Available online at www.isr-publications.com/jmcs
J. Math. Computer Sci. 16 (2016), 50–68
Research Article

Online: ISSN 2008-949x

Journal of Mathematics and Computer Science

Journal Homepage: www.tjmcs.com - www.isr-publications.com/jmcs

JIT scheduling problem on flexible flow shop with machine break down, machine eligibility and setup times

N. Kangarloo, J. Rezaeian, X. Khosrawi

Department of Industrial Engineering, Mazandaran University of Science and Technology, Babol, Iran.

Abstract

In this research, the problem of scheduling flexible flowshop system with machine breakdown, machine eligibility, machine-dependent setup time and sequence-dependent setup time to minimizing total weighted earliness and tardiness is studied. Firstly, the problem is formulated as a Mixed Integer Linear Programming model. With this mathematic model, small-sized instances are solved to optimality. The considered problem is too difficult to be optimally solved for large problem sizes, and hence two metaheuristics algorithms namely genetic algorithm (GA) and imperialist competitive algorithm (ICA) are proposed to afford large-sized instances. Due to the sensitivity of the proposed algorithms to parameter's values, the taguchi method as an optimization technique to widespread tune different parameters of applied algorithms is employed to improve solutions authenticity. These proposed algorithms were coded and tested on randomly generated examples, then to accredit the effectiveness of them computational results are examined in terms of relative percentage deviation. Moreover, some sensitive analyses are executed for comparing the performance of algorithms in various conditions. The computational evaluations expressly confirm the high performance of the proposed genetic algorithm against imperialist competitive algorithm for related scheduling problem.

Keywords: Flexible flowshop, mixed integer linear programming, genetic algorithm, imperialist competitive algorithm, taguchi.

1. Introduction

Scheduling is a kind of decision-making that has a prominent figure in industrial and manufacturing systems. In fact, scheduling and sequencing allocate the available resources to perform certain

Email addresses: kangarloonasim@gmail.com (N. Kangarloo), j_rezaeian@ustmb.ac.ir (J. Rezaeian), Xaniar.khosrawi@gmail.com (X. Khosrawi)

Received October 2015

activities in an efficient manner. Flexible flowshop is one of the most applied scheduling problem which has begun to be taken seriously in modern industry and received remarkable consideration from researchers in recent years [6, 7, 22, 28].

The flexible flowshop scheduling problem (FFSS), also called compound flowshop, multi-processor flowshop, or hybrid flowshop, consists of two or more processing stages in series with at least one stage having two or more parallel machines. Because of duplication of the number of machines in some stages additional flexibility is created, buffer between any two successive stages is unlimited, the overall capacities are increased and bottlenecks if some operations be too long are avoided [21]. In FFSS each job has to undergo a series of operations and visit all stages in the same order. Moreover, a machine can process at most one job at a time and a job has to be processed at each stage only on one of the machines. The problem consists of assigning the jobs to machines at each stage and sequencing the assigned jobs to the same machine so that some optimality criteria are minimized. FFSS is known to be a NP-hard problem [41].Therefore many algorithms based on computational intelligence are proposed for FFSSs [18, 19, 24]. Arthanari & Ramamurthy [2] and Salvador [31] are among the first who define the flexible flowshop problem. They applied a branch-and-bound method to optimization problem.

Botta-Genoulaz [8] used six new heuristics approach to solve the hybrid flowshop (HFS) scheduling problem with precedence constraints and time lags. The purpose was to minimize maximum lateness. Moursli &Pochet [25] presented a branch and bound algorithm to minimize makespan. Yang et al. [39] defined and analyzed three heuristic ways based on decomposition methods and local search to minimize the total weighted tardiness of jobs with release dates for the hybrid flowshop scheduling problem. Gupta et al. [15] proposed heuristics to solve the HFS problem where the processing times of the operations on some machines might be different and a due date assignment cost was included into the objective function. The aim was to minimize makespan.

Su [33] studied a two-stage hybrid flowshop problem with a batch processor in stage 1 and a single processor in stage 2. Engin & Döyen [10] used a computational method based on clonal selection principle and affinity maturation mechanism of the immune response for the hybrid flowshop scheduling problem. Tang et al. [34] proposed a neural network model and algorithm to solve the dynamic HFS scheduling problem. They employed a two stages approach to tackle the problem. Xie & Wang [38] considered the two-stage flexible flowshop scheduling problem with availability constraints. They discussed the complexity and the assessments of the problem and provided some approximation algorithms. Ruiz & Maroto [30] presented a metaheuristic, in the form of a genetic algorithm, for a complex generalized flowshop scheduling problem that results from the addition of unrelated parallel machines at each stage, sequence dependent setup times and machine eligibility. Tang et al. [35] investigated the problem of scheduling n jobs in s-stage hybrid flowshops with parallel identical machines at each stage. The objective was to minimize the sum of weighted completion times of the jobs. Jin et al.[19] considered the multistage hybrid flowshop scheduling problem, in which each stage consisted of parallel identical machines. The problem was to determine a schedule that minimizes the makespan for a given set of jobs over a finite planning horizon. They proposed two metaheuristic algorithms that the procedure of them was based on simulated annealing and the variable-depth search.

Vob & Witt [36] considered a real-world multi-mode multi-project scheduling problem in which the resources form a hybrid flowshop consisting of 16 production stages. Allahverdi et al. [1] had put together a much more updated and comprehensive review of scheduling research with setup times in which other relevant papers related to the sequence dependent setup flowshop can be found. Naderi et al. [26] studied general hybrid flow shops with the NSDST and transportation times to separately minimize total completion and tardiness times. They consider a single-transporter sys-

tem with job-dependent transportation times and then propose a simulated annealing (SA) with advanced operators to solve the problem. Figielska [11] proposed a heuristic that combined column generation technique with a genetic algorithm or a simulated annealing algorithm for solving the problem of scheduling in a two-stage flowshop with parallel unrelated machines and additional renewable resources at the first stage and a single machine at the second stage. The objective was the minimization of makespan.

Jungwattanakit et al. [20] proposed a GA for bi-objective hybrid flowshop with unrelated machines and setup times. Gholami & Zandieh [13] presented an immune algorithm for SDST hybrid flow shops scheduling with machines suffering stochastic breakdowns to optimize expected makespan. Yaurima et al. [40] considered an important production scheduling problem and presented a genetic algorithm to solve it. They focused on suboptimal scheduling solutions for the hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints, and limited buffers. Wang & Tang [37] investigated the hybrid flowshop scheduling with finite intermediate buffers, whose objective was to minimize the sum of weighted completion time of all jobs and proposed a tabu search heuristic.

Behnamian et al. [5] considered the problem of sequence-dependent setup time hybrid flowshop scheduling with the objectives of minimizing the makespan and sum of the earliness and tardiness of jobs. They presented a multi-phase method to solve the problem. Nishi et al. [27] addressed a new Lagrangian relaxation method for solving the hybrid flowshop scheduling problem to minimize the total weighted tardiness. Heydari & Mohammadi [16] generalized Johnson's results for more than two-machine flow shop problems with the objective of minimizing makespan and used a fuzzy heuristic algorithm.

Choi et al. [9] developed a real time scheduling algorithm with a decision tree selecting one of multiple dispatching rules for a flexible flowshop with reentrant flows for the objectives of the throughput, the mean flow and tardiness, and the number of tardy jobs. Behnamian & Fatemi Ghomi [4] considered sequence-dependent setup time hybrid flowshop scheduling problems. They presented a new solution presentation method and a robust hybrid metaheuristic for combination of two multiple objective decision-making methods, min–max and weighted techniques. Bellanger et al. [7] studied a three-stage hybrid flowshop problem that the first stage corresponded to the receiving docks, the second stage corresponded to the sorting stations, and the third stage corresponded to the shipping docks. The objective of the problem was to find a schedule that minimizes the completion time of the latest batch. They proposed a branch-and-bound algorithm to solve the problem. Seidgar et al., [32] considered a two stages HFS problem with sequence dependent set up times. The objective was to minimize the weighted sum of completion time and maximum tardiness. They used a genetic algorithm approach to solve the problem.

