Flexible Job Shop Scheduling using Gravitational Search Algorithm

ISBN 978-1-943295-24-1

Nikhil Aditya
Siba Sankar Mahapatra
National Institute of Technology, Rourkela
(nike.aditya@gmail.com)
(mahapatrass2003@gmail.com)

Flexible job shop scheduling problem (FJSP) is one of the complex and important problems in operations management. Over time, many heuristic and metaheuristic approaches have been used to find solutions of FJSP. However, the potential of the gravitational search algorithm (GSA) is still unknown while solving FJSP. Therefore, the present work uses real number encoding-based GSA (RGSA) and chaotic GSA (RCGSA) to solve FJSP. The results of 35 benchmark problems and one industrial test case show that RCGSA performs significantly better than RGSA regarding the quality of solutions and convergence in a limited number of iterations.

Keywords: GSA, Metaheuristic, job shop scheduling, Real encoding, Chaotic GSA

1. Introduction

Smart manufacturing in Industry 4.0 focuses on meeting the dynamic changes in demand and supply. Therefore, smart manufacturing targets shop floor operations, making flexible job shop scheduling problems (FJSP) one of the critical problems in operations management (Elnadi and Abdallah, 2024). The FJSP is an extension of the job shop scheduling problem (JSP), where the routes of the jobs are not fixed since an operation can be processed on multiple machines. The extension of JSP to FJSP was first introduced by (Brucker and Schlie, 1990). The complexity of FJSP is greater than that of JSP, as it includes two problem statements of assignment and sequencing. Due to two subproblems, solution approaches can be hierarchical and integrated. In hierarchical methods, the two subproblems are treated separately, such that sequencing occurs after assignment. In an integrated approach, both subproblems are handled simultaneously (Roshanaei et al., 2013). The FJSP comes under the category of combinatorial optimization and is NP-hard in nature, i.e., as the problem size increases, the time required to find the exact solution to the problem increases exponentially (Jain and Meeran, 1999). Therefore, several attempts have been made over the past three decades to achieve quality solutions in a reasonable amount of time. The solution methodologies for FJSP can be divided into exact, heuristics, and metaheuristics. In exact approaches, a mixed integer linear programming model is usually formulated, which is further solved by branch and bound algorithm or constraint programming. However, exact approaches are not well-suited for large-sized instances. Several heuristics have been used to overcome the limitations of the exact approaches, such as scheduling based on the shortest processing time, earliest due date, shortest remaining processing time, etc. Heuristics provide results faster than exact approaches but are of inferior quality. Therefore, researchers started exploring another class of methods under the approximate approaches known as metaheuristics (Xie et al., 2019). Metaheuristics are intelligent algorithms that work on the principle of exploration and exploitation. Recently, metaheuristic algorithms have been widely used to handle complex FJSP instances with lower flexibility. Metaheuristic algorithms provide quality solutions for large-sized instances in a reasonable amount of time (Dauzère-Pérès et al., 2024).

2. Literature Review

Some of the earliest works reporting the application of a single solution-based metaheuristic for FJSP, such as the tabu search (TS) algorithm, is by (Brandimarte, 1993). Dauzère-Pérès and Paulli (1997) presented an integrated approach to solve FJSP using a TS algorithm. Hurink et al. (1994) presented a more sophisticated neighborhood structure than Brandimarte that used an integrated approach while performing assignment and sequencing in an FJSP. Fattahi et al. (2007) used integrated and hierarchical approaches with simulated annealing (SA) and TS algorithms to solve FJSP of small to medium sizes. The results concluded that hierarchical SA is better than other approaches taken in the literature. Yazdani et al. (2010) proposed a parallel variable neighborhood search (VNS) to improve the exploration rate through multiple searches. The proposed algorithm proved its effectiveness in solving FJSP. Apart from single solution-based metaheuristic approaches, population-based algorithms have also gained importance over the past two due to their competitive nature. Some of the most widely used population-based metaheuristic algorithms are genetic algorithm (GA) and particle swarm optimization (PSO). Lei (2010) used GA with two vector encoding to optimize FJSP with fuzzy processing time. The proposed algorithm performed significantly better than PSO and SA. Pezzella et al. (2008) adopted different initialization, crossover, and mutation strategies to optimize the FJSP. The proposed method performed better than some TS algorithms. Recently, Xue et al. (2024) proposed an improved version of GA with a multi-population search mechanism for FJSP with parallel batch machines. The proposed algorithm performed better than some of the baseline algorithms. Zhang et al. (2020) proposed a discrete PSO by mapping the properties of continuous PSO using crossover and mutation operators for multiobjective optimization of FJSP. Zarrouk et al. (2019) developed a two-

