Optimization of Job Shop Scheduling Problem using Jaya Algorithm

Shah H. K.[1], Raj Abhishek K.[2], Mehta A. A.[3], H. S. Keesari[4]
[1, 2, 3, 4] Department of Mechanical Engineering
[1, 2, 3, 4] Sinhgad Institute of Technology, Lonavala

Abstract- *Optimization algorithms are search methods where the goal is to find an optimal solution to a problem in order to satisfy one or more objective functions which are possibly subjected to a set of constraints. In this study, a recently developed Jaya algorithm is tested for solving the constrained and unconstrained optimization problems which is based on the concept that the solution obtained for a given problem should move towards the best solution and should avoid the worst solution. By applying the same concept, we are reducing the ideal time of different Job Shop Scheduling Problem (JSSP) benchmark functions by rescheduling the jobs through Position Based Crossover method and minimizing the overall makespan thus optimizing the entire process by Jaya Algorithm.*

The JSSP can be stated as follows that n jobs to be processed through m machines. Each job should be processed through the machines in a particular order without precedence constraints among different job operations

Keywords- Jaya Algorithm, Job Shop Scheduling Problem, Makespan, Position Based Crossover Mechanism.

I. INTRODUCTION

In the phase of global competition, the requirement for dimension accuracy, mechanical properties and surface properties have become a major challenge on manufacturing industries since almost every service that we use in our day-to-day life is somehow linked to the optimization of resources.

Scheduling (production process) is an important tool for manufacturing and engineering. This importance stems from the desire to lower production costs, increase profits, save time, increase production rate and thus optimizing resources. Also, Scheduling problems exist in many real-life situations, such as transportation, management, engineering, biomedical, construction, production processes. Our way of life, made possible by modern computer technology and networking, demands efficient scheduling. And so Scheduling for job shops is an important topic in production management. It is concerned with determining the release order and times of a set of jobs on the relevant machines subject to the processing constraints in an effort to improve the production efficiency and reduce the processing duration so as to gain as high profits as possible. The processing time of an operation cannot be known precisely and the due-date may be flexible in real-world, processing time and due-date with fuzzy value is quite usual nowadays in practice. The fuzzy job-shop scheduling problem (fJSSP) extends the JSSP by considering the processing time or the due-date to be fuzzy value. The fuzzy flexible job-shop scheduling problem (fFJSSP) is a combination of the FJSSP and fJSSP, which is more close to the production reality [1]. By introducing more real-world constraints, the problems can be solved by optimizing the real time data in hand.

JSSP is one of many types of scheduling problems that researchers from many fields are currently attempting to solve optimally using various meta-heuristic algorithms. The solution to these scheduling problems is simply the determination of the optimal assignment of a finite number of resources to a finite number of operations, while adhering to many pre-defined constraints, usually precedent constraints. Precedent constraints, or technological constraints, dictate the order of operations for each job, or the order of machines a job must visit. A solution to a JSSP is a schedule specifying when each machine is to start processing certain operations that does not violate any precedent constraints. The Job Shop Problem belongs to the class of NP-Hard problems, and is commonly thought of as one of the harder problems in that class. NP-Hard problems have an exponentially growing search space as the problem increases in dimension. Therefore, methods and algorithms must be developed to provide good search directions within this space for our modern computers to perform their calculations. Where exact methods of optimization search out the best solution exhaustively by using mathematical formulations, methods of computational intelligence which rely on certain heuristic principles and ideas to explore and then converge to the best found solution. Optimization algorithms are search methods where the goal is to find an optimal solution to a problem, in order to satisfy one or more objective functions, possibly subject to a set of constraints. Thus a simple yet powerful optimization algorithm is used in this paper for solving the discrete optimization problems. This algorithm is based on the concept that the solution obtained for a given problem should

move towards the best solution and should avoid the worst solution. The performance of this algorithm is investigated by implementing it on four unconstrained benchmark functions, having different characteristics from the literature. In addition to solving the unconstrained benchmark problems, this algorithm also investigates real time problems on JSSP. The ultimate goal is to minimize the makespan of the problem, or the minimum time required for all jobs to finish processing, thus optimizing the entire job shop scheduling problem through Jaya Algorithm.

