APEM journal
Advances in Production Engineering & Management
Volume 15 | Number 4 | December 2020 | pp 377–389
https://doi.org/10.14743/apem2020.4.372
ISSN 1854-6250
Journal home: apem-journal.org
Original scientific paper

A layered genetic algorithm with iterative diversification for optimization of flexible job shop scheduling problems

Amjad, M.K.[a,*], Butt, S.I.[a], Anjum, N.[a], Chaudhry, I.A.[b], Faping, Z.[c], Khan, M.[a]

[a]School of Mechanical and Manufacturing Engineering, National University of Sciences and Technology, Islamabad, Pakistan
[b]Department of Industrial Engineering, College of Engineering, University of Ha'il, Kingdom of Saudi Arabia
[c]School of Mechanical Engineering, Beijing Institute of Technology, Beijing, P.R. China

ABSTRACT

Flexible job shop scheduling problem (FJSSP) is a further expansion of the classical job shop scheduling problem (JSSP). FJSSP is known to be NP-hard with regards to optimization and hence poses a challenge in finding acceptable solutions. Genetic algorithm (GA) has successfully been applied in this regard since last two decades. This paper provides an insight into the actual complexity of selected benchmark problems through quantitative evaluation of the search space owing to their NP-hard nature. A four-layered genetic algorithm is then proposed and implemented with adaptive parameters of population initialization and operator probabilities to manage intensification and diversification intelligently. The concept of reinitialization is introduced whenever the algorithm is trapped in local minima till predefined number of generations. Results are then compared with various other standalone evolutionary algorithms for selected benchmark problems. It is found that the proposed GA finds better solutions with this technique as compared to solutions produced without this technique. Moreover, the technique helps to overcome the local minima trap. Further comparison and analysis indicate that the proposed algorithm produces comparative and improved solutions with respect to other analogous methodologies owing to the diversification technique.

ARTICLE INFO

Keywords:
Scheduling;
Flexible job shop scheduling problem (FJSSP);
Complexity;
Diversity;
Combinatorial optimization;
Genetic algorithm

**Corresponding author:*
kamal.amjad@smme.edu.pk
(Amjad, M.K.)

Article history:
Received 19 May 2020
Revised 27 November 2020
Accepted 30 November 2020

1. Introduction

Modern manufacturing processes consist of several carefully planned sub-processes that require to be completed in a predefined manner to achieve the intended product [1]. With the continuously changing business flux and fluctuating product demand, it is imperative that manufacturing flexibility may be added to the shop floor so that a maximum variety of operations can be performed. Flexible manufacturing systems (FMS) can handle a great deal of product variety with reasonable volumes as they are capable of variable routing among different workstations. Modern concepts of group technology (GT) and cellular manufacturing have specifically been designed to incorporate flexibility in the manufacturing process [2]. The decision making regarding allocation of tasks or set of activities to available resources is termed as scheduling [1, 3]. Optimum utilization of resources is possible if the tasks are efficiently performed according to predefined criteria. Since many sequences can be executed for a said product, many schedules can be developed. Optimization of schedules is conventionally evaluated through benchmark problems against a predefined cost function. The cost function, in this case, is scheduling an objective that is used to assess the optimality of the generated schedule concerning the said objective.

Job shops are popular because they can handle a variety of processes in a single facility [4], thereby offering a significant advantage over classical product or process based layouts. They can handle the different sequences of operations on various fixed machines. The flexible job shop (FJS) offers flexibility in the job shop through the introduction of flexible machines whereby required operations may be performed on several alternative machines [5]. Therefore, the problem can be decomposed as an assignment and scheduling problem. Additionally, cases of partial and total flexibility are also formulated, whereby all operations can be performed on all machines in the case of total flexibility and only some operations can be performed on several machines in the case of partial flexibility [6]. Whereas this extension has provided effective resource utilization and added manufacturing flexibility, it also has increased the complexity of scheduling optimization manifold. Therefore, flexible job shop scheduling problem (FJSSP) has been studied in a dedicated and detailed manner owing to its NP-hard nature [7, 8] and complexity [9].

It has been reported that there are $(n!)^m$ possible sequences for generating a schedule for n jobs on m machines in case of a JSSP. Consequently, the computational resources either expire or become scarce when attempting large instances of problems. Where FJSSP offers more flexibility in terms of assignment, the problem becomes more complex as an additional layer of alternatives is available in this case. Thus, exact solution methods are seldom attempted [10, 11] and artificial intelligence-based approaches have gained extreme popularity. Although, various dedicated case studies of FJSSP have been reported in [12], traditionally, benchmark problems are used to test the developed algorithms [13, 14].

Schedules are developed keeping in view a certain objective. Thus, a schedule meant for optimum use of resources may not be applicable for optimum workload minimization. Many objectives have been addressed in literature with regards to FJSSP; however, makespan has been addressed the most [15], which is the maximum time required to complete all operations of the selected dataset.

FJSSP is one of the most challenging optimization problems [16]. To achieve optimum solutions in a reasonable time, meta-heuristic algorithms have gained tremendous popularity. Whereas, a lot of studies have been conducted using various algorithms; Genetic algorithm (GA) has gained outstanding attention in this regard [17]. In a comprehensive study considering various approaches for solving FJSSPs, it has been pointed out that GA is the most popular algorithm with publications amounting to 34 % [18]. It has also been pointed out that 26.4 % of studies performed on FJSSPs have been conducted using GA [19].

GA mimics the phenomena of human evolution based on the "survival of the fittest" rule [20]. It provides an effective mechanism to conduct a directed random search for finding optimal solutions and therefore it has used effectively for sequencing problems in flexible manufacturing systems, gaining exceptional popularity in the last decade [15]. An approach by localization along with benchmark problems of FJSSP with total flexibility was presented [21]. Goa *et al.* [22] addressed the FJSSP with multi-objective optimization and proposed GA for the solution of selected benchmarks. Similarly, Pezzella *et al.* [23] suggested a GA with various strategies for algorithm improvement. Gu *et al.* [24] presented an improved GA with a hybrid population initialization method.

Population diversity plays an extremely important role in solution quality. Traditionally, crossover and mutation operators have been used to introduce diversity within the principles. A cluster of population at a local minimum enforces the algorithm to converge prematurely. Alternatively, an extremely diverse population may not allow the algorithm to converge. Therefore, population diversity on one side provides solution quality, while on the other side; it may allow the algorithm to run for long periods. Research has thus been carried out in order to propose mechanisms to attain a balance between these two different ends. Wang *et al.* [25] have introduced a population diversity technique through the conservation of a single elitist solution. Xiong *et al.* [26] introduced a crowding distance measure to ensure population diversity. Teekeng *et al.* [27] proposed a modified version of mutation to ensure population diversity.

