IEEE Access
The journal for rapid open access publishing

Received August 22, 2015, accepted September 6, 2015, date of current version October 8, 2015.

Digital Object Identifier 10.1109/ACCESS.2015.2481463

Solving the Flexible Job Shop Scheduling Problem With Makespan Optimization by Using a Hybrid Taguchi-Genetic Algorithm

HAO-CHIN CHANG, YEH-PENG CHEN, TUNG-KUAN LIU, AND JYH-HORNG CHOU
Institute of Engineering Science and Technology, National Kaohsiung First University of Science and Technology, Kaohsiung 824, Taiwan
Corresponding authors: T.-K. Liu (tkliu@nkfust.edu.tw) and H.-C. Chang (u0115905@nkfust.edu.tw)

This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grant 103-2221-E-327-031-, and in part by the Bureau of Energy, Ministry of Economic Affairs, Taiwan, under Grant 102-D0629.

ABSTRACT Enterprises exist in a competitive manufacturing environment. To reduce production costs and effectively use production capacity to improve competitiveness, a hybrid production system is necessary. The flexible job shop (FJS) is a hybrid production system, and the FJS problem (FJSP) has drawn considerable attention in the past few decades. This paper examined the FJSP and, like previous studies, aimed to minimize the total order completion time (makespan). We developed a novel method that involves encoding feasible solutions in the genes of the initial chromosomes of a genetic algorithm (GA) and embedding the Taguchi method behind mating to increase the effectiveness of the GA. Two numerical experiments were conducted for evaluating the performance of the proposed algorithm relative to that of the Brandimarte MK1–MK10 benchmarks. The first experiment involved comparing the proposed algorithm and the traditional GA. The second experiment entailed comparing the proposed algorithm with those presented in previous studies. The results demonstrate that the proposed algorithm is superior to those reported in previous studies (except for that of Zhang *et al.*: the results in experiment MK7 were superior to those of Zhang, the results in experiments MK6 and MK10 were slightly inferior to those of Zhang, and the results were equivalent in other experiments) and effectively overcomes the encoding problem that occurs when a GA is used to solve the FJSP.

INDEX TERMS Flexible job shop, genetic algorithm, optimization, Taguchi method.

I. INTRODUCTION

The flexible job shop scheduling problem (FJSP) has received considerable attention in recent years, because advanced machines have become prevalent. Assigning job operations to different machines for processing yields different results, and operations may be assigned incorrectly because of differences in machine type. In addition, the time for each operation required by different machines is different, and some machines can execute only specific operations. Therefore, an effective scheduling method is required for increasing the productivity of enterprises.

In the traditional manual scheduling approach, managers rely on their experience or relatively simple dispatching rules, and the result often is limited. For practical use, a set of nonhomogeneous and conflicting targets must be optimized; therefore, increasing complexity of scheduling problems has rendered manual scheduling and traditional metaheuristic algorithms inadequate, and developing a new scheduling method is necessary. In general, the FJSP involves the assumption that multiple parallel machines exist in a factory. Each type of machine has a different processing time for different operations; each job comprises several operations with a fixed route; and each machine can execute several operations.

For searching for solutions to solve the FJSP, this study proposes an algorithm in which a Taguchi–genetic method is embedded in the evolution phase to facilitate obtaining a high-quality offspring and effectively increase the convergence speed of the genetic algorithm (GA); therefore, the proposed algorithm is more likely to identify a near-optimal solution and avoid local optimal solutions. In recent years, numerous studies have used the GA to solve the FJSP. Borne *et al.* (2002) [5] proposed a localization approach to enable a user to assign each operation according to the process time and workloads of machines. They then applied advanced genetic manipulation based on

their assignment schema to improve the solution quality. To determine a robust and stable solution for the FJSP, Hinai and ElMekkawy (2011) [1] used a two-stage hybrid GA to generate a predictive schedule. The first stage involves optimizing the makespan, and the second stage involves optimizing the bi-objective function and integrating machine assignments and operations in sequence with the expected machine breakdown in the decoding space. The results indicated that different measures exhibited substantially different performance. Wang and Chu (2012) [13] proposed a novel encoding method that divides a chromosome into two parts, operation assignment (OA) and machine selection (MS), and designed different strategies for the crossover and mutation operators. A scenario with eight jobs and eight machines was used to test the application of the approach. The results showed that the proposed method can effectively solve FJSPs. Tung-Kuan *et al.* (2014) [11], [12] used two GAs to solve FJSPs for a midscale screw manufacturer. The proposed refined GA was evaluated, and satisfactory results were obtained through two-stage validation; in the first stage, a classical DFJSP was used to show the effectiveness of the algorithm, and in the second stage, the algorithm was used to solve a real-world case. The results for the real-world case were satisfactory. Karimi-Nasab [6] proposed a modified particle swarm optimization (PSO) algorithm for the FJSP. He tested various benchmark data from the literature. Li [7] presented a novel discrete artificial bee colony algorithm. This is a unique solution representation in which a food source is represented by two discrete vectors, and a tabu search (TS) is applied to each food source to generate neighboring food sources for employed bees, onlooker bees, and scout bees. Nasiri suggested a modified ABC algorithm. The effective neighborhood of the stage shop problem and PSO were used for the employed and onlooker bee phases, respectively [9]. Zeng proposed two-integer nonlinear programming models that combine an improved timetabling method and local search for solving the blocking job shop automated guided vehicle problem [14]. Zhao proposed an intelligent algorithm called the improved shuffled complex evolution algorithm. However, this algorithm obtains a poor solution and low convergence rate. Therefore, a new strategy was used to change the individual's evolution in the basic shuffled complex evolution algorithm. This strategy makes the new individual closer to the optimal individual in the current population [16]. Moin *et al.* presented multiparent crossover in the hybrid GA. The multiparent crossover operator recombines more than two parents to generate a single new offspring [17]. Xiong *et al.* solved the job shop problem (JSP) by using a novel quadspace cultural genetic tabu algorithm. This algorithm provides a structure different from the original cultural algorithm with respect to double brief spaces and population spaces [18]. Tasi *et al.* proposed a hybrid Taguchi–genetic algorithm (HTGA) for solving the JSP [4]. Xu *et al.* proposed a two-level batch chromosome coding scheme for solving the FJSP by using the hybrid discrete differential evolution algorithm [19]. Tasi *et al.* used the Taguchi method to solve global numerical optimization problems with continuous variables [20].

In contrast to the traditional JSP, the FJSP considers the selection of a machine, which increases the complexity of the scheduling problem. This makes encoding solution spaces into the initial chromosome difficult, causing the generation of an infeasible solution; in other words, incorrect machine assignment easily occurs during evolution. Therefore, a special mechanism is required for crossover and mutation. In addition, although the traditional genetic algorithm (TGA) has a strong search capability, its performance can be improved further when the problem scale is large; thus a method for determining an effective near-optimal solution with robust and rapid convergence should be developed. Therefore, this study proposes a refined HTGA to address these concerns. The Taguchi method is used to identify the optimal chromosome combination after crossover for expediting convergence.

To verify the performance of the proposed refined HTGA, 10 benchmarks proposed by Brandimarte [2], MK1–MK10, were used. Overall, the results of this study were more favorable than those of previous studies [3], [4], [10], [15]. The remainder of this paper is organized as follows: Section 2 reviews studies on the FJS. Section 3 describes the HTGA and explains how our algorithm is used to solve the FJSP. Section 4 discusses the results obtained using the MK1–MK10 benchmarks and a comparison of convergence speeds and optimal fitness values. Section 5 concludes the paper and provides suggestions for future research.

II. PROBLEM FORMULATION AND HYPOTHESIS

The FJSP entails organizing the execution of N jobs on M machines. A set of machines is represented by U. Each job J_i contains a number of unchangeable ordered operations $O_{i,j} \subseteq O_i$, where $O_{i,j}$ represents the jth operation of the ith job. Each $O_{i,j}$ requires at least one machine for processing from a set of available machines $U_{i,j} \subseteq U$. The processing time of operation $O_{i,j}$ assigned to machine M_k ($M_k \in U_{i,j}$) is represented as $P_{i,j,k}$ [5], where job $J_i = \{1 \leq i \leq N\}$, operations $O_{i,j} = \{1 \leq i \leq N; 1 \leq j \leq O_i\}$, and machines $M_k = \{1 \leq k \leq M\}$.