II. LITERATURE REVIEW

The Job Shop Scheduling Problem (JSSP) is one of the most popular scheduling models existing in practice which is among the hardest combinatorial optimization problems.

Keesari and Rao concluded that the TLBO algorithm can be effectively used for job shop scheduling problems. From the experimental results it can be said that when the size of the JSSP is low then TLBO gives the better solutions in terms of best make-span, relative percent error and standard deviation [2].

Rao and Waghmare provided the verification of performance of the TLBO algorithm with well-known other optimization methods, by experimenting with different multi-objective unconstrained and constrained benchmark functions [3].

Gao exhibited the remanufacturing scheduling problem as two stage FJSP with new job inserting. A two-stage artificial bee colony algorithm is proposed to solve scheduling and rescheduling with new job inserting. In this paper, three re-scheduling strategies are proposed for rescheduling and compared with standards [4].

The standard job shop scheduling problem has been widely adopted as a model in the research of optimization algorithms. There are several objective functions to be considered in theoretical investigations, of which the most frequently studied is the make span criterion i.e. maximum completion time of all jobs. Ziaee examined the FJSSP with preventive maintenance constraints. The objective of this paper is to minimize the make-span, the total workload of machines and the workload of most loaded machine. The main purpose is to produce reasonable schedules very quickly. The proposed approach uses an accurate, relatively comprehensive and flexible criterion for scheduling job operations constructing a feasible high-quality solution [5].

Pierre examined an unexplored approach to the design of heuristics: change of neighbourhood in the search. He called this method variable neighbourhood search method (VNS). Contrary to other metaheuristics based on local search methods, VNS does not follow a trajectory but explores increasingly distant neighbourhoods of the current incumbent solutions and jumps from this solution to a new one if and only if an improvement has been made. This is useful in scheduling and finding the optimum solution [6].

Lixin studied two-machine flowshop scheduling with batching and release time, whose objective is to minimize the makespan. He derived a lower bound and developed a dynamic programming-based heuristic algorithm to solve the scheduling problem. The numerical results show that some of the heuristic algorithms can indeed find effective solutions for the scheduling problem [7].

Rao developed the Jaya Algorithm. The algorithm always tries to get closer to success (i.e. reaching the best solution) and tries to avoid failure (i.e. moving away from the worst solution). The algorithm strives to become victorious by reaching the best solution and hence it is named as Jaya (a Sanskrit word meaning victory). He accentuated that the proposed Jaya algorithm is not claimed as the 'best' algorithm among all the optimization algorithms available in the literature. In fact, there may not be any such 'best' algorithm existing for all types and varieties of problems. What can be said with more confidence at present about the Jaya algorithm is that it is simple to apply without any algorithm-specific parameters and provides the optimum results in comparatively less number of function evaluations [8].

It has been observed from the literature review that different researchers proposed different optimization methods to solve the job shop scheduling problems. Many researchers have focused on minimizing the makespan. In this study, an attempt is made to apply a recently developed advanced optimization algorithm known as Jaya algorithm to solve the job shop scheduling problems.

III. MATHEMATICAL MODEL OF JOB SHOP SCHEDULING PROBLEM (JSSP)

The optimization process contains the combination of four main points, namely, JSSP, Jaya Algorithm, Position Based Crossover (PBC/PBX) mechanism and Minimizing Makespan.

The jobs are scheduled through PBX mechanism and the best/minimum makespan is calculated for the best combination. Further it is combined with Jaya Algorithm and the overall process is optimized for makespan.

A solution to a Job Shop scheduling Problem is a schedule specifying when each machine is to start processing certain operations that does not violate any preceding constraints. The ultimate goal is to minimize the makespan of the problem, or the minimum time required for all jobs to finish processing.

