Solving Flexible Job-Shop Scheduling Problem Using Improved Real Coded Genetic Algorithms

Wayan Firdaus Mahmudy
Department of Computer Science, University of Brawijaya (UB)
Malang, Indonesia
wayanfm@ub.ac.id

***Abstract*—This paper addresses optimization of the flexible job-shop problem (FJSP). This NP-hard problem is solved by using improved real-coded genetic algorithm (IRCGA). The real-coded genetic algorithm (RCGA) uses a vector of real numbers as chromosome representation. The previous research proved that the representation could be used to effectively explore the search space. In this paper, the RCGA is further improved by adding a mechanism to adaptively changing crossover and mutation rates. Numerical experiments show that the mechanism could improve the performance of the RCGA to solve various test bed problems from literature.**

Keywords— flexible job-shop problem (FJSP); improved real-coded genetic algorithm; crossover; mutation, manufacturing

I. INTRODUCTION

Jobs scheduling is an important decision making process in manufacturing systems. The process allocates limited resources such as machines and tools to perform operations [1]. The sequence of operations is called a job. To achieve optimum resources utilization and throughput of the system, a good schedule is required.

The flexible job-shop problem (FJSP) is one of scheduling types in manufacturing system. The FJSP is generalized form of the classical job-shop problem (JSP) and exists in modern manufacturing system where a machine can perform different operations to produce different products. Thus, instead of an operation must be processed on a dedicated machine as can be found in the JSP, in the FJPS an operation can be processed on several alternatives machines [2]. This flexibility makes solving the FJSP is more difficult than solving the FJP that is well known as NP-hard problem. Therefore, good approaches are required to address the FJPS [3, 4].

Various meta-heuristic algorithms have been proposed to deal with the large search space of the FJSP. The proposed approaches include simulated annealing (SA) [5], tabu search (TS) [6], variable neighborhood search (VNS) [7], and genetic algorithm (GAs) [8-11]. Due to the complexity of the FJPS, more powerful approaches were developed by combining two methods such as hybrid GAs with TS [12], hybrid particle swarm optimization (PSO) with TS [13], and hybrid ant colony algorithm (ACO) with PSO [14].

Previous researches have proven that real-coded genetic algorithm (RCGA) was very effective to explore a large search space of the FJSP [3, 4]. In this paper, the RCGA is further improved by adding a mechanism to adaptively change crossover and mutation rate. The mechanism allows the RCGA to balance its power of exploring and exploiting the search space for better solutions.

Rest of the paper is organized as follow. Next section discusses related works related to heuristic algorithms for the FJSP. Section 3 introduces definition and assumptions of the FJSP. Section 4 details the development of the RCGA. Section 5 discusses the results of computational experiments. Finally, the conclusion and future works are presented in Section 6.

II. RELATED WORKS

Solving the FJSP requires two decisions. The first decision is choosing a machine for a job's operation. The decision is required since there are several possible machines for the operation. The second decision is determining the sequence of operations in the machines. Due to the existence of the two decisions, approaches to solve the FJSP can be classified into two classes: hierarchical approaches and integrated approaches.

The hierarchical approaches decompose the FJSP into two stages. Machine selection for all operations is carried out in the first stage by using dispatching rules or heuristic algorithms. In the second stage, meta-heuristic algorithms are applied to obtain the optimum operations sequence. On the other hand, the integrated approaches search the optimum solution in a one stage. The integrated approaches require higher computational time as they should explore a larger search space. However, some studies proved that the integrated approaches tend to produce better solutions comparable to those achieved by the hierarchical approaches [8].

The hierarchical approaches were implemented in several earlier studies. For example, Brandimarte [6] solved both two sub-problems in the FJSP by using TS. A special strategy was developed to consider information flow between two levels of decision. The integrated approaches were developed in later studies. For example, Yazdani et al. [5] solved the FJSP by using simulated annealing. Their approach was supported by two neighborhood structures to generate new solutions. A parallel variable neighborhood search (PVNS) algorithm was proposed in [7]. The parallelization was designed to enable the VNS simultaneously explore the large search space of the FJSP.

