AN EVOLUTIONARY ALGORITHM FOR JOB-SHOP SCHEDULING

Adnan Tariq,[1] Iftikhar Hussain,[2] Abdul Ghafoor,[3] Sahar Noor[2]

ABSTRACT

It has been a general perception that the hardness level of job-shop scheduling problems is comparatively high and that is why effective, efficient and accurate procedures for scheduling are required to substantiate its usefulness. Therefore, in this paper a hybrid approach, primarily based on Genetic Algorithm (GA), is presented to handle the problem of scheduling job-shop that consists of m number of machines and n number of jobs. This approach is actually a combination of a Local Search Heuristic (LSH) with standard GA and optimizes the value of makespan. Computational experience, that includes some case studies and a number of problems from literature, shows that the LSH has the tendency to minimize the makespan value and help the algorithm to find out the optimum solution for the problem in fewer generations.

Keywords: *Genetic Algorithm, Scheduling, Job-shop, Local Search Heuristic.*

INTRODUCTION

Manufacturing industries play an important role in the national economies, both in terms of GDP/GNP contribution and level of employment. To enhance productivity and maximize the benefits from a manufacturing system, optimized utilization of resources becomes very essential. Therefore, scheduling is of paramount importance to ensure proper utilization of resources.

Scheduling has wide area of application in scheduling production, trains and flights.

In the class of combinatorial optimization problems, the handling of Job-Shop Scheduling Problem (JSSP) is generally considered to be the hardest of all. Considering the fact that JSSP belongs to the group of NP-complete problems, a considerable margin for further improvement, in current solution approaches, is still available. Due to the size of its solution space handling of JSSP is considered to be extremely difficult. In case of *m* number of machines and *n* number of jobs, $(n!)^m$ are (theoretically) considered to be the total number of possible solutions. Generally, because of the higher degree of difficulty in finding optimum solution, it becomes extremely hard for the conventional search techniques to reach the optimum value in polynomial time. Therefore, such problems can be handled efficiently by the techniques based on heuristic optimization.

A variety of methods based on heuristics can be found in literature to tackle the JSSP[1]. Most renowned ones are Tabu Search (TS)[2 - 4], Simulated Annealing (SA)[5,6], and Genetic Algorithms (GA)[7 - 12]. Also, in[13] a detailed survey regarding solution techniques for JSSP is presented. An optimization based approach of hybrid nature was presented in[14], a Greedy Randomized Adaptive Search Procedure (GRASP) for JSSP was presented in[15]. Also, a parallel GRASP for handling JSSP was presented in[16]. None of the above has been able to solve the JSSP optimally and struggle in different directions is still underway to solve the JSSP.

Genetic Algorithms (GA) is known for solution of combinatorial nature of problems like JSSP but its performance has not been satisfactory for solution of the JSSP. The flexibility level of standard GA, in case of practical applications, is low due its evolutionary nature. This effect of lower flexibility is magnified when problems of complicated nature that posses conflicts and multi-tasking are dealt with. Therefore, a number of local search based algorithms[6,4,1,3] are available in literature. According to[17] though GAs are better equipped to reach the optimum solution rapidly but in the process there is every chance of premature

[1] *Sarhad University of Science & IT, Peshawar, Pakistan.*

[2] *NWFP University of Engineering and Technology, Peshawar, Pakistan.*

[3] *National University of Science and Technology, Rawalpindi, Pakistan.*

convergence and genetic drift. The problem can be somewhat solved by developing hybrid approaches that do not simply depend upon the evolutionary abilities of GA but also benefit from local search heuristics. According to[18] a local search procedure follows a pre-specified set of rules and following those it explores the immediate neighborhood carefully. They also argued that local search combined with GA develops such kind of a hybrid tool that normally has the ability to carryout optimization more efficiently and relieves pressure off the GA parameters. In other words, GA depends far less on its parameters because of having an effective local search heuristic.

A hybrid GA based algorithm is presented in this paper to handle the JSSP which in fact is the combination of standard GA and a procedure for local improvement (LSH). To check the effectiveness of each solution makespan is used as the criteria of measuring performance. The reason, of using makespan as the criteria, is that it has very frequently been used in literature and thus facilitates the comparison of results with those already available in literature. For the purpose of validation of the hybrid tool developed during this research we considered some industrial case studies along with a number of benchmark problems from literature. It has been observed while solving these benchmark problems that the hybrid approach proposed in this paper has the ability to reach the optimum or near optimum solution in the earlier generations. Since in case of more than 90% of the problems the algorithm has been able to reach the optimum makespan value, therefore it proves that the hybrid technique developed during this research is effective and accurate.

THE STANDARD JOB-SHOP SCHEDULING PROBLEM

The standard job-shop scheduling problem consists of the scheduling of *n* number of different jobs on *m* number of different machines with the objective of minimizing makespan while satisfying the following constraints:

1. Each job visits each machine only once.
2. The operation sequence of different jobs is independent of each other.
3. No interruptions are allowed in between.
4. The capacity of each machine is such that it can accommodate only one job at a time.
5. No specifications for due dates and release times are given.

The different notations/ abbreviations used:

Best [i,j]	Best solution found in a generation
Chrom [i,j]	A two dimensional chromosome/solution
C_{max}	Makespan
Count	A counter to count the number of times LSH is being executed
CT_o	Completion Time for operation 'o'
EST_o	Earliest Start Time for operation 'o'
Gen	Generation
i, a	Counter for rows of a solution/chromosome
j, b	Counter for columns of a solution/ Chromosome
JAT_x	Availability time job 'x'
Jobs	Total number of jobs
Machs	Total number of machines
MAT_k	Availability time of machine 'k'
Max Gen.	Maximum number of generations
PT_o	Processing Time for operation 'o'

METHODOLOGY

This has been a general observation of the researchers in general and those in the area of manufacturing systems in particular that the performance, both in terms of accuracy and time, of hybrid algorithms is better than standard GAs. Keeping this aspect of computation in view a hybrid approach is presented in this paper, which is the combination of standard GA and the LSH, proposed during this research. In every generation of GA the solution having the minimum makespan value is further refined (minimized) by applying the LSH to it. This process is repeated for a maximum number of generations the value of which is determined by a sensitivity analysis.

The methodology is developed in software known as AM (Applications Manager)[19]. AM uses a highly visual interactive interface that enables the developer to produce applications easily and quickly.