Pan et al. [28] present an effective discrete artificial bee colony algorithm for hybrid flowshop scheduling with the objective of minimizing the make span. Figielska [12] proposes a heuristic for solving a two-stage hybrid flowshop with one machine at the first stage and parallel unrelated machines at the second stage. The heuristic first sequences jobs on the machine at stage 1 and then solves the preemptive scheduling problem at stage 2.

To the best of our knowledge, FFSS problems considering machine breakdown constraint, machine-dependent setup time and sequence-dependent setup time, machine eligibility at the same time have not been investigated. Hence, in this research two metaheuristic algorithms namely genetic algorithm (GA) and imperialist competition algorithm (ICA) for solving the FFSS to minimizing the total weighted earliness and the total weighted tardiness are proposed. The remainder of this study is arranged as follows: in Section 2 the problem outline and assumptions are presented. Section 3 formulates the problem as a mixed integer linear program. In Section 4 the proposed algorithms for solving the considered problem is elaborated. Section 5 addresses the computational evaluation

including parameter calibration, data generation and experimental results. Finally, the conclusion and some further research suggestions are presented in Section 6.

2. Problem definition

In this section, first the assumptions of the studied problem are elaborated then the notations which are used to describe the problem are defined.

2.1. Assumptions

The FFSS with machine and sequence dependent setup times, machine breakdown and machine eligibility can be described as follows:
This problem consists of performing a set of n independent jobs, $J= \{1,...,j,...,n\}$ which have to process through k subsequent stages. There is a due date d_j and r_j for each job j. The number of machines in parallel at each stage k is shown by m^t. Processing time of job j on machine i is described by P_{ij}. A setup time of a job is machine-dependent if it depends on the machine to which the job is assigned. It is assumed to occur only when the job is the first job assigned to the machine. ch_{ij}^t denotes the machine-dependent setup time of job j if job j be the first job assigned to the machine i at stage t . Sequence-dependent setup time between job j and job l at stage t is depicted by S_{jl}^t. Each machine maybe breakdown after processing each job and probability of breakdown depends on the type of job and complexity of operation. The probability of breakdown for machine at stage t after processing job j is denoted by pd_j^t . All m machines are not capable of processing job j. Set M_j $(M_j \, ? \, M_i)$ denotes the set of machines that can process job j.
It is also assumed that machine at stage t requires a constant time R^t to repair after breakdown.
The characteristics of the considered problem are as follows:

1. Preemption of processing is not allowed. It means that, when a job is started on the first machine, it must be processed through all machines without any preemption and interruption.
2. Each stage has at least one machine and at least one stage must have more than one machine.
3. Each machine can process only one job at a time.
4. Each job can be performed by at most one machine at a time.
5. There is an unlimited buffer between every two consecutive stages.
6. Machines may not always be available during the scheduling period.
7. Setup times depend on sequencing of jobs which means the setup times are sequence dependent and the length of time required to do the setup depends on the previous and current jobs and the machine in mentioned stage to be processed (S_{jl}^t).
8. Machine-dependent setup time will be occurring only when the job is the first job assigned to the machine (ch_{ij}^t).
9. The penalty weights could be different for jobs based on their priority importance.

The scheduling problem under consideration has objective, namely minimizing the cost criterion consisting of the total weighted earliness and the total weighted tardiness.

2.2. Notations

The notations are defined as below:
n The number of jobs to be scheduled (j=1, 2, ..., n)
t The number stage (t = 1, 2, 3,..., k)
m^t The number of parallel machines at stage t, (i = 1, 2, 3,...,m^t)

r_j Release date of job j
d_j Due date of job j
P_{ij} Processing time of job j on machine i
S_{jl}^t Setup time of job l if l is immediately processed after job j at stage t
ch_{ij}^t Setup time of job j if job j is sequenced as the first job on machine i at stage t
pd_j^t Probability of machine breakdown at stage t after processing job j
R^t Repair time of machine at stage t after breakdown
av_i^t Ready time for machine i at stage t
C_j^t Completion time of job j at stage t
C_l^t Completion time of job l at stage t
E_j Earliness of job j, E_j =max $\{0, d_j - C_j^k\}$
T_j Tardiness of job j, T_j =max $\{0, C_j^k - d_j\}$
α_j Earliness penalty for job j
β_j Tardiness penalty for job j
M' A large positive integer number

$$M_{ij}^t = \begin{cases} 1 & \text{if job } j \text{ can be processed on machine } i \text{ at stage } t \\ 0 & \text{otherwise} \end{cases}$$

$$X_{ij}^t = \begin{cases} 1 & \text{if job } j \text{ is sequenced on machine } i \text{ at stage } t \\ 0 & \text{otherwise} \end{cases}$$

$$Y_{ijl}^t = \begin{cases} 1 & \text{if job } j \text{ is sequenced on machine } i \text{ before job } l \text{ at stage } t \\ 0 & \text{otherwise} \end{cases}$$

3. The mathematical model

The objective function and constraints can be formulated as follows:
Minimize Z=$\sum_{j=1}^{n} (\alpha_j E_j + \beta_j T_j)$
Subject to:

$$\sum_{i=1}^{m^t} X_{ij}^t = 1, \ j = 1 \ldots n, t = 1 \ldots k \tag{3.1}$$

$$\sum_{l=1}^{n} Y_{ijl}^t \leq X_{ij}^t, i = 1 \ldots m, j = 1 \ldots n, t = 1 \ldots k \tag{3.2}$$

$$\sum_{j=0}^{n} Y_{ijl}^t = X_{il}^t, i = 1 \ldots m, l = 1 \ldots n, t = 1 \ldots k \tag{3.3}$$

$$\sum_{l=1}^{n} Y_{i0l}^t = 1, i = 1 \ldots m, t = 1 \ldots k \tag{3.4}$$

$$X_{ij}^t \leq \ M_{ij}^t, i = 1 \ldots m, j = 1 \ldots n, t = 1 \ldots k \tag{3.5}$$

$$C_l^t - \ C_j^t \geq \ \ S_{jl}^t + \ P_{il}^t \ * \ Y_{ijl}^t + \ pd_j^t \ * R^t + \ \left(\ Y_{ijl}^t \ - \ 1 \right) * \acute{M}, j = 1 \ldots n, i = 1 \ldots m, t = 1 \ldots k \tag{3.6}$$

$$C_j^t \geq 0, j = 1 \ldots n, t = 1 \ldots k \tag{3.7}$$

$$\begin{aligned} C_l^t - \ C_l^{t-1} \geq \sum_{i=1}^{m^t} \sum_{j=1}^{n} Y_{ijl}^t * S_{jl}^t \ + \sum_{i=1}^{m^t} ch_{il} * \ Y_{i0l}^t + \sum_{i=1}^{m^t} \sum_{j=1}^{n} P_{il} * Y_{ijl}^t + \sum_{i=1}^{m^t} P_{il} * Y_{i0l}^t \\ + \ \sum_{j=1}^{n} pd_j * R^t * Y_{ijl}^t, l = 1 \ldots n, t = 1 \ldots k, j \neq l \end{aligned} \tag{3.8}$$

$$C_l^t \geq \sum_{i=1}^{m^t} av_i^t * \ Y_{i0l}^t + \ \sum_{i=1}^{m^t} ch_{il}^t * \ Y_{i0l}^t + \sum_{i=1}^{m^t} P_{il} * Y_{i0l}^t, l = 1 \ldots n, t = 1 \ldots k \tag{3.9}$$