Original Article
Mahmudy, WF 2014, 'Solving flexible job-shop scheduling problem using improved real coded genetic algorithms', *International Conference on Science and Technology for Sustainability*, Batam, Indonesia, 22 – 23 October 2014, pp. 181-188.

To deal with the large search space of the FJPS, hybrid approaches were developed in several studies. For instance, Li et al. [13] enhanced a power of PSO with TS. In each iteration, TS improved solutions of PSO by exploring their neighbors' solutions. Moreover, a hybrid ACO and PSO was proposed in [14]. The merit of PSO for achieving a fast convergence was used to determine initial positions of ants in ACO. The ants that represent solutions of the FJSP then were driven by ACO to reach the global optimum solution.

Due to their effectiveness for solving complex combinatorial problems, GAs were implemented to solve the FJSP in several studies. For instance, Pezzella et al. [8] proposed a GA that was supported by a mechanism to produce a good initial population and an intelligent mutation to enhance the quality of solution. To build more powerful GAs, hybrid strategies were also implemented. For example, Ho and Tay [15] improved their GA with a cultural evolution. The cultural evolution preserved patterns of good schedules to be inherited to the next generations. Moreover, a local search was implemented to enhance the quality of chromosomes (solutions) for GA in a study by Al-Hinai and ElMekkawy [16]. A similar strategy was used by Zhang et al. [17] that hybridized their GA with VNS.

All approaches discussed in this section have common similarity; the approaches encoded a solution as an array of integer numbers. Most of them used task sequencing list representation proposed by Kacem et al. [18]. The representation is composed by a list of vector of three integer numbers that represent job, operation and machine. Various smart strategies such as hybridization and generating good initial solutions were developed to support the approaches obtained good solutions. Even if near optimum solutions were obtained, the strategies might require a high computational time.

This paper is an extension of previous studies [3, 4]. The studies proved that by using simple genetic operators (crossover and mutation) and random initial population, the real-coded genetic algorithm (RCGA) could produce promising results. In this study, the RCGA is further improved by adding a mechanism to adaptively changing crossover and mutation rate. The mechanism is quite simple and do not increase computational time of the RCGA significantly.

III. THE FJSP

This paper addresses a FJSP in a manufacturing environment with the following characteristics:

- There are J independent jobs and M machines.
- Job j (j=1..J) is composed by a number of nonpreemptable ordered operations ($O_{j,k}$, k=1...n_j, n_j=number of operations of job j).
- Operation k of job j ($O_{j,k}$) can be processed on a set of alternative machines ($M_{j,k}$).
- Processing operation $O_{j,k}$ on machine m ($m \in M_{j,k}$) requires $T_{j,k,m}$. time units (including setup time for the machine).

Several assumptions exist as follows:

- All jobs are ready to be processed at time zero.
- An operation can be assigned to only a single machine among several possible machines.
- Machines are never breakdown and always available during production cycle.

The scheduler must make two decisions. The first decision is selecting machine m from $M_{j,k}$ for operation $O_{j,k}$. The second is determining the order $O_{j,k}$ in the machines. Minimizing the completion time of all operations (makespan) is the objective of the FJSP in this study.

IV. DEVELOPMENT OF THE IMPROVED RCGA

Genetic algorithms (GAs) are well known as the powerful method to solve complex problems with a large search space [19]. GAs have an ability to escape from local optima as they can jump randomly from one sequence to another sequence [20]. Solutions in GAs are encoded as chromosomes. In this paper, the chromosomes are represented as an array of real number. Thus, the GA is called real-coded GA (RCGA).