A pure GA based approach is presented in the current study to solve the FJSSP. The diversification and intensification regimes are used side-by-side to increase the capability of the algorithm to further explore the search space while conserving the best solutions and deferring premature convergence. When local minima are encountered after several iterations, re-initialization is invoked. The algorithm is then implemented and tested on selected standard benchmark problems of FJSSP. An in-depth analysis of the algorithm efficacy for FJSSP is then discussed and conclusions are presented.

2. Problem formulation and complexity

The FJSSP is formulated as a set of N jobs ($J = J_1, J_2, J_3,\dots, J_N$) to be processed on M machines ($M = M_1, M_2, M_3,\dots, M_M$). Each job J_i consists of predefined operations O_{ij} to be processed on any of the available machines, where O_{ij} is the operation j of job i. The processing time required for completion of operation O_{ij} on machine M_k is a known aspect termed as P_{ijk}. As number of operations for each job may differ, the total number of operations for all operations is as follows:

$$L = \sum_{i=1}^{N} J_{io} \tag{1}$$

where J_{io} is the total number of operations for a single job J_i. Accordingly, a sequence may be assigned to an operation O_{ij} such that:

$$n_{ij} = \sum_{x=1}^{i-1} J_{xo} + j \tag{2}$$

where n_{ij} is the sequence number for a said operation. The operations can be scheduled on any machine depending upon the condition that the previous operation is complete, and the machine is available at the said time. Other assumptions are formulated below [18].

i. All resources are available at time $t = 0$:

$$t_{ijk} \geq 0 \qquad O_{ijk} \,\epsilon\, N \tag{3}$$

$$r_{ijk} \geq 0 \qquad \forall\, O_{ijk} \,\epsilon\, N \tag{4}$$

ii. Only one operation can be performed on one machine at a provided time:

$$t_{ijk} - t_{i'j'k} \geq P_{i'j'k} \qquad \forall \left(O_{ij}, O_{i'j'}\right) \epsilon\, S_k \tag{5}$$

iii. Operations are performed in a predefined order:

$$t_{ijk} - t_{ij'k'} \geq P_{ij'k'} \qquad \forall \left(O_{ijk}, O_{ij'k'}\right) \epsilon\, J_{io} \tag{6}$$

iv. Operations are not interrupted once they are started, i.e. pre-emption or cancellation is not considered:

$$E_{ijk} - t_{ijk} = P_{ijk} \qquad \forall\, O_{ij} \epsilon J_{io} \;\&\; M_k \,\epsilon\, M \tag{7}$$

v. There is no free time between any two operations

$$t_{ijk} = \max\left(C_k, r_{ijk}\right) \tag{8}$$

Where, t_{ijk} is the start time of O_{ij} on machine M_k, r_{ijk} is the release time of O_{ij} on machine M_k, $t_{i'j'k}$ is the start time of the previous operation on machine M_k, $P_{i'j'k}$ is the processing time of the previous operation on machine M_k and E_{ijk} is the end time of O_{ij} on machine M_k. Additionally, all jobs have equal priorities and the setup times are either zero or considered in the operation time.

FJSSP is one of the most challenging combinatorial optimization problems. Even for the simpler JSSP, the computational effort increases in an exponential manner with the increase of problem size and computational time for an exact solution may rise to millions of years [28]. The actual depiction of problem complexity lies in the evaluation of search space. The size of the search space depends upon the chromosome length and level of flexibility U_{ij} of the problem. Henceforth, changing the length or definition affects the search space. In every gene, there is an upper bound U_{ij}, i.e. the number of machines on which a said operation can be performed. According to the chromosome definition of Zhang *et al*. [29], search space (SS) is the combination of all possible values of a gene. The same is formulated as Eq. 9.

$$SS = SS(MS) \cdot SS(OS) = \prod_{i=1}^{N} \prod_{j=1}^{J_{io}} U_{ij} \cdot \frac{L!}{\prod_{i=1}^{N} J_{io}!} \tag{9}$$

Where $SS(MS)$ and $SS(OS)$ are the search space for machine selection (MS) and operation selection (OS) parts of the chromosome respectively. For the MS part, SS is all possible valid combinations for all genes, i.e. product of U_{ij} for all operations. For the OS part, SS is the ratio of OS part chromosome length and product of all $J_{io}!$. The computations of search space for Fattahi [30] and Kacem [21] are presented in Table 1 and graphically shown in Fig. 1. Here, M, N, and L represent the total number of jobs, machines, and sequences, respectively. It is evident that the search space size increases manifold in an exponential manner. This depiction of problem complexity has not been attempted before.

Table 1 Search space for Kacem and Fattahi datasets

Instance	N	M	L	SS	Instance	N	M	L	SS
SFJS1	2	2	4	9.60E+01	MFJS3	6	7	18	4.67E+18
SFJS2	2	2	4	2.40E+01	MFJS4	7	7	21	1.12E+23
SFJS3	3	2	6	1.44E+03	MFJS5	7	7	21	7.45E+22
SFJS4	3	2	6	1.44E+03	MFJS6	8	7	24	1.81E+27
SFJS5	3	2	6	5.76E+03	MFJS7	8	7	32	3.00E+36
SFJS6	3	3	9	1.08E+05	MFJS8	9	8	36	2.82E+42
SFJS7	3	5	9	8.60E+05	MFJS9	11	8	44	1.35E+55
SFJS8	3	4	9	8.60E+05	MFJS10	12	8	48	6.28E+61
SFJS9	3	3	9	8.60E+05	Kacem 1	4	5	12	6.77E+13
SFJS10	4	5	12	9.46E+07	Kacem 2	10	7	29	1.41E+48
MFJS1	5	6	15	1.39E+13	Kacem 3	10	10	30	4.39E+54
MFJS2	5	7	15	2.12E+14	Kacem 4	15	10	56	2.03E+112

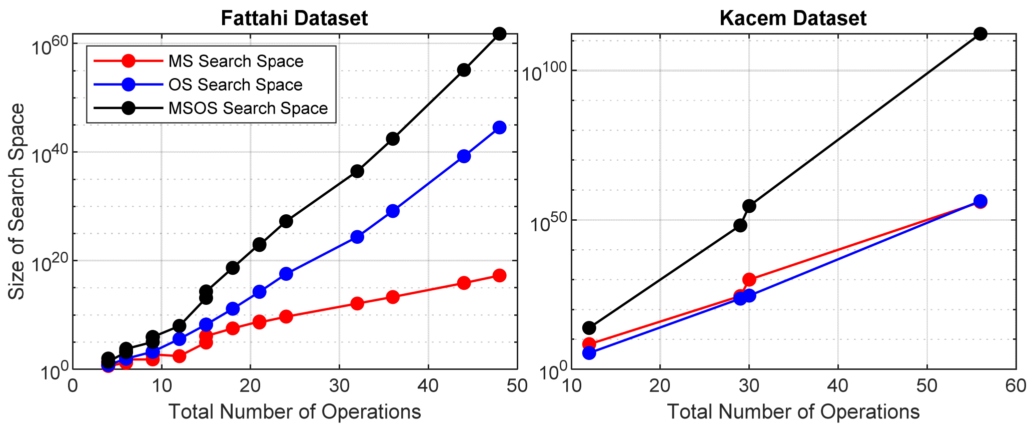

Fig. 1 Problem size vs. search space

3. Proposed algorithm

3.1 Methodology

To explore the problem search space sufficiently, crossover and mutation operators are introduced in GA [6]. These operators bring diversity in the population through exchanging information between individual chromosomes.