$$C_j^0 \geq r_j, j = 1 \ldots n \tag{3.10}$$

$$T_j \geq \ \ C_j^k - d_j, j = 1 \ldots n \tag{3.11}$$

$$T_j \geq 0, j = 1 \ldots n \tag{3.12}$$

$$E_j \geq \ d_j - C_j^k, j = 1 \ldots n \tag{3.13}$$

$$E_j \geq 0, j = 1 \ldots n \tag{3.14}$$

In the above formulation, constraints (3.1)–(3.5) ensure that the partial schedule on each machine at each stage is feasible. Constraint (3.1) states that each job can be processed by only one machine at each stage. Constraints (3.2) and (3.3) ensure that each job be processed immediately before and immediately after only one job on each machine at each stage. Constraint (3.4) specifies the first job that is assigned to each machine at stage t. Relation (3.5) introduces the machine eligibility. As mentioned in notation part, M_{ij}^t will be 1 if job j can be processed on machine i at stage t; otherwise the value of this parameter will be zero. The possibility of processing job j on machine i at stage t specifies by processing set of job j (M_j^t). Set M_j (M_j ? M_i) denotes the set of machines that can process job j. This constraint stipulates model to allocate machine i for job j and considering M_{ij}^t. If M_{ij}^t be 1, value 1 will be assigned to decision variable X_{ij}^t. Constraints (3.6)–(3.10) find the completion time of every job. Constraint sets (3.11) and (3.12) determine the correct value of the tardiness (T_j). Constraint set (3.11) determines the correct value of the lateness (L_j) and (3.12) specifies only the positive lateness as the tardiness (T_j=max $\{0, C_j^k - d_j\}$). In relations (3.13) and (3.14) the correct value of the earliness (E_j) are determined (E_j=max $\{0, d_j - C_j^k\}$).

4. Proposed algorithms

In this section, two proposed algorithms, entitled genetic algorithm and imperialist competitive algorithm are elaborated respectively for solving the addressed problem.The frameworks of these algorithms are explained in the following:

4.1. Genetic algorithm

Genetic algorithm (GA) as a search process, is more general and synopsis than the other optimization methods that can be used to solve the scheduling problems. The interested reader can consult Goldberg [14] and Reeves [29] for more details about the GA approach and its applications. In this study a convenient representation is used in order to apply GA to determine a special problem. The solution is represented by a set of parameters known as genes joined together and referred to as chromosomes or individuals. The chromosomes define the current population. The quality of each chromosome is defined by a fitness function. New population is obtained from the current one using selection, crossover and mutation mechanisms in iterations. Only fittest chromosomes are selected from new and current population and used in next iteration [17, 23]. The factors which characterize the applied GA to the considered problem in this study are determined as follows.

4.1.1. Solution representation: The proposed representation for the GA to solve the scheduling problem is based on decoding all n jobs and t stages as chromosomes in a t-by-$(n + M^t – 1)$ matrix witch M^t is the maximum number of machine in the problem. In this type of representation, in each row, $(m^t – 1)$ genes (m^t is the number of machine in each row) consist of '*'s used to differentiate from one machine to the other one. *i-th* gene at *t-th* row contains 1 if job j was processed on machine i.

4.1.2. The initial Population: An Initial population contains chromosomes that the quantity of these chromosomes is equivalent to population size (Pop_{size}), is generated by the mentioned method.

4.1.3. Decoding method: Decoding process is based on a heuristic way. In this method, firstly, number of jobs that can be processed on every machine at each stage is specified. Afterward, jobs are randomly allocated on the machines. An example for chromosome decoding is shown in Fig. 1 In this example; there are 6 jobs, 3 identical parallel machines at the first stage and 2 machines at second stage. As shown in Figure jobs 5, 2 and 1 with order 5→2→1 are assigned to machine 1; jobs 6 and 4 are assigned to machine 2 and Job 3 is assigned to the last machine at stage 1. It is important to note that, the sequence of jobs is represented by the numbers of genes from the left side to the right side.

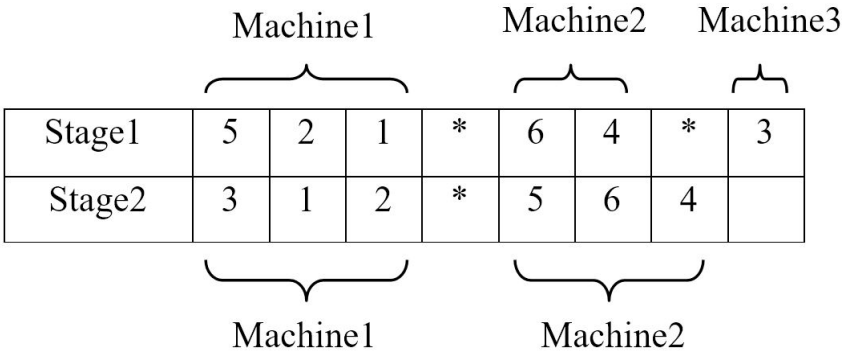

Figure 1: Decoding chromosome scheme

4.1.4. Evaluation: Fitness function for allocating the selecting probability of chromosomes is defined as follow:

$$f_k = \frac{1}{1 + \sum_{j=1}^{n} \alpha_j E_j + \beta_j T_j} \tag{4.1}$$

The defined fitness function is according to the objective function of the proposed problem.

4.1.5. Selection strategy: The Roulette wheel selection procedure is used for selecting individuals. A probability of selection is assigned to each chromosome based on its fitness value. Not only better individuals will have a higher probability for selection but also all individuals in the population will have a chance to be selected. After ranking individuals based on their corresponding fitness and giving weighted probability of selection to each individual in the population, the parents will be randomly selected.

4.1.6. Crossover: A random crossover operator with probability P_c(crossover rate) is applied to each pair of parent chromosomes. According to the number of the stage of the problem, one or more row of parents is randomly selected and exchanged for each other (Fig. 2).

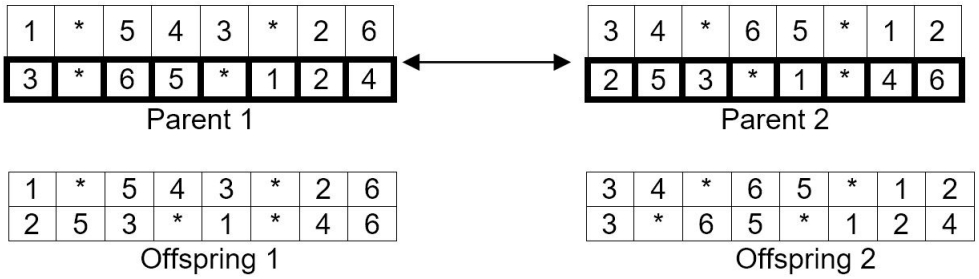

Figure 2: Crossover operation

4.1.7. Mutation: One individual in the population and one stage of it is randomly considered. Then two jobs in the considered stage are randomly selected. Mutation operator is worked by swapping two jobs in the selected chromosome with a probability P_m (mutation rate). Selected jobs may be on one machine or two different machines. The selected jobs on two different machines can be exchanged if can be processed on the other machines. Mutation operation is shown at Fig. 3.

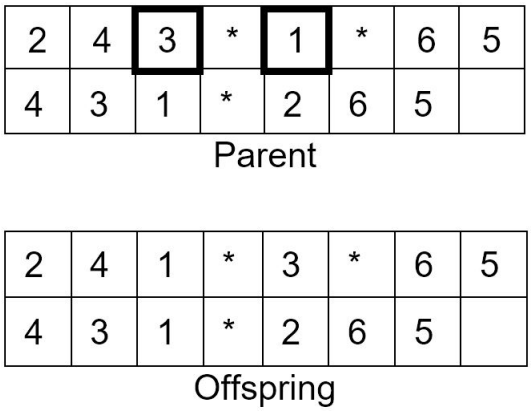

Figure 3: Mutation operation

4.2. Imperialist competitive algorithm

Imperialist competitive algorithm (ICA) is a new evolutionary algorithm to solve the optimization problems. ICA uses the sociopolitical evolution of humans as a source of revelation for expanding a strong optimization strategy [3]. This algorithm starts with an initial population named 'Country' in which these countries are divided into two types of colonies and imperialists according to their

power. Some of the best (most powerful) countries are selected to be the imperialist and the rest of the colonies are dispensed among the imperialists. The more powerful imperialists will have more colonies. The power of each country is a simulation of its objective function value in the considering model. Following this approach, initial empires are created and competition will begin. Imperialists endeavor to get more colonies and colonies tend to move towards their imperialists. This competition leads to the decline of weak empires and the development of more powerful ones. At the end just one imperialist will remain which has the same position as its colonies.