The assumptions regarding the FJSP are listed as follows:

1) The job J has N operations that must be processed according to a predefined sequence.
2) During the operation of a machine, regardless of crashes, damage, or material shortages, once each operation is in line, neither stop action nor reprocessing is allowed.
3) The execution of operation $O_{i,j} \in O_i$, $O_i = \{O_{i,1}, O_{i,2}, \ldots, O_i\}$ of job $J_i \in J$, $J = \{J_1, J_2, \ldots, J\}$ selected from a set of available machines $U_{i,j}$.
4) Each machine can perform only one operation $O_{i,j}$ at a time and the transportation times between machines are neglected.

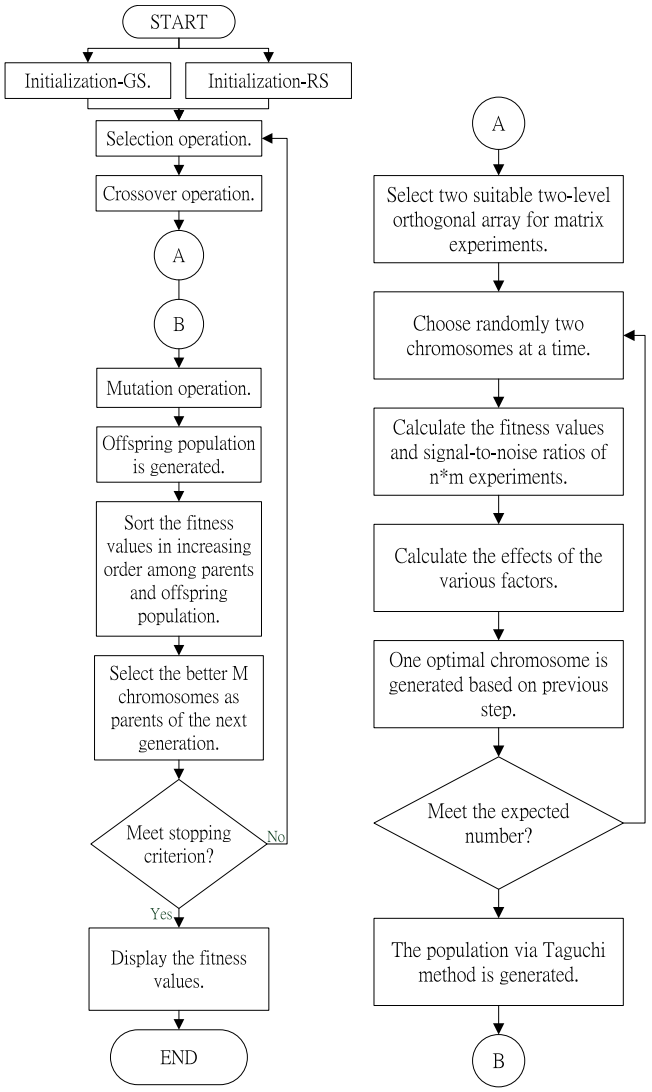


FIGURE 1. HTGA flow chart.

5) All operations with the same job number enter the production facility in sequence.
6) Time is discretized into the set $d_{i,j,k}$ of time slots.

III. PROPOSED APPROACH

This study aimed to minimize the total order completion time (makespan) by using the HTGA to solve the FJSP. The results obtained using TGAs to solve JSPs are usually satisfactory. However, because FJSPs belong to a more complex scheduling problem category (FJSs involve unrelated parallel machines), the method used for solving FJSPs must consider constraints; for example, a specific machine can execute only specific operations rather than all operations. Therefore, TGAs cannot be used to solve the FJSP. Thus, for solving the GA problem in FJS scenarios, we propose a novel method that involves encoding the FJSP solution in the initial chromosome. In addition, we investigated the difference between the HTGA and the TGA. Fig. 1 shows a flow chart of the proposed HTGA. The entire procedure of the proposed algorithm is detailed as follows:

A. ENCODING THE PROBLEM

A chromosome comprises two sections: OA and MS. In OA, each gene represents an operation of a job; in MS, each gene represents a machine number. The lengths of both OA and MS chromosomes are equal to the total number of operations of jobs. OA and MS then merge to form a single chromosome, and each chromosome represents a feasible solution.

The encoding of the FJSP solutions into the GA chromosome is detailed as follows.

In the OA encoding method, the first step is ranking all operations of jobs according to the job and operation numbers. For example, the classic FJSP instance MK1 comprises 55 operations in 10 jobs; therefore, the OA length is 55, and a number between zero and one is randomly generated for each operation until the length of the gene is equal to the OA length. Subsequently, rank and job numbers are assigned to each OA gene. Fig. 2 shows the entire process and its results.

Regarding the order of the process and initial OA chromosome, Fig. 2 illustrates that the value of the second gene is 0.0540 and the job number is 7 because job number 7 appears for the first time. This means that this gene represents Operation 1 of Job 7, O_{71}. The value of the third gene is 0.0759 and the job number is 7; thus, this gene represents Operation 2 of Job 7, O_{72}.

In the MS encoding method, the first step is the generation of an initial array, the length of which is equal to the total number of operations based on the order of job and operation numbers. A number between zero and one is then randomly generated for each operation to determine the probability for selecting a feasible machine until a suitable length of MS is obtained (Fig. 3(a)).

Fig. 3(b) shows the entire process and results. Operation 2 of Job 1, O_{12}, can be performed by Machine 2, 3, or 5, according to the MK1 instance. Three machines, ordered from small to large, with the same probability can be used to execute O_{12}. The value of O_{12} was 0.0838, and Machine 2 was located between 0 and 0.333; therefore, Machine 2 was selected.

B. FITNESS FUNCTIONS

In this study, we used makespan as a main parameter of the fitness function; its formula is as follows:

$$C_{\max} = Min\left\{Max.\sum_{i=1}^{n}\sum_{j=1}^{m}\left(T_{i,j,k}+P_{i,j,k}\right)\right\}, \quad (3.1)$$

where $C_{\max}$ represents the minimal total order makespan, $T_{i,j,k}$ represents the start time of operation j for job i in machine k; and $P_{i,j,k}$ represents the processing time of operation j for job i in machine k.

Fig. 4 shows a Gantt diagram of instance MK1. In the first pink rectangle of Fig. 4, "1-1" represents the first

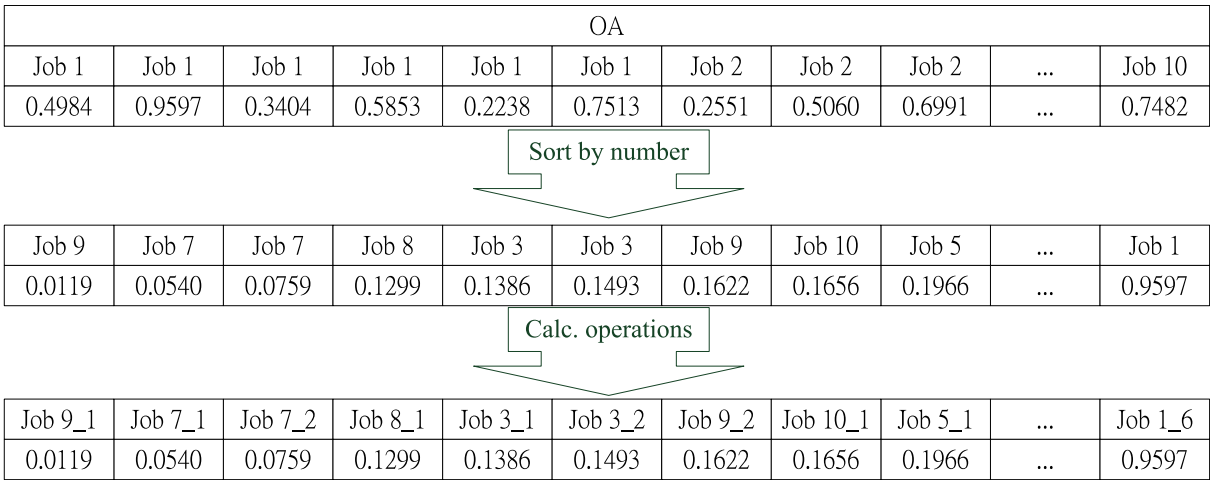


FIGURE 2. OA chromosome of the instance MK1.