level PSO where level I handles assignments, whereas level II is responsible for the sequencing of operations. Two-level PSO outperformed some standard algorithms in terms of the quality of the solutions and CPU time. Liu et al. (2021) used three neighborhood structures based on a critical path with a hybrid of GA and PSO to optimize FJSP. Xu et al. (2024) recently proposed a real-number encoding-based quantum PSO (QPSO) with different neighborhood structures to optimize FJSP. The proposed performed significantly better than several state-of-the-art algorithms. Based on the critical analysis of the literature, it can be deduced that the most favorable algorithms for optimizing FJSP are GA, PSO, TS, and VNS (Jiang et al., 2023). However, despite having decent performance in the continuous domain, the performance of the gravitational search algorithm (GSA) has not yet been evaluated for optimization of FJSP. Therefore, the present study aims to quantify the performance of GSA and its variant chaotic GSA (CGSA) for optimization of FJSP.

3. Problem Formulation and Mathematical Modelling

The present study optimizes one of the regular criteria of FJSP, i.e., to minimize the makespan (C_{max}) of *n* jobs to be processed on *m* machines. The mixed integer linear programming formulation of FJSP can be given according to (Chen et al., 2020). Let there be *n* jobs $J_1, J_2, J_{3,\ldots\ldots}, J_n$ to be processed on m machines $M_1, M_2, M_{3,\ldots\ldots}, M_m$. Let O_{ij} represent the j^{th} operation of i^{th} job and $t_{i,j,k}$ denotes the processing time of *Oij* on one of its capable machines M_k *(k=1,2,3,……, m)*. C_i represents the completion time of the i^{th} job. Therefore, the MILP formulation of FJSP can be given as follows:

$$Minimize\ C_{max} = Max(C_i) \tag{1}$$

Subjected to:

$$p_{i,j,k} + t_{i,j,k} \leq p_{i,j+1,k}\ i = 1,2,3,\ldots.n, j = 1,2,3,\ldots.,h, k = 1,2,3,,,,,m \tag{2}$$

$$t_{i,j,k} \geq 0\ i = 1,2,3,\ldots.n, j = 1,2,3,\ldots.,h\ k = 1,2,3,\ldots.,m \tag{3}$$

$$\sum_{k=1}^{m} X_{i,j,k} \geq 1\ i = 1,2,3,\ldots.,n,\ j = 1,2,3,\ldots.h \tag{4}$$

$$\sum_{i=1}^{n}\sum_{j=}^{h} X_{i,j,k} = 1,\ k = 1,2,3,\ldots..m \tag{5}$$

$$X_{i,j,k} = \begin{cases} 1, & if\ O_{ij}\ is\ processed\ on\ machine\ k \\ 0, & otherwise \end{cases} \tag{6}$$

In the MILP formulation of FJSP, C_{max} is the makespan that represents the maximum completion time of all the jobs. In equation (1), $p_{i,j,k}$ is the starting time of the j^{th} operation of i^{th} job on the k^{th} machine, whereas $p_{i,j+1,k}$ is the starting time of $j+1^{th}$ operation of i^{th} job on the k^{th} machine. Therefore, constraint 1 ensures the precedence relationship between two operations of the same job on the same machine. Equation (3) depicts the non-zero processing time of all the operations. Equation (4) shows that each operation should have at least one allowable machine. Equation (5) ensures that each machine can process one operation at a time. In equation (6), $X_{i,j,k}$ is the binary variable that shows the capability of a machine to process the operation. Apart from the abovementioned constraints, certain assumptions must be maintained in the classical FJSSP. These assumptions are mentioned below

1. All jobs and machines are available at a time equal to zero.
2. Once the operation is processed on a machine, it cannot be interrupted. Therefore, pre-emption is not allowed.
3. No machine breakdowns are considered.
4. The machine installation time and transportation time between the operations are negligible.
5. The precedence relationship should be strictly followed.
6. There should not be any overlapping of operations assigned on the same machine.