At the initialization phase of the algorithm, the population is fairly diverse as the algorithm generates initial chromosomes on a random basis. When convergence is achieved after certain generations in a GA routine, the population tends to have a large amount of better solutions and variance of the population therefore decreases. Where the convergence is an indication of the best solution, there is also a known tendency of GA to get trapped in local minima. Literature proposes to run the algorithm for extended generations [23] or employ local search techniques to overcome this issue [24]. Although both techniques have proven to be successful in obtaining acceptable solutions, they have their inherent drawbacks. Extended generations come at the cost of computational effort and time. Similarly, local search not only increases the algorithm complexity with regards to implementation but also increases the computational effort. This study proposes a Re-initialization based genetic algorithm (R-GA) to overcome these observations. Fig. 2 presents an overall information exchange four-layer scheme followed in R-GA. A detailed description of these layers is presented in subsequent sections.

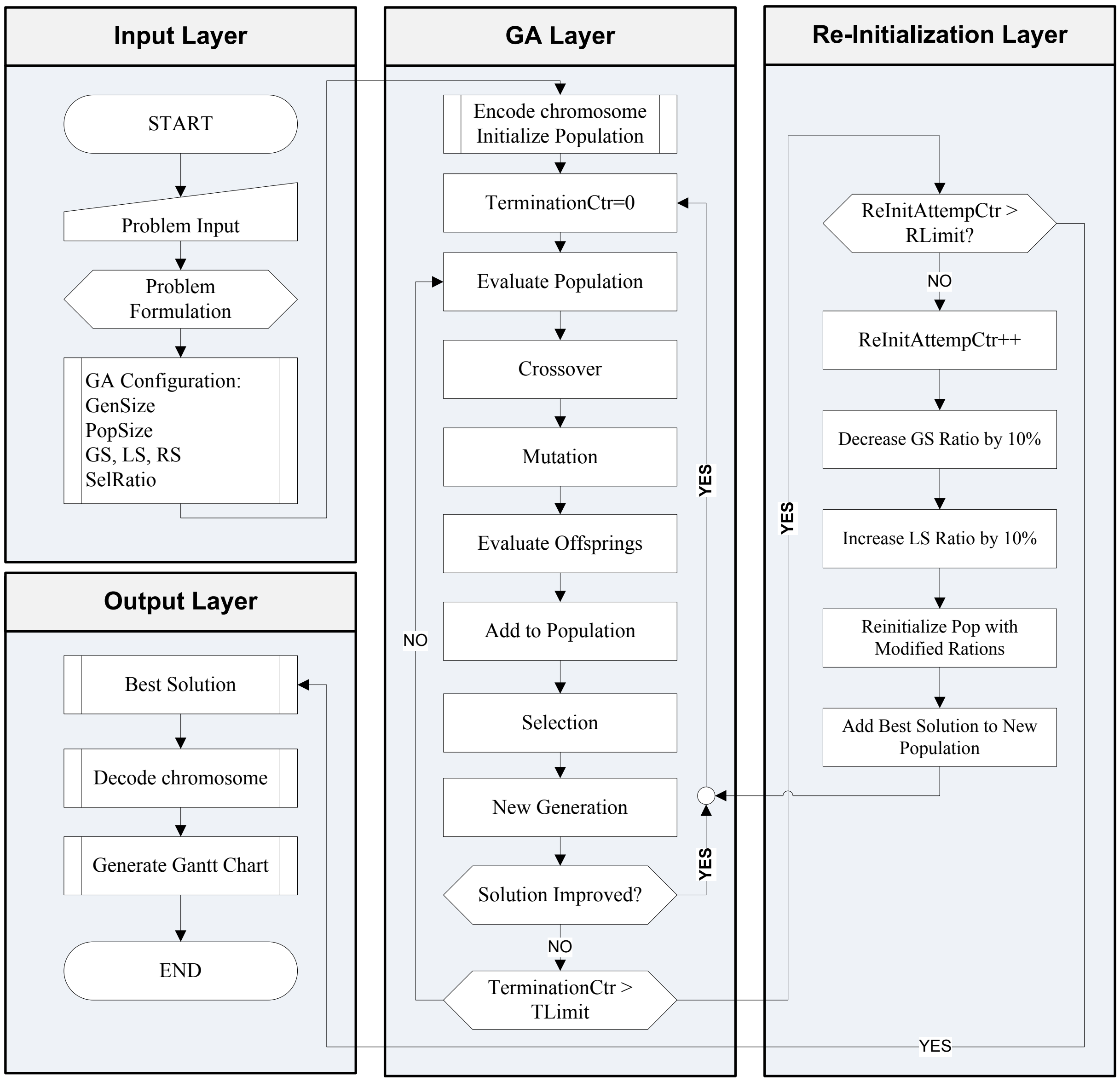

Fig. 2 Flowchart of proposed algorithm (R-GA)

3.2 Input layer

The input layer takes various pre-defined parameters from the user as input to GA. The values of parameters taken for the current implementation and testing is listed in Table 2. The adaptability of applicable parameters is explained in relevant sections.

Table 2 Input parameters for R-GA

Parameter	Description	Value
Population size	Total chromosomes in a population	1500
Generation size	Number of iterations in GA	250
Crossover probability	Likelihood for chromosome crossover	Adaptive
Mutation probability	Likelihood for chromosome mutation	Adaptive
Global selection ratio	Population initialization factor for global selection	Adaptive
Local selection ratio	Population initialization factor for local selection	Adaptive
Random selection ratio	Population initialization factor for random selection	Adaptive
Elitism ratio	Elite chromosome selection factor	20
Roulette wheel ratio	Factor for roulette wheel selection	80
Termination counter limit	Limit for GA before re-initialization	100
Re-initialization counter limit	Number of re-initialization attempts	4

3.3 GA layer

This layer contains the implementation of GA. The chromosome representation of machine selection (*MS*) and operation selection (*OS*) as proposed by Zhang *et al.* [29] is used in this study. This representation avoids generating infeasible chromosomes during the evolution process and used a single chromosome for handling the routing and scheduling aspects.