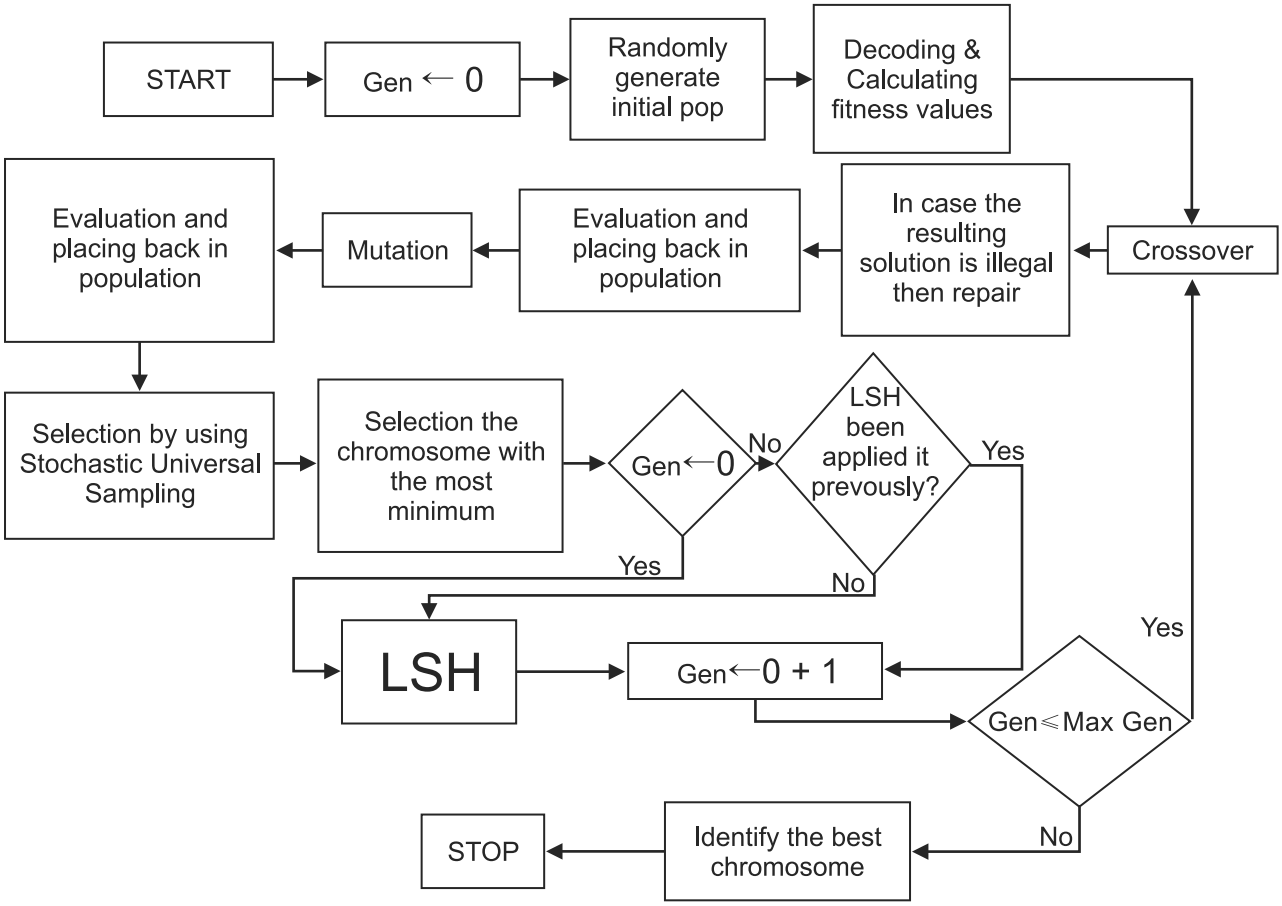

Figure 1: Hybrid methodology for JSSP

AM supports modular development of larger applications. Each module is saved as a separate file and can be linked dynamically with one another.

GENETIC ALGORITHM (GA)

GA is stochastic in nature as it starts with a randomly generated set of solutions and also replicates the principles of natural genetics and natural selection. The GA applied during this research uses a population size of 75, maximum number of generations = 100 and rate of crossover and mutation is 60% and 10% respectively. All these values, of different GA parameters, are determined by carrying out a sensitivity analysis.

Representation

While applying GA to a problem the first thing that needs to be taken care off is devising a representation scheme. The scheme used during this research is integer based. The size of each chromosome, representing a solution to a problem, would be equal to the product of total number of machines and total number of parts in the problem. Each integer in a solution would represent a particular operation of a particular job and that is why each integer is reflected operation times in each solution i.e. if a problem consists of 6 jobs and 6 machines then each integer (from 1 to 6) would exist 6 times in each solution. Table 1 represents a solution for a 6 × 6 problem.

Table 1: Chromosome representation

3	3	1	5	4	3
4	2	1	2	5	5
1	4	6	6	6	4
2	6	5	2	5	1
6	2	4	1	2	5
4	3	3	1	6	3

At each position an integer having a value less than or equal to 6 is inducted and each integer from 1 to 6 would not exist more than 6 times in each solution as no job can have more than 6 operations. It means there are six 1's, six 2's and so on. While moving in the direction of arrow, as shown alongside Table 1, the first integer is '3'. This represents the first operation of job 3. After scheduling the first operation of job 3 on its respective machine, we move to the next integer in the same column that is 4. Since this is the first time integer 4 has been read by the algorithm therefore it represents Job 4's first operation and would be scheduled after Job 3's first operation. Following the same procedure all the other operations would be scheduled one after the other and finally the makespan value is calculated.

Fitness function

The objective of the problem is the minimization of makespan (C_{max}).Therefore the fitness function used here is the reciprocal of the objective function:

$$Fitness\ Function = F = 1/C_{max}$$

Genetic Operators

Genetic operators are the essential components of GA and help us to produce new solutions from the existing population and are therefore mainly responsible for the evolutionary nature of GA. This process of evolution continues until no further improvement can be observed. The different genetic operators used during this research are elaborated in the following sections.

Crossover

Crossover is one of the principal genetic operators of GA. It produces two new chromosomes by combining portions of two existing chromosomes. The procedure for crossover, adopted here, is elaborated by an example as follows:

Table2: Chromosome A

3	3	1	5	4	3
4	2	1	2	5	5
1	4	6	6	6	4
2	2	5	1	5	1
6	6	4	2	2	5
4	3	3	1	6	3

Table 3: Chromosome B

4	5	5	1	2	2
6	1	4	4	6	6
6	4	2	6	2	5
5	4	3	3	3	1
5	5	2	2	1	1
4	3	3	3	1	6

Table 4: Child A

3	3	5	1	4	3
4	2	4	4	5	5
1	4	2	6	6	4
2	2	3	3	5	1
6	6	2	2	2	5
4	3	3	3	6	3

Table 5: Child B

4	5	1	5	2	2
6	1	1	2	6	6
6	4	6	6	2	5
5	4	5	1	3	1
5	5	4	2	1	1
4	**3**	**3**	**1**	**1**	**6**

To carryout the crossover procedure, the first step is to select two parent chromosomes. Here, in this research elitist strategy is adopted for selecting parents. Once the parents are selected (Table 2 and Table 3) then that portions of the two selected chromosomes are identified (randomly) which are to be crossed over, as shown in Table 2 and Table 3 (columns 3 and 4). After this the selected portions of the two chromosomes are interchanged hence producing two new chromosomes (Table 4 and Table 5) which are termed as children. Such a crossover procedure has a side effect of having a tendency of sometimes producing illegal (where some integers may reflect less than or more than the specified limit i.e. operation times) solutions.