4. Gravitational Search Algorithm (GSA)

Among all the nature-inspired algorithms, the gravitational search algorithm (GSA) proposed by (Rashedi et al., 2009) is one of the famous metaheuristics in the physics-based category. The GSA is inspired by Nweton's law of gravity and the law of motion. According to Newton's law of gravity, the force acting between the two masses is directly proportional to the product of their masses and inversely proportional to the square of the distance between them. Therefore, in GSA, each solution of the population is allotted some mass that reflects the quality of the solution. A heavier mass indicates a solution with a high fitness value and vice-versa. The mathematical expression for calculating the mass is given in equations (7) and (8).

$$mass_p(t) = \frac{fitness_p(t) - worst_{fitness_{value(t)}}}{best_{fitness_{value(t)}} - worst_{fitness_{value(t)}}} \tag{7}$$

$$Mass_P(t) = \frac{mass_p(t)}{\sum_{l=1}^{N} mass_l(t)} \tag{8}$$

In equation (7), $mass_p(t)$ represents the mass value of the p^{th} solution at iteration 't'. The $best\_fitness\_value(t)$ and $worst\_fitness\_value(t)$ denotes the best and the worst solutions present in the population at iteration 't'. From equation (7), it can be easily observed that a candidate solution close to the best solution in the population is rewarded with a higher mass value; on the contrary, the worst solution in the population is allotted a zero mass value. Normalization is done to bring all the mass values to a particular range, as shown in equation (8). After calculating the masses, these masses interact with each other by applying force on each other in accordance with Newton's law of gravity. Therefore, the solutions with poor quality are attracted towards the heavier mass, which improves their quality. Mathematically, force interaction between the two masses is shown in equation (9).

$$F_{ij}^{d}(t) = G(t)\frac{Mass_i(t)\times Mass_j(t)}{R_{ij}(t)+\epsilon}\left(x_j^d(t) - x_i^d(t)\right) \tag{9}$$

In equation (9), $F_{ij}^{d}(t)$ is the force interaction between masses 'i' and 'j' for dimension 'd'. The gravitational constant at iteration 't' is represented by $G(t)$, whereas the distance between the masses is given by the Euclidean distance $R_{ij}(t)$. To prevent the denominator from becoming zero in equation (9), ϵ with a minimal value is added. The gravitational constant in conventional GSA follows an exponential trend, as shown in equation (10). In the gravitational constant ($G(t)$), G_o is 100, and α is 20 (Rashedi et al., 2009).

$$G(t) = G_o e^{-\alpha\frac{t}{T}} \tag{10}$$

In GSA, due to force interaction, the best solutions share information with the average or poor solutions. However, to promote exploitation in basic GSA, only top Kbest masses interact with the other solutions. The Kbest parameter is designed so that it decreases linearly over the iterations. Therefore, over the iterations, the number of top-best masses interacting with the population decreases, promoting exploitation and giving an elitist approach to the algorithm. To bring some stochasticity during the force interaction process between masses 'i' and 'j', a random component is added as given in equation (11).

$$F_i^d(t) = \sum_{j=1,j\neq i}^{N} rand_j F_{ij}^d(t) \tag{11}$$

After calculating the forces between the masses, the solutions are given an acceleration according to equation (12).

$$a_i^d(t) = \frac{F_i^d(t)}{Mass_i(t)} \tag{12}$$

Once the solutions are given an acceleration, their velocities and positions can be calculated according to equations (13) and (14).

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t) \tag{13}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \tag{14}$$

The GSA outperformed PSO, real genetic algorithm (RGA), and central force optimization algorithm (CFO) in the continuous domain (Rashedi et al., 2009). However, premature convergence or stocking into sub-optimal solutions are major concerns with GSA (Joshi, 2022). The premature convergence in GSA is mainly due to the masses getting heavier as the iterations pass. As the masses get heavier, the acceleration of the agents decreases, decreasing the velocity and step size of the movement. Mirjalili and Gandomi (2017) identified the gravitational constant as the parameter responsible for premature convergence in GSA, which controls the intensity of force interaction between the masses over the iterations. The gravitational constant in the conventional GSA follows an exponential trend, which decays very fast and results in smaller step movements of the search agents in the search space, which leads to the trapping of the GSA in the local optima. To resolve the issue of premature convergence due to the gravitational constant in the GSA, Mirjalili and Gandomi (2017) embedded the ten chaotic maps to the gravitational constant to enhance the exploration power of the GSA and named the corresponding algorithm as chaotic GSA (CGSA). To enhance the exploitation power of CGSA, Mirjalili and Gandomi (2017) adopted adaptive normalization while embedding the chaotic maps to the gravitational constant. The adaptive normalization technique reduced the normalization range over the iterations, reduced the normalized value of chaotic numbers, and boosted the exploitation later in the search. Mathematically, the normalization of chaotic sequences in the range [*x y*] can be normalized into the interval [0 *a(t)*] as shown in equation (15) (Mirjalili and Gandomi, 2017).