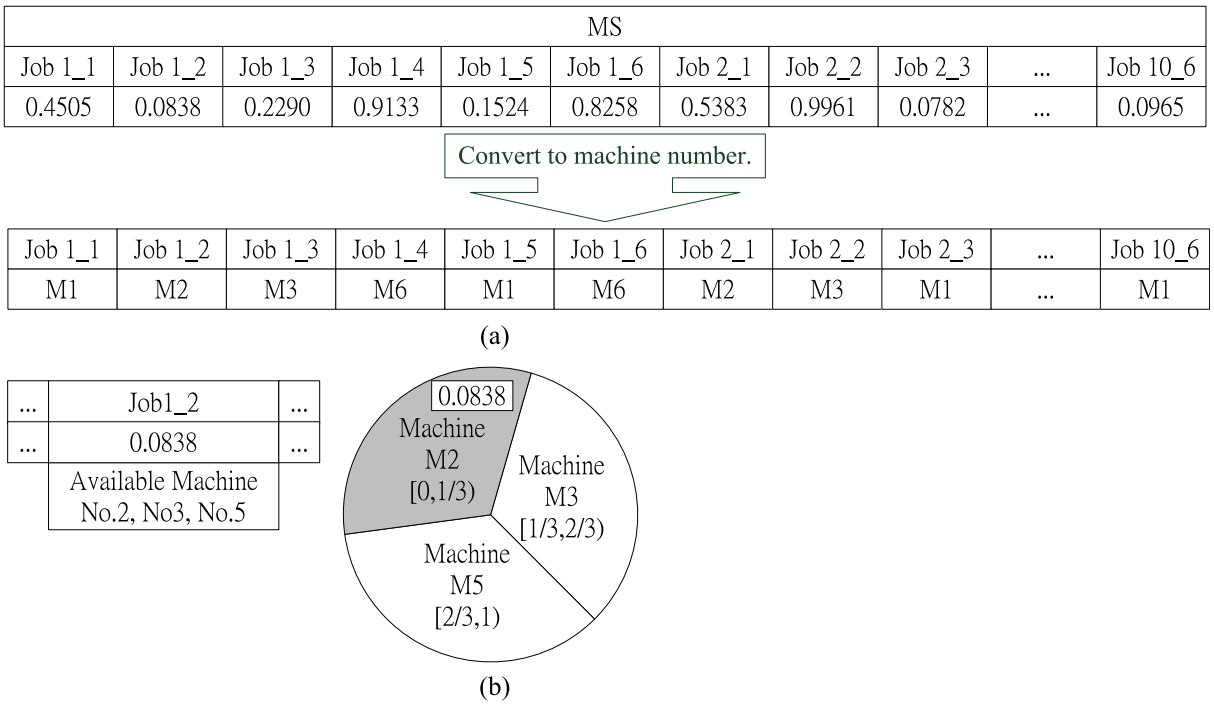


FIGURE 3. (a) MS chromosome of the instance MK1. (b) MS chromosome (second gene) of the instance MK1.

operation of Job 1. Among all machines in Fig. 4, the maximum completion time occurs on the second and fourth machines. The time (makespan) is 74 time units, meaning that the fitness value is 74 time units.

This study proposes a method for optimizing the permutation and combination of machine operations and optimizing the makespan.

C. INITIAL POPULATION

Calculating the initial population is crucial for GAs because it directly influences the convergence rate of fitness values and the quality of optimal solutions. Two selection methods (global selection and random selection) are embedded in the initial population of the proposed HTGA to generate the initial population (parent chromosomes). These two methods are used to generate 50% of the initial population.

1) GLOBAL SELECTION: THE GLOBAL SELECTION METHOD IS AS FOLLOWS

Step 1: The sequential order of operations for individual jobs is determined randomly.

Step 2: The matrix that shows processing times on all machines is established with an initial value of zero.

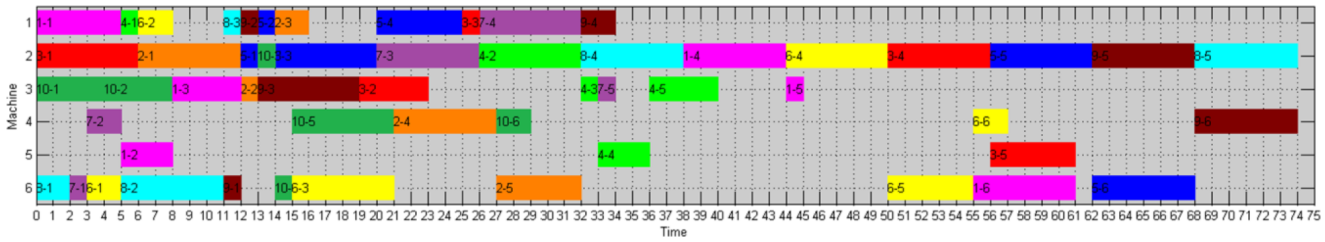


FIGURE 4. Gantt diagram for the instance MK1 obtained using proposed algorithm.

Step 3: Genes in the chromosome are read sequentially.
Step 4: The processing times of the available machines for the corresponding operations of the genes and the current cumulative times of the corresponding machines are shown in the matrix established in Step 2. These times are then added to obtain the cumulative matrix, Temp.
Step 5: Times on individual available machines in Temp are compared, and the machine k with the shortest length of cumulative time is selected. If available machines have identical cumulative and processing times, one of them is randomly selected.
Step 6: The processing time of machine k selected in the previous step is renewed and added to the matrix in Step 2.
Step 7: The next gene in the chromosome is read. Steps 3–6 are repeated until the last gene of the chromosome is read.
Step 8: The process is repeated from Step 2 and all remaining chromosomes are read.

2) RANDOM SELECTION: THE RANDOM SELECTION METHOD IS AS FOLLOWS

Step 1: Identify all feasible machines for each operation and compile them into set O.
Step 2: According to the number of operations, randomly generate OA chromosomes until population N is reached.
Step 3: On the basis of the number of OA chromosomes and available machine table, randomly generate MS chromosomes until population N is reached.
Step 4: Calculate the fitness of chromosomes according to the OA chromosomes, MS chromosomes, and processing time table.

D. REPRODUCTION

The roulette wheel method is used to generate an offspring; the greater the fitness function value, the greater the area of the wheel is. This means that higher fitness values have a higher probability of being selected.

E. CROSSOVER

The two methods used to perform OA and MS are two-point crossover and uniform crossover.

1) MS

(1) Two-point crossover:

Step 1: Two nonrepetitive genes are randomly selected from the chromosome.

Step 2: The two genes and those between them are arrayed in reverse order to generate a new chromosome.

(2) Uniform crossover:

Step 1: A number is randomly generated to determine the even- or odd-numbered genes in the chromosome to be used in the crossover operation.

Step 2: Another chromosome is randomly selected, and genes at identical positions are exchanged.

2) OA

(1) Two-point crossover:

Step 1: Two nonrepetitive genes are randomly selected from the chromosome.

Step 2: The two genes and those between them are arrayed in reverse order to generate a new chromosome.

(2) Uniform crossover:

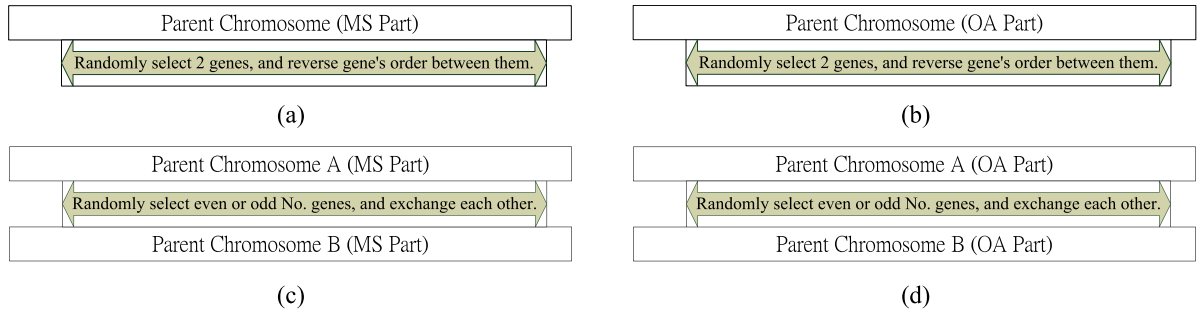
Step 1: A number is randomly generated to determine the even- or odd-numbered genes in the chromosome to be used in the crossover operation.