A. *The cycle of the RCGA*

The cycle of the RCGA are detailed in the form of pseudo code in Fig. 1.

```
Determine parameters of the RCGA
   population size pop_size
   crossover rate cr
   mutation rate mr
   number of generations n_gen
Initialization
   Let generation gen←0
   Create pop_size of random chromosomes
While gen<n_gen
   Reproduction
     crossover: produce pop_size×cr offspring
     mutation: produce pop_size×mr offspring
   Selection
     Select pop_size chromosomes from parents
     (population) and offspring pool for the
     next generation.
   Adjust Crossover and Mutation Rates
   Let gen←gen + 1.
End while
```

Fig. 1. Pseudo code of the RCGA

All stages in the cycle of the RCGA are explained in the following sub-sections:

1) Initialization

In this stage, *pop_size* of initial chromosomes are randomly generated. The chromosomes are placed in a pool called *population*. A simple and efficient technique to determine proper value for *pop_size* is not available [21]. Thus, a series of preliminary experiments is required [22].

2) Reproduction

On each generation, new chromosomes (*offspring*) are produced by using two reproduction operators: crossover and mutation. As in the previous studies [3, 4], two crossover methods, *extended-intermediate-crossover* and *one cut point*

crossover, and two mutation methods, *simple random mutation* and *random exchange mutation* are used. Parent chromosomes for the reproduction operators are randomly chosen from the population pool. All new chromosomes produced in this stage are placed in a pool called *offspring pool*.

3) Selection

In this stage, *pop_size* chromosomes from current population (parents) and offspring pool are selected to be passed to the next generation. To ensure that better next generations are obtained, a proper selection method is required. The previous studies [4, 23, 24] found that replacement selection is the most suitable method for the RCGA. Thus, the replacement selection is used in this study.

B. Chromosome Representation

A good chromosome representation is required to effectively explore a large search space of the FJSP. In the previous step of this research [3], the chromosome was composed of an array of real numbers. The representation always produced feasible solutions. This feature reduced the computational time needed by GA to repair infeasible chromosomes. The other benefit of using real numbers representation was simple mechanism of crossover and mutation methods can be applied.

A simple case of FJSP is created to explain a conversion mechanism of a real-coded chromosome into a solution. As shown in Table 1, there are 4 jobs and their operations sequence. Here, the total number of operations (denoted by *nop*) is 10. Three machines are required to complete all operations. An example of the flexibility of machining operations is shown by operation 1 of job 3. Here, the operation can be done in machine 1 or machine 2 with different processing times. In this case, maximum number of alternative machines of operations (denoted by *aMac*) is equal to 2. This value is indicated by job 3 operation 1, job 4 operation 1, and job 4 operation 2.

TABLE I. JOBS AND THEIR OPERATIONS SEQUENCE

job	operation	machine	time
1	1	1	6
	2	3	4
	3	1	6
2	1	1	4
	2	2	4
3	1	1	4
		2	5
	2	3	5
4	1	3	4
		2	5
	2	1	5
		3	4
	3	2	3

An operation index and its corresponding job are required in the chromosome conversion as shown in Table 2. Here, job 1 that has 3 operations is associated with operation indexes 1, 2, and 3.

TABLE II. JOBS AND THEIR OPERATION INDEXES

operation index	job
1	1
2	1
3	1
4	2
5	2
6	3
7	3
8	4
9	4
10	4

An example of chromosome (X) is shown in the first column of Table 3. Length of X is equal with the total number jobs' operations (*nop*). Elements of $X=(x_1,x_2,\ldots,x_p)$ represent the continuous position values for *nop* operations. x_i (i=1,2,…,p) has value in range $[0, 2^{bitOp+bitMac}]$. *bitOp* is number of bits required to represent a binary number in range [0, *nop*]. *bitMac* is number of bits required to represent a binary number in range [0, *aMac*]. Thus, x_i has value in range $[0, 2^{5+2}]$. x_i is treated as a real number in the cycle of GA. However, x_i will be rounded to a nearest integer value in the conversion process.

Jobs (*job*) associated with the chromosome are shown in the second column. The column is taken from Table 2. The sequence of operations to be processed in the machines can be obtained by sorting *x* and *job* in ascending order according to the value of *x*. The result is shown in the fourth column (*sorted job*). The fifth column (*op*) shows the operation sequence of each job. By using values of sorted *job* and *op* we obtain that operation 1 of job 4 is firstly processed in the machine, followed by job 4 operation 2, job 1 operation 1, and so on.