$$chaos_{normalized}(t) = \frac{(chaos(t)-x)\times(a(t)-0)}{y-x} + 0 \tag{15}$$

In equation (15), *a(t)* is adaptive to the number of iterations, and its value changes between a particular maximum (*max*) and minimum value (*min*). Therefore, the mathematical expression for *a(t)* can be given as follows:

$$a(t) = max - \frac{t}{T}(max - min) \tag{16}$$

From equation (16), it can be confirmed that the normalization interval keeps on decreasing as time passes. The range of the chaotic sequences varies between 0 and 1, except for Chebyshev and the iterative map, which ranges between -1 and 1 (Mirjalili and Gandomi, 2017). Therefore, the gravitational constant with chaos can be represented as shown in equation (17).

$$G(t)_{chaos} = G(t) + chaos_{normalized}(t) \tag{17}$$

According to Mirjalili and Gandomi (2017), the results obtained by the sinusoidal map are better than the other maps taken in the literature since most part of the sinusoidal map range between 0.5 and 1, which gives more exploration power to the algorithm and hence solution quality improves compared to other maps. Therefore, the present work adopts a sinusoidal map to test the performance of CGSA for FJSP.

5. Encoding and Decoding

Initially, the gravitational search algorithm is designed to solve continuous problems. Therefore, to make it suitable for FJSP, an encoding and decoding technique is required so that the solution vectors give some information about the FJSP. The present work adopts a real number-based encoding method (Xia and Wu, 2005) to map the properties of GSA to make it suitable for finding the schedules for an FJSP. Since the present method uses a real-number-based encoding method, the corresponding algorithms are named as real-coded gravitational search algorithm (RGSA) and real-coded chaotic GSA (RCGSA). A sample problem of fully flexible FJSP is given in Table 1 to demonstrate the real number-based encoding process. The problem in Table 1 contains 3 jobs (J1, J2, J3) and 3 machines (M1, M2, M3) with eight operations.

Table 1 *A 3*3 Sample Problem for FJSP*

Job	Operations	M1	M2	M3
J1	$O_{1,1}$	3	4	2
	$O_{1,2}$	5	2	3
	$O_{1,3}$	3	4	2
J2	$O_{2,1}$	2	1	3
	$O_{2,2}$	3	2	1
	$O_{2,3}$	5	4	6
J3	$O_{3,1}$	7	4	1
	$O_{3,2}$	2	1	3

In a real number-based encoding method, a priority matrix is designed based on the processing time of each operation on each machine. The priority matrix for the abovementioned problem is given in Table 2.

Table 2 *Priority Matrix for 3*3 Sample Problem*

Job	Operations	Priority order		
		1	**2**	**3**
J1	$O_{1,1}$	M3	M1	M2
	$O_{1,2}$	M2	M3	M1
	$O_{1,3}$	M3	M1	M2
J2	$O_{2,1}$	M2	M1	M3
	$O_{2,2}$	M3	M2	M1
	$O_{2,3}$	M2	M1	M3
J3	$O_{3,1}$	M3	M2	M1
	$O_{3,2}$	M2	M1	M3

After generating the priority matrix, the random population is generated with dimensions equal to the number of operations. Every candidate solution's lower and upper bounds lie between 1 and the maximum number of machines. Therefore, for the problem in Table 1, the lower bound is 1, and the upper bound is 3. Hence, the possible solution vector is shown in equation (18).

$$X = [2.2511, 3.1610, 1.0003, 1.9070, 1.4403, 1.2770, 1.5588, 2.0367] \tag{18}$$

The *X* vector represents the priority of the machine to be selected for a particular operation. Therefore, the *X* vector is rounded off, and then machines are allotted to each operation according to their priority. An example of the machine assignment is given in Table 3.

Table 3 *Allotment of Machines to the Operations According to the Priority Matrix*

Operations	$O_{1,1}$	$O_{1,2}$	$O_{1,3}$	$O_{2,1}$	$O_{2,2}$	$O_{2,3}$	$O_{3,1}$	$O_{3,2}$
***X* vector**	2.2511	3.1610	1.0003	1.9070	1.4403	1.2770	1.5588	2.0367
Priority	2	3	1	1	1	1	1	2
Machines	M1	M1	M3	M2	M3	M2	M3	M1

After the machines are assigned to the operations, sequencing is done by taking the nth operation of each job separately to avoid violating the precedence constraints. For example, the first operation of each job is taken and sequenced according to the ascending order of their processing time. Similarly, the second operation of each job is sequenced according to the ascending order of the completion time of the previous operations to sequence the operations at their earliest starting time. For example, the Gantt chart of the Kacem instance with four jobs and five machines (Kacem et al., 2002) in figure 1 shows that no precedence constraints are violated, and operations are sequenced according to their earliest starting time.