MS vector represents the machine number allocated for any operation O_{ij} out of all available machines. In *OS* vector, every operation is represented by its job number *i* and the schedule is decoded by the sequence of these numbers. Consider the problem presented in Table 3 consisting of 4 jobs that are to be scheduled on 4 machines. The candidate solution chromosome can be represented in MSOS format as shown in Fig. 3. The job, operation and machine routing have been color-coded to elaborate the representation, e.g. in the fifth gene of MS vector, O_{31} can be assigned on M_1, M_2 and M_4 however gene value 2 indicates that it has been assigned on 2nd machine from all available set of machines, i.e. M_2. Similarly, for OS vector, the first gene value 3 indicates that O_{31} will be scheduled first; second gene value 1 indicates that O_{11} will be scheduled second and so on.

The population initialization is important since it determines the quality of solutions in the search space. The global selection (*GS*), local selection (*LS*) and random selection (*RS*) are used in this study for the *MS* part [29]. Initially, *GS* is taken as 50, *LS* is taken as 20 and *RS* is taken as 30. For the OS part, the population is generated through random selection.

Table 3 A Sample FJSSP

Job	Operation	Machine			
		M_1	M_2	M_3	M_4
J_1	O_{11}	∞	10	∞	20
	O_{12}	25	30	10	∞
	O_{13}	15	60	15	10
J_2	O_{21}	15	∞	50	∞
J_3	O_{31}	80	10	∞	50
	O_{32}	∞	80	10	10
J_4	O_{41}	20	60	10	10
	O_{42}	20	∞	∞	∞

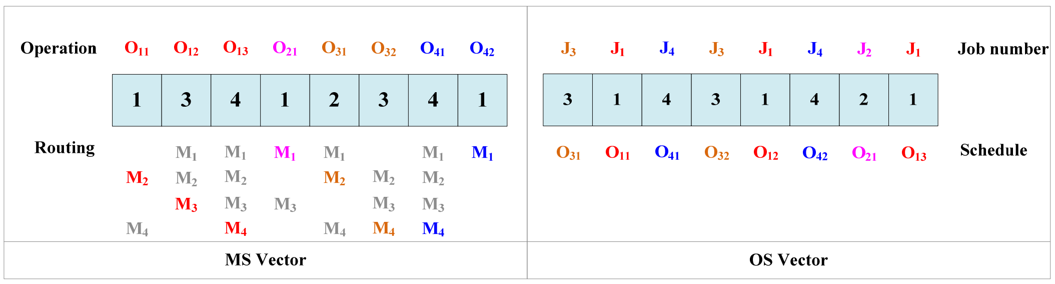

Fig. 3 An illustration of MSOS representation

Three different crossover methods have been used for the *MS* part. In the single point crossover (SPX) method [31], a random number *r* is generated in the range [1, *L*] and offspring are then generated by swapping genes [1, *r*] and [*r*, *L*] of two parents. In the two point crossover (TPX) method [32], two random numbers are generated in the range [1, *L*] and the *MS* chromosome is divided into three parts, i.e. [1, r_1], [r_1, r_2] and [r_2, *L*]. Offspring are then generated by swapping the resultant three parts. In uniform crossover (UX) method [29], even and odd number of genes of parent chromosomes is swapped to generate offspring.

The crossover of the OS vector is delicate due to its precedence order constraints. To avoid generating invalid chromosomes and preserving the scheduling constraints in OS vector, Improved precedence order crossover (iPOX) technique has been used [33] as shown in Fig. 4. This procedure generates two sets of jobs, i.e. J_{s1} and J_{s2} and then uses these sets to generate offspring from parent chromosomes. In this study, we generated job sets J_{s1} and J_{s2} by generating a random integer in the range [1, *n*]. Job sets are then generated as J_{s1} = [1, *r*] and J_{s2} = [*r*, *n*].

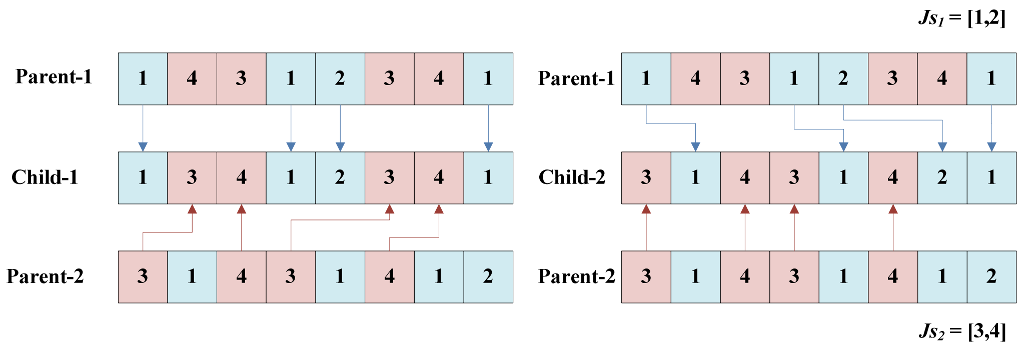

Fig. 4 Improved precedence order crossover

Conventionally, crossover probability (P_c) is kept at a predefined value, normally in the range of 0.8-0.9. In this study, P_c is adaptive in nature as formulated in Eq. 10. This increases the crossover probability as convergence is achieved, hence introducing the possibility for maximum diversification.

$$P_c = \frac{\overline{Ft}}{\max Ft} \tag{10}$$

Here, *Ft* and $\overline{Ft}$ is the overall and mean fitness of the entire population under consideration, respectively.

Random intelligent mutation has been used for the *MS* part. A random number r_1 is generated in the range [1, *L*]. A mutation is then performed at r_1 gene. To mutate r_1 gene, another random number generated in the range between one and the total number of machines on which operation at gene r_1 can be performed. The value of r_1 gene is then replaced with r_2. This procedure does not allow generation of infeasible chromosome and ensures generation of new child by restricting duplication of parent chromosomes. For the OS part, swap mutation has been used as proposed in [33]. Both procedures are presented in Fig. 5.