Step 2: Another chromosome is randomly selected, and genes at identical positions are exchanged.

F. TAGUCHI OPTIMIZATION METHOD

Because the orthogonal arrays of the Taguchi method, the Taguchi method is useful for reducing experiment time and increasing convergence speed. It was used in the proposed HTGA. The better combinations are decided by the orthogonal arrays and the signal-to-noise (S/N) ratios. The procedure is as follows:

Step 1: Select an appropriate two-level orthogonal array for matrix experiments and two chromosomes for each matrix experiment. The size of the orthogonal array is determined according to the size of the FJSP.
Step 2: Calculate the S/N ratios and the fitness values of all combinations in the matrix experiments. Select an optimal chromosome according to the S/N ratio.
Step 3: Verify that the generation of all populations in the crossover operation is complete. If not, return to Step 1.
Step 4: Generate a new population.



								MS									
								Exp. 1	Exp. 2	Exp. 3	Exp. 4	Exp. 5	Exp. 6	...	Exp. 64		
								Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	...	Chrom 2	Job 1_1	
								Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	...	Chrom 2	Job 1_2	
								Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	...	Chrom 1	Job 1_3	
								Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	...	Chrom 2	Job 1_4	
								Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	...	Chrom 1	Job 1_5	
	OA							⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
	Job 1	Job 1	Job 1	Job 1	Job 1	...	Job 10	Chrom 1	Chrom 2	Chrom 2	Chrom 1	Chrom 1	Chrom 2	...	Chrom 2	Job 10_6	
Exp. 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	...	Chrom 1	S/N(1,1)	S/N(1,2)	S/N(1,3)	S/N(1,4)	S/N(1,5)	S/N(1,6)	...	S/N(1,64)		
Exp. 2	Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	...	Chrom 2	S/N(2,1)	S/N(2,2)	S/N(2,3)	S/N(2,4)	S/N(2,5)	S/N(2,6)	...	S/N(2,64)		
Exp. 3	Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	...	Chrom 2	S/N(3,1)	S/N(3,2)	S/N(3,3)	S/N(3,4)	S/N(3,5)	S/N(3,6)	...	S/N(3,64)		
Exp. 4	Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	...	Chrom 1	S/N(4,1)	S/N(4,2)	S/N(4,3)	S/N(4,4)	S/N(4,5)	S/N(4,6)	...	S/N(4,64)		
Exp. 5	Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	...	Chrom 1	S/N(5,1)	S/N(5,2)	S/N(5,3)	S/N(5,4)	S/N(5,5)	S/N(5,6)	...	S/N(5,64)		
Exp. 6	Chrom 1	Chrom 1	Chrom 1	Chrom 1	Chrom 1	...	Chrom 2	S/N(6,1)	S/N(6,2)	S/N(6,3)	S/N(6,4)	S/N(6,5)	S/N(6,6)	...	S/N(6,64)		
⋮	⋮	⋮	⋮	⋮	⋮	...	⋮	⋮	⋮	⋮	⋮	⋮	⋮	...	⋮	⋮	⋮
Exp. 64	Chrom 2	Chrom 2	Chrom 1	Chrom 2	Chrom 1	...	Chrom 2	S/N(64,1)	S/N(64,2)	S/N(64,3)	S/N(64,4)	S/N(64,5)	S/N(64,6)	...	S/N(64,64)		
						OA EF 1								...		MS EF 1	MS EF 2
						OA EF 2								...			

(e)

FIGURE 5. **(a) Two-point crossover of machine selection. (b) Two-point crossover of operation assignment. (c) Uniform crossover of machine selection. (d) Uniform crossover of operation assignment. (e) Results obtained using the Taguchi method for selecting an optimal chromosome combination based on the instance MK1.**

Fig. 5(e) shows the result obtained using the Taguchi method to select the optimal chromosome combination based on the instance MK1. Because of the limitation of length, for more Taguchi method detail please refer to [4] and [20].

G. MUTATION METHOD

The mutation approach changes specific genes in the chromosome to increase the search space of the solution. In this study, two OA mutation methods, random selection and neighborhood search, were used for performing MS and OA mutation, respectively.

1) NEIGHBORHOOD SEARCH

Step 1: A chromosome is randomly selected from parent chromosomes, and a random number [0,1) is generated. If the corresponding random number of a gene is smaller than the preset mutation rate, Step 2 is performed. Otherwise, Step 2 is skipped.

Step 2: The machine with the shortest processing time among those available for the corresponding operation of the gene is selected, and the selected machine is restored to the decimal range in which the machine falls.

Step 3: A number N is randomly generated and must be lower than the chromosome length.

TABLE 1. **Parameters of the orthogonal array for benchmarks MK1–MK10.**

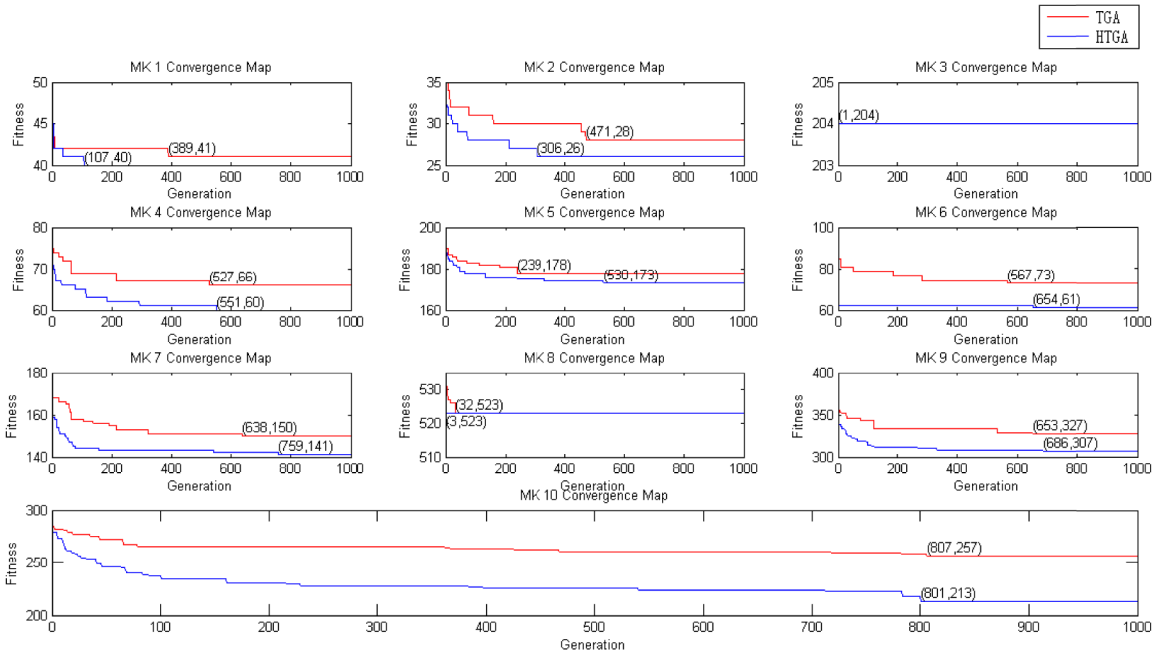
MK	Description			Orthogonal Arrays	
	Job	operation	Machine	Job	Machine
1	10	55	6	$L_{16}(2^{15})$	$L_{64}(2^{63})$
2	10	58	6	$L_{16}(2^{15})$	$L_{64}(2^{63})$
3	15	150	8	$L_{16}(2^{15})$	$L_{256}(2^{255})$
4	15	90	8	$L_{16}(2^{15})$	$L_{128}(2^{127})$
5	15	106	4	$L_{16}(2^{15})$	$L_{128}(2^{127})$
6	10	150	15	$L_{16}(2^{15})$	$L_{256}(2^{255})$
7	20	100	5	$L_{32}(2^{31})$	$L_{128}(2^{127})$
8	20	225	10	$L_{32}(2^{31})$	$L_{256}(2^{255})$
9	20	240	10	$L_{32}(2^{31})$	$L_{256}(2^{255})$
10	20	240	15	$L_{32}(2^{31})$	$L_{256}(2^{255})$

Step 4: Permutations and combinations based on N genes are assessed.