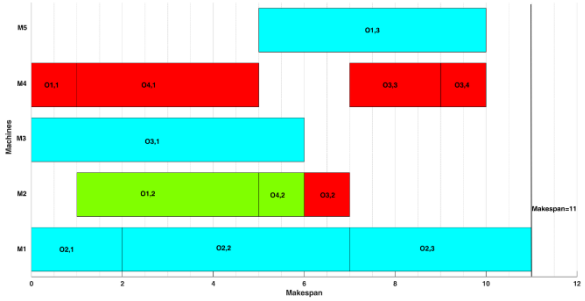

Figure 1 *Gantt Chart of the Schedule Produced by RGSA using Real Number-Based Encoding*

6. Results and Discussion

The proposed RGSA and RCGSA are tested on the following benchmark instances:

6.1 Kacem instances (Kacem et al., 2002)

The benchmark instances of Kacem et al. (2002) contain five problems, out of which four instances are fully flexible, whereas one is partially flexible. The number of jobs ranges from four to fifteen, whereas the machine number varies between five and ten.

6.2 Br instances (Brandimarte, 1993)

The 10 instances of Brandimarte (1993) are larger in size compared to Kacem's instances, with the number of jobs ranging from 10 to 20, while the number of machines varies between 4 to 15. All the instances are of medium flexibility.

6.3 Fa instances (Fattahi et al., 2007)

The 20 instances of Fattahi et al. (2007) are divided into ten small-scale and 10 medium-sized problems. The total instances contain problems that are both partially flexible and fully flexible.

Since FJSPs are computationally expensive, the population size is kept at 10 while the maximum number of iterations is maintained at 100. To analyze the performance of RGSA and RCGSA on the benchmark instances, the algorithms are run 10 times independently. The mean and standard deviation (std), along with the best values, are reported for the initial assessment of the algorithms. The size of each instance is mentioned with its lower bound (*lb*) or best-known solution obtained by any algorithm. A relative error (RE) is calculated using the expression given in equation (19) to measure the deviation of the best values obtained by the algorithms from the lower bound.

$$RE = \frac{C_{max} - lb}{lb} \tag{19}$$

In equation (19), *RE* is the relative error, C_{max} is the best makespan obtained by RGSA or RCGSA, and *lb* is the lower bound or the best-known solution obtained by any algorithm. Apart from *RE*, mean relative error (mean_*RE*) is also reported for each benchmark set of instances for comparison purposes. All the experiments were done on MATLAB R2021b with a computer system with an i7 processor and 10 GB RAM. The results of Fa instances are given in Table 4. The results of the Fa instances in Table 4 show that the performance of RCGSA is better than that of RGSA due to the enhanced exploration rate of RCGSA. It can be noted from Table 4 that as the size of the instance increases, the *RE* value increases for both RGSA and RCGSA. However, the mean_*RE* value of RCGSA is better than that of RGSA.

Table 4 *Results of RGSA and RCGSA for Fa Instances*

Problem	**Size**	***lb***	**RGSA**				**RCGSA**			
			Best	**Mean**	**std**	***RE***	**Best**	**Mean**	**std**	***RE***
Fa1	2*2	66	**66**	**66**	0	0	**66**	**66**	0	0
Fa2	2*2	107	**107**	**107**	0	0	**107**	**107**	0	0
Fa3	3*2	221	**221**	224.3	5.558	0	**221**	**221.5**	4.743	0

Fa4	3*2	355	**339**	342.1	6.539	-0.045	**339**	**341.2**	4.638	-0.045
Fa5	3*2	119	**119**	**119**	0	0	**119**	**119**	0	0
Fa6	3*3	320	**320**	**320.7**	2.213	0	**320**	**320.7**	2.213	0
Fa7	3*5	397	**397**	**397**	0	0	**397**	**397**	0	0
Fa8	3*4	253	**216**	**227.1**	10.587	-0.146	**216**	228.8	12.847	-0.146
Fa9	3*3	210	**210**	**212**	4.216	0	**210**	213	4.216	0
Fa10	4*5	516	**466**	491.7	17.575	-0.096	478	**490**	12.328	-0.073
Fa11	5*6	396	457	490.9	39.948	0.154	**447**	**472.2**	23.827	0.128
Fa12	5*7	396	457	468.8	12.788	0.154	**436**	**460.4**	16.153	0.101
Fa13	6*7	396	499	555.4	31.081	0.260	**491**	**533.2**	31.657	0.239
Fa14	7*7	496	**561**	676.7	65.892	0.115	576	**645.1**	38.118	0.161
Fa15	7*7	414	598	639.8	22.972	0.444	**549**	**601.8**	27.848	0.326
Fa16	8*7	469	**646**	**700.2**	30.684	0.377	649	702.7	44.136	0.383
Fa17	8*7	619	925	1012.2	82.221	0.494	**859**	**995.6**	117.833	0.387
Fa18	9*8	619	1038	1321.6	218.494	0.676	**925**	**1104.9**	209.370	0.494
Fa19	11*8	764	1393	1997.8	310.732	0.823	**1170**	**1465.5**	194.577	0.531
Fa20	12*8	944	1717	2453.3	525.979	0.818	**1508**	**1936.5**	475.876	0.597
mean_*RE*						0.2014				**0.1545**