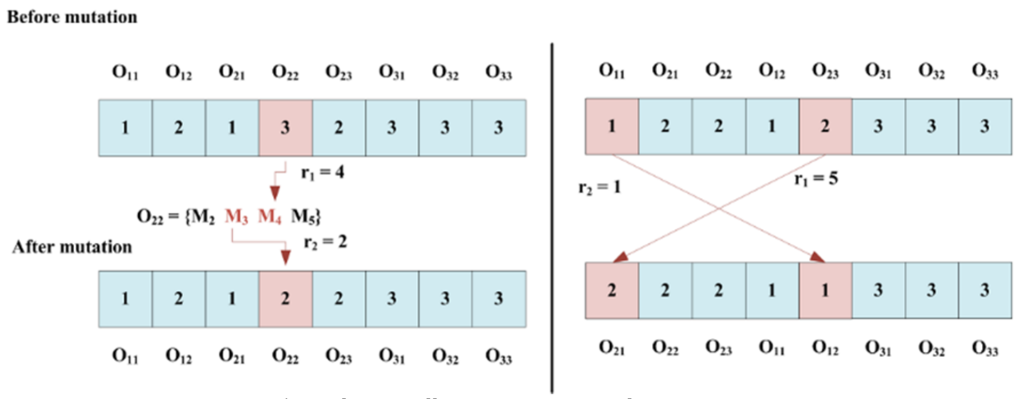

Fig. 5 Random intelligent mutation and swap mutation

Mutation probability (P_m) is generally kept at a lower side, ranging from 0.1-0.25. In this study, P_m is adaptive in nature as governed by Eq. 11. As the population is converged, P_m is increased as the values of minimum and maximum fitness comes close to each other. This further enhances the ability of the algorithm to achieve a diverse population.

$$P_m = \frac{\min Ft}{\max Ft} \tag{11}$$

Once crossover and mutation are carried out, offspring are generated and their fitness is evaluated with regards to makespan minimization. A combination of elite and roulette wheel selection is used to curtail the population down to population size. In this study, elitism ratio is kept at 20 % of the whole population. Chromosomes having improved fitness with respect to other competitor chromosomes are introduced into the population thereafter. GA routine is continued until improved solutions are found. The algorithm terminates if no improvement is found till 100 iterations consecutively and the final solution is fed into the re-initialization layer so that elite preservation is ensured.

3.4 Re-initialization layer

This layer receives the elite chromosomes and reinitializes the population to produce diversity for re-exploring the search space. During this process, GS and LS are decreased and increased by 10 % respectively. This approach enhances the generation of chromosomes by local search procedure and reduces global search in every re-initialization while preserving the elite solution. Thus, intensification around the best solution is carried out. Here, as the mutation probability is also adapted on the higher side to maximize the generation of new possible solutions. The new population is again returned to the GA layer until the re-initialization count limit is reached.

3.5 Output layer

This layer decodes the elite chromosome and generates the Gantt chart for the problem accordingly along with the best solution makespan.

4. Results and discussion

Before commencing with the large scale application, the parameters of R-GA were optimized such that algorithm efficiency was achieved. The algorithm was run on a Pentium Core i7 with 4GB RAM and experimental results are compared with known benchmark problems of Kacem [21] and Fattahi [30]. Since the proposed algorithm is pure GA, thus comparison is made either with pure GA based approaches or other standalone comparable evolutionary algorithms.

Fig. 4 shows the convergence pattern for MFJS8. R-GA initializes and generates a random population using the initialization procedures and starts to converge. The initial population is diverse and as convergence is achieved, it initially traps at local minima (922) at 120 generations and retains there for 100 iterations. As no improvement is observed until 100 generations, re-initialization is invoked to introduce diversity in population for search space evaluation. Additionally, the improved local selection in the new generation is also imposed for compensating diversification and intensification. The algorithm again starts to converge and just after re-initialization, attains 884 makespan. The algorithm then continues until termination criteria are met. It is remarkable that the similar algorithm without the proposed diversification technique does not perform to produce comparable results and retains trapped in local minima as obvious from Fig. 6. Similar behavior can be observed for MFJS2, MFJS 4, MFJS5 and MFJS6 as shown in Fig. 7 and Fig. 8. The algorithm, after initial convergence, reinitializes and attains a new minimum which is comparable to available benchmark problem solutions.

The proposed algorithm therefore successfully gets out of the local minima trap owing to the diversification methodology and produces comparable/better results as compared to other similar algorithms. In addition, the proposed scheme also minimizes the possibility of premature convergence as population diversity is initiated to check the local minima trap once the average fitness between generations becomes stable. However, the scheme requires more computational power accordingly owing to the additional features.