Repair Algorithm

Since the crossover procedure applied during this research has the tendency to sometimes produce illegal solutions therefore an algorithm to carryout repair of such solutions is proposed here after cross-over each solution is checked whether legal or not. After confirming illegality, the repair procedure applied is as follows:

1. After confirming illegality the procedure for repair is applied by first collecting information about the integers that are reflected either less than or greater than the specified limit (operation times). By collecting information about such integers we mean storing information about their positions (where they are placed) and their excess or shortage. In this case only those positions would be considered which are outside the crossed over portion.
2. Once the information to be collected in step 1 is available then one by one those integers that were less than the specified limit, are selected and inducted in place of an integer which is in excess. Important thing to note here is that integers in shortage are selected one by one (sequentially) but for placement positions are selected randomly.
3. Step 2 is kept on repeated till the time all the integers that were less than the specified limit are inducted into the solution.

For further clarification of the above procedure an example is presented below by applying the repair procedure to the solution presented in Table 4.

Table 6: Frequency of existence of each integer in the solution

Integer	Frequency of existence	Legal/ excess/ short	Shortage/ excess limit
1	3	Short	3
2	7	Excess	1
3	9	Excess	3
4	7	Excess	1
5	5	Short	1
6	5	Short	1

Step 1: The first thing that the repair strategy has to do is to verify the illegality of a solution. This can be done by collecting information about the frequency of existence of each integer in the solution and then comparing that with the specified limit, as shown in Table 6:

Step 2: Since there are some integers in excess and some in shortage, therefore it proves the illegality of the solution and thus requires repair.

Step 3: At this stage we know which integers are in shortage and which are in excess. So, at this stage information about the placement, of those integers which reflect more than the specified times, is collected and stored. As mentioned earlier only those positions would be considered which lie outside the crossed over portion. Table 7 presents this information.

Step 4: Now the integers in shortage are picked one by one and placed in a location that is randomly selected from the locations mentioned in Table 7, as shown in Table 8.

Table 7: Locations of those integers which are in excess

Integer that is in excess	Locations	
	Row	***Column***
2	2	2
	4	1
	4	2
	5	5
3	1	1
	1	2
	1	6
	6	2
	6	6
4	1	5
	2	1
	3	2
	3	6
	6	1

Table 8: Repair work

Integer in shortage	Frequency of shortage	Integer in excess	Frequency of excess	Locations selected randomly		Assigned values to selected locations
				Row	Column	
1	3	2	1	4	2	1
		3	3	1	2	1
				6	6	1
5	1			6	2	5
6	1	4	1	3	2	6

Step 5: It has been clearly represented in Table 8 that every single illegality has been removed by placing back those integers in the solution which were in shortage. The repaired version of child A (Table 4) is presented in Table 9.

Table 9: Repaired solution

3	1	5	1	4	3
4	2	4	4	5	5
1	6	2	6	6	4
2	1	3	3	5	1
6	6	2	2	2	5
4	5	3	3	6	1

Mutation

It is another principal genetic operator of GA. The role of mutation in GA is to keep the diversity level in a population at a certain level maintain a certain level that prevents the algorithm to get trapped at local optimum. It does that through changing the genetic structure of a solution by incorporating a random change in the value of one or more genes of the solution. The procedure adopted for mutation during this research is of swap mutation type, where in each column of the solution two genes are randomly picked and their values are swapped, as shown in Tables 10 and 11. The usefulness of adopting this mutation approach is that it does not produce illegal solutions.

Decoding procedure:

The decoding procedure used here, is elaborated in the form of a flow chart as shown in Figure 2.

Selection

A number of methods have been used in literature for selecting chromosomes from one generation into another. All these method are based on the evolution theory of Darwin. Stochastic Universal Sampling (SUS)[20] is one of these methods and is used for

