International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 05 Issue: 01 | Jan-2018 www.irjet.net p-ISSN: 2395-0072

Novel heuristics for no-wait two stage multiprocessor flow shop with probable rework and sequence dependent setup times

Ehsan Shalchi

Space Thrusters Research Institute, Iranian Space Research Center, Tabriz, Iran

---

Abstract - *This paper presents a number of novel heuristics for solving a no-wait two stage multiprocessor flow shop scheduling problem. To fill in a segment of gap in real scheduling problems, two practical assumptions including probable rework and sequence dependent setup times are considered this study. A number of evolutionary algorithms namely, discrete particle swarm optimization (DPSO), adapted imperialist competitive algorithm (AICA) and adapted invasive weed optimization (AIWO) are used in this investigation. Comprehensive calibrations of different parameters' values are performed. For this purpose, response surface method is employed to select the optimum parameters' values with the least possible number of experiments. The performances of the algorithms are studied in terms of relative percentage deviation of make-span. The results of the computational experiments confirm superiority of AIWO to the other algorithms.*

***Key Words*: no-wait, flexible flow shop, rework, sequence dependent setup times, ICA, IWO**

1.INTRODUCTION

Manufacturing systems involving multiple production lines with different machines are more complicated. The clue of mentioned complexity can be found in resource management process as well. One of the imperatively important technique to improve the productivity, resource utilization, profitability of the production lines and keeping the competitively in rapidly changing marketplace is scheduling and sequencing. Scheduling and sequencing are widely used techniques in different parts of a manufacturing systems from designing the product flow and processing orders in a manufacturing facility to modelling queues in service industries [1-3].

No-wait scheduling systems have recently been paid more attention by the researchers among the scheduling problems, in which the operations of a job have to be processed from start to end without interruptions in the production line or between them. Lack of the intermediate buffers and the nature of the production method are two most important reasons for designing a no-wait manufacturing system. Nowadays, some sort of the industries uses the no-wait production method such as: production of steel, plastics, aluminum products [4], pharmaceutical processing [5], chemical processing [6], food processing [7], and concrete ware production [8].

Reddi and Ramamurthy (1972) was the first paper in which the flow shop scheduling with no-wait was studied with make-span performance. To solve this problem, they proposed a heuristic algorithm. Although the performance of the presented approach was not investigated in detail, this can be seen as a milestone in this area as they have proposed the first heuristic algorithm for the problem [8]. Travelling salesman problem (TSP) was utilized by Gilmore and Gomory (1964) to analyze a two stage single processor no-wait flow shop problem, in which an optimal solution was reported due to a TSP based branch and bound algorithm. The time complexity of the proposed algorithm was $O(n^2)$ [9]. Another study in no-wait flow shop scheduling problem with make-span performance was done by Rajendran (1994). In this study a heuristic algorithm was suggested based on a preference relation and job insertion. This algorithm outperformed the previously presented algorithms [6].

Another heuristic algorithm called Least Deviation (LD) was presented by Zhixin et al. (2003) for a two-stage-no-wait hybrid flow shop scheduling in which there is a machine in each stage. Lower time complexity makes this algorithm one of the favorable in the associated applications [10]. Later, a novel heuristic algorithm known as Minimum Deviation Algorithm (MDA) proposed to minimize the make-span of a two-stage flexible flow shop with no-wait presented by Xie et al. (2004) [2].

Xie and Wang (2005) generalizes the two-stage flexible flow shop scheduling problem by considering availability constraints. The complexity and the approximations of the problem have been studied in this work, but the provided results indicate that the problems have been studied are more difficult to approximate than the cases without availability constraints [11]. Another study of these area have been completed by Huang et al. (2009). They propose an integer programming model and Ant Colony Optimization approach to solve a no-wait two stage flexible flow shop with setup time and minimum total completion time performance measure. They claim that the performance of the ACO is much better than the IP model in terms of running time and quality of the final solution [12].

A no-wait flexible flow line scheduling problem with time windows and job rejection has been presented by Jolai et al. (2009) to maximize the overall profit. Their work is a production extension and delivery scheduling problem with time windows. To solve their model, a MILP model has been presented and solved by LINGO. To speed up the

performance of the algorithm they used GA and Tabu search technique. Comparing the results proves that GA works better than others [13].

Two classes of no-wait flexible flow shop scheduling problem that are commonly used in automated manufacturing industry are adding: i) sequence-dependent setup times (SDST) [14] and, ii) rework for jobs in all stages. After completing an operation in a step, to proceed to the next step some types of sequence dependent setup (e.g. changing tools or adding some new devices) are needed. But, generally setting up a machine is one of the routine actions that might easily consume more than 20% of machine availability if it is mismanaged [15]. An important classification for sequence dependent setup time is ASDST and NSDST, which are anticipatory and non-anticipatory dependent setup time, respectively. ASDST refers to a setup that can be done even if the job is not available yet to be processed, on the other hand, NSDST is the setup that can be started if both the job and machine are available [16]. Second group of the no-wait flexible flow shop scheduling problem mentioned in the beginning of this paragraph was rework for jobs in all stages. Rework is transformation of rejected products into the re-usable products [17].