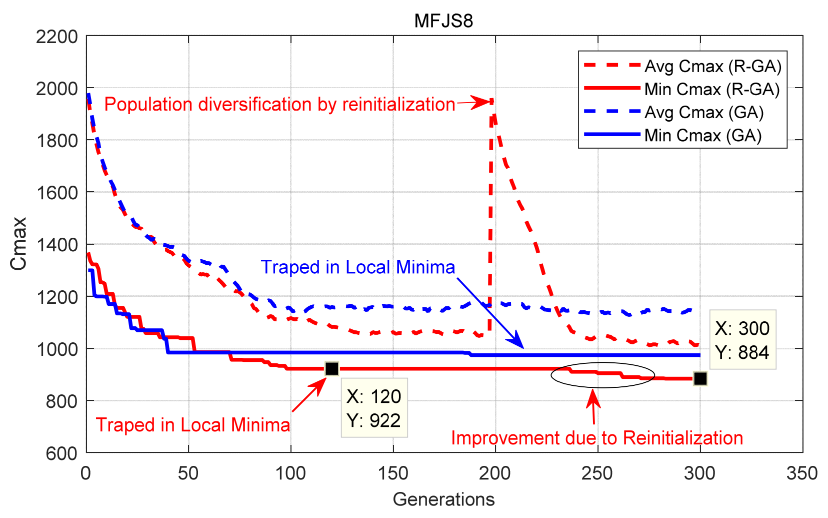

Fig. 6 Convergence pattern for MFJS8 and its comparison with GA

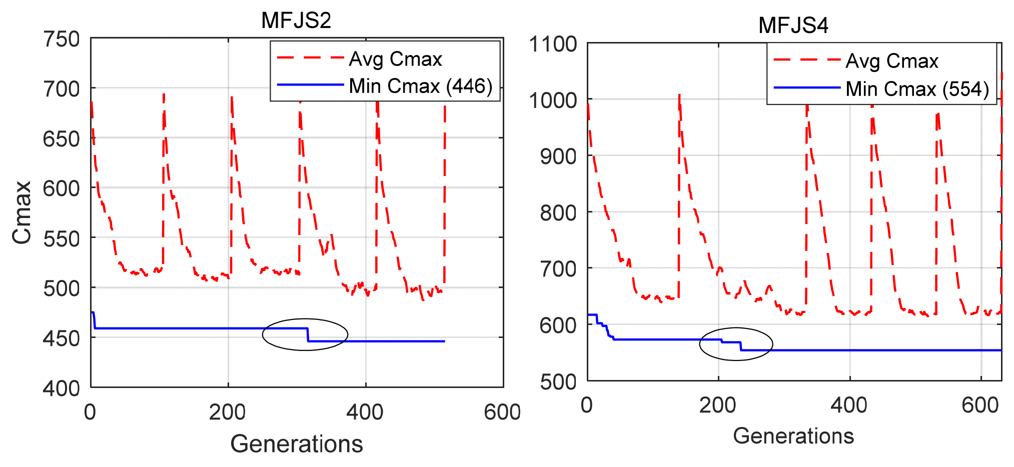

Fig. 7 Convergence pattern for MFJS2 and MFJS4

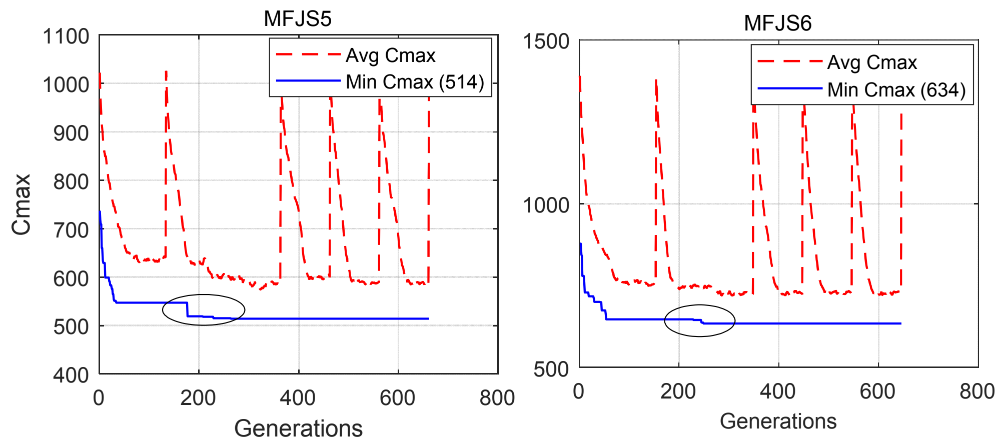

Fig. 8 Convergence pattern for MFJS5 and MFJS6

Table 4 presents the solutions of R-GA in contrast with other comparable algorithms. The algorithm results are compared with standalone optimization algorithms that have solved similar benchmarks. Notably, the algorithm outperforms other algorithms in various instances. Positive mean percentage deviation calculated from Eq. 12 shows that the algorithm performs competitively when tested against these benchmarks and even outperforms in some instances. Fig. 9 presents a final Gantt chart of MFJS8.

$$\%\Delta = \frac{Reference - Achieved}{Reference} \cdot 100 \qquad (12)$$

Table 4 Comparison of R-GA with other similar approaches

Problem	R-GA	HTS/TS [30]		HTS/SA[30]		GA[34]		AIA[35]		CP[36]	
		C_{max}	%Δ	C_{max}	%Δ	C_{max}	%Δ	C_{max}	%Δ	C_{max}	%Δ
SFJS1	66	66	0	66	0	66	0	66	0	66	0
SFJS2	107	107	0	107	0	107	0	107	0	107	0
SFJS3	221	221	0	221	0	221	0	221	0	221	0
SFJS4	355	355	0	355	0	355	0	355	0	355	0
SFJS5	119	119	0	119	0	119	0	119	0	119	0
SFJS6	320	320	0	320	0	320	0	320	0	320	0
SFJS7	397	397	0	397	0	397	0	397	0	397	0
SFJS8	253	253	0	256	1.2	253	0	253	0	253	0
SFJS9	210	210	0	210	0	210	0	210	0	210	0
SFJS10	516	516	0	516	0	516	0	516	0	516	0
MFJS1	468	469	0.2	469	0.2	468	0	468	0	468	0
MFJS2	446	482	7.5	468	4.7	448	0.4	448	0.4	446	0
MFJS3	466	533	12.6	538	13.4	466	0	468	0.4	466	0
MFJS4	554	634	12.6	618	10.4	554	0	554	0	554	0
MFJS5	514	625	17.8	625	17.8	514	0	527	2.5	514	0
MFJS6	634	717	11.6	730	13.2	634	0	635	0.2	634	0
MFJS7	879	964	8.8	947	7.2	881	0.2	879	0	931	5.6
MFJS8	884	970	8.9	922	4.1	891	0.8	884	0	884	0
MFJS9	1091	1105	1.3	1105	1.3	1094	0.3	1088	-0.3	1070	-2
MFJS10	1238	1404	11.8	1384	10.5	1286	3.7	1267	2.3	1208	-2.5
Kacem1	11	-	-	-	-	-	-	-	-	11	0
Kacem2	11	-	-	-	-	-	-	-	-	11	0
Kacem3	7	-	-	-	-	-	-	-	-	7	0
Kacem4	12	-	-	-	-	-	-	-	-	12	0
Mean %Δ			4.7		4.2		0.3		0.3		0

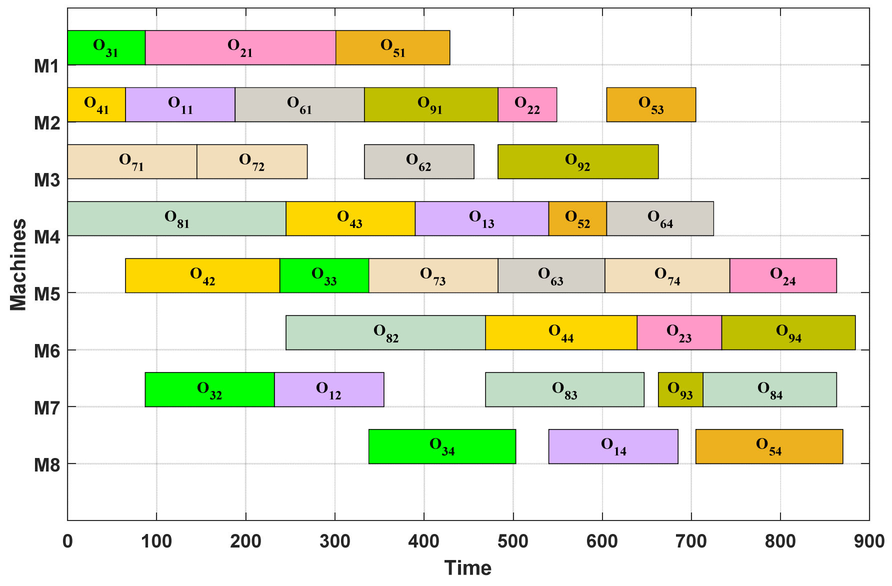

Fig. 9 Gantt chart for MFJS 8

Fig. 10 shows percentage deviation of R-GA from five different algorithms for ten problems, i.e. MFJS1 to MFJS10. Positive deviation indicates that R-GA performed better than reference algorithm while negative deviation indicates otherwise. It is notable that R-GA performs better than other algorithms for more complex problems e.g. MFJS10 etc. It is also evident that R-GA performs in a satisfactory manner as the problem complexity increases since significant positive deviation is achieved in nearly all problems as shown in Fig. 10. However, R-GA lagged for MJFS9 when compared with AIAA and CP and MFJS10 when compared with CP.