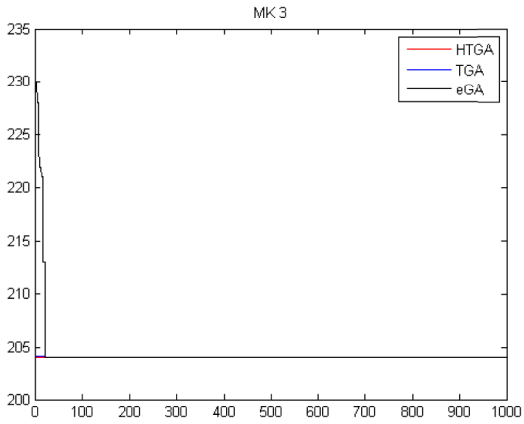
Step 5: The fitness values of all chromosomes are calculated after permutations and combinations, and the optimal chromosome is selected for generation. The process is now complete.

2) RANDOM SELECTION

Step 1: Genes on all parent chromosomes are read sequentially, and random numbers [0,1) are generated.



(a)



(b)

FIGURE 6. **(a) Fitness values obtained using the TGA and HTGA for benchmarks MK1–MK10. (b) Comparison of convergence speed among proposed enhanced HTGA, TGA and eGA.**

If the corresponding random number of a gene is smaller than the preset mutation rate, Step 2 is performed. Otherwise, Step 2 is skipped.

Step 2: The integer part of the gene is retained, and a random number [0,1) is generated again for the decimal.

Step 3: After all genes on all parent chromosomes are read, the process is complete. Otherwise, the process is repeated.

H. SELECTION

Step 1: A number X is predefined to perform chromosome selection.

Step 2: The fitness values of all parent chromosomes are calculated, and the chromosomes are sorted according to these values.

Step 3: Chromosomes with poor fitness values (i.e., values lower than X) are removed.

I. TERMINATION

Because of an excessive number of orders, determining the optimal solution is impossible. Therefore, the iterations were used as the termination conditions in this study.

IV. NUMERICAL EXPERIMENTS

Two experiments based on the classic FJSP Brandimarte [2] benchmarks MK1–MK10 were performed to validate the

IEEE Access
The journal for rapid open access publishing

TABLE 2. Comparison of the results obtained using the HTGA for benchmarks MK1–MK10 and those of algorithms developed in previous studies.

MK	Description			TGA [3]	EDPSO [10]	TS [10]	PSO [10]	PSO+TS [10]	MATSLO [10]	eGA [15]	HTGA
	Job	operation	Machine								
1	10	55	6	41	41	42	**40**	**40**	**40**	**40**	**40**
2	10	58	6	28	**26**	32	27	27	32	**26**	**26**
3	15	150	8	**204**	207	211	**204**	**204**	207	**204**	**204**
4	15	90	8	66	65	81	62	63	67	**60**	**60**
5	15	106	4	178	**171**	186	178	173	188	173	173
6	10	150	15	73	61	86	78	65	85	**58**	61
7	20	100	5	150	173	157	147	145	154	145	**141**
8	20	225	10	523	523	523	523	523	523	523	523
9	20	240	10	327	**307**	369	341	331	437	**307**	**307**
10	20	240	15	257	312	296	252	223	380	**198**	213

proposed HTGA. The first experiment was conducted to illustrate the difference between the convergence speeds obtained using the standard GA and HTGA. The second experiment was conducted to evaluate the proposed HTGA according to benchmarks MK1–MK10. In addition, several algorithms were compared. The algorithms were implemented in Matlab 7 on a CPU Intel®Core ™i7-2630 running Microsoft Visual Studio 2010 Express at 2.0 GHz. The visual C # programming language was used to code the production scheduling system.

A. RESULTS AND DISCUSSION

1) COMPARISON OF THE CONVERGENCE SPEED OBTAINED USING THE GA AND HTGA

The problem sizes of benchmarks MK1–MK10 are introduced in the first, second, third, and fourth columns of Table 1. Table 1 lists the parameters of the orthogonal array for benchmarks MK1–MK10. For example, two 2-level orthogonal arrays, $L_{16}(2^{15})$ and $L_{64}(2^{63})$, were used for OA and MS in MK1 because it comprised 16 jobs and 55 operations (16×55), and two 2-level orthogonal arrays, $L_{32}(2^{31})$ and $L_{256}(2^{255})$, were used for OA and MS in MK8 because it comprised 20 jobs and 225 operations. The size of the selected orthogonal table corresponds to the size of the problem.

The general symbol for two-level standard orthogonal arrays is [4]

$$L_n(2^{n-1}), \tag{3.2}$$

where

$n = 2k$	number of experiment runs;
k	a positive integer greater than 1;
2	number of levels for each factor;
$n - 1$	number of columns in the orthogonal array.

The letter "L" is derived from "Latin," the concept of using orthogonal arrays for experimental design having been associated with Latin square designs from the outset.

Fig. 6(a) shows fitness value and convergence speed diagrams obtained using the TGA and HTGA. The results indicate that the HTGA outperformed the TGA in both convergence speed and makespan. This experiment proved that embedding the Taguchi method can increase the convergence rate and the opportunity to discover near-optimal or optimal solutions.

2) COMPARISON OF THE MAKESPANS OBTAINED USING THE PROPOSED HTGA AND THOSE OF ALGORITHMS DEVELOPED IN PREVIOUS STUDIES

Table 2 shows a comparison of the results obtained using the HTGA for benchmarks MK1–MK10 and those obtained in previous studies.

Figs. 7–16 show the results for benchmarks MK1–MK10 obtained using the HTGA, respectively. The results indicate that the results obtained using the eGA [15] algorithm were the most satisfactory.

First, the HTGA and TGA were compared. The results obtained using these two algorithms were identical for MK3 and MK8. However, for MK1, MK2, MK4–MK7, MK9, and MK10, the makespans obtained using the TGA were 41, 28, 66, 178, 73, 150, 327, and 257, respectively, and those obtained using the HTGA were 40, 26, 60, 173, 61, 141, 307, and 213, respectively, indicating that the HTGA outperformed the TGA.

Second, the HTGA and EDPSO algorithm were compared, and the results obtained using these two algorithms were identical for MK2, MK3, MK6, MK8, and MK9. For MK5, the makespans obtained using the EDPSO algorithm and HTGA were 171 and 173, respectively. Thus, the EDPSO algorithm is superior to the HTGA in this scenario. However, for MK1, MK4, MK7, and MK10, the makespans obtained using the EDPSO algorithm were 41, 65, 173, and 312, respectively, and those obtained using the HTGA were 40, 60, 141, and 213, respectively, indicating that the HTGA outperformed the EDPSO algorithm.

Third, the HTGA and TS algorithms were compared. The results obtained using these two algorithms were identical for MK8. However, for MK1–MK7, MK9, and MK10, the makespans obtained using the TS algorithm were 42, 32, 211, 81, 186, 86, 157, 369, and 296, respectively, and those

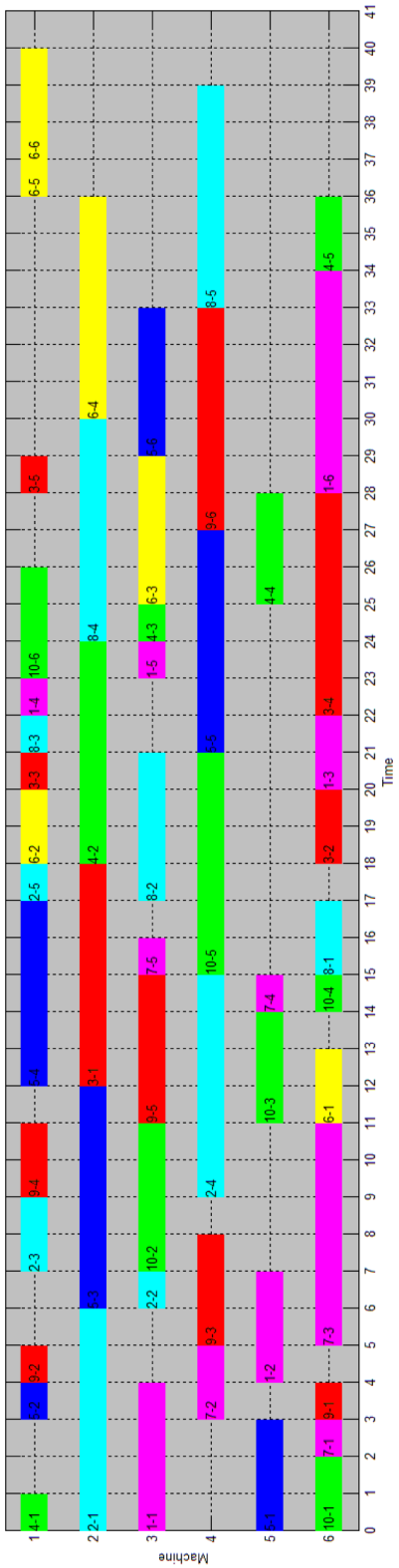


FIGURE 7. Gantt chart for MK1.