The results of Kacem instances are given in Table 5. From Table 5, it can be deduced that RGSA and RCGSA are optimal for Kacem1. However, an increase in the instance size deteriorated the performance of both RGSA and RCGSA. However, RCGSA provides better results than RGSA for each instance in terms of the mean and *RE* values.

Table 5 *Results of RGSA and RCGSA for Kacem Instances*

Problem	**Size**	***lb***	**RGSA**				**RCGSA**			
			Best	**Mean**	**std**	***RE***	**Best**	**Mean**	**std**	***RE***
Kacem1	4*5	11	**11**	12.9	1.100	0	**11**	**12.1**	1.100	0
Kacem2	8*8	14	20	25.4	3.627	0.428	**19**	**21.5**	2.173	0.357
Kacem3	10*7	11	21	25.4	3.062	0.909	**17**	**22.4**	3.777	0.545
Kacem4	10*10	7	19	21.6	1.955	1.714	**14**	**18.7**	2.406	1
Kacem5	15*10	11	31	35.1	2.806	1.818	**27**	**30.6**	2.988	1.454
Mean_*RE*						0.9738				**0.6712**

The results of Br instances are given in Table 6. Since Br instances are larger in size compared to Fa instances and Kacem instances, *RE* produced by RGSA is significantly higher. Although RCGSA maintains better mean values than RGSA, the best values obtained are still far from lb, as indicated by *RE* values.

Table 6 *Results of RGSA and RCGSA for Br Instances*

Problem	**Size**	***lb***	**RGSA**				**RCGSA**			
			Best	**Mean**	**std**	***RE***	**Best**	**Mean**	**std**	***RE***
Br1	10*6	36	49	55.6	5.910	0.361	**44**	**46.7**	2.311	0.222
Br2	10*6	24	39	43.9	3.695	0.625	**33**	**36.7**	1.946	0.375
Br3	15*8	204	457	481.7	19.471	1.240	**393**	**440.3**	26.348	0.926
Br4	15*8	48	184	203.8	17.806	2.833	**155**	**188.5**	19.642	2.229
Br5	15*4	168	228	265.5	21.593	0.357	**205**	**224.9**	13.698	0.220
Br6	10*15	33	207	220.8	9.390	5.272	**151**	**196**	20.725	3.575
Br7	20*5	133	190	226.6	29.993	0.428	**165**	**188.9**	26.530	0.240
Br8	20*10	523	1996	2074.7	48.327	2.816	**1805**	**1950**	91.057	2.451
Br9	20*10	299	1642	1720.9	50.034	4.491	**1509**	**1623.6**	61.339	4.046
Br10	20*15	165	1376	1406.7	20.028	7.339	**1360**	**1396.3**	21.234	7.242
mean_*RE*						2.57				**2.15**

The best and mean fitness curves are plotted for Kacem instances in figure 2 to track the search history of RGSA and RCGSA.

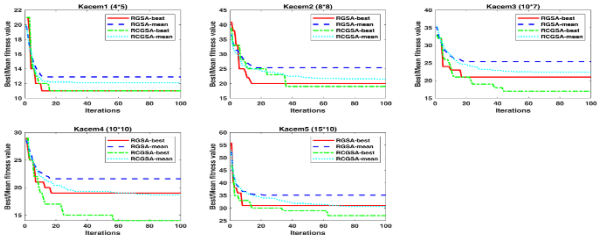

Figure 2 *Convergence Curves of RGSA and RCGSA for Kacem Instances*

Figure 2 shows that due to the enhanced exploration power of RCGSA, it reaches a better solution in a limited number of iterations than RGSA.

6.4 Performance Evaluation of the RGSA and RCGSA on An Industrial Test Case

An industrial problem from Xu et al. (2024) is taken to analyze the performance of RGSA and RCGSA in real-world test cases. The problem statement demands an optimal schedule for machining a hydraulic gear pump with six components and six machining centers. The detailed information of the problem, such as the number of operations, processing time, and flexibility, can be obtained from Xu et al. (2024). The number of components is increased by a multiple of six to analyze the impact of scalability on the performance of the algorithms. The results for the industrial test case are given in Table 7.