Shafaei et al. (2011) investigated the no-wait two stage flexible flow shop with a minimizing mean flow time performance measure. They developed six meta-heuristic algorithms to solve the problem [18]. Moradinasab et al. (2012) consider a no-wait two-stage flexible flow shop scheduling problem by considering unit setup times and rework probability for jobs after second stage and solved this problem with ICA and DPSO [19]. Moradinasab et al. (2013) studied a no-wait two-stage flexible flow shop scheduling problem with setup times aiming to minimize the total completion time. They used an adaptive imperialist competitive algorithm (AICA) and genetic algorithm (GA) to solve this problem and the performance of their proposed AICA and GA algorithms were tested by comparing with ant colony optimization, known as an effective algorithm in the literature [20].

In this study a no-wait two stage flexible flow shop scheduling problem with considering two realistic assumptions including sequence dependent setup times and rework probability of jobs is solved using three meta-heuristics algorithms namely discrete particle swarm optimization (DPSO), adapted imperialist competitive algorithm (AICA) and adapted invasive weed optimization (AIWO). The aim is to find an effective algorithm with minimum maximum completion time. Because to achieve a global optimal solution and guarantee the maximum amount of overall profit in each system, all these aspects should be considered in a single model [21, 22].

The rest of the article is formed as follows: In Section 2, the problem is described using a numerical example. Structures of the proposed algorithms are presented in Section 3. Parameters calibration of the suggested algorithm is presented in Section 4. In Section 5, a comparative analogy among the proposed algorithms is presented. Finally, Section 6 concludes the paper and proposes some directions for future works.

2. PROBLEM DEFINITION

The no-wait two stage flexible flow shop scheduling problem (NWTSFFSSP) is a typical scheduling problem with strong engineering background which can be described as follows: In a NWTSFFSSP, each of n jobs consists of two operations owning a predetermined processing time P(i, j) of stage i on job j and setup times S (j,k) between job j and job k , each of n jobs will be sequentially processed in stage 1, 2 respectively. At the same time, a NWTSFFSSP must meet some constraints as follows:

- The processing of each job has to be continuous.
- That is, once a job is started on the first machine, it must be processed through all machines without any preemption and interruption.
- Each machine can handle no more than one job at a time.
- Each job has to visit each machine exactly once.
- The release time of all jobs is zero. It means all jobs can be processed at the time 0.
- The set-up time of each machine is considered sequence dependent setup times S (j, k) between Job j and Job k.
- For both operations of each job after processing, an inspection is considered, inspection time included to processing time in both stages.
- After inspection, with predetermined probability (pre-work) each job it may be needed to reworking procedure.
- The problem is to find a sequence that the maximum completion time is minimized. The problem is shown by $F_2(m_1, m_2) \mid \text{no-wait, sdst, rework} \mid C_{\max}$.

The no-wait two stage flexible flow shop problems are NP-hard in the strong sense (23). So, all exact approaches for even simple problems will most likely have running times that increase exponentially with the problem Size. In this paper three meta-heuristic algorithms are proposed to the problem described above. The frameworks of these algorithms are explained in the next section.

3. Proposed algorithms

3.1 Adapted imperialist competitive algorithm

Atashpaz-Gargari and Lucas (2007) illustrated the imperialist competitive algorithm (ICA)[1]. ICA has been

widely applied for many non-polynomial hard optimization problems such as flow shop scheduling. Shokrollahpour et al. (2010) applied this method for solving the two-stage assembly flow shop scheduling problem with minimization of weighted sum of make-span and mean completion time as the objective. They calibrate the parameters of this algorithm using the Taguchi method. The results show satisfactory performance of the proposed algorithm [2].

ICA is originated similar to other evolutionary algorithms using an initial population and any individual of the population is named a country. Countries are divided in two groups: imperialists and colonies. Some of the best countries (countries with the least cost for minimization problems and with high cost for maximization problems) are chosen to be the imperialist countries and the rest (i.e. colonies) are divided among the mentioned imperialists based on their power. The power of each country is calculated based on the objective function. A set of one imperialist and their colonies forms one empire. The total power of an empire is equal to the power of the imperialist country plus a percentage of mean power of its colonies. After forming initial empires, the competition starts, the colonies in each of empires start moving toward their imperialist country, and the imperialists attempt to achieve more colonies. Hence during the competition, the weak imperialist will be collapsed. At the end just one imperialist will remain. The framework of the proposed adapted imperialist competitive algorithm (AICA) is described as follow:

3.1.1 Generating initial empires

Each solution in AICA is in a form of an array. Each array consists of variable values to be optimized. In GA terminology, this array is called "chromosome," but here, we use the term "country". In an N dimensional optimization problem, a country is a 1×N array. This array is defined by:

$$country = [p_1, p_2, p_3, ..., p_N] \quad (1)$$

Where Pi is the variable to be optimized. Each variable in a country denotes a socio-political characteristic of a country. From this point of view, all the algorithm does is to search for the best country that is the country with the best combination of socio-political characteristics such as culture, language, and economical policy [1].