In a JSSP, a job can be processed by any machine of an associated pre-specified subset of the machine set. Thus, these problems generalize problems in which a job can be processed by each machine of the machine set. The objective is to find a schedule that minimizes the makespan.

A JSSP can be described as follows. We have a set of n jobs need to be operated on a set of m machines. Each job has its own processing route, i.e. jobs visit machines in different sequences. Each job may need to be performed only on a fraction of m machines, not all of them.

The following assumptions are generally made in the Job Shop Scheduling problem:

- Each machine can perform only one operation at a time on any job.
- An operation of a job can be performed by only one machine at a time.
- Once an operation has begun on a machine, it must not be interrupted.
- An operation of a job cannot be performed until its preceding operations are completed.
- There are no alternate routings, i.e., an operation of a job can be performed by only one type of machine.
- Operation processing time and the index of operable machines are known in advance.
- The jobs are independent; that is, there are no precedence constraints among the jobs and they can be operated in any sequence.
- Setup times are sequence dependent.
- All the jobs are available for their process at various times.

The classical JSS problem can be described as follows: There are a set of m machines and a set of n jobs. Each job consists of a sequence of operations, each of which needs to be processed during an uninterrupted time period of a given length on a given machine. Each machine can process at most one operation at a time. We assume that any successive operations of the same job are processed on different machines. A schedule is an assignment of the operations to time intervals on the machines. The problem is to find a schedule which optimizes a given objective. Assume that three finite sets J, M, O are given where J is a set of jobs 1… n, M is a set of machines 1 … m, and O is a set of operations 1, …, N.

Consider the following denotations:

J_i = the job to which operation i belongs,
M_i = the machine on which operation i is to be processed,
t_i = the start time for operation i,
p_i = the processing time for operation i,
C_{max} = the makespan.

On O, a binary relation $\rightarrow$ is defined that represents precedence constraints between operations of the same job.

If $i \rightarrow j$, then $J_i=J_j$ and there is no $k \in \{i,j\}$ satisfying $i \rightarrow k$ or $k \rightarrow j$. (Operation i is the predecessor of operation j). Thus, if $i \rightarrow j$, then $Mi \neq M_j$ by the JSSP specifications.

The problem of optimal job scheduling is to find a starting t_i time for each operation $i \in O$ such that:

$$\text{Max } (t_i+p_i) \text{ as } i \in O \text{ is minimized,} \quad (1)$$

Subjected to:

$$\forall\, i \in O : t_i \geq 0 \quad (2)$$

$$\forall\, i, j \in O,\ i \rightarrow j : t_j \geq t_i+p_i \quad (3)$$

$$\forall i,j \in O,\ i \neq j,\ Mi = M_j : (t_j \geq t_i+p_i) \vee (t_j \geq t_i+p_j) \quad (4)$$

The conditions (3) express precedence constraints which represent technological link-up of operations within the same task. The conditions (4) express machine capacity constraints, i.e. each machine can process at most one operation at a time.

The described equations cannot be directly used for determining a schedule. We need to eliminate symbols of binary relation $\rightarrow$ and disjunction $\vee$ and try to get a formulation of integer programming.

The binary relation can be eliminated easily so that O will be decomposed into subsets of operations that correspond to tasks. Then we will assign to operations in each task numbers creating a sequence of consecutive integers by the operation order.

Denote n_j = the number of operations in job j, and N_j = the total number of operations of the first j jobs.
Evidently:

$$N_0 = 0, \quad N_j = \sum_{k=1}^{j} n_k, \quad N = \sum_{k=1}^{n} n_k \quad (5)$$

Using the denotation for total number of operations of the first j-1 jobs, we assign to n_j operations of the first j-1

jobs, we assign to n_j operations of task j numbers $N_{j-1}+1, \ldots, N_{j-1} + n_j$ where $N_{j-1} + n_j = N_j$.