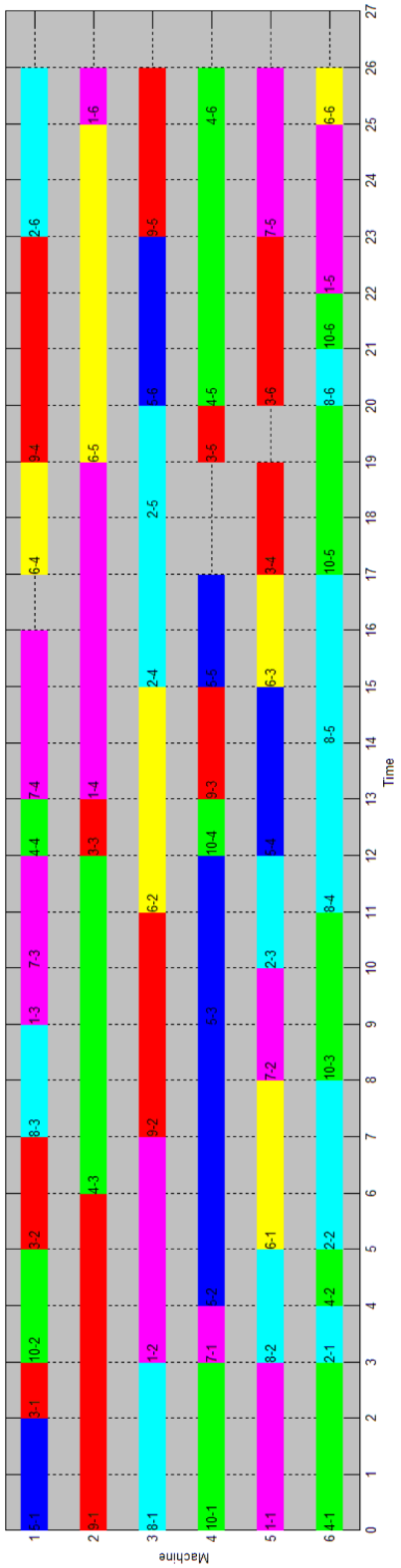


FIGURE 8. Gantt chart for MK2.

obtained using the HTGA were 40, 26, 204, 60, 173, 61, 141, 307, and 213, respectively, indicating that the HTGA outperformed the TS algorithm.

Fourth, the HTGA and PSO algorithms were compared. The results obtained using these two algorithms were identical for MK1, MK3, and MK8. However, for

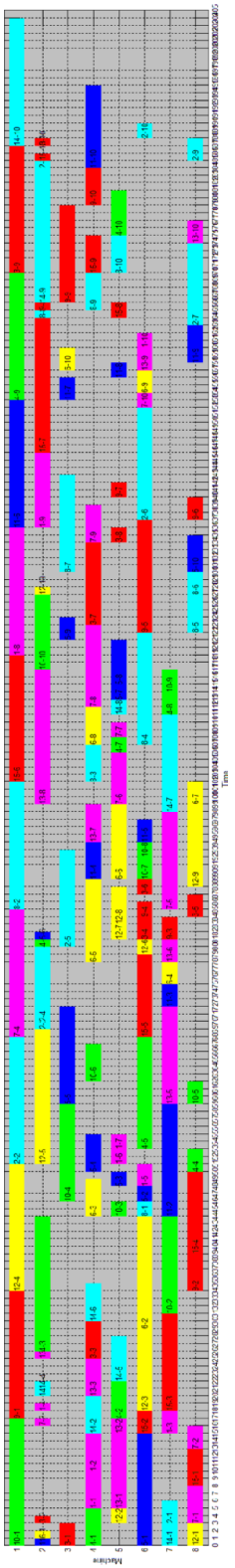


FIGURE 9. Gantt chart for MK3.

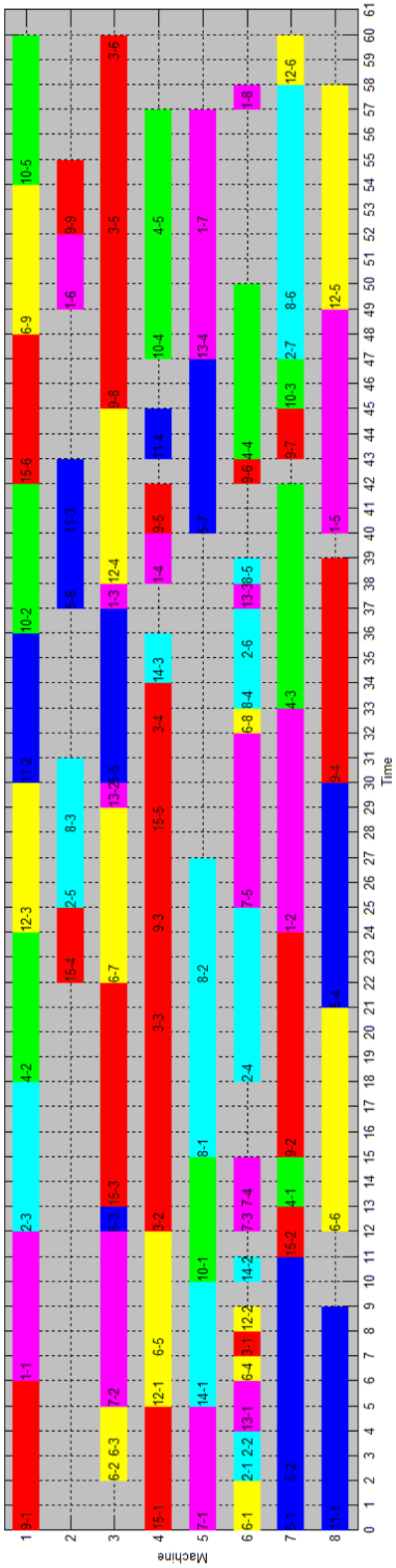


FIGURE 10. Gantt chart for MK4.

MK2, MK4–MK7, MK9, and MK10, the makespans obtained using the PSO algorithm were 27, 62, 178, 78, 147, 341, and 252, respectively, and those obtained using the HTGA were 26, 60, 173, 61, 141, 307, and 213, respectively, indicating that the HTGA outperformed the PSO algorithm.

Fifth, the HTGA and PSO+TS algorithms were compared. The results obtained using these two algorithms

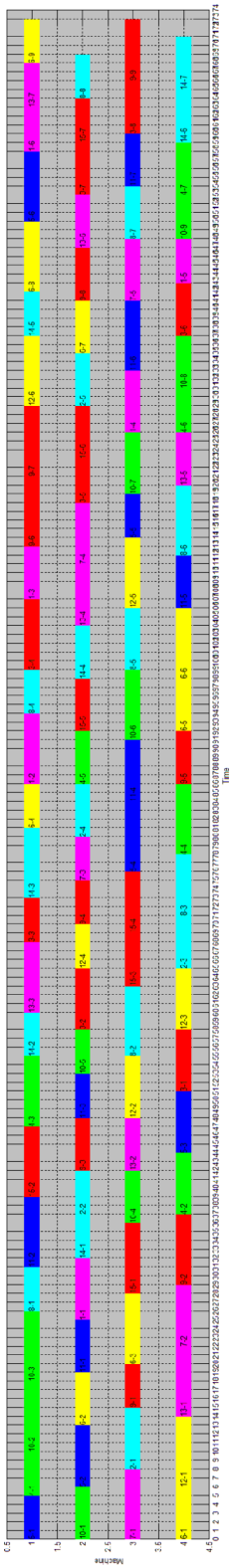


FIGURE 11. Gantt chart for MK5.