4.2.1. Generating initial empires: In this algorithm solution is denoted by a 2-dimension matrix (similar to chromosome in applied GA)

$$country = [p_1, p_2, \ldots, p_N]. \tag{4.2}$$

Where P_i is a socio-political specific of a country and is considered to be optimized. In this model N is equal to $(n + M^t – 1)$. The power of an empire which is the counterpart of the fitness value in GA is reversely proportionate to its cost. To evaluate the cost of each solution, a cost function (f) at the variables $(p_1, p_2, \ldots, p_N)$ is used as follows:

$$Cost_i = f(country_i) = f(p_1, p_2, p_3, \ldots, p_{iN}). \tag{4.3}$$

In this algorithm f is defined as follows:

$$f = \min\left(\sum_{j=1}^{n} e_j E_j + t_j T_j\right). \tag{4.4}$$

For creation of the initial empires, the number of N_{imp} countries are selected which have the minimum cost as imperialists and consider all remaining countries as colonies with the number of N_{col}, where these colonies are dispensed among the imperialists according to each imperialist normalized cost.The normalized cost of each imperialist represents imperialist power to absorb colonies and will be computed as follows:

$$C_n = max\{c_i\} - c_n\ . \tag{4.5}$$

Where c_n is the cost of nth imperialist and C_n is its normalized cost of all imperialists. The remaining N_{col} ($\mathrm{N}_{col} = N_{pop} – N_{imp}$) of the population will be the colonies of these empires.
To calculate the proportionate power for each imperialist (P_n) the following equation is used:

$$p_n = \left|\frac{C_n}{\sum_{i=1}^{N_{imp}} C_i}\right|. \tag{4.6}$$

Then the initial number of colonies which could be belonged to nth imperialist is:

$$N.C_n = round\{P_n.(N_{col})\}. \tag{4.7}$$

The number of $N.C_n$ of colonies are randomly selected to allocate them to each imperialist in which the more powerful imperialists have the more colonies.

4.2.2. Assimilation: In each empire, colonies tend to improve their power moving toward their imperialist which is called assimilating. To execute assimilating, crossover and mutation operators (similar to crossover and mutation operators in proposed GA) are used. Two countries are randomly selected, so that the first country is the imperialist and second one is the colony. After implementing

crossover operation two new countries will be generated and the objective value of them and the colonial country will be computed. If the objective value of the generated countries be better (lower cost) than current solution, the country with lower cost will be replaced as the colonial country. Now the mutation operation on the new solution (like proposed GA) will be implemented and the new cost of it will be computed. If the computed objective value be lower than the current solution, the mutant country will be replaced.

4.2.3. Exchanging positions between imperialists and colonies: During the competition and moving colonies towards the imperialists a colony might keep improving and get a cost less than the imperialist. It leads to swapped positions between the imperialist and the improved colony and creation of a new empire. This process will proceed during the algorithm (see Fig. 4).

4.2.4. Total Power of an Empire: Total power of an empire is basically affected by the power of imperialist country. In order to measure total power of an empire both the imperialist's power and cumulative power of colonies are considered, but the main affect effected by imperialist's power through the cumulative power of the colonies has a partial share with a positive constant factor between 0 and 1 called $\boldsymbol{\xi}$. So, the equation of total cost is defined as follows:

$$T.C_n = Cost(imperialist_n) + \xi mean\{Cost(coloniesofempire_n) \tag{4.8}$$

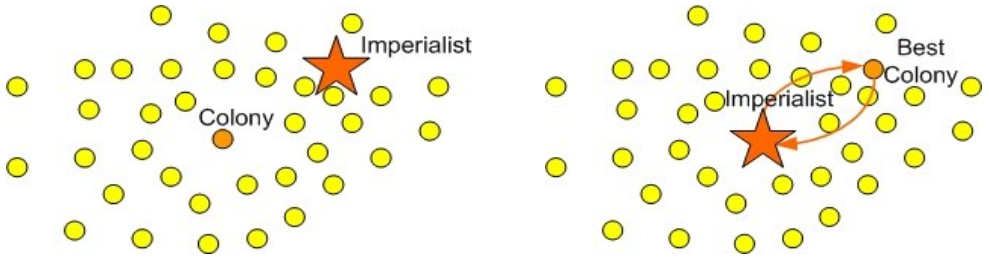

Figure 4: A: Exchanging the positions of a colony B: The entire empire after position exchange.

Where $T.C_n$ is the total cost of the nth empire and ξ is a positive number which is considered to be less than 1. A little value for ξ causes the total power of the empire to be determined by just the imperialist and increasing it will increase the role of the colonies in determining the total power of an empire. We have used the value of 0.1 for ξ in most of our implementation.

4.2.5. Imperialistic competition: As all empires tend to take the attainment of the colonies of the other empires and control them, the imperialistic competition gradually brings about a decrease in the power of weaker empires and an increase in the power of the more powerful one. Imperialistic competition executes by picking one (or some) colony or colonies. By this procedure, the most powerful empires will be more likely to possess weak colonies according to the possession probability of each empire. Possession probability of each empire is calculated based on its total power. Normalized total cost of nth empire is obtained by

$$N.T.C._n = max_i\,(T.C._i) - T.C._n\ , i = 1; 2; ...; N_{imp}. \tag{4.9}$$

Where $T.C._n$ and $N.T.C._n$ are normalized total cost and total cost of nth empire respectively. Now the possession probability of each empire is as

$$P_{Pn} = \left| \frac{N.T.C._n}{\sum_{i=1}^{N_{imp}} N.T.C._i} \right|. \tag{4.10}$$

To distribute colonies among empires based on their possession probability based on [3] a new method that has less computational effort than Roulette Wheel selection can be used. For this at the beginning vector P is formed as

$$P = [P_{p1}, P_{p2}, \ldots, P_{pN_{imp}}]. \tag{4.11}$$

Now a vector with the same size as P is created and its elements must be uniformly distributed random numbers between 0 and 1.

$$R = [r_1, r_2, \ldots, r_{N_{imp}}]. \tag{4.12}$$

Then vector D will be obtained by

$$D = P - R = \left[D_1, D_2, D_3, \ldots, D_{N_{imp}}\right] = [P_{p_1} - \quad r_1, P_{p_2} - r_2, \ldots, P_{pN_{imp}} - r_{N_{imp}}]. \tag{4.13}$$

Referring to vector D the mentioned colonies to an empire will be given with maximum index in D.

4.2.6. Eliminating the powerless empires: During imperialistic competition, powerless empires will collapse so that their colonies will be allocated to other powerful empires. Many criteria can be assumed to expurgate the powerless empires. In this research an empire will be collapsed when it misses all of its colonies.

4.2.7. Convergence: After a while all empires except the most powerful one will be terminated and all colonies will be under control of this unique empire. In such a situation, an end will be placed to the imperialistic competition and the algorithm will be stopped.

5. Computational experiments

5.1. Generation of test data

To inquire the effectiveness of the proposed approaches, 10 test problems are considered and each algorithm is run ten times. The problem data can be specified by eight factors in terms of the number of jobs, number of stages, distribution of number of machines at each stage, distribution of processing time, distribution of due dates, distribution of release date, distribution of weight of earliness and tardiness, distribution of processing speed of machines. (Uniform distribution is used to generate these parameters.) All parameters and their distributions are shown in Table 1 concisely.