Table 7 *Results of RGSA and RCGSA for An Industrial Test Case*

Problem	Size	*lb*	RGSA				RCGSA			
			Best	Mean	std	*RE*	Best	Mean	std	*RE*
Problem1	6*6	56	**65**	75.3	5.498	0.160	66	**74.3**	4.164	0.178
Problem2	12*6	92	120	130.6	8.971	0.304	**109**	**125.3**	9.809	0.184
Problem3	18*6	135	184	194.6	9.489	0.362	**161**	**173.3**	12.202	0.192
Problem4	24*6	180	230	270.9	26.074	0.277	**207**	**232.8**	19.611	0.15
Problem5	30*6	225	324	391.2	46.913	0.44	**251**	**292.4**	21.706	0.115
Problem6	36*6	267	403	479.9	37.793	0.509	**328**	**401.8**	39.563	0.228
Mean_*RE*						0.342				**0.1745**

From Table 7, it is evident that the search capability of RCGSA is better than that of GSA due to the chaotic behaviour embedded in it. The best and mean values of the algorithms are degrading as the problem size is scaling. However, the *RE* values produced by RCGSA are far better than those produced by RGSA. The convergence behaviour of the algorithms is analyzed by plotting the best and mean fitness curves against the number of iterations, as shown in figure 3. Except for problem 1, the RCGSA can achieve a better solution faster than RCGSA in fewer iterations. Convergence analysis proves that RCGSA better establishes the exploration and exploitation trade-off.

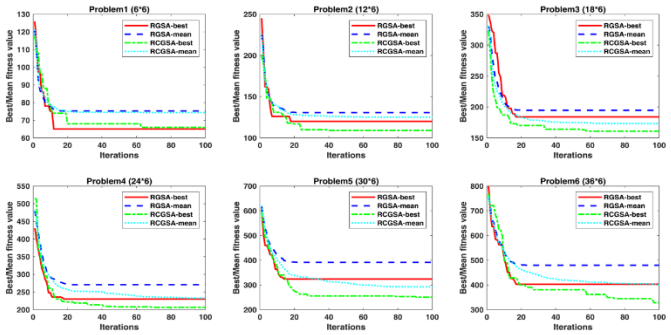

Figure 3 *Convergence Curves of RGSA and RCGSA for an Industrial Test Case*

A Wilcoxon's signed rank test is done to have a pair-wise comparison for each problem to statistically analyze the best and mean values obtained by RGSA and RCGSA. The R^+ (sum of positive ranks), R^- (sum of negative ranks), and p-values are reported in Table 8 for statistical validation of the performance of RCGSA against RGSA. From Table 8, it can be confirmed that RCGSA is significantly better than RGSA while considering both the best and mean values. For the best values, RCGSA outperforms RGSA 27 times out of 41, whereas 33 times, the average performance of RCGSA is better than RGSA. The p-value indicates significant improvement while combining chaos with GSA for FJSPs.

Table 8 *Results of Wilcoxon's Signed Rank Test for best and Mean Values Obtained by RGSA and RCGSA*

Algorithm	Winner	Ties	R^+	R^-	p-value
RCGSA_best vs RGSA_best	27	10	466.50	29.50	1.80e-05
RCGSA_mean vs RGSA_mean	33	5	649	17	6.88e-07

7. Conclusion

The present work evaluates the performance of GSA and chaotic GSA (CGSA) for FJSP using a real number-based encoding method, and the corresponding algorithms are named RGSA and RCGSA. The performance of the algorithms is analyzed on 35 benchmark instances and 6 real-world test cases. It is observed that the RGSA is well-suited for small-scale instances. However, as the size of an instance increases, the relative error (*RE*) from the lower bound increases in case of RGSA, but RCGSA maintains a better value for almost every case compared to RGSA. The significant improvement in the performance of the RCGSA, as proved by Wilcoxon's signed rank test, is due to the better exploration-exploitation characteristics than RGSA due to the presence of the sinusoidal chaotic sequences. Furthermore, the convergence analysis proves that RCGSA

converges to a better solution in a shorter time than RGSA. Moreover, the scope of the present work can be extended to improve the performance of RCGSA further by incorporating some local search methods based on the critical paths.