In the AICA each solution (country) is a 1×N array of integer variables that N represents the number of jobs. The array of the country represents a sequence of jobs to be assigned to earliest available machines in both stages. The structure of one solution for a seven-job problem is shown in Figure 1.

6	5	7	2	1	3	4

Fig – 1: Structure for a seven-job problem in HICA

The cost of a country is calculated using a cost function f at the variables $(P_1, P_2, ..., P_N)$ as follow:

$$c_i = f(country_i) = f(p_{i1}, p_{i2}, ..., p_{iN}) \quad (2)$$

The algorithm starts with initial countries that are generated randomly by a number of population size (PopSize) and the most powerful countries (countries with minimum cost) are selected as the imperialists by a number of N-imp. The remaining countries are colonies each of which belongs to an empire. The colonies are distributed among imperialists based on imperialist's power. For calculating the imperialists power, the normalized cost of an imperialist is applied based on follow definition:

$$C_n = \max_i c_i - c_n \quad (3)$$

Where, c_n is the cost of nth imperialist and C_n is its normalized cost which is equal to the deviation of the maximum total completion time from the nth imperialist cost. The power of each imperialist is calculated according to Equation 4:

$$p_n = \left| \frac{C_n}{\sum_{i=1}^{N_{imp}} C_i} \right| \quad (4)$$

Having obtained the imperialist power, the colonies are distributed among the imperialist accordingly. In addition, the initial number of colonies of an imperialist is calculated as follow:

$$NC_n = round\{P_n . N_{col}\} \quad (5)$$

Where, NC_n is the initial number of colonies of nth imperialist and N_{col} is the number of all colonies. We randomly select NC_n of colonies and designate them for each imperialist. Imperialist with the bigger power has a greater number of colonies while imperialist with weaker power has less.

3.1.2 Updating the colonies (assimilating)

Colonies start improving their power by capturing more Imperialist countries. Some part of a colony's structure will be similar to the empire's structure as it is created from the nature of this movement. Figure 2 illustrates the imperialist's and colony's arrays. In the AICA, the percent of job numbers from colony's array are chosen to be same the imperialist's array. For this purpose, a new array with the cells value equal to one and zero is randomly generated (Figure 3). Noted this, the number of the ones are equal to percent of jobs that have the positions equal to that of the imperialist array and named

as Prct-Assimilate. Then the subsequent jobs are determined based on the orders defined in the colonies. The resulting job sequence is shown in Figure 4.

Imperialist	6	5	7	2	1	3	4
Colony	4	3	1	6	7	5	2

Fig - 2: Imperialist's and colony's arrays

0	1	1	0	0	0	0

Fig - 3: New array

2	5	7	4	3	1	6

Fig - 4: Assimilated colony

3.1.3 Exchanging positions of the imperialist and a colony

A colony might reach to a position with lower cost than the imperialist when colony moved toward the imperialist. In these situations, the position of the imperialist and the colony is swapped as showed in Figure 5a. Afterward the algorithm will continue and the colonies will be moved toward imperialist in its new position. The resulting empire, after swapping the position of the imperialist and the colony, is depicted in Figure 5b.

(a) (b)

Fig - 5: Exchanging positions of the imperialist and a colony

3.1.4 Total power of an empire

Total power of an empire is mainly affected by the power of the imperialist country, but the power of the colonies of an empire has an indigent effect on the total power of that empire. Therefore, the equation of the total cost is defined as follow:

$$TC_n = \cos t(imperialist_n) + \xi mean\{\cos t(colonies\ of\ empire_n)\} \quad (6)$$

Where TC_n is the total cost of the nth empire and zeta (ξ) is a positive number which is considered to be less than 1. The total power of the empire to be determined by just the imperialist when the value of ξ is small. The role of the colonies, which determines the total power of an empire, becomes more important as the value of ξ increases.

3.1.5 Imperialistic competition

All empires attempt to take the possess and control of colonies of other empires. In the imperialistic competition the power of weaker emprises will gradually reduce and the power of more powerful ones will rise. In other words, picking some (usually one) of the weakest colonies of the weakest empire and making a competition among all empires to possess these colonies are the imperialistic competition. In this competition, the most powerful empires will not definitely possess these colonies, but these empires will be more likely to possess them. This competition is modelled by picking one of the weakest colonies from the weakest empire. Then in order to calculate the possession probability of each empire, first the normalized total cost is calculated as follows:

$$NTC_n = TC_n - \max\{TC_i\} \quad (7)$$

Where, TC_n is the normalized total cost of n^{th} empire and TC_n is the total cost of nth empire. Having normalized the total cost, the possession probability of each empire is calculated as below:

$$p_{Pn} = \left| \frac{NTC_n}{\sum_{i=1}^{N_{imp}} NTC_i} \right| \quad (8)$$

We use Roulette wheel method for assigning the mentioned colony to the empires.