Now we can express equation (3) as follows:

$$(\forall\, j \in J)\ (N_{j-1}+1 \le i \le N_j-1) : t_{i+1} \ge t_i + p_i \qquad (6)$$

The makespan is then determined as the maximum of the completion times of the last operations in jobs. Hence, we get:

$$\forall j \in J : C_{max} \ge t_n + p_{Nj} \qquad (7)$$

Let us define capacity constraints using binary variables $x_{i,j} \in \{0,1\}$ as follows:

$$\forall\, i, j \in O,\ i \ne j,\ Mi = M_j :$$

$x_{ij} = 1, t_j \ge t_i + p_i$, operation i precedes operation j. (8)

$= 0,\ t_j \ge t_j + p_j$, operation j precedes operation i

If T is an upper bound of the makespan, then, using x, we can replace equation (4) by pairs of inequalities as follows:

$$\forall\, i, j \in O,\ i \ne j,\ M_i = M_j : t_j \ge t_i + p_i\, x_{ij} - T(1-x_{ij}) \text{ or } t_i \ge t_j + p_i\,(1-x_{ij}) - T x_{ij} \qquad (9)$$

Hence, the job shop scheduling problem with makespan objective can be formulated as follows:

Minimize $C_{max,}$ Subjected to,

$\forall\, i \in O : t_i \ge 0$

$(\forall\, j \in J)\ (N_{j-1}+1 \le i \le N_j-1) : t_{i+1} \ge t_i + p_i$

$\forall j \in J : C_{max} \ge t_n + p_{Nj}$

$\forall\, i, j \in O,\ i \ne j,\ Mi = M_j : x_{i,j} \in \{0,1\}$

$: t_j \ge t_i + p_i\, x_{ij} - T(1-x_{ij})$ or $t_i \ge t_j + p_i\,(1-x_{ij}) - T x_{ij}$

IV. JAYA ALGORITHM

The algorithm always tries to get closer to success (i.e. reaching the best solution) and tries to avoid failure (i.e. moving away from the worst solution). The algorithm strives to become victorious by reaching the best solution and hence it is named as Jaya (a Sanskrit word meaning **victory**).

Let f(x) is the objective function to be minimized or maximized. At any iteration i, assume that there are 'm' number of design variables (i.e. j=1, 2,…, m) & 'n' number of candidate solutions (i.e. population size, k=1, 2,…,n). Let the best candidate obtains the best value of f(x) (i.e. f(x) best) in the entire candidate solutions and the worst candidate obtains the worst value of f(x) (i.e. f(x) worst) in the entire candidate solutions. If Xj,k,i is the value of the jth variable for the kth candidate during the ith iteration, then this value is modified as per the following Eq. (1). X'j,k,i= Xj,k,i+ r1,j,i (Xj,best,i- |Xj,k,i|) - r2,j,i (Xj,worst,i- |Xj,k,i|).....(1) where, Xj,best,i is the value of the variable j for the best candidate and Xj,worst,i is the value of the variable j for the worst candidate. X'j,k,i is the updated value of Xj,k,i and r1,j,i and r2,j,i are the two random numbers for the jth variable during the ith iteration in the range [0, 1]. The term "r1,j,i ((Xj,best,i- |Xj,k,i|)" indicates the tendency of the solution to move closer to the best solution and the term "-r2,j,i (Xj,worst,i- |Xj,k,i|)" indicates the tendency of the solution to avoid the worst solution. X'j,k,I is accepted if it gives better function value. All the accepted function values at the end of iteration are maintained and these values become the input to the next iteration [8].

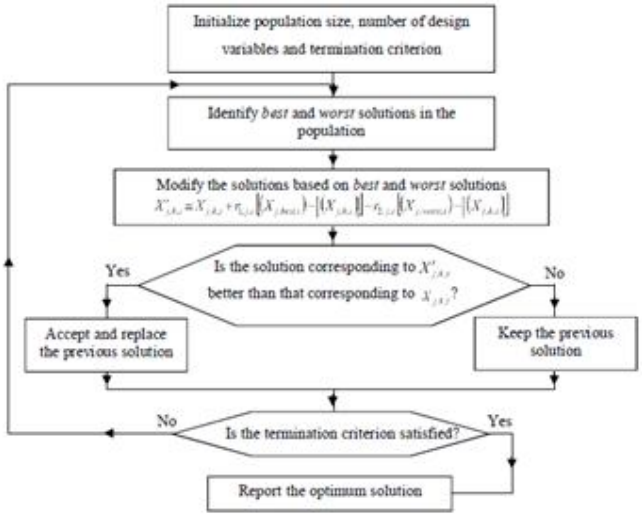

Fig. 4.1 Flowchart of the Jaya Algorithm [8].