8. References

1. Elnadi, M., and Abdallah, Y. O., (2024), 'Industry 4.0: critical investigations and synthesis of key findings', Management Review Quarterly, Vol.74(2), 711-744.
2. Brucker, P., and Schlie, R., (1990), 'Job-shop scheduling with multipurpose machines', Computing.
3. Roshanaei, V., Azab, A., and ElMaraghy, H., (2013), 'Mathematical modelling and a meta-heuristic for flexible job shop scheduling', International Journal of Production Research, Vol.51(20), 6247-6274.
4. Jain, A. S., and Meeran, S., (1999), 'Deterministic job-shop scheduling: Past, present and future', European journal of operational research, Vol.113(2), 390-434.
5. Xie, J., Gao, L., Peng, K., Li, X., and Li, H., (2019), 'Review on flexible job shop scheduling', IET collaborative intelligent manufacturing, Vol.1(3), 67-77.
6. Dauzère-Pérès, S., Ding, J., Shen, L., and Tamssaouet, K. (2024), 'The flexible job shop scheduling problem: A review', European Journal of Operational Research, Vol.314(2), 409-432.
7. Brandimarte, P., (1993), 'Routing and scheduling in a flexible job shop by tabu search', Annals of Operations research, Vol.41(3), 157-183.
8. Dauzère-Pérès, S., and Paulli, J., (1997), 'An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search', Annals of Operations Research, Vol.70(0), 281-306.
9. Hurink, J., Jurisch, B., and Thole, M., (1994), 'Tabu search for the job-shop scheduling problem with multi-purpose machines', Operations-Research-Spektrum, Vol. 15, 205-215.
10. Fattahi, P., Saidi Mehrabad, M., and Jolai, F., (2007), 'Mathematical modeling and heuristic approaches to flexible job shop scheduling problems', Journal of intelligent manufacturing, Vol. 18, 331-342.
11. Yazdani, M., Amiri, M., and Zandieh, M., (2010), 'Flexible job-shop scheduling with parallel variable neighbourhood search algorithm', Expert Systems with Applications, Vol.37(1), 678-687.
12. Lei, D., (2010), 'A genetic algorithm for flexible job shop scheduling with fuzzy processing time', International Journal of Production Research, Vol.48(10), 2995-3013.
13. Pezzella, F., Morganti, G., and Ciaschetti, G., (2008), 'A genetic algorithm for the flexible job-shop scheduling problem', Computers & operations research, Vol.35(10), 3202-3212.
14. Xue, L., Zhao, S., Mahmoudi, A., and Feylizadeh, M. R., (2024), 'Flexible job-shop scheduling problem with parallel batch machines based on an enhanced multi-population genetic algorithm', Complex & Intelligent Systems, Vol.10(3), 4083-4101.
15. Zhang, Y., Zhu, H., and Tang, D., (2020), 'An improved hybrid particle swarm optimization for multi-objective flexible job-shop scheduling problem', Kybernetes, Vol.49(12), 2873-2892.
16. Zarrouk, R., Bennour, I. E., and Jemai, A., (2019), 'A two-level particle swarm optimization algorithm for the flexible job shop scheduling problem', Swarm Intelligence, Vol. 13, 145-168.
17. Liu, Z., Wang, J., Zhang, C., Chu, H., Ding, G., and Zhang, L., (2021), 'A hybrid genetic-particle swarm algorithm based on multilevel neighbourhood structure for flexible job shop scheduling problem', Computers & Operations Research, Vol. 135, 105431.
18. Xu, Y., Zhang, M., Yang, M., and Wang, D., (2024), 'Hybrid quantum particle swarm optimization and variable neighborhood search for flexible job-shop scheduling problem', Journal of Manufacturing Systems, Vol.73, 334-348.\
19. Jiang, B., Ma, Y., Chen, L., Huang, B., Huang, Y., and Guan, L., (2023), 'A Review on Intelligent Scheduling and Optimization for Flexible Job Shop', International Journal of Control, Automation and Systems, Vol.21(10), 3127-3150.
20. Chen, R., Yang, B., Li, S., and Wang, S., (2020), 'A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem', Computers & industrial engineering, Vol. 149, 106778.
21. Rashedi, E., Nezamabadi-Pour, H., and Saryazdi, S., (2009), 'GSA: a gravitational search algorithm', Information sciences, Vol.179(13), 2232-2248.
22. Joshi, S. K., (2023), 'Chaos embedded opposition based learning for gravitational search algorithm', Applied Intelligence, Vol.53(5), 5567-5586.
23. Mirjalili, S., & Gandomi, A. H., (2017), 'Chaotic gravitational constants for the gravitational search algorithm', Applied soft computing, Vol. 53, 407-419.
24. Xia, W., & Wu, Z., (2005), 'An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems', Computers & industrial engineering, Vol.48(2), 409-425.
25. Kacem, I., Hammadi, S., & Borne, P., (2002), 'Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems', IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), Vol.32(1), 1-13.