Table 1: Factors and their levels

factor	Level
Processing times	U$\sim$ [10,30]
Release dates	U$\sim$ [0,8]
Number of machines in each stage	U$\sim$ [1,6]
Number of jobs	4, 6, 10, 20, 40, 60
Due dates	$d_j = (1 + \mathrm{U} \sim [0,1] \times 3) \times \left(\frac{\mathrm{k}}{min_{t \epsilon k}}(m^t)\right) \times (s_i + p_i)$, $\mathrm{i} = 1 \ldots \mathrm{n}$
Weight of earliness and tardiness	U$\sim$[1,5]
Machines's speed	U$\sim$[2,8]
Number of stages	3,5,7

The total processing time of each job on all k stages, average setup time for all possible subsequent jobs and sum of them for all k stages and due date for each job are computed as follow:

$$p_j = \sum_{t=1}^{k} p_j^t, j = 1...n, t = 1...k \ , \tag{5.1}$$

$$s_j = \sum_{t=1}^{k} \frac{\sum_{l=1,j\neq l}^{n} s_{jl}^t}{n-1}, j = 1...n, \tag{5.2}$$

$$d_j = (r_j + random \times 3) \times (k/min_{t\epsilon k}\ (m^t)) \times (s_i + p_i). \tag{5.3}$$

Where random is a random number from a uniform distribution over range [0, 1].

5.2. Parameter tuning

It is known that how to choose the parameters has a considerable influence on the performance of algorithms. In this study the configuration of parameters is down by the Taguchi method. The taguchi method accentuates a mean performance specification value close to the objective value rather than a value within certain characteristic limits, hence prospering the final quality .This is the supremacy of Taguchi method on the other calibrations methods. Additionally, this method for experimental design is straight and easy to apply for many engineering situations. This makes it a powerful yet simple tool. In order to procure preferable and more sturdy results for the proposed algorithms, 7 parameters are considered for tuning. These parameters are Pop_{GA}, P_c, P_m, G_{max}, Pop_{ICA}, decade and P_{ICA}.

Some primary tests are run to find proportionate parameter levels before calibration of the applied algorithms. These parameters and their levels, which appertain to two algorithms, are given in Table 2.

All combination test problems based on the taguchi method, solutions and computation times of the proposed algorithms are presented in Table 3.

By computing all of the experimental results in the Taguchi method, the average S/N ratio and average solution have been acquired for both algorithms and are shown in Figs 5–8, respectively.

Table 2: Algorithm parameters and their levels

factor	**level**
Pop_{GA}	200, 300, 400
P_c	0.6, 0.7, 0.8
P_m	0.10, 0.12, 0.15
G_{max}	200, 300, 400
P_{mu}	0.1, 0.15, 0.2
Pop_{ICA}	300, 400, 500
$decade$	300, 200, 400
P_{ICA}	0.08, 0.10, 0.12

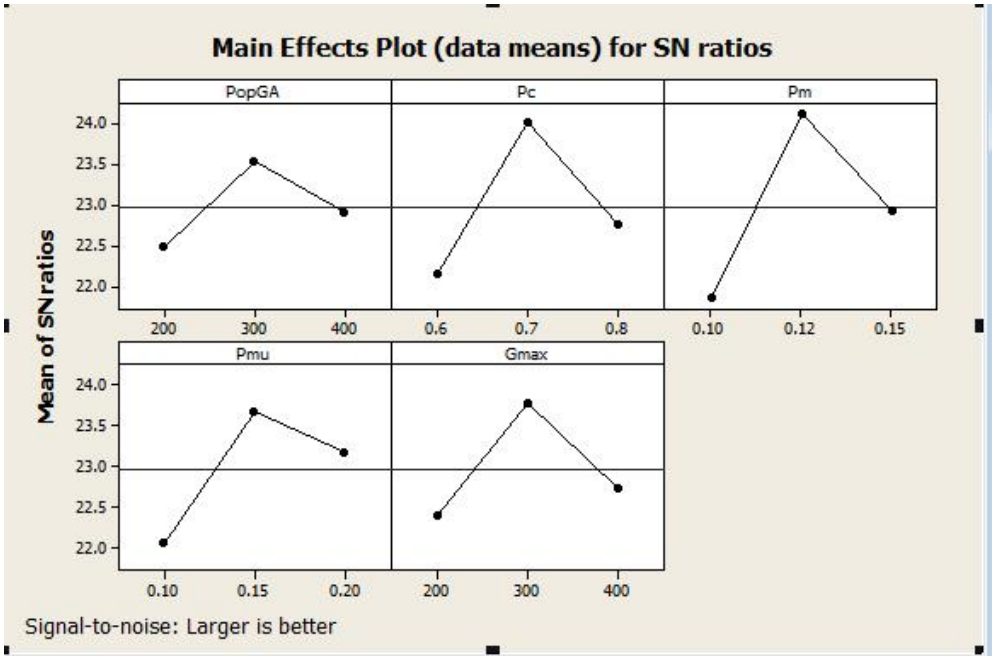

Figure 5: The S/N ratio plot for GA in Taguchi methodology.

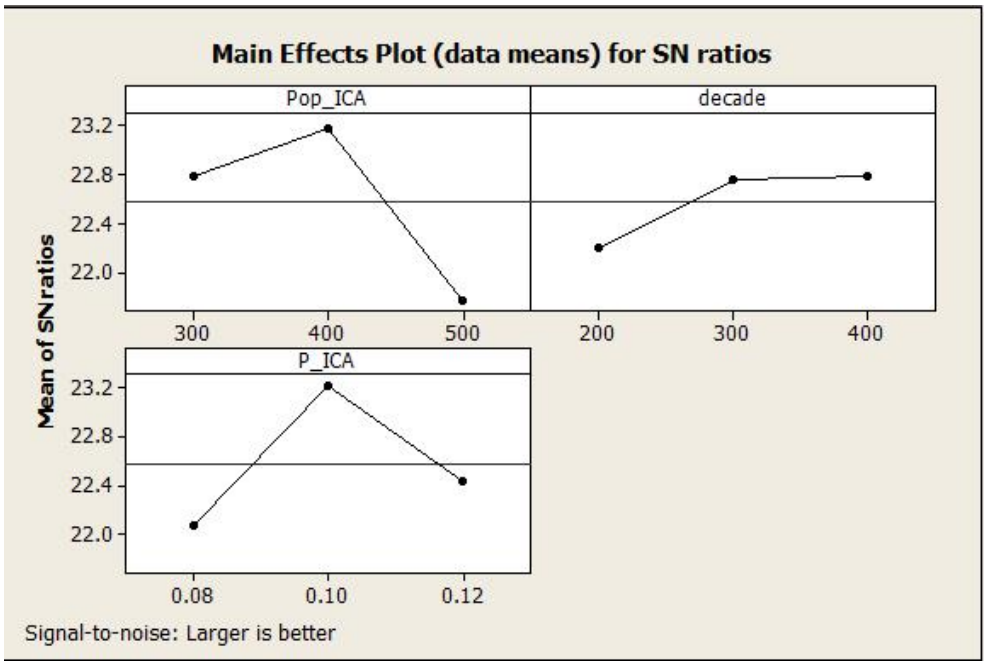

Figure 6: The S/N ratio plot for ICA in Taguchi methodology

Table 3: Results of the proposed algorithms and CPU times (in seconds) for 27 test problems.