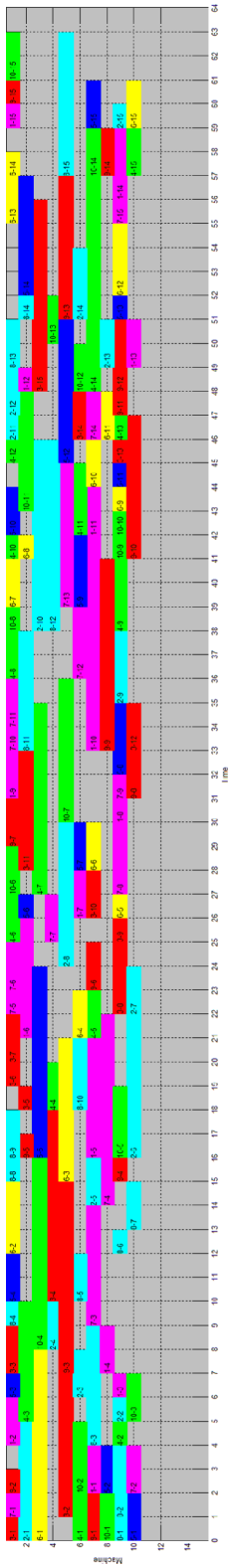


FIGURE 12. Gantt chart for MK6.

were identical for MK1, MK3, MK5, and MK8. However, for MK2, MK4, MK6, MK7, MK9, and MK10, the makespans obtained using the PSO+TS algorithm were 27, 63, 65, 145, 331, and 223, respectively, and those obtained using the HTGA were 26, 60, 61, 141, 307, and 213, respectively,

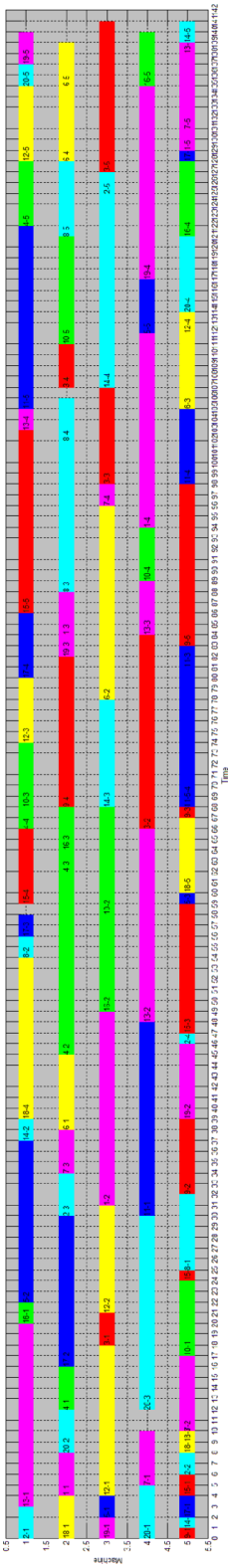


FIGURE 13. Gantt chart for MK7.

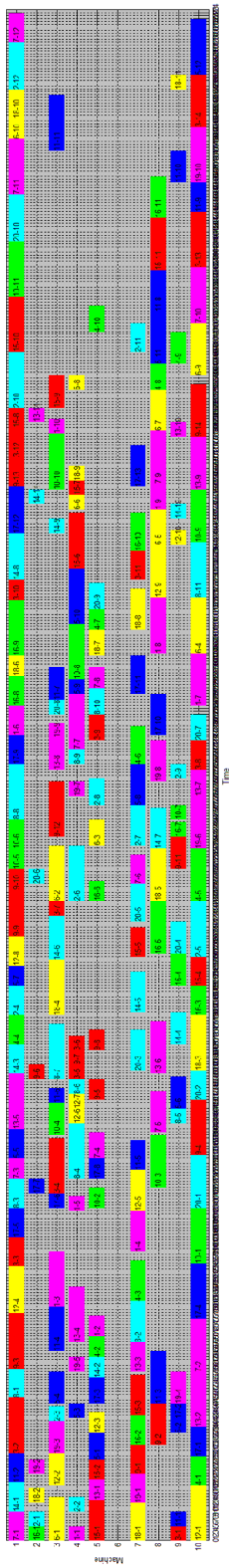


FIGURE 14. Gantt chart for MK8.

indicating that the HTGA outperformed the PSO+TS algorithm.

Sixth, the HTGA and MATSLO algorithms were compared. The results obtained using these two algorithms were identical for MK1 and MK8. However, for MK2–MK7, MK9, and MK10, the makespans obtained using the MATSLO algorithm were 32, 207, 67, 188, 85, 154, 437, and 380, respectively, and those obtained using the HTGA

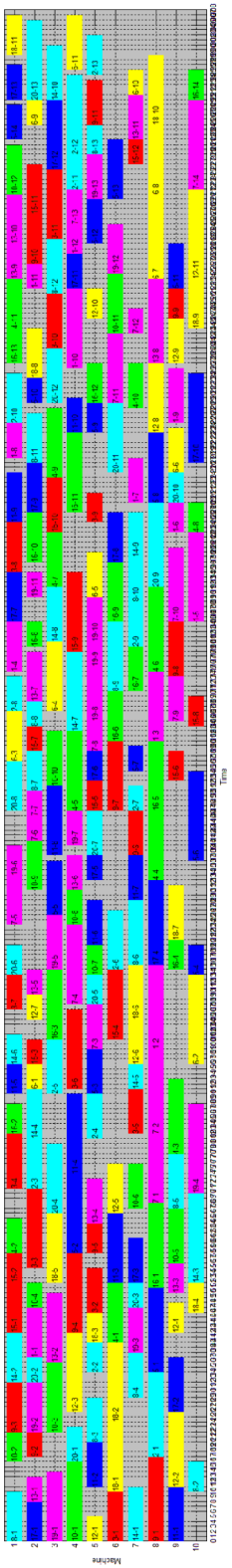


FIGURE 15. Gantt chart for MK9.

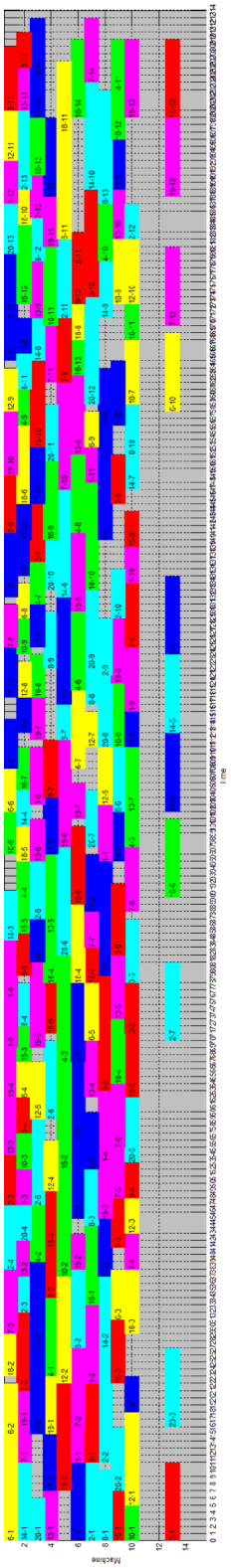


FIGURE 16. Gantt chart for MK10.

were 26, 204, 60, 173, 61, 141, 307, and 213, respectively, indicating that the HTGA outperformed the MATSLO algorithm.

Finally, the HTGA and eGA algorithms were compared. The results obtained using these two algorithms were identical for MK1–MK5, MK8, and MK9. For MK6 and MK10,

the makespans obtained using the eGA algorithm were 58 and 198, respectively, and those obtained using the HTGA were 61 and 213, respectively. Thus, the eGA algorithm is superior to the HTGA in these scenarios. However, for MK7, the makespans obtained using the eGA algorithm and HTGA were 145 and 141, respectively, indicating that the HTGA outperformed the eGA algorithm.