3.1.6 Revolution

In each iteration, for every imperialist two positions of imperialist's array are chosen and these positions are exchanged together and new imperialist is replaced with the weakest imperialist colony's. These processes are repeated by a percentage of jobs for each imperialist named as Prct-Imp -R. Furthermore, some of the colonies are selected and then two positions of the colony's array are chosen and these positions are exchanged. These processes are repeated for a percentage of jobs for each colony named as Prct- Col -R. The replacement ratio is identified as the revolution rate and named as P-R.

3.1.7 Eliminating the powerless empires

Powerless empires will collapse and their colonies will be distributed among other empires in the imperialistic competition. In this paper, when an empire loses all of its colonies, we consider it as a collapsed empire.

3.1.8 Global war

After preceding a number of iterations, a global war occurs, named as I-Global-War. Then new countries equal to the number of population are produced. This is followed by merging new countries with the old population and sorting the countries based on their accenting cost functions. Finally, from the sorted countries, a number of countries equal to the old population are selected. This process is repeated a number of times known as N-Global-War.

3.1.9 Stopping criteria

In this paper stopping criteria or end of imperialistic competition is considered when there is only one empire for all of the countries.

3.2 Adapted invasive weed optimization

The invasive weed optimization (IWO) is a numerical stochastic search algorithm that is proposed by Mehrabian and Lucas (2006) [3]. IWO mimics natural behaviour of weeds in colonizing and finding suitable place for growth and reproduction. IWO has been widely applied for a number of optimization problems. Karimkashi and Kishk (2010) applied IWO to the array antenna synthesis problems. The results showed that IWO outperforms the particle swarm optimization (i.e. PSO)[4]. Furthermore, Ghalenoei and Hajimirsadeghi (2009) proposed discrete invasive weed optimization (DIWO) algorithm and compared its performance with five other evolutionary algorithms in time cost trade-off (TCT) problem [5]. The result revealed that DIWO outperforms the other algorithms. Such superiority encouraged the authors to apply an adapted version of DIWO to a no-wait two stage flexible flow shop scheduling problem with sequence dependent setup times and probable reworks. The framework of the proposed algorithm namely 'adapted invasive weed optimization' (AIWO) algorithm is as follows:

3.2.1 Initialize a population

A limited number of weeds, called PopSize, are randomly created and considered as initial population. The weed structure is similar to the country structure defined in AICA that is shown in Figure 1. Fitness value of each weed is calculated as its objective function as soon as it is generated. In this study, the fitness function for a weed is defined as the minimization of make-span.

3.2.2 Reproduction

Every seed grows to become new plant (Weed) then these weeds are allowed to produce other seeds depending on their fitness function. The weed with minimum fitness function will produce maximum possible seed (S_{max}). Likewise, the weed with maximum fitness function will produce minimum possible seed (S_{min}). The number of seeds associated with each weed is produced depending on the value of the fitness function and is generated using a linear function varying in a range between S_{min} and S_{max}.

In this step seeds for each weed are produced using a neighbourhood definition applied in Simulated Annealing (SA). In other word the seeds around of each weed are produced using one of the three policies namely "swap, reversion and insertion".

To perform the seed generation around the given weed, assume two jobs of given weed are elected randomly. Finally, among three policies including swap, reversion and insertion, one of them is selected, randomly. The structures of these operators are described at below:

- Swap: the positions of selected jobs are exchanged.
- Reversion: In this policy besides conducting swap, the jobs located in between the swapped jobs are reversed, too.
- Insertion: In this case the job in the second position is located immediate after the job in the first location and the other jobs are shifted right hand side accordingly.

3.2.3 Spatial distribution

The produced seeds in the previous step are distributed randomly in the problem space. For to discrete the solution space of problem, in this paper instead of using the normal distribution to generate the seeds in around of each weed, neighborhood concept is applied. Therefore, at first beyond neighborhoods is examined and with increasing the iterations, nearer neighborhoods is checked. and are the most far and nearest neighborhoods of the problem, respectively. The value of is equal to the percent of job numbers that is obtained using Equation 9 as bellow:

$$NE_{\max} = \eta \times number \text{ of jobs} \tag{9}$$

The number of neighborhood ($N_{NE}^{iter,Weed}$) for a particular iteration for each weed is calculated using Equation 10:

$$N_{NE}^{iter,Weed} = Discret\ \ Uniform \left(NE_{\min}, round\left(\left[\frac{iter_{\max} - iter}{iter_{\max}} \right]^{pow} (NE_{\max} - NE_{\min}) + NE_{\min} \right) \right) \tag{10}$$

Where $iter_{\max}$ is the maximum number of iteration ($iter$) is the current iteration and pow is a fixed number.

3.2.4 Competitive exclusion

In each iteration, assume that all of the seeds that are produced are created new population. Therefore, the new population and previous population are merged together

then among this population the weeds or seeds with minimum fitness function are selected by number of population size. Finally, selected population is the final population that is used in next iteration.

3.2.5 Stopping criteria

The processes of weed generating are stopped when a fixed number of generations are satisfied that is called *MaxIt*.