Table 10: Chromosome selected for mutation

3	3	1	5	4	3
4	2	1	2	5	5
1	4	6	6	6	4
5	4	3	3	3	1
5	5	2	2	1	1
4	3	3	1	6	3

Table 11: Mutated chromosome

3	3	1	2	6	3
4	5	1	2	5	5
5	4	3	6	4	1
1	4	3	3	3	4
5	2	2	5	1	1
4	3	6	1	6	3

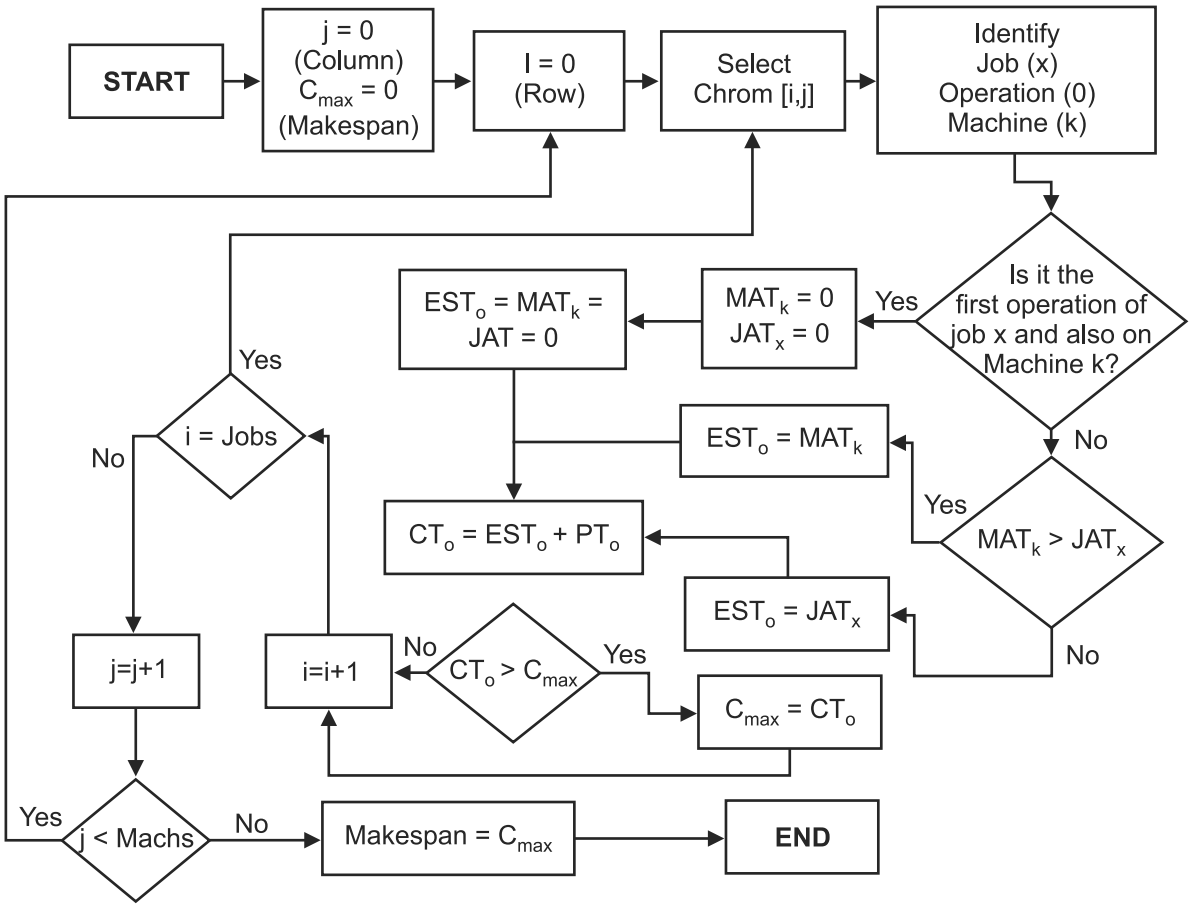

Figure 2: Decoding Procedure

selecting a complete new generation out of the current generation. The main advantage of SUS is its minimum spread and zero bias.

LOCAL SEARCH HEURISTIC (LSH)

The LSH developed during this research, is placed inside the standard GA loop. In each generation, the solution with the minimum makespan value (best of the lot) is further improved by subjecting it to the LSH. The process of local improvement is started with the first two genes in the first row of the solution provided by GA, as the best of the population. These two genes are swapped and after that the solution is decoded and the corresponding makespan value is determined. If this makespan value is smaller than the original makespan value of the solution then the change is stored otherwise genes are reverted back to their original positions. Now the same procedure is repeated with the first and third gene of the same solution in the same row. This process is kept repeated until processing for the first gene against all the other genes in the solution is completed. On completion, the next gene is considered and the same process is repeated. This repetition is kept continued until at least half of the genes are tested against all the other genes. The reason to keep it down to half of the total number of genes is that by the time first 50% of the operations are scheduled a trend has developed and the last 50% follow the same trend and therefore do not affect the makespan value.

Though LSH is very effective but it is helped a great deal by the evolution of GA. It is GA that is responsible to search out a comparatively better solution, which after being subjected to local improvement is converted into an even better one. The possibility getting trapped in local optimum is remote. It is to be noted that this locally improved procedure, in each generation, does not replace any solution in the main population and therefore plays no role in the evolution of GA. In other words the local improvement procedure and the evolution of GA are kept separate so that the natural evolution of GA is not affected by local improvement. This prevents GA from

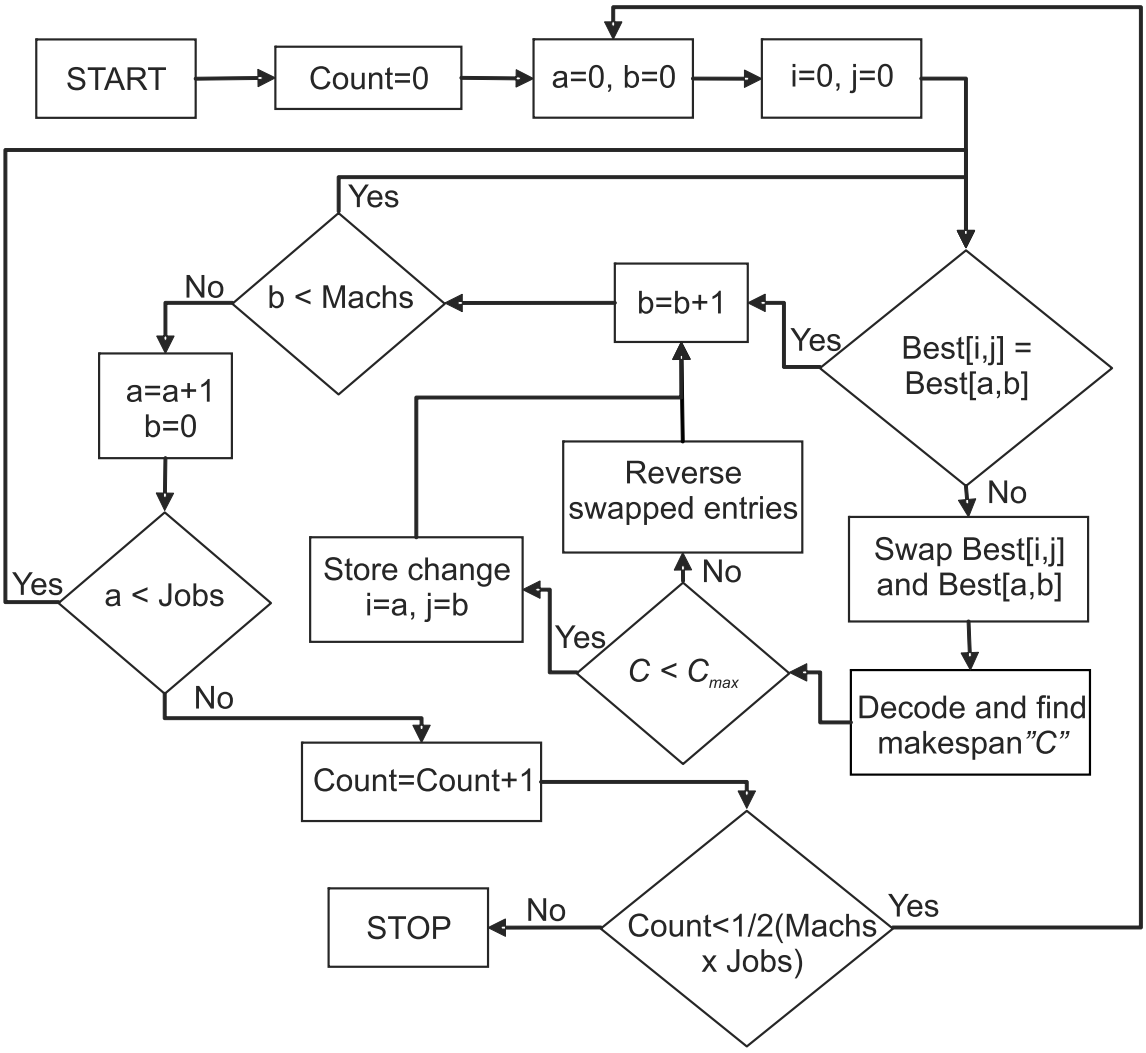

Figure 3: Local Search Heuristic (LSH)

getting trapped in local minimum. Further explanation of the LSH is given in the form of a detailed block diagram as shown in Figure 3.