Pseudo code for Jaya Algorithm

Objective Function= fx(i,j)=x(i,1)2+a(i,2)2+….+x(i,j)2 (1)

Design Variables= x1(i,j,Ni) =

x(i,j)+rand*(bestsolution(Ni,j)-abs(x(i,j)))-rand* (worstsolution(Ni,j)-abs(x(i,j))) (2)

New Objective Function= fx1(i,j) =

x1(i,1)2+x1(i,2)2 +……+x1(i,j)2 (3)

Begin

Initialize N (Population Size), D (Design Variables), Ng (Generations) and Ni (Iterations)

for all design variables

Randomly the values are taken within the range

Evaluate the solutions/objective_functions for each member of the population size

for i=1:N (Population size)

Update the value of solutions/objective_functions according to (1)

end for

while (stopping condition is mentioned)

Identify the best solution and worst solution as bestsolution

and worstsolution
Evaluate the New Design Variables
for i=1:N (Population size)
for j=1:D (Design Variables)
Update the design variables according to (2)
while (condition for keeping the design variables in the range)
Again update the design variables according to (2) if not within the range
end while
end for
end for
Evaluate the new solutions/objective functions for each member of the population size
for i=1:N (Population size)
Update the new values for objective function according to (3)
end for
for i=1:N (Population size)
if new solution value is less than the old solution value
Replace the old solution value with the new solution value
end if
end for
Find the position and value of the minimum objective function from the whole process
if Ng==Ni
Process needs to stop (condition mentioned in while)
else
Continue with the process/cycle (condition mentioned in while) end if
Display the value of design variables for which we get the optimum solution
end while
end for
end

V. MAPPING OF JAYA ALGORITHM ON JSSP

In our JSSP, the goal is to find a global optimization of the makespan, i.e. we try to find the job operation scheduling list that minimizes the makespan value. For this, the steps of operation can be described as follows:

The First Step:

The initial parameters i.e. algorithm parameters such as the number of students and number of generations are set. Next, the job's processing time on each machine and the job's machine sequence will be given at this step.

Table 1. An example of job processing time on each machine for 10 jobs and 5 machines.

	M1	M2	M3	M4	M5
J1	21	53	95	55	34
J2	21	52	16	26	71
J3	39	98	42	31	12
J4	77	55	79	66	77
J5	83	34	64	19	37
J6	54	43	79	92	62
J7	69	77	87	87	93
J8	38	60	41	24	83
J9	17	49	25	44	98
J10	77	79	43	75	96

In our solution representation, a solution in JSSP is an operation scheduling list, which is represented as a student in our Jaya algorithm. Each dimension in a student represents one operation of a job. Each job appears exactly m times in an operation scheduling list. For the n-job and m-machine problem, each food source contains nxm dimensions corresponding to nxm operations. Here we have taken one of the instances (problems) from a set of 64 JSSP test instances i.e. instance la01 which is given below in the Table 1.

This is the representation of sequence in which each job will be visiting each machine at a time. Since each job has five operations, it occurs ten times in the operation scheduling list. The interpretation of the example above is as follows:

Table 2. A sequence of job processing time on each machine for 10-job, 5-machine.

	OPR1	OPR2	OPR3	OPR4	OPR5
J1	1	0	4	3	2
J2	0	3	4	2	1
J3	3	4	1	2	0
J4	1	0	4	2	3
J5	0	3	2	1	4
J6	1	2	4	0	3
J7	3	4	1	2	0
J8	2	0	1	3	4
J9	3	1	4	0	2
J10	4	3	2	1	0

The Second Step:

Now calculate the mean of the makespan and select any one solution which is very nearer to the mean (M_D). Now this solution will act as the mean (M) for that iteration.