Pop_{GA}	Pop_{ICA}	P_{ICA}	P_c	P_m	P_{mu}	G_{max}	decade	Solution (GA)	$Solution\ (ICA)$	$GA.CPU\ time(s)$	$ICA.CPU\ time(s)$
200	300	0.08	0.6	0.10	0.10	200	200	164.00	85.2	6.56	0.66
200	300	0.08	0.6	0.10	0.10	300	200	42.00	40.6	11.80	0.82
200	300	0.08	0.6	0.10	0.10	400	200	46.70	40.2	13.73	1.13
200	300	0.10	0.7	0.12	0.15	200	300	40.00	36.0	8.50	0.65
200	300	0.10	0.7	0.12	0.15	300	300	13.67	23.0	12.60	0.83
200	300	0.10	0.7	0.12	0.15	400	300	32.33	27.8	13.75	1.14
200	300	0.12	0.8	0.15	0.20	200	400	66.33	46.2	12.45	0.65
200	300	0.12	0.8	0.15	0.20	300	400	26.33	31.4	16.58	0.80
200	300	0.12	0.8	0.15	0.20	400	400	27.00	30.8	20.70	1.15
300	400	0.10	0.6	0.12	0.20	200	200	20.67	19.0	14.17	1.29
300	400	0.10	0.6	0.12	0.20	300	200	14.33	19.0	23.07	1.86
300	400	0.10	0.6	0.12	0.20	400	200	48.33	25.6	12.80	0.98
300	400	0.12	0.7	0.15	0.10	200	300	20.33	30.6	16.21	1.27
300	400	0.12	0.7	0.15	0.10	300	300	21.67	37.8	22.89	1.87
300	400	0.12	0.7	0.15	0.10	400	300	44.67	36.2	13.21	0.99
300	400	0.08	0.8	0.10	0.15	200	400	18.67	19.0	22.58	1.28
300	400	0.08	0.8	0.10	0.15	300	400	22.33	21.8	23.54	1.92
300	400	0.08	0.8	0.10	0.15	400	400	49.10	23.0	12.51	0.98
400	500	0.12	0.6	0.15	0.15	200	200	15.00	21.4	28.12	2.55
400	500	0.12	0.6	0.15	0.15	300	200	45.20	24.6	14.30	1.25
400	500	0.12	0.6	0.15	0.15	400	200	26.57	19.0	20.01	1.77
400	500	0.08	0.7	0.10	0.20	200	300	14.87	22.6	27.95	2.59
400	500	0.08	0.7	0.10	0.20	300	300	14.65	26.0	27.80	1.24
400	500	0.08	0.7	0.10	0.20	400	300	13.00	23.4	29.14	1.66
400	500	0.10	0.8	0.12	0.10	200	400	8.60	32.2	34.00	2.60
400	500	0.10	0.8	0.12	0.10	300	400	9.20	28.2	31.23	1.30
400	500	0.10	0.8	0.12	0.10	400	400	7.60	22.6	31.46	1.74

Table 4: S/N ratio table for GA and ICA.

Level	$\boldsymbol{Pop_{GA}}$	$\boldsymbol{P_c}$	$\boldsymbol{P_m}$	$\boldsymbol{P_{mu}}$	$\boldsymbol{G_{max}}$	$\boldsymbol{Pop_{ICA}}$	***decade***	$\boldsymbol{P_{ICA}}$
1	14.33	13.51	12.92	13.30	14.02	22.79	22.20	22.07
2	15.20	16.18	16.41	15.59	15.74	23.17	22.75	23.17
3	13.98	13.82	14.18	14.62	13.74	21.77	22.78	21.77
Delta	1.22	2.67	3.49	2.29	2.00	1.40	0.59	1.15
Rank	5	2	1	3	4	1	3	2

As illustrated in Figs 5–8, optimal levels of Pop_{GA}, P_c, P_m, P_{mu}, G_{max}, Pop_{ICA}, decade and P_{ICA} are 300, 0.6, 0.12, 0.15, 300, 400, 300, 0.10 respectively.
Moreover, Table 4 exhibits the order of factor in minimizing objective function in rank row for used algorithms.

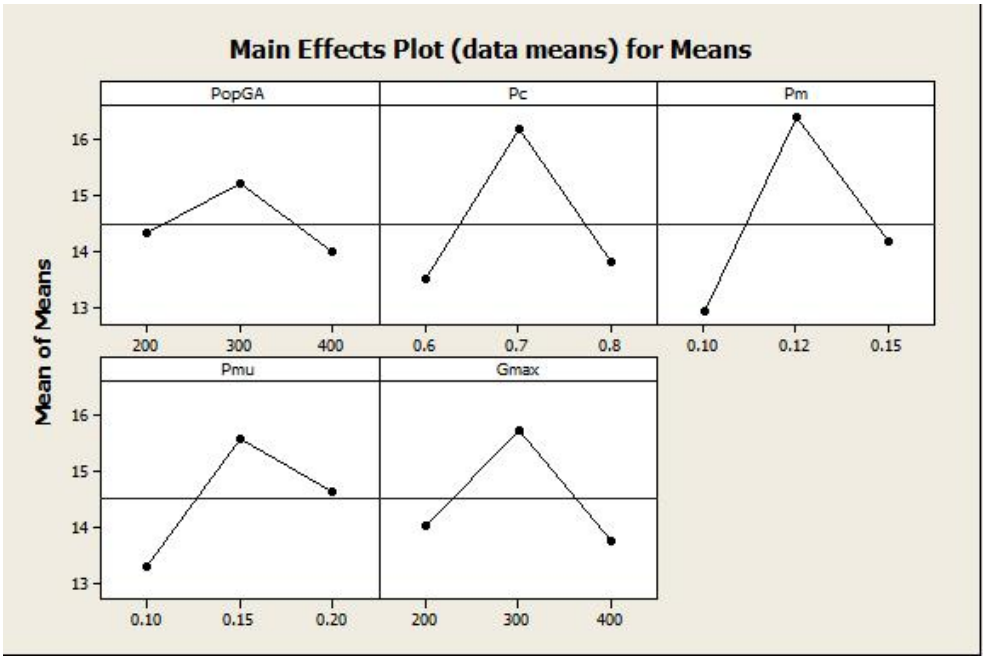

Figure 7: Diagram of mean effect of parameters for GA.

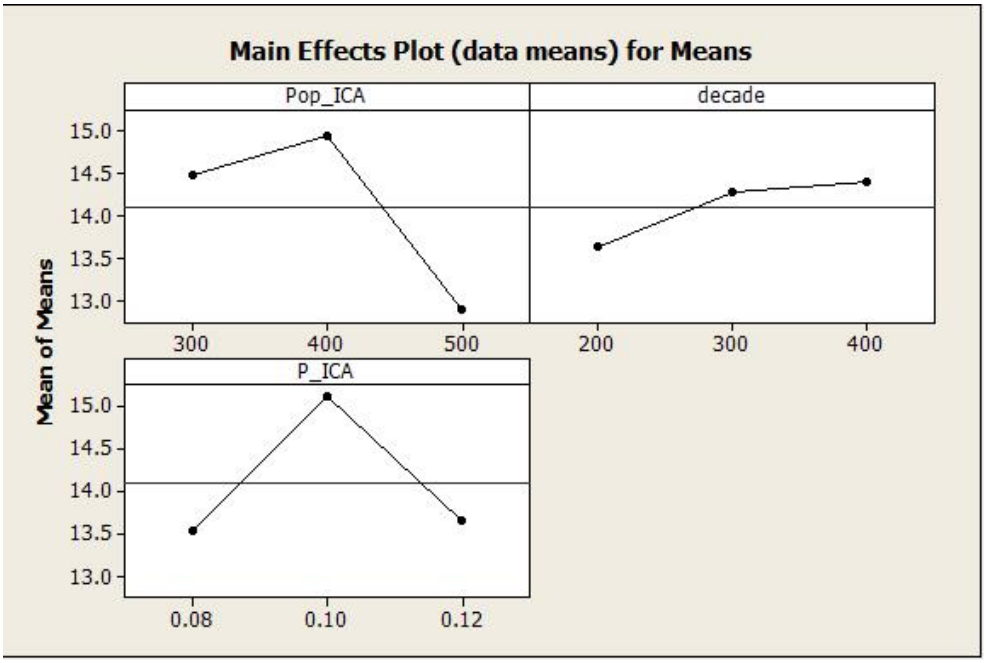

Figure 8: Diagram of mean effect of parameters for ICA.