1-Initialization a population

1-1-Set Parameters (*MaxIt* ,*PopSize*, η , *Smax*, *Pow*)

1-2-Generate initial weeds (Randomly)

2-Evaluate fitness of each weed

3- Reproduction

3-1-determine the number of seeds for each weed

3-2-produce the seeds for weeds by

4--merge the population of seeds and weeds with together

5--Sort merged population and choice the first weeds or seeds with the size of the *PopSize* as new population

6- Stop if stopping criteria is met, otherwise go to step 3.

3.3 Discrete particle swarm optimization

Particle swarm optimization (PSO) is a population based stochastic optimization technique which is developed by Eberhart and Kennedy (1997) (29). PSO is inspired by the behavior of bird flocking and fish schooling.

Recently, PSO algorithm and a discrete particle swarm optimization (DPSO) algorithm have shown a wide application in optimization problems. Pan et al. (2008) applied DPSO algorithm to solve the no-wait flow shop scheduling problem with both make-span and total flow time criteria [30-40]. The framework of the adapted version of discrete particle swarm optimization (DPSO) is as follows:

3-3-1- Initialize swarm

In DPSO, the potential solutions are called particles. Initial population of particle is randomly generated by a number of population size (*PopSize*) and dispread randomly over the problem space. The representation of particle is similar to country representation that is shown in Figure 1. Also, fitness value of each particle is evaluated by objective function when it is generated. In this study, the fitness function for the particle is defined as the minimization of make-span.

3.3.2 Individual and the global best

In the end of each iteration, the particle best solution and the global best solution are determined. The best solution for each particle over the iterations to the presented iteration is determined as particle best solution and the best solution for all the particles over the iterations to the presented iteration is determined as global best solution.

3.3.3 Update of the particle's position

Each particle moves according to the pervious velocity of particle and the both distance between the current position of particle with the global best solution and current position of particle with particle best solution. The two major targets for particles in DPSO are movement toward the global best solution and particle best solution.

Therefore, in each iteration the position of each particle is updated three times. First position update is done by its current position and particle best solution. For the first position updated, first a new array is randomly generated in a range between one and zero with size equal to job number. The number of the ones is equal to percent of job numbers and this is shown by *P-BestPosition*. For every position in the new array that the position value's is one, the same position value's of the particle best solution's array is inserted in the first position updated. Afterword, the other works, not assign to first position updated, in the same order of the current position's array is chosen to assign the first position updated array. The second position updated is done by its first position update and global best solution similar to the first position updated. Noted this, the number of the ones are equal to the percent of job numbers and this is shown by *G-BestPosition*.

Finally, the last position updated is done according to the previous velocity, for this target the final position updated is done by generation new particle at the neighbourhood of the second position updated. New particle is produced by swap or insertion operator on the second position updated. These operators for generating new particle are repeated by percent of job numbers that is shown by *P-prcnt*. The new particle is the final position updated.

3.3.4 Stopping criteria

When a fixed number of generations are satisfied that is called Max-It, the processes of searching are stopped.

1-Initialization a population

1-1-Set Parameters (*MaxIt*, *PopSize*, *P-BestPosition*, *G-BestPosition*, *P-prcnt*)

1-2-Generating initial particles (Randomly)

2-Evaluate fitness of each particle

3- Determinate the global best solution and the particle best solution

4- Update the position of each particle by pervious velocity of particle, current position of particle, the global best solution and particle best solution.

5-Stop if stopping criteria is met, otherwise go to step 3.

4. Computational experiments

4.1 Data generation

The data required for this problem consist of number of jobs, number of machines in each stage, processing time in both stages, sequence dependent setup times in both stages, rework probability for both stages and rework times in both stages. Number of jobs and number of machines generated for the test problems are shown in Table 1. The processing times and sequence dependent setup times are generated using a uniform distribution ranging [1, 30] in both stages. Furthermore, rework probability for each job is generated using an exponential distribution ($\lambda e^{-\lambda}$) with the mean equal to 0.05 (i.e. $\lambda = 20$). Also rework times in both stages are generated based on Equation 11:

$$\text{rework time} = round(\text{unif}(0.3, 0.6) \times p_{i,j}) \quad (11)$$

Table – 1: Size of the generated problems

No. Jobs	small	5	10	15	20	25
	large	40	80	120	160	200
No. Machines (in both stages)	small	M_1=2 M_2=3		M_1=2 M_2=2		M_1=3 M_2=2
	large	M_1=16, M_2=20		M_1=20 M_2=20		M_1=24 M_2=20

4.2 Parameter setting

In this section, for optimizing the behavior of the proposed algorithms, appropriate tuning of their parameters has been carried out. For this purpose, response surface methodology (RSM) is employed. RSM is defined as a collection of mathematical and statistical method-based experiential, which can be used to optimize processes. Regression equation analysis is used to evaluate the response surface model. First of all, parameters of each algorithm that statistically have significant impact on the algorithms are recognized. To identify significant parameters, two levels for each parameter are considered. Each factor is measured at two levels, which can be coded as −1when the factor is at its low level (L) and+1 when the factor is at its high level (H). The coded variable can be defined as follows:

$$x_i = \frac{X_i - X_0}{\Delta x} \quad (12)$$

Where X_i and x_i are the actual value and codified value, respectively, X_0 is the value of X_i at the center point, and Δx is the step change value (Myers 2003), The levels and tuned value for each factor of each algorithm is provided in Table 2. It is noticeable that the values of $S_{\min}$ and $NE_{\min}$ are equal to one.