To further clarify the working of LSH with the help of numerical example, a 4×4 problem is generated randomly as shown in Table 12.

It is required to schedule the system while minimizing the makespan (C_{max}). For this purpose the problem is run on a computer program developed as per the methodology described in Figure 1. The solution that represents the minimum makespan value in first generation of GA is presented in Table 13.

Table 12: A randomly generated 4×4 problem

Jobs	Operations							
	1		2		3		4	
	Machine	Time	Machine	Time	Machine	Time	Machine	Time
1	3	5	2	7	4	6	1	4
2	2	6	1	8	3	7	4	5
3	4	9	3	3	1	4	2	2
4	1	5	4	8	2	6	3	3

Table 13: Solution having the minimum makespan value in 1st generation of GA

2	2	2	4
4	3	4	3
3	4	1	2
1	1	1	3

The chromosome shown in Table 13 is having *makespan* = C_{max} = 33. Its decoded form is shown in Figure 4.

This solution is now subjected to LSH. On completion, the final result found by the LSH is presented in Table 14.

Figure 5 shows the decoded from of the solution, presented in Table 14, on Gantt chart.

SENSITIVITY ANALYSIS:

The sensitivity analysis of the hybrid GA based approach for JSSP is carried out in this section. A problem of the size 10×10^{21} is solved while varying the values of different GA parameters and recording corresponding variation in %age solution gap. This variation in values of different GA parameters in fact helps us in fine tuning the combination in which these parameters would finally be used. Once this process of fine tuning is completed the resulting values of the parameters are selected and then used for solving the rest of the problems. Figures 6, 7, 8 and 9 presents the sensitivity analysis to fine tune the working of the algorithm, developed during this research.

Table 14: Locally improved solution

2	1	3	1
3	1	3	2
4	4	1	4
4	2	3	2

Figure 6 shows the effect of variation in the number of generations on %age Solution Gap. It clearly shows that the minimum value of %age solution gap for the problem is achieved at a generation number 100. On the other hand, Figure 7 shows that suitable population size for the algorithm, after allowing it to complete 100 generations with a 60% crossover rate and 10% mutation rate, is 75.

The justification of using 10% mutation rate, 60% crossover rate, size of population = 75 and the total number of generations = 100, is given in Figures 8 and 9.

Keeping the above analysis in view it is safe to say that the algorithm would perform satisfactorily, with population size = 75, total number of generations = 100, mutation rate = 10%, crossover rate = 60%, in case of all the other tested problems.

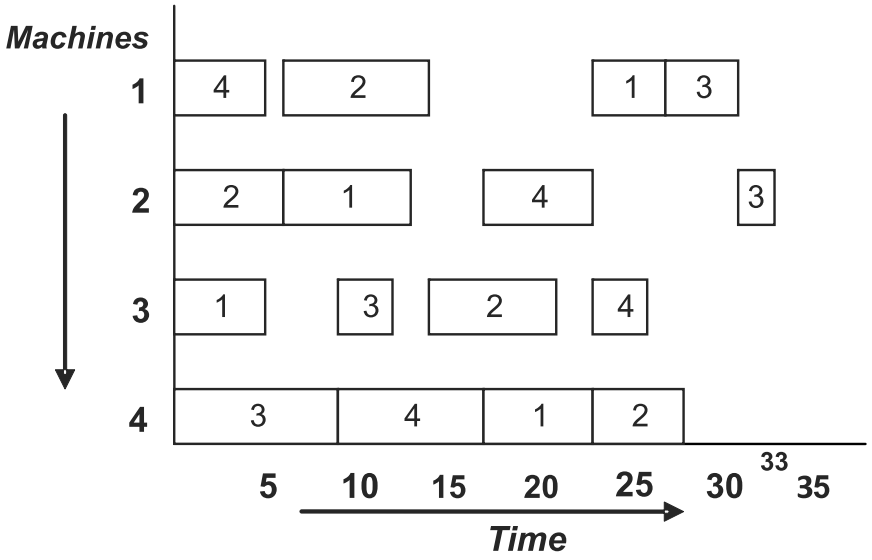

Figure 4: Schedule developed for the best chromosome found in 1st generation of GA

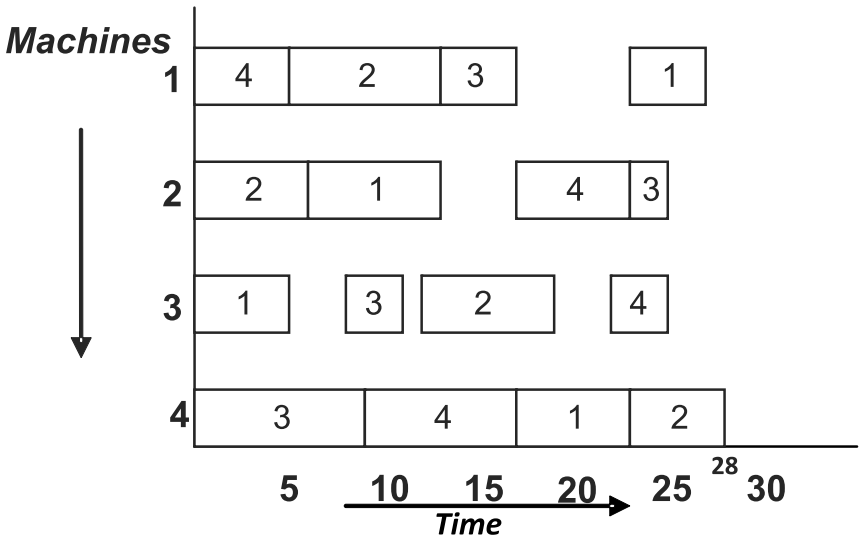

Figure 5: Schedule developed for the solution improved by LSH

NUMERICAL EXAMPLE

To further elaborate the accuracy and effectiveness of the technique proposed in this paper a numerical example (bench mark problem) is selected from literature[21]. The processing requirement and corresponding processing times for each operation are shown in Table 15.

The main objective here is to minimize the makespan value. Though a number of other performance measures can also be used but makespan is considered to be the simplest and more frequently used measure

The data given in Table 15 is given a run on a computer code developed in AM[19] according to the already described hybrid approach, in Figure 1. The optimum value of makespan for the problem was found during the first generation. The best chromosome obtained is shown in Table 16, whereas the decoded schedule is shown in Figure 10.

