	Beatrice and Mario		Croce *et al.* (LAS)	Adams *et al.*	
						LSGA	Hybrid		SB I	SB II
la01	666	**666**	667	**666**	666	666	–	666	666	–
la02	655	**655**	**655**	**655**	655	655	–	666	720	669
la03	597	605	617	**597**	604	597	–	666	623	605
la04	590	607	607	**590**	590	590	–	–	597	593
la05	593	**593**	**593**	**593**	593	593	–	–	593	–
la06	926	**926**	**926**	**926**	926	926	–	926	926	–
la07	890	**890**	**890**	**890**	890	890	–	–	890	–
la08	863	**863**	**863**	**863**	863	863	–	–	868	863
la09	951	**951**	**951**	**951**	951	951	–	–	951	–
la10	958	**958**	**958**	**958**	958	958	–	–	959	–
la11	1222	**1222**	**1222**	**1222**	1222	1222	–	1222	1222	–
la12	1039	**1039**	**1039**	**1039**	1039	1039	–	–	1039	–
la13	1150	**1150**	**1150**	**1150**	1150	1150	–	–	1150	–
la14	1292	**1292**	**1292**	**1292**	1292	1292	–	–	1292	–
la15	1207	**1207**	**1207**	**1207**	1207	1207	–	–	1207	–
la16	945	982	994	**945**	946	–	959	979	1021	978
la17	784	793	785	787	784	–	792	–	796	787
la18	848	861	861	861	848	–	857	–	891	859
la19	842	885	881	850	842	–	860	–	875	860
la20	902	915	915	907	907	–	907	–	924	914
la21	1046	1114	1115	1091	1091	1114	1097	1097	1172	1084
la22	927	993	1021	970	960	989	980	–	1040	944
la23	1032	1037	**1032**	**1032**	1032	1035	1032	–	1061	1032
la24	935	1022	978	986	978	1032	1001	–	1000	976
la25	977	1084	1066	991	1028	1047	1031	–	1048	1017
la26	1218	1305	1301	1226	1271	1307	1295	1231	1304	1224
la27	1235	1343	1328	1292	1320	1350	1306	–	1325	1291
la28	1216	1316	1311	1260	1293	1312	1302	–	1256	1250
la29	1157	1294	1296	1238	1293	1311	1280	–	1294	1239
la30	1355	1469	1454	**1355**	1368	1451	1406	–	1403	1355
la31	1784	1784	1784	**1784**	1784	1784	1784	1784	1784	–
la32	1850	1850	1855	**1850**	1850	1850	1850	–	1850	–
la33	1719	1719	1719	**1719**	1719	1745	1719	–	1719	–
la34	1721	1744	1747	**1721**	1753	1784	1758	–	1721	–
la35	1888	1907	1903	**1888**	1888	1958	1888	–	1888	–

The first two columns of Table III indicate the problem instances and the best known solutions found from literature. The next three columns represent the best result found by our algorithms. The next columns show the result found from Binato *et al.* [21] who solved JSSP by using GRASP; Beatrice and Mario [20] who implemented a local search based GA and tabu search based algorithm; Della Croce *et al.* [10] who introduced a lookahead simulation based encoding scheme and Adams *et al.* [19] who used the shifting bottleneck heuristic approach for solving JSSP, respectively. In most cases, our GUT is better than any other algorithms. If the average relative deviation (ARD) and standard deviation of the relative deviations (SDRD) are considered, our algorithm performs better than each algorithm considered in this paper. It indicates that the algorithm gives steady solutions than other algorithms.

Table IV shows the comparison of the ARD and SDRD with other results found in literature. As each author did not consider the same number of problems, ARD and SDRD are calculated by the number of problems they considered.

TABLE IV
COMPARISON OF THE AVERAGE AND STANDARD DEVIATION OF THE RELATIVE DEVIATIONS

Author(s)	No. of Problems Considered	Average of Relative Deviations	Standard Deviation of Relative Deviations
Our Work			
SGA	35	0.028125	0.037925
RBT	35	0.026800	0.035823
GUT	35	**0.010040**	**0.019105**
Binato *et al.* (GRASP)	35	0.014758	0.027332
Beatrice and Mario			
LSGA	30	0.029861	0.039164
Hybrid	20	0.032561	0.031529
Croce *et al.* (LAS)	9	0.025309	0.038179
Adams *et al.*			
SB I	35	0.031839	0.040136
SB II	19	0.021833	0.019411

Table V shows the detailed result of our experiments.