M = the solution of the makespan which is very nearer to

the mean (M_D).

$X_t = X_{f(x) = min}$.

The best solution will try to shift the mean from M towards X_t which will act as a new mean for the iteration. So,

$M_{new} = X_t$.

The difference between two means is expressed as

$D_D = r (M_{new} – T_F * M_D)$.

The difference (D_D) is updated by old mean (M_D) and new mean (M_{new}) solutions using, Variable Neighbourhood Search Method.

Then the current solutions are updated by using the relation shown below

$X_{new} = X_{old} + D_D$

The obtained difference is used to the current solution to update its values using Variable Neighbourhood Search Method. By considering difference (D_D) as the new mean and each solution (X_{old}) in the population as old mean one at a time, JAYA updates the old solutions. Then after calculating the makespan by applying the greedy selection method improves the solutions.

The Third Step:

The new solutions (X_{new}) are improved by Variable Neighbouring Search method (VNS). A local search based on the Variable Neighbouring Search method (VNS) is performed on the new solutions to improve the solution quality. The pseudo code of VNS method is shown below.

Although it seems that VNS would actually find the best solution by itself, it sometimes takes a long time to reach useful solutions whilst solving large scale Job Shop Scheduling.

Variable Neighbouring Search method (VNS):

VNS Procedure:

Get Initial solution, x'=xb
Set Step =0 and p=1
n= number of jobs
m = number of machines
i = random integer number [1, n*m]
j = random integer number [1, n*m], i ≠ j
x' = exchanging process (x', i, j)
i = random integer number [1, n*m]
j = random integer number [1, n*m], i ≠ j
x' = inserting process (x', i, j)
i = random integer number [1, n*m]
j = random integer number [1, n*m], i ≠ j
x' = exchanging process (x', i, j)
While (step ≤ (n*m)*(n*m-1))
i = random integer number [1, n*m]
j = random integer number [1, n*m], i ≠ j
If (p=1) then x'' = exchanging process (x', i, j)
Else if (p=0) then x'' = inserting process (x', i, j)
If (fitness(x'') ≥ fitness(x')) then x' = x''
Else p = |p-1|
Step=step+1
End while
If (fitness(x') ≥ fitness (xb)) then xb= x'
End procedure

i and j are the random integer numbers between 1 and n*m, Exchanging Process (x, a, b) means exchanging the job operations in solution x between ith and jth dimensions, i ≠ j. Inserting Process (x, a, b) means removing the job operation in solution x from the ith dimension and inserting it in the jth dimension. The example of the exchanging process and the inserting process are shown in Figs. 5.2 and 5.3, respectively.

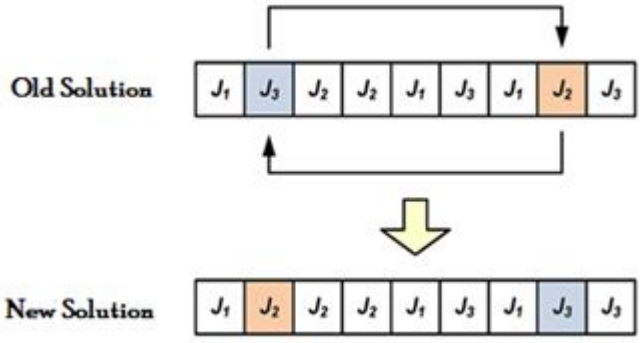

Figure 5.2: Exchanging process in VNS method for new solution [2].

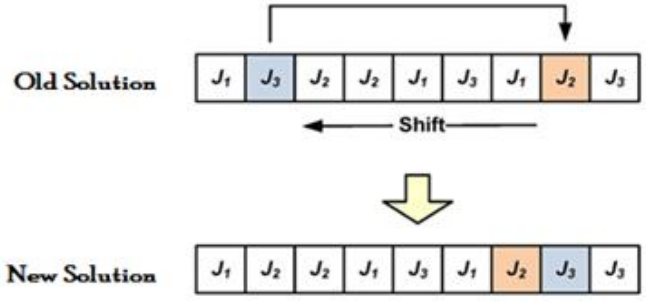

Figure 5.3: Inserting process in VNS method for new solution [2].