Table -2: Tuned parameters of the algorithms

Alg.	Parameters	Problem size					
		Small			Large		
		L	H	T	L	H	T
DPSO	*PopSize*	100	150	146	300	400	367
	MaxIt	200	300	288	400	600	565
	P-BestPosition	0.2	0.3	0.28	0.25	0.4	0.34
	G-BestPosition	0.2	0.3	0.26	0.25	0.4	0.36
	P-prcnt	0.15	0.2	0.16	0.15	0.2	0.19
AICA	*PopSize*	100	150	150	400	600	451
	Max-It	500	600	565	1000	1200	1156
	N-Imp	6	8	7	12	15	14
	P-Asimlt	0.3	0.4	0.33	0.3	0.4	0.35
	ξ	0.03	0.04	0.033	0.03	0.04	0.038
	Pr-Imp-R	0.2	0.3	0.21	0.2	0.3	0.26
	Pr- Col -R	0.1	0.15	0.13	0.1	0.15	0.15
	P-R	0.3	0.5	0.34	0.4	0.6	0.53
	T-GW	100	150	124	150	200	187
	N-GW	2	3	3	4	6	6
AIWO	*PopSize*	100	150	146	300	400	378
	MaxIt	200	300	294	400	600	583
	η	0.1	0.15	0.12	0.2	0.3	0.27
	S_{max}	6	10	9	10	14	14
	Pow	2	3	2	2	3	3

4.3. Experimental results

In this section, the performances of the proposed algorithms are compared with ACO which was proposed by Huang et al. (2009) [6] and GA applied by Jolai et al. (2009)[7]. The purpose of this paper is to find a sequence which minimizes maximum completion time. In order to conduct the experiment, algorithms are coded in MATLAB 2009b language and run on a PC with 2.66 GHz and 4GB RAM memory computer.

After calculation of the objective function of the proposed algorithms, the best solution obtained for each test problems, named as $Best_{sol}$, is calculated. Relative percentage deviation (RPD) is obtained using Equation 13.

$$RPD = \frac{|A\lg_{sol} - Best_{sol}|}{Best_{sol}} \times 100 \quad (13)$$

Where $A\lg_{sol}$ is the objective value obtained for a given algorithm and test problem. It is clear that lower values of RPD are preferred. Also, average relative percentage deviation ($\overline{RPD}$) is defined according to Equation 14.

$$\overline{RPD} = \frac{\sum_{i=1}^{No.Run} RPD}{No.Run} \quad (14)$$

The performance of the proposed algorithms was examined by solving 30 different problems in two scales (15 problems in small scale and 15 problems in large scale). The problems are replicated thirty times for each combination (i.e. in total 30×30×5 times are implemented). The comparative results in terms of RPD are shown in Tables 3 and 7 for small and large scales problems, respectively. The results show that for in all cases, AIWO obtained better performance than the other algorithms.

Table -3: Results in terms of $\overline{RPD}$ (small scale problems)

			Algorithms				
No. jobs	M_1	M_2	GA-Jolai	ACO-Huang	AICA	AIWO	DPSO
5	2	3	0	8.35	0	0	0
	2	2	0	12.06	0	0	0
	3	2	0	1.47	0	0	0
10	2	3	4.95	17.87	2.07	0	12.76
	2	2	1.88	16.67	1.00	0	11.35
	3	2	6.96	16.10	1.22	0.19	14.33
15	2	3	6.01	23.26	1.75	0	19.58
	2	2	3.30	23.03	1.57	0.03	17.73
	3	2	11.76	24.33	1.12	0.03	17.76
20	2	3	13.68	37.69	3.38	0	26.32
	2	2	11.31	39.75	1.63	0.11	36.41
	3	2	14.94	40.06	2.85	0	34.35
25	2	3	12.92	37.17	2.60	0	29.78
	2	2	5.49	30.42	1.07	0	27.43
	3	2	8.25	35.40	1.12	0	29.07

To validate the statistical significance of the observed differences in the solution quality, % 95 confidence interval for RPD performance measure in both small scale and large scale were applied. These show that AIWO statistically outperform the other algorithms in both scales. Furthermore, AIWO and AICA obtained better results compared with ACO and GA. In addition, the influences of the number of jobs over the different algorithms are studied based on % 95 confidence interval. The results demonstrate that the change in the number of jobs has no impact on AIWO performance but for the other algorithms it could be seen as an asymmetric trend. In order to evaluate the impact of machine balancing on the algorithms performance, $\overline{RPD}$ of algorithms are performed. This shows that machine balancing also has no impact on AIWO performance whereas for AICA.