TABLE V
EXPERIMENTAL RESULTS OF THE PROPOSED ALGORITHMS WITH TWO DIFFERENT PERFORMANCE METRICS

Instances	Problem Size	Population Size	Relative Deviation ($\times 10^{-2}$)			Normalized STDEV ($\times 10^{-2}$)		
			SGA	RBT	GUT	SGA	RBT	GUT
la01	10×5	625	0.0000	0.1502	0.0000	0.0121	0.0100	0.0077
la02	10×5	625	0.0000	0.0000	0.0000	0.0153	0.0195	0.0043
la03	10×5	625	1.3400	3.3501	0.0000	0.0269	0.0251	0.0009
la04	10×5	625	2.8814	2.8814	0.0000	0.0064	0.0073	0.0062
la05	10×5	625	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
la06	15×5	750	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
la07	15×5	750	0.0000	0.0000	0.0000	0.7589	0.7941	0.0000
la08	15×5	750	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
la09	15×5	750	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
la10	15×5	750	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
la11	20×5	875	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
la12	20×5	875	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
la13	20×5	875	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
la14	20×5	875	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
la15	20×5	875	0.0000	0.0000	0.0000	1.7352	1.5144	0.6287
la16	10×10	1000	3.9153	5.1852	0.0000	1.4725	1.5633	1.5174
la17	10×10	1000	1.1480	0.1276	0.3827	1.6388	1.7453	0.1442
la18	10×10	1000	1.5330	1.5330	1.5330	1.9154	1.9035	0.0440
la19	10×10	1000	5.1069	4.6318	0.9501	1.5386	1.4760	1.0111
la20	10×10	1000	1.4412	1.4412	0.5543	2.6764	2.1820	0.6442
la21	15×10	1125	6.5010	6.5966	4.3021	1.8672	1.4949	0.9588
la22	15×10	1125	7.1197	10.1402	4.6386	3.3898	1.4413	1.0788
la23	15×10	1125	0.4845	0.0000	0.0000	1.8883	1.9498	0.1229
la24	15×10	1125	9.3048	4.5989	5.4545	1.4394	1.8134	0.9621
la25	15×10	1125	10.9519	9.1095	1.4330	1.8978	2.2257	1.0202
la26	20×10	1250	7.1429	6.8144	0.6568	1.3997	1.7733	1.4640
la27	20×10	1250	8.7449	7.5304	4.6154	1.8787	1.9062	0.5772
la28	20×10	1250	8.2237	7.8125	3.6184	0.9280	0.9674	0.9802
la29	20×10	1250	11.8410	12.0138	7.0009	1.4336	1.3016	1.0173
la30	20×10	1250	8.4133	7.3063	0.0000	2.3267	2.0810	0.7850
la31	30×10	1500	0.0000	0.0000	0.0000	0.4275	0.4801	0.0000
la32	30×10	1500	0.0000	0.2703	0.0000	1.5421	1.3137	0.0000
la33	30×10	1500	0.0000	0.0000	0.0000	0.9378	0.6685	0.0000
la34	30×10	1500	1.3364	1.5107	0.0000	1.3251	1.3905	0.3658
la35	30×10	1500	1.0064	0.7945	0.0000	1.6941	1.7218	0.1278
		Average	2.8125	2.6800	1.0040	1.0335	0.9649	0.3848

The first three columns represent the problem instance, problem size and population size respectively. The next three columns represent the relative deviation i.e. deviation of a result with the best known result; of three of our algorithms. The next three columns represent the normalized standard deviation which is calculated by the formula –

$$NSTD = \left(Standard\ Devation \Big/ Best\ Fitness \right) \times 10^2 \qquad (4)$$

Normalization is important due to the different ranges of fitness values. The multiplying factor 10^2 is used simply to show more digits within the limited space.

We use the fixed crossover rate xP=0.85 and mutation rate mP=0.15 and set the population size nP=25×N+75×M where N is the number of jobs and M is the number of machines. We apply more weight on M because the problem complexity increases with the increase of number of machines. The number of generations nI was fixed at 1000. From our experiments and observation, we chose these parameters that are good enough for these algorithms. We applied both RBT and ABT on 10% of the population.