The Fourth Step :

If the termination criterion is satisfied then the algorithm stops otherwise goes to the next run or iteration.

The following three heuristic rules are implemented on them for the selection:

1) If one solution is feasible and the other is infeasible, then the feasible solution is preferred.
2) If both the solutions are feasible, then the solution having the better objective function value is preferred.
3) If both the solutions are infeasible, then the solution having the least constraint violation is preferred.

These rules are implemented at the end of Steps 2 and 3 [2].

In short, this is the mapping of Jaya algorithm on JSSP problems.

VI. RESULTS AND CONFIRMATION OF THE EXPERIMENT

Our goal is to solve the job shop scheduling problem using advanced optimization techniques and to evaluate the performance of the Jaya algorithm.

Jaya algorithm is based on the concept that the solution obtained for a given problem should move towards the best solution and should avoid the worst solution. The performance of this algorithm is investigated by implementing it on unconstrained benchmark functions, having different characteristics from the literature.

Also, while solving the JSSP we are interested only in the makespan of a particular solution to the JSSP, which is just a number. It is possible to judge the complexity of a problem by looking at the complexity of its solution. However, it is interesting and helpful in understanding the problem, when we look at the schedule in the form of a Gantt chart. In this section Gantt charts of some of the solutions which are given by JAYA algorithm are produced to give the reader some visual feedback.

A 10x5 problem was solved until optimality is reached. In this we can notice that most of the time all machines are processing an operation, this is a result of how the problem happens to be defined, the precedent constraints and processing times happen to allow for very efficient use of time.

In Table 3 actual problem is shown. Also in Table 4 the optimal solution for the problem is shown.

This shows that machine 5 is working more time than the other machines. That means to finish all jobs we must wait until the 5th machine is finished working. So this becomes the makespan for the problem.

‘‘BKS’’ means the best known solution for the instance and ‘‘Best’’ means the best solution found by each algorithm, ‘‘Average’’ and ‘‘S.D.’’ means the average and standard deviation, respectively, of the results over 20 runs, and ‘‘RPE’’ means the relative percent error with respect to the best known solution. RPE is calculated from the equation below

RPE =((Best−BKS)/BKS)x100 [2].

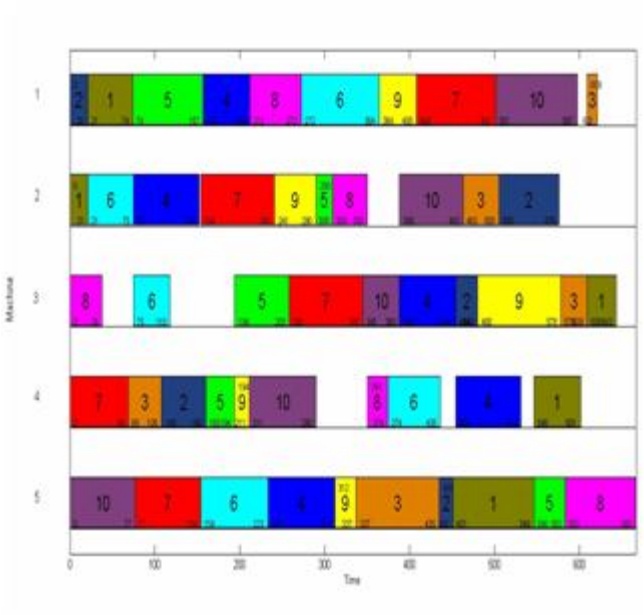

Figure 6.1: Gantt chart of the LA01 (OR-Library) Problem[2]

Table 3: Job Shop Problem Machine Sequence (Time)