Table 5: Received values from different runs of lingo for two proposed algorithm.

job	stage	*Lingo*		*GA*				*ICA*				*GA*		*ICA*	
		Global Optimal	Time	*Best*	*Worst*	*Mean*	*Time*	*Best*	*Worst*	*Mean*	*Time*	*RPD*	Average computation time(s)	*RPD*	Average computation time(s)
4	3	1	0:00:06	1	1	1	0:00:16	1	1	1	0:00:14	0.00	0:00:16	0.00	0:00:14
	5	4	0:00:09	4	4	4	0:00:17	4	4	4	0:00:17	0.00	0:00:17	0.00	0:00:17
	7	9	0:00:11	9	9	9	0:00:20	9	9	9	0:00:19	0.00	0:00:20	0.00	0:00:19
6	3	6	0:00:10	6	6	6	0:00:30	6	6	6	0:00:30	0.00	0:00:30	0.00	0:00:30
	5	12	0:05:42	12	12	12	0:00:37	11	12	11.33	0:00:33	0.00	0:00:37	0.00	0:00:33
	7	11	0:19:11	11	11	11	0:00:43	11	11	11	0:00:43	0.00	0:00:43	0.00	0:00:43
10	3	15	0:56:01	15	15	15	0:01:21	15	15	15	0:01:47	0.00	0:01:21	0.00	0:01:47
	5	–	–	20	26	21.2	0:02:33	21	26	22.2	0:02:29	0.06	0:02:33	0.06	0:02:29
	7	–	–	18	22	19.4	0:03:00	14	17	15.6	0:03:21	0.08	0:03:00	0.11	0:03:21
20	3	–	–	47	51	47.66	0:21:36	48	51	49.8	0:23:04	0.01	0:21:36	0.04	0:23:04
	5	–	–	46	48	46.2	0:29:58	49	50	49.5	0:27:50	0.004	0:29:58	0.01	0:27:53
	7	–	–	19	25	22	0:28:46	19	24	23.33	0:28:49	0.16	0:28:46	0.23	0:28:49
40	3	–	–	35	40	36.2	0:32:16	35	39	37.8	0:30:10	0.03	0:32:16	0.03	0:30:10
	5	–	–	34	37	35.6	0:38:36	36	40	39.6	0:39:23	0.05	0:38:36	0.10	0:39:23
	7	–	–	16	18	17.2	0:36:34	15	17	16.8	0:37:12	0.08	0:36:34	0.12	0:37:12
60	3	–	–	70	86	72.2	0:37:45	69	73	71.2	0:36:19	0.03	0:37:45	0.03	0:36:19
	5	–	–	55	61	56.6	0:38:44	58	64	63.6	0:40:55	0.07	0:38:44	0.10	0:40:55
	7	–	–	68	73	70	0:40:08	68	74	70.2	0:41:17	0.03	0:40:08	0.03	0:41:17

5.3. Computational evaluation

In this section, the efficiency of employed GA and ICA are appraised. These algorithms are implemented in MATLAB 11 and run on PC with 1.6 GHZ processor and 4 GB RAM memories. The comparison of algorithms is accomplished with a common performance measure which is known as RPD (relative percentage deviation) to evaluate them. In order to evaluate performance of suggested algorithms, a number of random problems in three types of small, medium and large size are used. The algorithms will be finished after 100 seconds. The best obtained solution, worst obtained solution, mean of obtained solutions and mean run times for each instance are calculated (see Table 5).

RPD is computed by the given formula as below:

$$RPD = \frac{sol_{avg} - sol_{min}}{sol_{min}} \times 100 \tag{5.4}$$

Where sol_{avg} is the average of the obtained solutions for given algorithms and sol_{min} is the best value through algorithms in a related problem. It could be inferred from Table 5 that, GA yields more proper solutions than ICA.

In order to have a significant statistical analysis of difference between algorithms, the 95% confidence interval for computed values in RPD is calculated in two algorithms. Fig. 9 illustrates the 95% confidence interval for related RPD to each algorithm. As it can be seen in Fig. 9, GA works better than ICA. Also, to scrutinize algorithms a sensitivity analysis for ARPD values by considering variations of the number of jobs and the number of stages are shown in Fig. 10 and 11, respectively. The results demonstrate that there exist some insignificant difference between performances of the algorithms with increasing the number of jobs and stages.

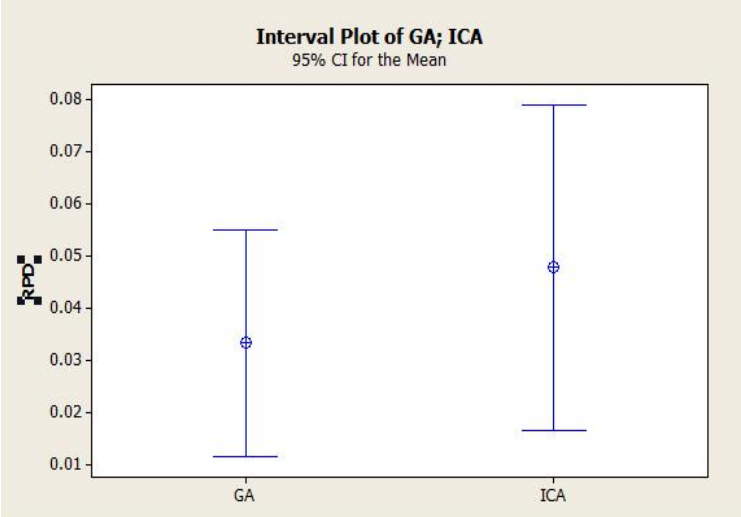

Figure 9: Means plot and LSD intervals (at the 95% confidence level) for algorithms.

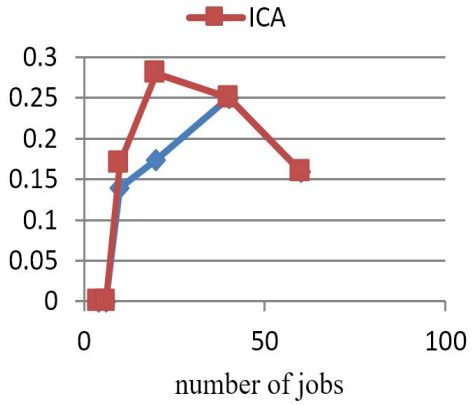

Figure 10: Interaction between performances of algorithms and number of jobs in terms of RPD.

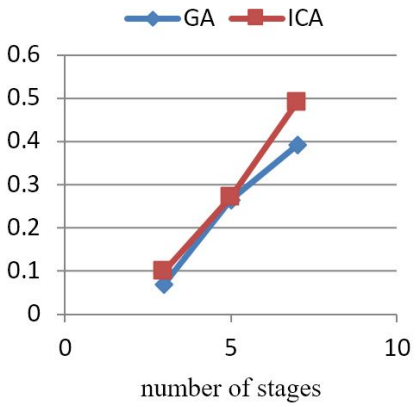

Figure 11: Interaction between performances of algorithms and number of stages in terms of RPD.

6. Conclusion

This study has considered solving a flexible flowshop scheduling problem with sequence and machine dependent setup times, machine breakdown and machine eligibility. In order to find proper schedules that minimize sum of the weighted earliness and tardiness, genetic algorithm (GA) and

imperialist competitive algorithm (ICA) are studied for mentioned problem. In order to reinforce the algorithm and achieve reliable results, also to prevent carrying out extensive experiments to find optimum parameters of the algorithm, the Taguchi method was applied for these goals. To assess the effectiveness of our proposed algorithms some random problems were generated and results obtained by algorithms were analyzed and compared with each other. Furthermore, sensitive analysis of the performance of the proposed algorithms demonstrated that GA in all cases was better than ICA. As an interesting future research, some practical assumptions can be added, including unrelated machine in each stage, limited buffer, preemption and job permutation. Moreover, hybridization of these algorithms could be addressed in future.