TABLE III. CHROMOSOME CONVERSION

X	*job*	sorted		*op*	binary(*X*)	dec	n	*mac idx*	*mac*
		X	*job*						
18	1	8	4	1	00001**00**	0	2	1	3
144	1	10	4	2	00001**01**0	1	2	2	3
182	1	18	1	1	00010**01**0	1	1	1	1
228	2	107	4	3	01101**01**1	1	1	1	2
174	2	144	1	2	10010**00**0	0	1	1	3
212	3	146	3	1	10010**01**0	1	2	2	2
146	3	174	2	1	10101**10**0	3	1	1	1
8	4	182	1	3	10110**10**0	3	1	1	1
10	4	212	3	2	11010**10**0	2	1	1	3
107	4	228	2	2	11100**10**0	2	1	1	2

Machine index (*mac-idx*) is obtained from a binary operation of sorted X. For instance, x_2=10 is converted into $(00001010)_2$. The last bit is reserved for the next research to deal with alternative production plan which refer to possibility of processing jobs on alternative operation sequence [24]. As *bitMac*=2, the remaining 2 right bits (01) are used. The binary

number $(01)_2$ is equal to 1. As there are 2 possible machines (n=2) for operation 2 of job 4, the following formula is applied:

$$mac\text{-}idx = (01)_2 \ mod\ n + 1 = 1\ mod\ 2 + 1 = 2$$

mod is modulus operator which gives the remainder of a division. As *mac-idx*=2, operation 2 of job 4 is processed on the second alternative machine that is machine 3. Finally, the operation sequences can be stated as the sequence of triples (*j*,*o*,*m*) which represent job *j*, operation *o*, and machine *m* as follows: (4,1,3)→(4,2,3) →(1,1,1) →(4,3,2) →(1,2,3) →(3,1,2) →(2,1,1) →(1,3,1) →(3,2,3) →(2,2,2). The sequence is used to generate an active schedule as can be seen in Gantt chart in Fig. 2. Here, the completion time of all operations (*makespan*) is 18.

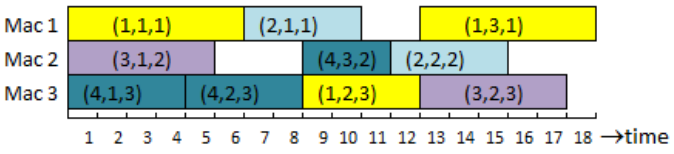

Fig. 2. Gantt chart of the active schedule

Note that the conversion process always produces feasible solutions. This feature will prevent GA to do two procedures that require a high computational time, moving its population toward a feasible search space and repairing infeasible chromosomes [25].

C. Fitness Function

The RCGA requires a fitness function to measure the quality of chromosomes according to the objective of the problem. A bigger fitness value means a better chromosome. In this paper, minimizing of the makespan is changed into maximizing of the fitness value as shown in Eq. 1.

$$fitness = 1/makespan \quad (1)$$

D. Adaptive Crossover and Mutation Rates

The power of GAs to explore and exploit the search space depend on their crossover rate and mutation rate [26]. A low value of crossover rate may cause the RCGA depend on its mutation rate and tend to work as a random search method. Here, the RCGA may not effectively learn from previous generations and the search space may not be exploited effectively.

On contrary, the RCGA will lose its ability to maintain population diversity if using a high crossover rate and a low mutation rate. By using a high crossover rate the offspring will have a high similarity with their parents. Thus, only in few generations the HGA achieves a premature convergence and loses the chance to explore other areas in the search space.

Several preliminary experiments are required to obtain the best combination of crossover and mutation rates [23]. It is possible that different values of crossover rate and mutation rate are required for different problem sizes.