The result shows that the performance of RBT leads over SGA in most cases. And the GUT gives a highly competitive result with the average and standard deviation of the relative deviations which are 0.010040 and 0.019105 respectively. The algorithm also gives a promising average normalized standard deviation of 0.003848. The result can compete with most of the genetic approaches for solving JSSP.

VIII. Conclusion

Though JSSP is a very old and popular problem, still no algorithm can assure the optimal solution for all test problems, specifically for larger problems appearing in the literature. However, GAs are well-known for producing good quality solutions for smaller problems within a reasonable computational time. Although our algorithms also cannot guarantee optimal solutions, the results are very competitive as compared to the existing algorithms, while testing for different sizes of test problems. We would like to extend our research by introducing constraints such as machine breakdown, dynamic job arrival, machine addition and removal, and due date restrictions. Currently we are testing our algorithms by solving larger problems as proposed by Storer *et al.* (swv11-swv20) [8].

References

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

[2] 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.

[3] A. D'Ariano, D. Pacciarelli, and M. Pranzo, "A branch and bound algorithm for scheduling trains in a railway network," *European Journal of Operational Research,* vol. In Press, Corrected Proof.

[4] B. Peter, J. Bernd, and S. Bernd, "A branch and bound algorithm for the job-shop scheduling problem." vol. 49: Elsevier Science Publishers B. V., 1994, pp. 107-127.

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

[6] H. Fisher and G. L. Thompson, "Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules," in *Industrial Scheduling* Englewood Cliffs, New Jersey: Prentice Hall, 1963, pp. 225-251.

[7] D. Lawrence, "Job Shop Scheduling with Genetic Algorithms," in *Proceedings of the 1st International Conference on Genetic Algorithms*: Lawrence Erlbaum Associates, Inc., 1985.

[8] R. H. Storer, S. D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling." vol. 38: INFORMS, 1992, pp. 1495-1509.

[9] 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," *INFORMS Journal on Computing,* vol. 6, pp. 118-125, March 1994 1994.

[10] F. Della Croce, R. Tadei, and G. Volta, "A genetic algorithm for the job shop problem," *Computers & Operations Research,* vol. 22, p. 15, 1995.

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

[12] 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.

[13] E. D. Taillard, "Parallel Taboo Search Techniques for the Job Shop Scheduling Problem," *ORSA Journal on Computing,* vol. 6, p. 108, Spring 1994.

[14] E. Nowicki and C. Smutnicki, "A Fast Taboo Search Algorithm for the Job Shop Problem," *Management Science,* vol. 42, pp. 797-813, 1996.

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

[16] J.-P. Vacher, T. Galinho, F. Lesage, and A. Cardon, "Genetic algorithms in a multi-agent system," in *Intelligence and Systems, 1998. Proceedings., IEEE International Joint Symposia on*, Rockville, MD, USA, 1998, pp. 17-26.

[17] A. Cardon, T. Galinho, and J.-P. Vacher, "An Agent Based Architecture for Job-Shop Scheduling Problem Using the Spirit of Genetic Algorithm," in *Proceedings of Evolutionary Algorithms in Engineering and Computer Science, EUROGEN'99* Jyväskylä, Finland, 1999, pp. 12-19.

[18] T. Galinho, A. Cardon, and J.-P. Vacher, "Genetic integration in a multiagent system for job-shop scheduling," *Progress in Artificial Intelligence - IBERAMIA '98,* vol. 484, pp. 76-87, 1998.

[19] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling." vol. 34: INFORMS, 1988, pp. 391-401.

[20] M. O. Beatrice and V. Mario, "Local Search Genetic Algorithms for the Job Shop Scheduling Problem." vol. 21: Kluwer Academic Publishers, 2004, pp. 99-109.

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

[22] 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.

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

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

[25] 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.

[26] 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.