Table -4: Results in terms of $\overline{RPD}$ (large scale problems)

			Algorithms				
No. jobs	M_1	M_2	GA-Jolai	ACO-Huang	AICA	AIWO	DPSO
40	16	20	25.09	38.33	4.21	0	32.88
	20	20	20.64	35.59	4.52	0	28.19
	24	20	19.15	30.25	1.59	0.09	27.87
80	16	20	20.81	34.31	5.302	0	26.29
	20	20	13.87	30.47	2.29	0.07	22.97
	24	20	17.99	30.18	6.12	0	21.01
120	16	20	18.28	29.87	2.691	0	21.75
	20	20	16.76	26.05	4.495	0	21.33
	24	20	18.55	26.26	3.698	0	19.92
160	16	20	12.25	23.64	4.883	0	17.58
	20	20	11.31	21.40	5.30	0	14.99
	24	20	14.79	26.14	4.39	0	16.22
200	16	20	18.30	23.85	6.34	0	18.61
	20	20	11.27	25.09	4.57	0	17.52
	24	20	13.83	17.83	3.93	0	14.93

3. CONCLUSIONS

In this paper, a couple of novel metaheuristics (i.e. AICA and AIWO) and a popular metaheuristic (i.e. DPSO) were proposed for solving no-wait two stage flexible flow shop scheduling problems with probable rework and sequence dependent setup times to minimizing make-span. The performance of the proposed algorithms was studied in terms of relative percentage deviation performance measure in both small and large scale problems. Results in both scales indicate that AIWO statistically outperforms the other algorithms. Furthermore, interaction between the number of jobs and algorithm shows that AIWO in all of experimental cases dominates the other algorithm. Sensitive analysis was performed to study the interaction between machine balancing and the proposed algorithms performance. The results indicated that machine balancing has no impact on AIWO performance whereas for AICA. As a further work, it is suggested that new assumptions such as machines with different speeds, released time and availability constraints to be considered.

ACKNOWLEDGEMENT

We thank Hassan Jafarzadeh for his assistance and comments that greatly improved the manuscript.

REFERENCES

[1] Atashpaz-Gargari, E., & Lucas, C. (2007, September). Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In Evolutionary computation, 2007. CEC 2007. IEEE Congress on (pp. 4661-4667). IEEE.

[2] Shokrollahpour, E., Zandieh, M., & Dorri, B. (2011). A novel imperialist competitive algorithm for bi-criteria scheduling of the assembly flowshop

problem. International Journal of Production Research, 49(11), 3087-3103.

[3] Mehrabian, A. R., & Lucas, C. (2006). A novel numerical optimization algorithm inspired from weed colonization. Ecological informatics, 1(4), 355-366.

[4] Karimkashi, S., & Kishk, A. A. (2010). Invasive weed optimization and its features in electromagnetics. IEEE transactions on antennas and propagation, 58(4), 1269-1278.

[5] Ghalenoei, M. R., Hajimirsadeghi, H., & Lucas, C. (2009, December). Discrete invasive weed optimization algorithm: application to cooperative multiple task assignment of UAVs. In Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on (pp. 1665-1670). IEEE.

[6] Jafarzadeh, H., Moradinasab, N., & Gerami, A. (2017). Solving no-wait two-stage flexible flow shop scheduling problem with unrelated parallel machines and rework time by the adjusted discrete Multi Objective Invasive Weed Optimization and fuzzy dominance approach. Journal of Industrial Engineering and Management, 10(5), 887.

[7] Jolai, F., Sheikh, S., Rabbani, M., & Karimi, B. (2009). A genetic algorithm for solving no-wait flexible flow lines with due window and job rejection. The International Journal of Advanced Manufacturing Technology, 42(5-6), 523.

[8] Aldowaisan, T., & Allahverdi, A. (2004). New heuristics for m-machine no-wait flowshop to minimize total completion time. Omega, 32(5), 345-352.

[9] Flapper, S. D. P., & Teunter, R. H. (2004). Logistic planning of rework with deteriorating work-in-process. International journal of production economics, 88(1), 51-59.

[10] Jafarzadeh, H., Moradinasab, N., & Elyasi, M. (2017). An Enhanced Genetic Algorithm for the Generalized Traveling Salesman Problem. Engineering, Technology & Applied Science Research, 7(6), 2260-2265.

[11] Grabowski, J., & Pempera, J. (2000). Sequencing of jobs in some production system. European Journal of Operational Research, 125(3), 535-550.

[12] Hall, N. G., & Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. Operations research, 44(3), 510-525.

[13] Hasani, R., Jafarzadeh, H., & Khoshalhan, F. (2013). A new method for supply chain coordination with credit option contract and customers' backordered demand. Uncertain Supply Chain Management, 1(4), 207-218.

[14] Jafarzadeh, H., Gholami, S., & Bashirzadeh, R. (2014). A new effective algorithm for on-line robot motion planning. Decision Science Letters, 3(1), 121-130.

[15] Kennedy, J., & Eberhart, R. C. (1997, October). A discrete binary version of the particle swarm algorithm. In Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on (Vol. 5, pp. 4104-4108). IEEE.