In this paper, a rule for adjusting crossover rate (*cr*) and mutation rate (*mr*) is developed and presented in Fig. 3. On each generation the average of fitness values (*fAvg*) is calculated. If there is significant improvement of *fAvg* then *cr* is increased whereas *mr* is decreased. This change will enable the RCGA to exploit local search areas. On the other hand, if there is no significant improvement of *fAvg* then *cr* is decreased whereas *mr* is increased. This change will enable the RCGA to explore the search space by jumping out from local search areas. Additional checking is required to preserve the values of *cr* and *mr* are always within predetermined reasonable intervals of [*crMin*, *crMax*] and [*mrMin*, *mrMin*] respectively.

```
IF (fAvg - fAvgPrev) > threshold THEN
   cr ← cr + 0.01
   mr ← mr - 0.01
ELSE
   cr ← cr - 0.01
   mr ← mr + 0.01
END IF

/* checking interval */
IF cr < crMin THEN
   cr ← crMin
ELSE IF cr > crMax THEN
   cr ← crMax
END IF

IF mr < mrMin THEN
   mr ← mrMin
ELSE IF mr > mrMax THEN
   mr ← mrMax
END IF
```

Fig. 3. Pseudo code of the adaptive crossover and mutation rates

V. Computational Experiments

The performance of the improved RCGA is evaluated by using data set from Brandimarte [6] available at http://www.idsia.ch/~monaldo/fjsp.html. As can be seen in Table 4, the data set has 10 problem instances with various numbers of jobs (*jobs*), machines (*macs*), number of operations for each job, and total number of operations (*ops*).

TABLE IV. Characteric of the Data Set

problem	jobs	macs	ops
Mk01	10	6	55
Mk02	10	6	58
Mk03	15	8	150
Mk04	15	8	90
Mk05	15	4	106
Mk06	10	15	150
Mk07	20	5	100
Mk08	20	10	225
Mk09	20	10	240
Mk10	20	15	240

The best combination parameters value for the RCGA was obtained in the previous research as follows:

– Initial crossover rate is 0.7. This value will be changed adaptively during the cycle of the RCGA.

- Initial mutation rate is 0.3
- Population size is 4000.
- Iterations will be terminated after 500 successive generations no longer produces better results or after reaches generation of 5,000.

The experimental results are presented in Table 5. Makespans in the column '*iRCGA*' is the best result from five runs results of the improved RCGA using different initial populations. The makespans are compared to the results produced by other heuristic approaches that are available in the literature. The approaches include a cultural-based GA (*GENACE*) developed by Ho and Tay [15], a specialized GA (*sGA*) proposed by Pezzella et al. [8], hybrid particle swarm optimization and tabu search algorithm (*hPSO*) developed by Li et al. [13], and hybridized genetic algorithm (*hGA*) developed by Al-Hinai and ElMekkawy [16].

To measure quality of solutions produced by iRCGA, a relative deviation (*dev*) of makespan obtained by iRCGA to the best solution obtained by the other heuristic method is calculated using Eq. 2.

$$dev = \frac{(makespan_{iRCGA} - makespan_{\min})}{makespan_{\min}} \times 100\% \quad (2)$$

TABLE V. THE RESULTS COMPARISON

problem	dev (%)	iRCGA	other results			
			GENACE	sGA	hPSO	hGA
Mk01	0	**40**	41	40	40	40
Mk02	3.8	27	29	26	27	26
Mk03	0	**204**	-	204	204	204
Mk04	0	**60**	67	60	63	61
Mk05	0	**173**	176	173	173	173
Mk06	6.5	66	68	63	65	62
Mk07	1.4	141	148	139	145	141
Mk08	0	**523**	523	523	523	523
Mk09	0	**307**	328	311	331	307
Mk10	7.1	227	231	212	223	214
average	1.88					

Table 5 clearly shows that the proposed iRCGA can produce promising results. The iRCGA achieves minimum makespan as those achieved by the four other approaches (dev=0%) in 6 out of 10 problems (Mk01, Mk03, Mk04, Mk05, Mk08, and Mk09). In the other problems, the iRCGA produces relatively small values of *dev* with the average of 1.8%. Note that the promising results of iRCGA are achieved by using only simple genetic operators and mechanism to adjust production rate adaptively. Thus, it proves the effectiveness of chromosome representation of the iRCGA.