COMPUTATIONAL RESULTS

A reasonable computational experience (Table 18) shows that the approach developed during this research has been able to determine the optimum value of makespan in more than 90% of the problems tried so far. This shows that the technique developed here has the tendency to produce accurate results and can be implemented in practical situations. The CPU Time presented in the table is for the number of generations in which the minimum value of makespan is obtained while using Intel® T2130 machine with 1.86 GHz processor and 1.0 GB RAM.

Table 15: A 6 × 6 bench mark problem

Jobs	Operations											
	1		2		3		4		5		6	
	Machine	Time	Machine	Time	Machine	Time	Machine	Time	Machine	Time	Machine	Time
J1	3	1	1	3	2	6	4	7	6	3	5	6
J2	2	8	3	5	5	10	6	10	1	10	4	4
J3	3	5	4	4	6	8	1	9	2	1	5	7
J4	2	5	1	5	3	5	4	3	5	8	6	9
J5	3	9	2	3	5	5	6	4	1	3	4	1
J6	2	3	4	3	6	9	1	10	5	4	3	1

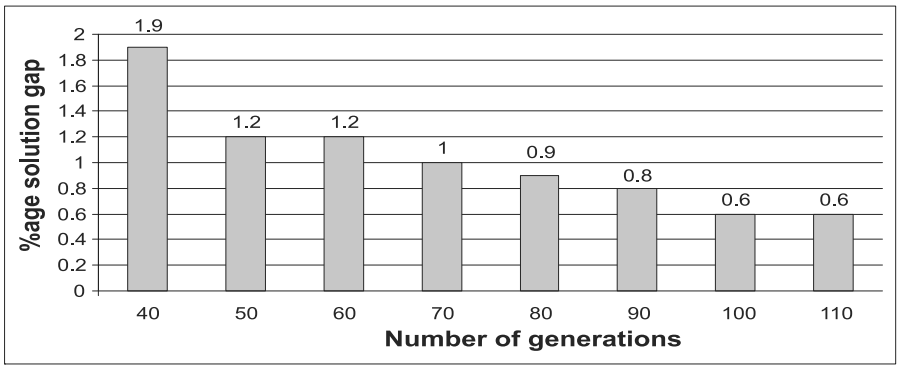

Figure 6: Effect of the number of generations on %age solution gap

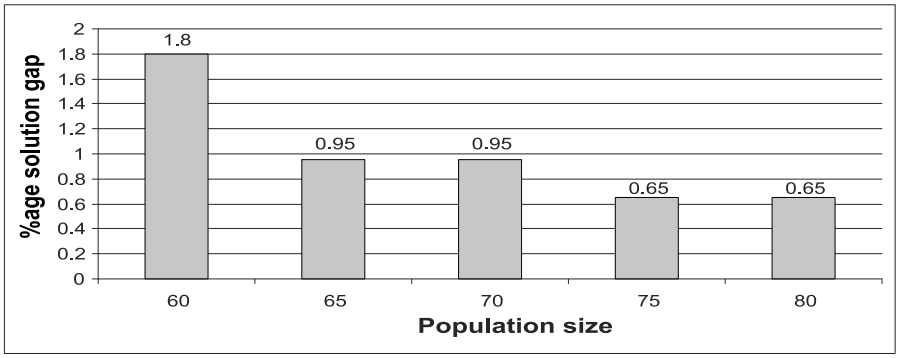

Figure 7: Effect of population size on %age solution gap
Number of generations = 100, Mutation rate = 10% Crossover rate = 60%

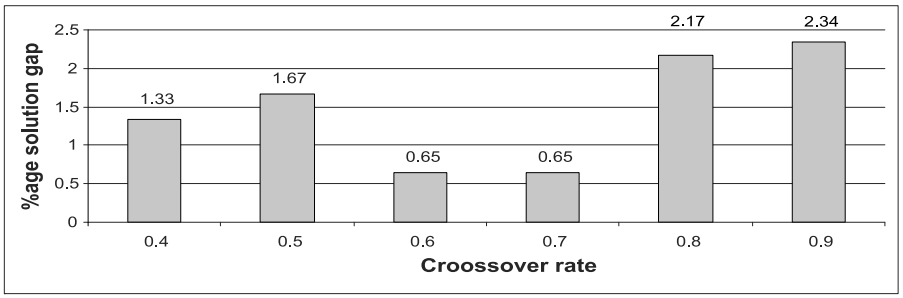

Figure 8: Effect of crossover rate on %age solution gap
Size of Population = 75, Number of generations = 100, Mutation rate = 10%

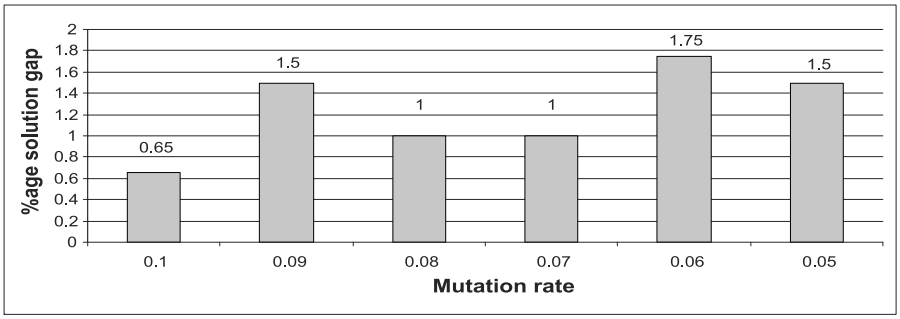

Figure 9: Effect of mutation rate on %age solution gap
Size of Population = 75, Number of generations = 100, Crossover rate = 60%

Table 16: Best Chromosome

Makespan = 55 time units

1	6	2	1	3	3
2	1	5	4	6	3
4	5	3	6	2	1
3	3	4	2	6	2
2	5	4	5	5	5

Figure 11 and Figure 12 shows the total generations and corresponding CPU time consumed by the algorithm to reach the optimum or near optimum solution, respectively.

Figure 11 is good representation of the fact that this algorithm, having an effective LSH at the heart of the GA loop, has the ability to reach the optimum or near optimum solution in earlier generations.

The hardness of a problem depends on the number of machines, number of operations and parts/machines ratio. Also, square problems are comparatively harder even if the number of operations is same. As the number of machines and operations are comparatively maximum in the case of 10x10 problem and at the same time it is a square problem too, therefore the algorithm has taken longer time to find its optimum.