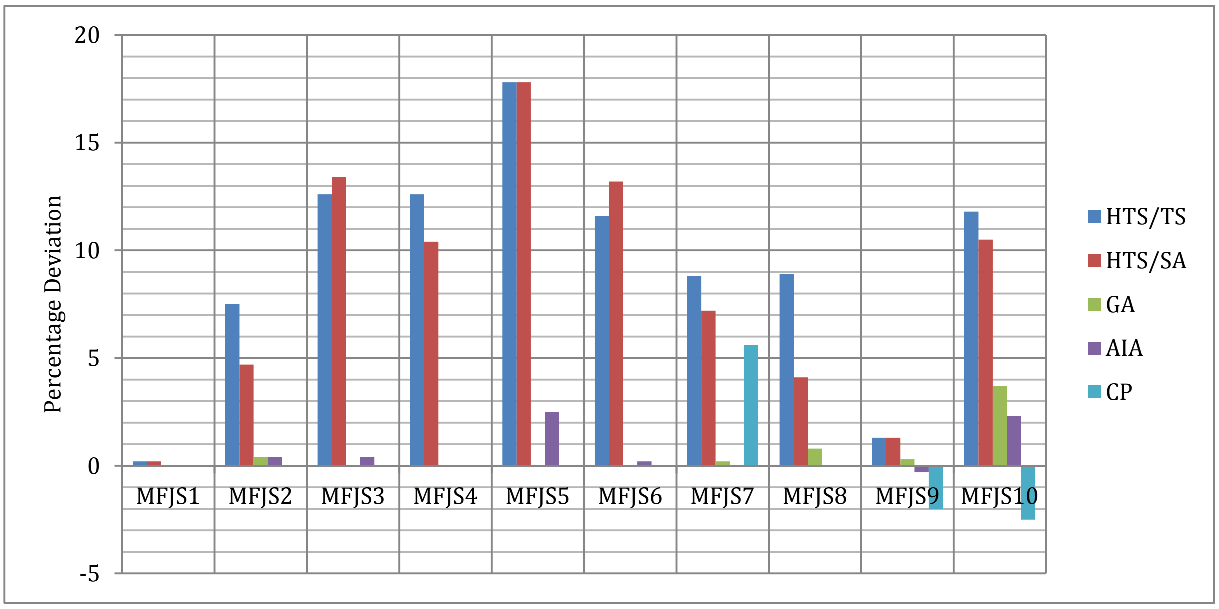

Fig. 10 Percentage deviation of R-GA from different algorithms

5. Conclusion

In this paper, the search space of well-known benchmark problems is addressed qualitatively by proposing a chromosome-based formulation and an insight into the actual problem complexity is presented owing to the NP-hard nature of the problem. A modified GA is then developed and implemented for solving the selected benchmark problems of FJSSP for makespan optimization. In this approach, GA is initialized based upon global, local and random selection techniques and adaptive reproductive operators are applied to intelligently evolve the algorithm. Adaptability has been incorporated in the parameters so that the algorithm may adhere to the current population diversity and acquire additional benefits out of the intensification and diversification regimes. The algorithm converges to a certain limit and traps in local minima while preserving the best available solution. To divulge the algorithm from this point, diversification methodology is employed with revised adaptive parameters and the algorithm converges until termination criteria are encountered. The algorithm is tested extensively on selected benchmarks and it is concluded that the proposed algorithm not only performs effectively for solving the FJSSP but also escapes out of local minima trap at various instances. The convergence patterns along with solution quality further endorse the algorithm efficacy. It is also identified that the proposed diversification methodology produces better results when integrated with the GA and surpasses most of the other standalone comparable approaches.

This work enhances the utility of GA through the effective use of various diversification techniques and provides a framework for effectively exploring the huge search space with an easy-to-program approach. A user-friendly software is developed in this work which requires minimum input from the user and can be used in other similar optimization applications. Future research involves the expansion of algorithm application on multi-objective problems and other objective functions.

Acknowledgement

The authors gratefully acknowledge the support of National University of Science and Technology, Islamabad, Pakistan for funding the studies

References

[1] Pinedo, M.L. (2012). *Scheduling: Theory, algorithms, and systems*, Springer, Boston, USA, doi: 10.1007/978-1-4614-2361-4.

[2] Jain, A., Jain, P.K., Chan, F.T.S., Singh, S. (2013). A review on manufacturing flexibility, *International Journal of Production Research*, Vol. 51, No. 19, 5946-5970, doi: 10.1080/00207543.2013.824627.

[3] Dauzère-Péres, S., Lasserre, J.-B. (2012). *An integrated approach in production planning and scheduling*, Springer-Verlag, Berlin, Germany, doi: 10.1007/978-3-642-46804-9.

[4] Groover, M.P. (2015). *Automation, production systems, and computer-integrated manufacturing*, 4th edition, Pearson Education, Delhi, India.

[5] Azzouz, A., Ennigrou, M., Said, L.B. (2017). A hybrid algorithm for flexible job-shop scheduling problem with setup times, *International Journal of Production Management and Engineering*, Vol. 5, No. 1, 23-30, doi: 10.4995/ijpme.2017.6618.

[6] Kacem, I. (2013). Genetic algorithms for solving flexible job shop scheduling problems, In: Jarboui, B., Siarry, P., Teghem, J. (eds.), *Metaheuristics for Production Scheduling*, John Wiley & Sons, New York, USA, 19-44, doi: 10.1002/9781118731598.ch2.

[7] Chaudhry, I.A., Usman, M. (2017). Integrated process planning and scheduling using genetic algorithms, *Tehnički Vjesnik – Technical Gazette,* Vol. 24, No. 5, 1401-1409, doi: 10.17559/TV-20151121212910.

[8] Cheng, T.C.E., Shafransky, Y., Ng, C.T. (2016). An alternative approach for proving the NP-hardness of optimization problems, *European Journal of Operational Research*, Vol. 248, No. 1, 52-58, doi: 10.1016/j.ejor.2015.06.076.

[9] Brucker, P., Sotskov, Y.N., Werner, F. (2007). Complexity of shop-scheduling problems with fixed number of jobs: A survey, *Mathematical Methods of Operations Research*, Vol. 65, No. 3, 461-481, doi: 10.1007/s00186-006-0127-8.

[10] Candan, G., Yazgan, H.R. (2015). Genetic algorithm parameter optimisation using Taguchi method for a flexible manufacturing system scheduling problem, *International Journal of Production Research*, Vol. 53, No. 3, 897-915, doi: 10.1080/00207543.2014.939244.