[16] Liu, B., Wang, L., & Jin, Y. H. (2007). An effective hybrid particle swarm optimization for no-wait flow shop scheduling. The International Journal of Advanced Manufacturing Technology, 31(9-10), 1001-1011.

[17] Liu, Z., Xie, J., Li, J., & Dong, J. (2003). A heuristic for two-stage no-wait hybrid flowshop scheduling with a single machine in either stage. Tsinghua Science and Technology, 8(1), 43-48.

[18] Moradinasab, N., Shafaei, R., Rabiee, M., & Ramezani, P. (2013). No-wait two stage hybrid flow shop scheduling with genetic and adaptive imperialist competitive algorithms. Journal of Experimental & Theoretical Artificial Intelligence, 25(2), 207-225.

[19] Naderi, B., Zandieh, M., & Shirazi, M. A. H. A. (2009). Modeling and scheduling a case of flexible flowshops: Total weighted tardiness minimization. Computers & Industrial Engineering, 57(4), 1258-1267.

[20] Pan, Q. K., Tasgetiren, M. F., & Liang, Y. C. (2008). A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. Computers & Operations Research, 35(9), 2807-2839.

[21] Raaymakers, W. H., & Hoogeveen, J. A. (2000). Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing. European Journal of Operational Research, 126(1), 131-151.

[22] Rajendran, C. (1994). A no-wait flowshop scheduling heuristic to minimize makespan. Journal of the Operational Research Society, 45(4), 472-478.

[23] Shafaei, R., Moradinasab, N., & Rabiee, M. (2011). Efficient meta heuristic algorithms to minimize

mean flow time in no-wait two stage flow shops with parallel and identical machines. International Journal of Management Science and Engineering Management, 6(6), 421-430.

[24] Shafaei, R., Rabiee, M., & Mazinani, M. (2012). Minimization of maximum tardiness in a no-wait two stage flexible flow shop. International Journal of Artificial Intelligence™, 8(S12), 166-181.

[25] Sriskandarajah, C., & Ladet, P. (1986). Some no-wait shops scheduling problems: complexity aspect. European journal of operational research, 24(3), 424-438.

[26] Jafarzadeh, H., Moradinasab, N., Eskandari, H., & Gholami, S. (2017). Genetic Algorithm for A Generic Model of Reverse Logistics Network. International Journal of Engineering Innovation & Research, 6(4), 174-178.

[27] Weiss, G. (1995). Scheduling: Theory, Algorithms, and Systems.

[28] Xie, J., & Wang, X. (2005). Complexity and algorithms for two-stage flexible flowshop scheduling with availability constraints. Computers & Mathematics with Applications, 50(10-12), 1629-1638.

[29] Xie, J., Xing, W., Liu, Z., & Dong, J. (2004). Minimum deviation algorithm for two-stageno-wait flowshops with parallel machines. Computers & Mathematics with Applications, 47(12), 1857-1863.

[30] Zandieh, M., Ghomi, S. F., & Husseini, S. M. (2006). An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. Applied Mathematics and Computation, 180(1), 111-127.

[31] Zeyghami, S., Babu, N., & Dong, H. (2016). Cicada (Tibicen linnei) steers by force vectoring. Theoretical and Applied Mechanics Letters, 6(2), 107-111.

[32] Zeyghami, S., & Dong, H. (2015). Coupling of the wings and the body dynamics enhances damselfly maneuverability. arXiv preprint arXiv:1502.06835.

[33] Bode-Oke, A. T., Zeyghami, S., & Dong, H. (2017). Aerodynamics and flow features of a damselfly in takeoff flight. Bioinspiration & biomimetics, 12(5), 056006.

[34] Zeyghami, S., Bode-Oke, A. T., & Dong, H. (2017). Quantification of wing and body kinematics in connection to torque generation during damselfly yaw turn. Science China Physics, Mechanics & Astronomy, 60(1), 014711.

[35] Joghataie, A., & Dizaji, M. S. (2009, July). Nonlinear analysis of concrete gravity dams by neural networks. In Proceedings of the World Congress on Engineering (Vol. 2).

[36] Soltani, R. (2017). Modeling integrated flow shop Scheduling problem and air transportation in supply chain. International Academic Journal of Science and Engineering, 4(3), 10-19.

[37] Joghataie, A., & Dizaji, M. S. (2010). Transforming results from model to prototype of concrete gravity dams using neural networks. Journal of Engineering Mechanics, 137(7), 484-496.

[38] Sojoudi, M., & Sojoudi, M. (2017). Designing Mathematical Modeling of location Network and Optimal Planning for Supply Chain Demand. International Academic Journal of Science and Engineering, 4(3), 75-86.

[39] Sojoudi, M., & Saeedi, H. (2017). The Problem of Locating-Allocation of Facilities and Central Warehouse in the Supply Chain with Bernoulli Demand. International Academic Journal of Science and Engineering, 4(2), 152-165.

[40] Lashgari Y. (2017). Proposing a hierarchical approach based on fuzzy logic to choose a contractor in the bank. International Academic Journal of Science and Engineering, 4(3), 28-38.