The superiority of the iRCGA is clear if compared to the GENACE and hPSO individually. The iRCGA outperforms GENACE in 8 problems and outperforms hPSO in 3 problem.

To prove the effectiveness of adaptive crossover and mutation rates, complete results are presented in Table 6. The table shows the best result (*best*), average result (*avg*), and standart deviation (*std*) of makespans produced in five runs of iRCGA and RCGA. The average of computational time (*time*) in seconds to complete all iterations is also presented.

Table 6 clearly shows that iRCGA produces better average solutions in all problems. Thus, it proves the effectiveness of mechanism to adjust production (crossover and mutation) rate adaptively. The mechanism requires less than 100 seconds additional computational time in average.

TABLE VI. RESULTS COMPARISON OF IRCGA AND RCGA

problem	iRCGA				RCGA			
	best	avg	std	time	best	avg	std	time
Mk01	40	40.0	0.0	106	40	40.0	0.0	87
Mk02	27	27.8	0.4	186	27	28.6	0.9	117
Mk03	204	204.0	0.0	427	204	204.0	0.0	324
Mk04	60	61.6	1.1	333	60	63.4	2.1	231
Mk05	173	173.6	0.5	437	173	173.8	0.8	364
Mk06	66	68.6	1.7	849	66	70.6	3.0	708
Mk07	141	143.4	1.3	317	142	143.6	0.9	303
Mk08	523	523.0	0.0	852	523	523.0	0.0	542
Mk09	307	309.8	1.8	1661	307	310.6	2.2	1515
Mk10	227	235.8	5.3	1915	227	240.0	7.5	1993
average				**708.3**				**618.4**

VI. CONCLUSION

This paper presents the development of real-coded genetic algorithm (RCGA) to solve the FJSP. The RCGA is further improved by adding mechanism to adjust production (crossover and mutation) rate adaptively. The improved RCGA (iRCGA) produce promising results that proves the effectiveness of its chromosome representation. The effectiveness of mechanism to adjust production (crossover and mutation) rate adaptively is shown by comparing solutions produced by iRCGA and RCGA.

The next research will deal with alternative production plan which refer to possibility of processing jobs on alternative operation sequence. Thus, the mechanism for conversion of a chromosome into a solution will be modified.

REFERENCES

[1] G. Tuncel, "A heuristic rule-based approach for dynamic scheduling of flexible manufacturing systems," in *Multiprocessor Scheduling: Theory and Applications*, E. Levner, Ed., ed Vienna, Austria: I-Tech Education and Publishing, 2007.

[2] G. Zhang, X. Shao, P. Li, and L. Gao, "An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem," *Computers & Industrial Engineering*, vol. 56, pp. 1309-1318, 2009.

[3] W. F. Mahmudy, R. M. Marian, and L. H. S. Luong, "Real coded genetic algorithms for solving flexible job-shop scheduling problem – Part I: modeling," *Advanced Materials Research*, vol. 701, pp. 359-363, 2013.