[11] Ojstersek, R., Lalic, D., Buchmeister, B. (2019). A new method for mathematical and simulation modelling interactivity: A case study in flexible job shop scheduling, *Advances in Production Engineering & Management*, Vol. 14, No. 4, 435-448, doi: 10.14743/apem2019.4.339.

[12] Borreguero-Sanchidrián, T., Pulido, R., García-Sánchez, Á., Ortega-Mier, M. (2018). Flexible job shop scheduling with operators in aeronautical manufacturing: A case study, *IEEE Access*, Vol. 6, 224-233, doi: 10.1109/ACCESS.2017.2761994.

[13] Nidhiry, N.M., Saravanan, R. (2014). Scheduling optimization of a flexible manufacturing system using a modified NSGA-II algorithm, *Advances in Production Engineering & Management*, Vol. 9, No. 3, 139-151, doi: 10.14743/apem2014.3.183.

[14] Hecker, F.T., Hussein, W.B., Paquet-Durand, O., Hussein, M.A., Becker, T. (2013). A case study on using evolutionary algorithms to optimize bakery production planning, *Expert Systems with Applications*, Vol. 40, No. 17, 6837-6847, doi: 10.1016/j.eswa.2013.06.038.

[15] Amjad, M.K., Butt, S.I., Kousar, R., Ahmad, R., Agha, M.H., Faping, Z., Anjum, N., Asgher, U. (2018). Recent research trends in genetic algorithm based flexible job shop scheduling problems, *Mathematical Problems in Engineering*, Vol. 2018, No. 32, Article ID 9270802, doi: 10.1155/2018/9270802.

[16] Zhang, J., Ding, G., Zou, Y., Qin, S., Fu, J. (2019). Review of job shop scheduling research and its new perspectives under Industry 4.0, *Journal of Intelligent Manufacturing*, Vol. 30, No. 4, 1809-1830, doi: 10.1007/s10845-017-1350-2.

[17] Janes, G., Perinic, M., Jurkovic, Z. (2017). An efficient genetic algorithm for job shop scheduling problems, *Tehnički Vjesnik – Technical Gazette,* Vol. 24, No. 4, 1243-1247, doi: 10.17559/TV-20150527133957.

[18] Chaudhry, I.A., Khan, A.A. (2016). A research survey: Review of flexible job shop scheduling techniques, *International Transactions in Operational Research*, Vol. 23, No. 3, 551-591, doi: 10.1111/itor12199.

[19] Çaliş, B., Bulkan, S. (2015). A research survey: Review of AI solution strategies of job shop scheduling problem, *Journal of Intelligent Manufacturing*, Vol. 26, No. 5, 961-973, doi: 10.1007/s10845-013-0837-8.

[20] Ida, K., Oka, K. (2011). Flexible job-shop scheduling problem by genetic algorithm, *Electrical Engineering in Japan*, Vol. 177, No. 3, 28-35, doi: 10.1002/eej.21194.

[21] 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, No. 1, 1-13, doi: 10.1109/TSMCC.2002.1009117.

[22] Gao, J., Gen, M., Sun, L., Zhao, X. (2007). A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems, *Computers & Industrial Engineering*, Vol. 53, No. 1, 149-162, doi: 10.1016/j.cie.2007.04.010.

[23] Pezzella, F., Morganti, G., Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem, *Computers & Operations Research*, Vol. 35, No. 10, 3202-3212, doi: 10.1016/j.cor.2007.02.014.

[24] Gu, X., Huang, M., Liang, X. (2019). An improved genetic algorithm with adaptive variable neighborhood search for FJSP, *Algorithms*, Vol. 12, No. 11, Article No. 243, doi: 10.3390/a12110243.

[25] Wang, L., Luo, C., Cai, J. (2017). A variable interval rescheduling strategy for dynamic flexible job shop scheduling problem by improved genetic algorithm, *Journal of Advanced Transportation*, Vol. 2017, Article ID 1527858, doi: 10.1155/2017/1527858.

[26] Xiong, J., Tan, X., Yang, K.-W., Xing, L.-N., Chen, Y.-W. (2012). A hybrid multiobjective evolutionary approach for flexible job-shop scheduling problems, *Mathematical Problems in Engineering*, Vol. 2012, Article ID 478981, doi: 10.1155/2012/478981.

[27] Teekeng, W., Thammano, A. (2012). Modified genetic algorithm for flexible job-shop scheduling problems, *Procedia Computer Science,* Vol. 12, 122-128, doi: 10.1016/j.procs.2012.09.041.

[28] Framinan, J.M., Leisten, R., García, R.R. (2014). *Manufacturing scheduling systems*, Springer-Verlag, London, United Kingdom, doi: 10.1007/978-1-4471-6272-8.

[29] Zhang, G., Gao, L., Shi, Y. (2011). An effective genetic algorithm for the flexible job-shop scheduling problem, *Expert Systems with Applications*, Vol. 38, No. 4, 3563-3573, doi: 10.1016/j.eswa.2010.08.145.

[30] Fattahi, P., Saidi Mehrabad, M., Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems, *Journal of Intelligent Manufacturing*, Vol. 18, No. 3, 331-342, doi: 10.1007/s10845-007-0026-8.

[31] Song, W.J., Zhang, C.Y., Lin, W.W., Shao, X.Y. (2014). Flexible job-shop scheduling problem with maintenance activities considering energy consumption, *Applied Mechanics and Materials*, Vol. 521, 707-713, doi: 10.4028/www.scientific.net/AMM.521.707.

[32] De Giovanni, L., Pezzella, F. (2010). An improved genetic algorithm for the distributed and flexible job-shop scheduling problem, *European Journal of Operational Research,* Vol. 200, No. 2, 395-408, doi: 10.1016/j.ejor.2009.01.008.

[33] Nouri, H.E., Driss, O.B., Ghédira, K. (2017). Solving the flexible job shop problem by hybrid metaheuristics-based multiagent model, *Journal of Industrial Engineering International*, Vol. 14, 1-14, doi: 10.1007/s40092-017-0204-z.

[34] Zandieh, M., Mahdavi, I., Bagheri, A. (2008). Solving the flexible job-shop scheduling problem by a genetic algorithm, *Journal of Applied Sciences*, Vol. 8, No. 24, 4650-4655, doi: 10.3923/jas.2008.4650.4655.

[35] Bagheri, A., Zandieh, M., Mahdavi, I., Yazdani, M. (2010). An artificial immune algorithm for the flexible job-shop scheduling problem, *Future Generation Computer Systems*, Vol. 26, No. 4, 533-541, doi: 10.1016/j.future.2009.10.004.

[36] Behnke, D., Geiger, M.J. (2012). *Test instances for the flexible job shop scheduling problem with work centers*, Universitätsbibliothek der Helmut-Schmidt-Universität, Hamburg, Germany, from *https://d-nb.info/1023241773/34*, accessed May 2020.