Job 1: 2 (21) 1 (53) 5 (95) 4 (55) 3 (34)
Job 2: 1 (21) 4 (52) 5 (16) 3 (26) 2 (71)
Job 3: 4 (39) 5 (98) 2 (42) 3 (31) 1 (12)
Job 4: 2 (77) 1 (55) 4 (79) 2 (66) 3 (77)
Job 5: 1 (83) 4 (34) 3 (64) 2 (19) 5 (37)
Job 6: 2 (54) 3 (43) 5 (79) 1 (92) 3 (62)
Job 7: 4 (69) 5 (77) 2 (87) 3 (87) 1 (93)
Job 8: 3 (38) 1 (60) 2 (41) 4 (24) 5 (83)
Job 9: 4 (17) 2 (49) 5 (25) 1 (44) 3 (98)
Job 10: 5 (77) 4 (79) 3 (43) 2 (75) 1 (96)

Table 4: Optimal solution for the problem

Machine 1: 2 1 9 5 10 6 4 3 8 7
Machine 2: 1 6 3 4 2 7 9 5 10 8
Machine 3: 10 2 5 4 1 8 6 7 9 3
Machine 4: 7 3 1 2 5 8 9 6 4 10
Machine 5: 9 6 4 7 3 2 1 5 8 10

Similarly, we solved 5 other problems. The results for all the problems are shown in the table below:

Table 5: Result table for all the problems.

Sr No.	Problem Name	Worst Solution	Best Solution	Mean	Standard Deviation	Relative Percent Error
1	La01	666	666	666	0	0
2	La02	673	655	665.40	6.0222	-1.6517
3	La03	617	597	609.30	8.4334	-10.3604
4	La04	596	590	591.5	2.1213	-11.4114
5	La05	593	593	593	0	-10.9610

VII. CONCLUSION

This paper has investigated Jaya Algorithm. Experiments on benchmark functions were carried out. Further, comparison of the results of those benchmark functions with standard optimized output concluded Jaya Algorithm to be a simpler and yet an efficient algorithm. The main goal is to optimize JSSP using Jaya Algorithm. The optimization was done by minimizing the makespan with the help of Position Based Crossover mechanism and Variable Neighbourhood Search Method. Keeping the results in mind, so far, this paper proved that the Jaya Algorithm fetched better results than the existing algorithms in optimizing a Job Shop Scheduling Problem.

REFERENCES

[1] Xu Y., Wang L., Wang S., Liu M., "An effective teaching-learning-based optimization algorithm for the flexible job-shop scheduling problem with fuzzy processing time", Neurocomputing , 2013, Vol. 148, pp. 260-268.

[2] Keesari H. S., Rao R. V.; "Optimization of job shop scheduling problems using teaching-learning-based optimization algorithm"; Operational Research Society of India, 2013, Vol. 51, pp. 545-561.

[3] Rao R. V., Waghmare G.G.; "A comparative study of a teaching–learning-based optimization algorithm on multi-objective unconstrained and constrained functions"; Journal of King Saud University – Computer and Information Sciences , 2014, Vol. 26, pp. 332–346.

[4] Gao K. Z., Suganthan P. N., Chua T. Jin, Chang C. Soon, Cai T. Xiang, Pan Q. Ke,; "A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job-shop insertion", Expert Systems with Applications, 2015, Vol. 42, pp. 7652-7663.

[5] Ziaee M., "An efficient heuristic algorithm for flexible job shop scheduling with maintenance constraints"; Applied Mathematics and Sciences: An International Journal (MathSJ), 2014, Vol. 1, No. 1.

[6] Pierre H., Nenad M.; "Variable neighbourhood search: Principles and applications"; European Journal of Operational Research, 2001, Vol. 130, pp. 449-467.

[7] Lixin T, Peng L, "Minimizing makespan in a two-machine flowshop scheduling with batching and release time"; Mathematical and Computer Modelling, 2009, Vol. 49, pp. 1071-1077.

[8] Rao R. V., "Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems"; International Journal of Industrial Engineering Computations, 2015, Vol. 7, pp. 19-34.