The ability of the algorithm to reach the optimum value in more than 90% of the problems

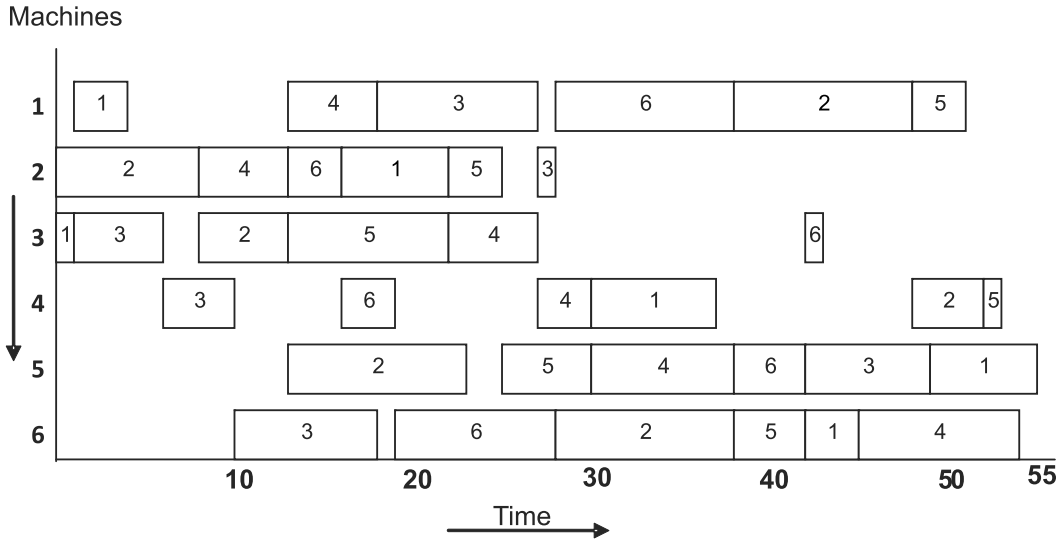

Figure 10: Schedule developed after decoding the best chrom.

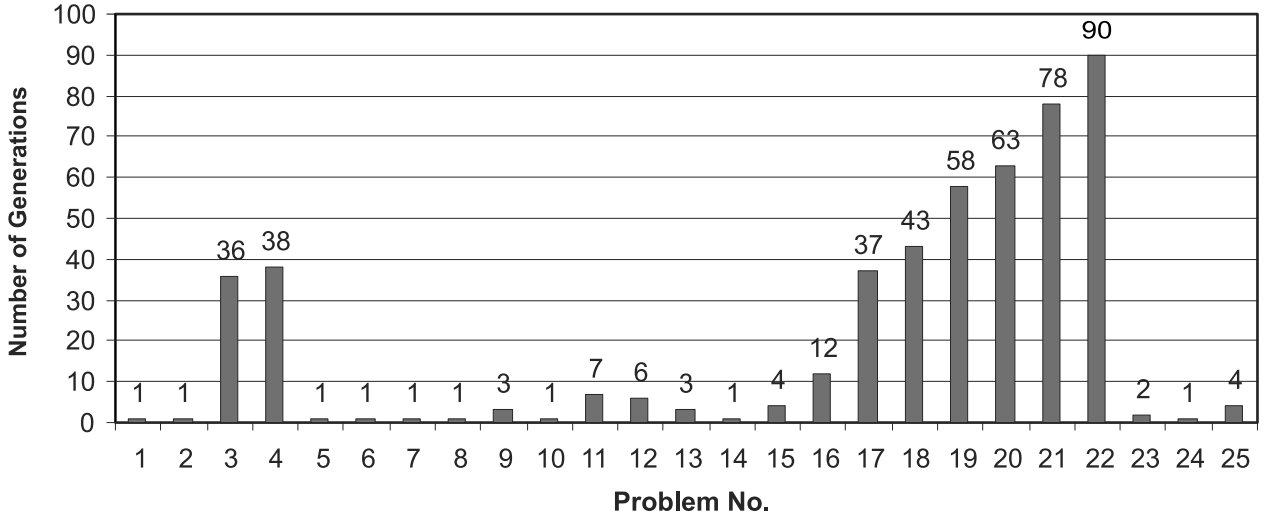

Figure 11: Number of generations to reach the optimum/near optimum result

Table 17: Computational Results

S/ No.	Problem	Size Jobs x Macks	Source	Optimal Makespan (OM)	Makespan found (M)	%age solution gap = (M-OM)* 100/0m	Num of Gens	CPU time (Sec)
1	FT 6	6 × 6	Fisher and Thompson, 1963.		55	55	0	1 1
2	LA 1	10 × 5	S. Lawrence, 1984.	666	666	0	1	4
3	LA 2	10 × 5	S. Lawrence, 1984.	655	655	0	36	328
4	LA 3	10 × 5	S. Lawrence, 1984.	597	597	0	38	431
5	LA 4	10 × 5	S. Lawrence, 1984.	590	590	0	1	13
6	LA 5	10 × 5	S. Lawrence, 1984.	593	593	0	1	1
7	LA 6	15 × 5	S. Lawrence, 1984.	926	926	0	1	4
8	LA 7	15 × 5	S. Lawrence, 1984.	890	890	0	1	12
9	LA8	15 × 5	S. Lawrence, 1984.	863	863	0	3	15
10	LA9	15 × 5	S. Lawrence, 1984.	951	951	0	1	5
11	LA10	15 × 5	S. Lawrence, 1984.	958	958	0	7	65
12	LA11	20 × 5	S. Lawrence, 1984.	1222	1222	0	6	72
13	LA12	20 × 5	S. Lawrence, 1984.	1039	1039	0	3	35
14	LA 13	20 × 5	S. Lawrence, 1984.	1222	1222	0	1	59
15	LA14	20 × 5	S. Lawrence, 1984.	1292	1292	0	4	40
16	LA15	20 × 5	S. Lawrence, 1984.	1207	1207	0	12	75
17	LA16	10 × 10	S. Lawrence, 1984.	945	945	0	37	476
18	LA17	10 × 10	S. Lawrence, 1984.	784	784	0	43	734
19	LA18	10 × 10	S. Lawrence, 1984.	848	848	0	58	965
20	LA19	10 × 10	S. Lawrence, 1984.	842	842	0	63	1234
21	LA20	10 × 10	S. Lawrence, 1984.	902	902	0	78	3245
22	FT 10	10 × 10	Fisher and Thompson, 1963.	930	936	0.645	90	6653
23	Case Study-1	8×6	S. Noor and M. K. Khan, 2007.	505	505	0	2	8
24	Case Study-2	6×6	S. Noor and M. K. Khan, 2007.	444	444	0	1	3
25	Case Study-3	6×6	S. Noor and M. K. Khan, 2007.	379	379	0	4	18