References

[1] A. Allahverdi, T. C. Ng, T. C. E. Cheng, M. Y. Kovalyov, *A survey of scheduling problems with setup times or costs*, European Journal of Operational Research, **187** (2008), 985–1032. 1

[2] T. S. Arthanari, K. G. Ramamurthy, *An extension of two machines sequencing problem*, Operation search, **8** (1971), 10–22. 1

[3] E. Atashpaz Gargari, C. Lucas, *Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition*, IEEE Congress on Evolutionary Computation, Singapore, (2007), 4661–4667. 4.2, 4.2

[4] J. Behnamian, S. M. T. Fatemi Ghomi, *Hybrid flowshop scheduling with machine and resource-dependent processing times*, Applied Mathematical Modeling, **35** (2011), 1107–1123. 1

[5] J. Behnamian, S. M. T. Fatemi Ghomi, M. Zandieh, *A multi-phase covering Pareto-optimal front method to multi-objective scheduling in a realistic hybrid flowshop using a hybrid metaheuristic*, Expert Systems with Applications, **36** (2009), 11057–11069. 1

[6] J. Behnamian, S. M. T. Fatemi Ghomi, M. Zandieh, *Development of a hybrid metaheuristic to minimize earliness and tardiness in a hybrid flowshop with sequence-dependent setup times*, International Journal of Production Research, **48** (2010), 1415–1438. 1

[7] A. Bellanger, S. Hanafi, C. Wilbaut, *Three-stage hybrid-flowshop model for cross-docking*, Computers & Operations Research, **40** (2013), 1109–1121. 1

[8] V. Botta-Genoulaz, *Hybrid flowshop scheduling with precedence constraints and time lags to minimize maximum lateness*, International Journal of Production Economics, **64** (2000), 101–111. 1

[9] H. S. Choi, J. S. Kim, D. H. Lee, *Real-time Scheduling for Reentrant Hybrid Flow Shops: a Decision Tree Based Mechanism and Its Application to a TFT-LCD Line*, Expert Systems with Applications, **38** (2011), 3514–3521. 1

[10] O. Engin, A. Döyen, *A new approach to solve hybrid flowshop scheduling problems by artificial immune systemv*, Future Generation Computer Systems, **20** (2004), 1083–1095. 1

[11] E. Figielska, *A genetic algorithm and a simulated annealing algorithm combined with column generation technique for solving the problem of scheduling in the hybrid flowshop with additional resources*, Computers & Industrial Engineering, **56** (2009), 142–151. 1

[12] E. Figielska, *A heuristic for scheduling in a two-stage hybrid flowshop with renewable resources shared among the stages*, European Journal of Operational Research, **236** (2014), 433–444. 1

[13] M. Gholami, M. Zandieh, *An immune algorithm for scheduling a hybrid flowshop with sequence-dependent setup times and machines with random breakdowns*, International Journal of Production Research, **47** (2009), 6999–7027. 1

[14] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning, Reading*, MA: Addison-Wesley, (1989). 4.1

[15] J. N. D. Gupta, K. Krüger, V. Lauff, F. Werner, Y. N. Sotskov, *Heuristics for hybrid flow shops with controllable processing times and assignable due dates*, Computers & Operations Research. **29** (2002), 1417–1439. 1

[16] M. Heydari, E. Mohammadi, *A fuzzy heuristic algorithm for the flow shop scheduling problem*, The Journal of Mathematics and Computer Science, **4** (2010), 349–354. 1

[17] Y. Honghong, W. Zhiming, *The application of Adaptive Genetic Algorithms in FMS dynamic rescheduling*, International Journal of Computer Integrated Manufacturing, **16** (2003), 382. 4.1

[18] A. Janiaka, E. Kozanb, M. Lichtensteina, C. Oguzc, *Metaheuristic approaches to the hybrid flowshop*

scheduling problem with a cost-related criterion, International Journal of Production Economics, **105** (2007), 407–424. 1

[19] Z. Jin, Z. Yang, T. Ito, *Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem*, International Journal of Production Economics, **100** (2006), 322–334. 1

[20] J. Jungwattanakit, M. Reodecha, M. P. Chaovalitwongse, F. Werner, *A comparison of scheduling algorithms for flexible flowshop problems with unrelated parallel machines, setup times, and dual criteria*, Computer & Operation Research, **36** (2009), 358–378. 1

[21] S. Khalouli, F. Ghedjati, A. Hamzaoui, *A meta-heuristic approach to solve a JIT scheduling problem in hybrid flow shop*, Engineering Applications of Artificial Intelligence, **23** (2010), 765–771. 1

[22] H. R. Kia, H. Davoudpour, M. Zandieh, *Scheduling a dynamic flexible flow line with sequence-dependent setup times: a simulation analysis*, International Journal of Production Research, **48** (2010), 4019–4042. 1

[23] F. S. C. Lam, B. C. Lin, C. Sriskandarajah, H. Yan, *Scheduling to minimize product design time using a genetic algorithm*, International Journal of Production Research, **37** (1999), 1369–1386. 4.1

[24] R. Logendran, S. Carson, E. Hanson, Group scheduling in flexible flowshops, International Journal of Production Economics, **96** (2005), 143–155. 1

[25] O. Moursli, Y. Pochet, *A branch-and-bound algorithm for the hybrid flowshop*. International Journal of Production Economics, **64** (2000), 113–125. 1

[26] A. Naderi, M. Zandieh, A. Khaleghi Ghoshe Balagh, V. Roshanaei, *An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness*, Expert Systems with Applications, **36** (2008), 9625–9633. 1

[27] T. Nishi, Y. Hiranaka, M. Inuiguchi, *Lagrangian relaxation with cut generation for hybrid flowshop scheduling problems to minimize the total weighted tardiness*, Computers & Operations Research, **37** (2010) 189–198. 1

[28] Q. K. Pan, L. Wang, J. Q. Li, J. H. Duan, *A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimization*, Omega, **45** (2014), 42–56. 1

[29] C. R. Reeves, *Genetic algorithms for the operations researcher*, INFORMS Journal on Computing, **9** (1997), 231–250. 4.1

[30] R. Ruiz, C. Maroto, *A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility*, European Journal of Operational Research, **169** (2006), 781–800. 1

[31] M. S. Salvador, *A solution to a special case of flowshop scheduling problems*, In: Elmaghraby SE, editor, Symposium of the theory of scheduling and its applications, New York: Springer, (1973), 83–91. 1

[32] H. Seidgar, M. Ezzati, M. kiani, R. Tavakoli-Moghaddam, *An Efficient Genetic Algorithm for Two-stage Hybrid Flow Shop Scheduling with Preemption and Sequence Dependent Setup Time*, Journal of Mathematics and Computer Science, **6** (2013), 251–259. 1

[33] L. Su, *A hybrid two-stage flowshop with limited waiting time constraints*, Computers & Industrial Engineering, **44** (2003), 409–424. 1

[34] L. Tang, W. Liu, J. Liu, *A neural network model and algorithm for the hybrid flowshop scheduling problem in a dynamic environment*, Journal of Intelligent Manufacturing, **16** (2005), 361–370. 1

[35] L. Tang, H. Xuan, J. Liu, *A new Lagrangian relaxation algorithm for hybrid flowshop scheduling to minimize total weighted completion time*, Computers & Operations Research, **33** (2006), 3344–3359. 1

[36] S. Vob, A. Witt, *Hybrid flowshop scheduling as a multi-mode multi-project scheduling problem with batching requirements*, International Journal of Production Economics, **105** (2007), 445–458. 1

[37] X. Wang, L. Tang, *A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers*, Computers & Operations Research, **36** (2009), 907–918. 1

[38] J. Xie, X. Wang, *Complexity and algorithms for two-stage flexible flowshop scheduling with availability constraints*, Computer and Mathematics with Application, **50** (10–12) (2005), 1629–1638. 1

[39] Y. Yang, S. Kreipl, M. Pinedo, *Heuristics for minimizing total weighted tardiness in flexible flowshop*, Journal of Scheduling, **3** (2000), 71–88. 1

[40] V. Yaurima. L. Burtseva, A. Tchernykh, *Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers*, Computers & Industrial Engineering, **56** (2009), 1452–1463. 1

[41] M. Zandieh, S. M. T. Fatemi Ghomi, S. M. Moattar Husseini, An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times, Journal of Applied Mathematics and Computation, **180** (2006), 111–127. 1