APPENDIX

ALGORITHM I
ALGORITHM TO FIND OUT THE BOTTLENECK JOB

Let $Q_p(m,k)$ is the k^{th} job in machine m for the phenotype p and $C(m,n)$ is the finishing time of n^{th} operation of machine m; where m varies from 1 to M and n varies from 1 to N. *getBottleneckJob* is a function that returns the job which is taking maximum time in the schedule.

getBottleneckJob (*void*)

1. Set m:=1and max:=-1
2. Repeat while $m \leq M$
   A. If $max < C(m,N)$ then
      i. Set $max := C(m,N)$
      ii. Set $j' := Q_p(m,N)$
      [End of Step A If]
   B. Set $m := m+1$
   [End of Step 2 Loop]
3. Return j'

[End of Algorithm]

ALGORITHM II
ALGORITHM FOR THE REDUCING BOTTLENECK TECHNIQUE (RBT)

Let $D(j,m)$ is the m^{th} machine in job j in the predefined machine sequence D. $O_p(j,m)$ is the order of job j in machine m for the phenotype p.

1. Set $j' :=$ *getBottleneckJob* and $m' := D(j',1)$
2. Set $k := O_p(j',m')$
3. Repeat until k>1
   A. Swap between $C_p(m',k)$ and $C_p(m',k\text{-}1)$
   B. Set $k := k\text{-}1$
   [End of Step 3 Loop]

[End of Algorithm]

ALGORITHM III
ALGORITHM FOR THE GAP-UTILIZATION TECHNIQUE (GUT)

Let p be the phenotype of an individual i, M and N are the total number of machines and jobs respectively. S' and C' is the set of starting and finishing times of all the operations respectively those are already been scheduled. $T(j,m)$ is the execution time of the current operation of job j in machine m. $Q_p(m,k)$ is the k^{th} operation of machine m for phenotype p. *mFront*(m) represents the front operation of machine m, *jFront*(j) is the machine where schedulable operation of job j will be processed. *jBusy*(j) and *mBusy*(m) are the busy time for job j and machine m respectively. *max*(m,n) returns the m or n which is the maximum.

1. Set m:=1 and *mFront*(1:M):=0
2. Repeat until all operations are scheduled
   A. Set $Loc := mFront(m)$ and $jID := Q_p(m,Loc)$
   B. If $jFront(jID) = m$ then
      i. Set *flag*:=1 and k:=1
      ii. Repeat until $k \leq Loc$
         a. Set $X := max(C'(m,k\text{-}1), jBusy(jID))$
         b. Set $G := S'(m,k) - X$
         c. If $G \geq T(jID,m)$
            (1) Set $Loc := k$
            (2) Go to Step F
            [End of Step b If]
         d. Set $k := k+1$
         [End of Step ii Loop]
         Else Set *flag*:=*flag*+1
      [End of Step B If]
   C. Set j_1:=1
   D. Repeat while $j_1 \leq J$
      i. Set $mID := jFront(j_1)$
      ii. Find the location h of j_1 in machine mID
      iii. Put j_1 in the front position and do 1-bit right shift from location $mFront(mID)$ to h.
      iv. Set $j_1 := j_1 + 1$
      [End of Step D Loop]
   E. Go to Step A
   F. Place jID at the position Loc
   G. Set $S'(m,Loc) := X$
   H. Set $C'(m,Loc) := S'(m,Loc) + T(jID,m)$
   I. Set $m := (m+1) \bmod M$
   [End of Step 2 Loop]

[End of Algorithm]

ALGORITHM IV
ALGORITHM FOR THE ADJUSTING BOTTLENECK TECHNIQUE (ABT)

Let $Q_p(m,k)$ is the k^{th} job in machine m and $O_p(j,m)$ is the order of job j in machine m particularly for the phenotype p. *nonConflict*(m,i,j) is a function that returns true if the ending time of immediate predecessor operation of j does not overlap with the modified starting time of the same job in machine m and starting time of the immediate following operation of job j does not conflict with the ending time of the same job in machine m.

1. Set $j' :=$ *getBottleneckJob* and $k := N\text{-}1$
2. Repeat while $k \geq 1$
   A. Set $m := S(j',k)$
   B. If $O_p(j',m) \neq 1$ then
      i. Set $j'' := Q_p(m,(O_p(j',m)-1))$
      ii. If *nonConflict*(m,j',j'')=*true*
         a. Swap j' with j'' in phenotype p.
         b. Go to Step C
         [End of Step ii If]
      [End of Step B If]
   C. Set $k := k\text{-}1$
   [End of Step 2 loop]

[End of Algorithm]