[4] W. F. Mahmudy, R. M. Marian, and L. H. S. Luong, "Real coded genetic algorithms for solving flexible job-shop scheduling problem – Part II: optimization," *Advanced Materials Research*, vol. 701, pp. 364-369, 2013.
[5] M. Yazdani, M. Gholami, M. Zandieh, and M. Mousakhani, "A simulated annealing algorithm for flexible job-shop scheduling problem," *J. Applied Sci*, vol. 9, pp. 662-670, 2009.
[6] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations Research*, vol. 41, pp. 157-183, 1993.
[7] M. Yazdani, M. Amiri, and M. Zandieh, "Flexible job-shop scheduling with parallel variable neighborhood search algorithm," *Expert Systems with Applications*, vol. 37, pp. 678-687, 2010.
[8] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Computers & Operations Research*, vol. 35, pp. 3202-3212, 2008.
[9] F. Defersha and M. Chen, "A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups," *The International Journal of Advanced Manufacturing Technology*, vol. 49, pp. 263-279, 2010.
[10] K. Ida and K. Oka, "Flexible job-shop scheduling problem by genetic algorithm," *Electrical Engineering in Japan*, vol. 177, pp. 28-35, 2011.
[11] G. Zhang, L. Gao, and Y. Shi, "An effective genetic algorithm for the flexible job-shop scheduling problem," *Expert Systems with Applications*, vol. 38, pp. 3563-3573, 4// 2011.
[12] M. Gonzalez, C. R. Vela, and R. Varela, " An efficient memetic algorithm for the fexible job shop with setup times," in *Proc. of ICAPS-2013, 2013*, 2011.
[13] J.-q. Li, Q.-k. Pan, S.-x. Xie, B.-x. Jia, and Y.-t. Wang, "A hybrid particle swarm optimization and tabu search algorithm for flexible job-shop scheduling problem," *International Journal of Computer Theory and Engineering*, vol. 2, pp. 1793-8201, April 2010 2010.
[14] L. Li, W. Keqi, and Z. Chunnan, "An improved ant colony algorithm combined with particle swarm optimization algorithm for multi-objective flexible job shop scheduling problem," in *Machine Vision and Human-Machine Interface (MVHI), 2010 International Conference on*, 2010, pp. 88-91.
[15] N. B. Ho and J. C. Tay, "GENACE: an efficient cultural algorithm for solving the flexible job-shop problem," presented at the IEEE international conference on robotics and automation, 2004.
[16] N. Al-Hinai and T. ElMekkawy, "An efficient hybridized genetic algorithm architecture for the flexible job shop scheduling problem," *Flexible Services and Manufacturing Journal*, vol. 23, pp. 64-85, 2011.
[17] G. Zhang, L. Gao, X. Li, and P. Li, "Variable neighborhood genetic algorithm for the flexible job shop scheduling problems," in *Intelligent Robotics and Applications*. vol. 5315, C. Xiong, H. Liu, Y. Huang, and Y. Xiong, Eds., ed: Springer Berlin / Heidelberg, 2008, pp. 503-512.
[18] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 32, pp. 1-13, 2002.
[19] W. Shen, "Genetic algorithms in agent-based manufacturing scheduling systems," *Integr. Comput.-Aided Eng.*, vol. 9, pp. 207-217, 2002.
[20] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*. New York: John Wiley & Sons, Inc., 2000.
[21] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Heidelberg: Springer-Verlag, 1996.
[22] W. F. Mahmudy, R. M. Marian, and L. H. S. Luong, "Solving part type selection and loading problem in flexible manufacturing system using real coded genetic algorithms – Part II: optimization," in *International Conference on Control, Automation and Robotics*, Singapore, 2012, pp. 706-710.
[23] W. F. Mahmudy, R. M. Marian, and L. H. S. Luong, "Optimization of part type selection and loading problem with alternative production plans in flexible manufacturing system using hybrid genetic algorithms – Part 2: genetic operators & results," in *5th International Conference on Knowledge and Smart Technology (KST)*, Chonburi, Thailand, 2013, pp. 81-85.
[24] W. F. Mahmudy, R. M. Marian, and L. H. S. Luong, "Hybrid genetic algorithms for part type selection and machine loading problems with alternative production plans in flexible manufacturing system," *ECTI Transactions on Computer and Information Technology (ECTI - CIT)*, vol. 8, pp. 80-93, 2014.
[25] R. M. Marian, L. H. S. Luong, and K. Abhary, "A genetic algorithm for the optimisation of assembly sequences," *Comput. Ind. Eng.*, vol. 50, pp. 503-527, 2006.
[26] M. Lozano and F. Herrera, "Fuzzy adaptive genetic algorithms: design, taxonomy, and future directions," *Soft Computing*, vol. 7, pp. 545–562, 2003.