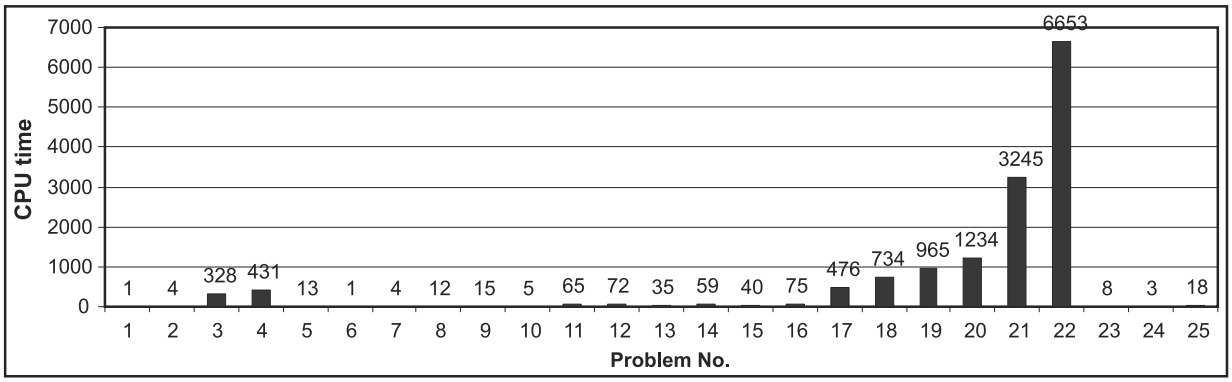

Figure 12: CPU time (sec) taken by each problem to reach the optimum/ near optimum result

is due to the fact that the algorithm has an effective LSH inside the GA loop. Because of its placement inside the GA loop in each generation the best solution of the algorithm is subjected to local improvement and afterwards the resulting make-span value is compared with the overall makespan value and if found lesser than that then the overall minimum solution found, so far, is replaced by the locally improved solution. Another aspect of the algorithm that keeps intact the evolutionary ability of GA, besides being hybrid in nature, is that it does not place the improved solution back into population thus avoiding any randomness creeping into the search.

CONCLUSION AND FUTURE WORK

This research has come up with an approach that is the combination of an LSH with standard GA by using integer based representation scheme, swap mutation, multipoint crossover and stochastic universal sampling (sus) for selection. The organization of the algorithm is such that in each generation the solution with the minimum makespan value (best of the lot) is further improved by subjecting it to the LSH and after that the minimized makespan value is compared with the overall minimum value found so far. If this new makespan value is smaller than the overall value then this locally improved solution replaces the overall minimum solution found so far and for future generations now this would act as the benchmark. Computational experience with the algorithm shows that LSH has the ability to lead the algorithm to the optimum or near optimum solution in earlier generations. Further since in more than 90% of the problems the optimum results have been achieved, this proves the accuracy of the approach.

REFERENCES

1. *Vaessens, R. J. M., Aarts, E. H. L., Lenstra, J. K., 1996. "Job-shop scheduling by local search". INFORMS Journal on computing 8, 302-317.*

2. *Taillard, E. D., 1994. "Parallel Tabu search techniques for the job-shop scheduling problem". ORSA Journal on Computing, 6(2), 108-117.*

3. *Nowicki, E., Smutnicki, C., 1996. "A fast tabu search algorithm for the job-shop problem". Management Science, 42(6), 797-813.*

4. *Lourenco, H. R., Zwijnenburg, M., 1996. "Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem". In: Osman, I. H., Kelly, J. P. (Eds), Metaheuristics: Theory and applications. Kluwer Academic Publishers, 219-236.*

5. *Laarhoven, P. J. M. V., Aarts, E. H. L., Lenstra, J. K., 1992. "Job-shop scheduling by simulated annealing". Operations Research, 40, 113-125.*

6. *Lourenco, H. R., 1995. "Local optimization and job-shop scheduling problem". European Journal of Operations Research, 83, 347-364.*

7. *Davis, L., 1985. "Job-shop scheduling with genetic algorithm". In: Proceedings of the First International Conference on Genetic Algorithms and their Applications. Morgan Kaufmann, 136-140.*

8. *Storer, R. H., Wu, S. D., Wu, S. D., Park, I., 1992. "Genetic algorithms in problem space space for sequencing problems". In: Proceed-*

ings of a Joint US-German Conference on Operations Research in Production Planning and Control, 584-597.

9. *Aarts, E. H. L., Van Laarhoven, P. J. M., Lenstra, J. K., Ulder, N. L. J., 1994. "A computational study of local search algorithms for job-shop scheduling". ORSA Journal on Computing, 6, 118-125.*

10. *Dorndorf, U., Pesch, E., 1995. "Evolution based learning in job-shop environment". Computers and Operations Research, 22, 25-40.*

11. *Croce, F., Tadei, R., Volta, G., 1995. "A genetic algorithm for job-shop problem". Computers and Operations Research, 22 (1), 15-24.*

12. *Noor, S., 2007. "Operational scheduling of traditional and flexible manufacturing systems using genetic algorithms, artificial neural networks and simulation. PhD Thesis, University of Bradford, UK.*

13. *Jain A. S. and Meeran S., 1999 "Deterministic job-shop scheduling: Past, Present and Future", European Journal of Operational research 113, 390-434.*

14. *Wang, L. Zheng, D., 2001. "An effective hybrid optimization strategy for job-shop scheduling problems". 8, 585-596.*

15. *Binato, S., Hery, W. J., Loewenstern, D. M., Resende, M. G. C., 2002. "A GRASP for job-shop scheduling". In: Ribeiro, C. C., Hansen, P. (Eds.), Essays and Surveys in Metaheuristics. Kluwer Academic Publishers.*

16. *Aeix, R. M., Binato, S., Resende, M. G. C., 2003. "Parallel GRASP with path-relinking for job-shop scheduling". Parallel Computing, 29, 393-430.*

17. *Uckun, S., Bagchi, S., Kawamura, K., Miyabi, Y., 1993. "Managing Genetic Search in Job-Shop Scheduling". IEEE Experts, 15-24.*

18. *Tsai, C. F., Lin, F.C., 2003. "A new hybrid heuristic technique for solving job-shop scheduling problem". IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 8-10 September 2003, Lviv, Ukraine.*

19. *Applications Manager (AM), 2001. Intelligent Environment, Middlesex, UK.*

20. *Chaperfield A., Flemming P., Pohlhein H., Fonseca C., 2001. "Genetic Algorithm MATLAB Tool Box – User's Guide" Version 1.2, Department of Automatic Control and Systems Engineering, University of Sheffield.*

21. *Muth, J. F., and Thompson, G.L., 1963. "Industrial scheduling". Prentice Hall, Eaglewood Cliffs, New Jersey, 1963.*