The results demonstrate that the proposed algorithm is not only relatively superior to previous studies reported except Zhang et al. [15] (the result of experiment MK7 is better than Zhang, the results of experiments MK6 and MK10 are slightly inferior to Zhang, the others are same), but also effective for overcoming the encoding problem that occurs when a GA is used to solve the FJSP; we also compared the convergence speed between HTGA and eGA, by coding the program based on the algorithm of eGA in [15]; the results show the HTGA outperformed eGA. The simplest instance MK3 was showed in Fig. 6(b). In addition, Figure 6 shows the convergence speed is outperformed TGA in numerical experiments MK1-Mk10 of FJSP.

V. CONCLUSION

In this study, an enhanced HTGA embedded a novel encoding method for FJSP has been proposed. The enhanced HTGA, were developed. The HTGA overcomes the limitation of the TGA in solving the FJSP; that is, it avoids unfeasible solutions and has an increased GA convergence speed. Although the optimal solution determined in this study is close to that determined by Zhang, but the scheduling results of the enhanced HTGA were presented by using Gantt chart (Figs 7 - 16), it proved the solutions are available. Our proposed algorithm is superior to other algorithms.

We successfully solved the FJSP and produced a feasible solution; however, in practice, some aspects of the method are still inadequate and thus much work, such as multiobjective optimization, remains for enhancing practicality.

REFERENCES

[1] N. Al-Hinai and T. Y. ElMekkawy, "Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm," *Int. J. Prod. Econ.*, vol. 132, no. 2, pp. 279–291, 2011.
[2] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Ann. Oper. Res.*, vol. 41, no. 3, pp. 157–183, 1993.
[3] H.-C. Chang, H.-T. Tsai, and T.-K. Liu, "Application of genetic algorithm to optimize unrelated parallel machines of flexible job-shop scheduling problem," in *Proc. 11th IEEE Int. Conf. Control Autom. (ICCA)*, Jun. 2014, pp. 596–599.
[4] J.-T. Tsai, T.-K. Liu, and J.-H. Chou, "Hybrid Taguchi-genetic algorithm for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 4, pp. 365–377, Aug. 2004.
[5] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 32, no. 1, pp. 1–13, Feb. 2002.
[6] M. Karimi-Nasab, M. Modarres, and S. M. Seyedhoseini, "A self-adaptive PSO for joint lot sizing and job shop scheduling with compressible process times," *Appl. Soft Comput.*, vol. 27, pp. 137–147, Feb. 2015.
[7] J.-Q. Li, Q.-K. Pan, and M. F. Tasgetiren, "A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities," *Appl. Math. Model.*, vol. 38, no. 3, pp. 1111–1132, Feb. 2014.
[8] T.-K. Liu, J.-T. Tsai, and J.-H. Chou, "Improved genetic algorithm for the job-shop scheduling problem," *Int. J. Adv. Manuf. Technol.*, vol. 27, nos. 9–10, pp. 1021–1029, 2006.
[9] M. M. Nasiri, "A modified ABC algorithm for the stage shop scheduling problem," *Appl. Soft Comput.*, vol. 28, pp. 81–89, Mar. 2015.
[10] M. Nouiri, A. Bekrar, A. Jemai, S. Niar, and A. C. Ammari, "An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem," *J. Intell. Manuf.*, pp. 1–13, Feb. 2015.
[11] T.-K. Liu, Y.-P. Chen, and J.-H. Chou, "Developing a multiobjective optimization scheduling system for a screw manufacturer: A refined genetic algorithm approach," *IEEE Access*, vol. 2, no. 4, pp. 356–364, 2014.
[12] T.-K. Liu, Y.-P. Chen, and J.-H. Chou, "Solving distributed and flexible job-shop scheduling problems for a real-world fastener manufacturer," *IEEE Access*, vol. 2, no. 1, pp. 1598–1606, 2014.
[13] J. Wang and K. Chu, "An application of genetic algorithms for the flexible job-shop scheduling problem," *Int. J. Adv. Comput. Technol.*, vol. 4, no. 3, pp. 271–278, 2012.
[14] C. Zeng, J. Tang, and C. Yan, "Scheduling of no buffer job shop cells with blocking constraints and automated guided vehicles," *Appl. Soft Comput.*, vol. 24, pp. 1033–1046, Nov. 2014.
[15] G. Zhang, L. Gao, and Y. Shi, "An effective genetic algorithm for the flexible job-shop scheduling problem," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 3563–3573, 2011.
[16] F. Zhao, J. Zhang, C. Zhang, and J. Wang, "An improved shuffled complex evolution algorithm with sequence mapping mechanism for job shop scheduling problems," *Expert Syst. Appl.*, vol. 42, no. 8, pp. 3953–3966, 2015.
[17] N. H. Moin, O. C. Sin, and M. Omar, "Hybrid genetic algorithm with multiparents crossover for job shop scheduling problems," *Math. Problems Eng.*, vol. 2015, 2015, Art. ID 210680.
[18] J. Xiong, X. Tan, K.-W. Yang, L.-N. Xing, and Y.-W. Chen, "A hybrid multiobjective evolutionary approach for flexible job-shop scheduling problems," *Math. Problems Eng.*, vol. 2012, May 2012, Art. ID 478981.
[19] X. Xu, L. Li, L. Fan, J. Zhang, X. Yang, and W. Wang, "Hybrid discrete differential evolution algorithm for lot splitting with capacity constraints in flexible job scheduling," *Math. Problems Eng.*, vol. 2013, Feb. 2013, Art. ID 986218.
[20] J.-T. Tsai, W.-H. Ho, T.-K. Liu, and J.-H. Chou, "Improved immune algorithm for global numerical optimization and job-shop scheduling problems," *Appl. Math. Comput.*, vol. 194, no. 2, pp. 406–424, 2007. [Online]. Available: http://dx.doi.org/10.1016/j.amc.2007.04.038

HAO-CHIN CHANG received the B.S. and M.S. degrees in marine engineering from National Kaohsiung Marine University, Kaohsiung, Taiwan, in 2009 and 2011, respectively, where he is currently pursuing the Ph.D. degree with the Institute of Engineering Science and Technology. His research interests include artificial intelligence and applications of multiobjective optimization genetic algorithms.

YEH-PENG CHEN received the B.S. degree in information engineering from I-Shou University, Kaohsiung, Taiwan, in 1996, and the M.S. and Ph.D. degrees in information management science and engineering science and technology from the National Kaohsiung First University of Science and Technology, Kaohsiung, in 2003 and 2015, respectively. His research interests include artificial intelligence, applications of multiobjective optimization genetic algorithms, power systems, and data mining.

TUNG-KUAN LIU received the B.S. degree in mechanical engineering from National Akita University, Akita, Japan, in 1992, and the M.S. and Ph.D. degrees in mechanical engineering and information science from National Tohoku University, Sendai, Japan, in 1994 and 1997, respectively. He is currently a Professor with the Mechanical and Automation Engineering Department, National Kaohsiung First University of Science and Technology, Kaohsiung, Taiwan. From 1997 to 1999, he was a Senior Manager with the Institute of Information Industry, Hsinchu, Taiwan. From 1999 to 2002, he was an Assistant Professor with the Department of Marketing and Distribution Management, National Kaohsiung First University of Science and Technology. His research and teaching interests include artificial intelligence, applications of multiobjective optimization genetic algorithms, and integrated manufacturing and business systems.

JYH-HORNG CHOU (SM'04–F'15) received the B.S. and M.S. degrees in engineering science from National Cheng Kung University, Tainan, Taiwan, in 1981 and 1983, respectively, and the Ph.D. degree in mechatronics engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 1988. He is currently the Chair Professor with the Department of Electrical Engineering, National Kaohsiung University of Applied Sciences, Kaohsiung, and a Distinguished Professor with the Institute of Electrical Engineering, National Kaohsiung First University of Science and Technology, Kaohsiung. He has co-authored three books, and authored over 255 refereed journal papers. He also holds six patents. His research and teaching interests include intelligent systems and control, computational intelligence and methods, automation technology, robust control, and quality engineering. He is a fellow of the Institution of Engineering and Technology, the Chinese Automatic Control Society, the Chinese Institute of Automation Engineers, and the Chinese Society of Mechanical Engineers. He was a recipient of the 2011 Distinguished Research Award from the National Science Council of Taiwan, the 2012 IEEE Outstanding Technical Achievement Award from the IEEE Tainan Section, the Research Award and the Excellent Research Award from the National Science Council of Taiwan 14 times, and numerous academic awards and honors from various societies.