FOUNDATIONS OF COMPUTING AND DECISION SCIENCES
Vol. 37 (2012) No. 3

DOI: 10.2478/v10209-011-0009-6

MINIMIZING TOTAL TARDINESS IN NO-WAIT FLOWSHOPS

Tariq ALDOWAISAN[1]
Ali ALLAHVERDI[1]

Abstract. We address the *m*-machine no-wait flowshop scheduling problem; where the objective is to minimize total tardiness. To the best of our knowledge, the considered problem has not been addressed so far. We propose heuristic solutions since the problem is *NP*-hard. Initially, we consider a number of dispatching rules commonly used for the considered objective in other scheduling environments. We identify through computational experiments the best performing dispatching rule; and then propose simulated annealing (*SA*) and genetic algorithms (*GA*) by using the best performing dispatching rule as an initial solution. This achieves at least 50% improvement in the *SA* and *GA* performances. Next, we propose enhanced versions of *SA* and *GA* and show through computational experiments that the enhanced versions provide over 90% further improvement. The performance of enhanced *GA* is slightly better than that of enhanced *SA*; however, the computation time of enhanced *GA* is about 10 times that of enhanced *SA*. Therefore, we conclude that the enhanced *SA* outperforms the enhanced *GA*.

Keywords: No-wait, scheduling, flowshop, total tardiness.

1. Introduction

In a flowshop environment, there is a set of *n* jobs that have to be processed on a set of *m* machines. All the jobs follow the same route in the machines starting with machine one until machine *m*. For some flowshop environments, the operations of a job have to be processed continuously from start to end without interruptions either on or between machines. In other words, if necessary, the start of a job on a given machine is delayed in order that the operation's completion coincides with the start of the next operation on the subsequent machine. Such flowshops are called no-wait flowshops.

[1] Department of Industrial and Management Systems Engineering, College of Engineering and Petroleum, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait, Fax: +965 2481 6137, Email:tariq.aldowaisan@ku.edu.kw, Email:ali.allahverdi@ku.edu.kw

Several industries exist, where the no-wait constraint applies including metal, plastic, chemical and food industries; which are common in Kuwait and Gulf countries. For instance, in the case of rolling of steel, the heated metal must continuously go through a sequence of operations before it is cooled in order to prevent defects in the composition of the material. Also in the food processing industry, the canning operation must immediately follow the cooking operation to ensure freshness. Hall and Sriskandarajah [12] provided a detailed presentation of applications and research on no-wait flowshop problems.

The *m*-machine no-wait flowshop problem has been addressed by many researchers, mainly with respect to makespan or total completion time performance measures. Aldowaisan and Allahverdi [3], Allahverdi and Aldowaisan [4], Framinan and Nagano [10], and Pan et al. [15] addressed the problem with respect to makespan performance measure. The performance measure of makespan is directly related to utilization of resources while the total completion time performance measure is directly related to the cost of inventory. The significance of minimizing the total cost of inventory has been discussed by many researchers, e.g., Warburton [19]. Some of the works on the problem with the performance measure of total completion time include Chen et al. [9], Aldowaisan and Allahverdi [4], Chang et al. [8], and Pan et al. [15].

The aforementioned performance measures of makespan and total completion time can be considered as related to job completion times. There are some other performance measures which are related to job due dates such as number of tardy jobs and total tardiness. Aldowaisan and Allahverdi [1] proposed several heuristics for the *m*-machine no-wait flowshop scheduling problem with respect to the objective of minimizing the number of tardy jobs. The minimization of the number of tardy jobs is important in scheduling since certain costs are incurred when jobs are completed after their due dates. As reported by Aldowaisan and Allahverdi [1], this performance measure is also directly related to the percentage of on-time shipments, which is often used to rate managers' performance. However, the performance measure of the number of tardy jobs does not differentiate among the late jobs (beyond their due dates), while it is a known fact that the cost increases as the gap between a job's due date and its completion time increases. Hence, late jobs should be differentiated as to how much their completion times are beyond their due dates. The costs as a result of late completion time of jobs may also be penalty costs in contracts, loss of goodwill, and damaged reputation. For such scheduling environments the performance measure of total tardiness is more appropriate than the number of tardy jobs. To the best of our knowledge, there has been no research addressing the *m*-machine no-wait scheduling problem with respect to the total tardiness performance measure. In this paper, we address this problem.

It should be noted that for other scheduling environments a lot of research has been conducted with respect to the total tardiness performance measure. For example, Vallada et al. [18] provided an excellent review and evaluation of heuristics and metaheuristics for the regular (no no-wait) *m*-machine flowshops with the objective of minimizing total tardiness. Framinan and Leisten [11] presented a greedy algorithm and Vallada et al. [17] presented a genetic algorithm for the same problem.

2. Notation

Every job $j = 1, \ldots, n$ has to be processed continuously in the same order on a set of $i = 1, \ldots, m$ machines without interruptions either on or between machines. We assume that the set of n jobs are ready for processing at time zero. The following notation will be used throughout the paper:

E: tardiness factor.
R: due date range factor.
d_j: due date of job j; which is generated using the E and R factors.
C_j: completion time of job j on the last machine.
T_j: tardiness of job j; where $T_j = max\ \{C_j\text{-}d_j , 0\}$.
P_{ij}: processing time of job j on machine i.
$\acute{P}_j$: average processing time of job j across all machines; i.e. $\acute{P}_j = \frac{\sum_{i=1}^{m} P_{ij}}{m}$
$\acute{\acute{P}}_j$: variation normalized average processing time of job j across all machines; i.e. $\acute{\acute{P}}_j = \frac{\sigma_j}{\min(\sigma_j)} * \acute{P}_j$ and $\sigma_j = \sqrt{\frac{\sum_{i=1}^{m}(P_{ij}-\acute{P}_j)^2}{m}}$
$\acute{d}_j$: due date of job j for reduced m-machine problem to single-machine one; i.e. $\acute{d}_j = d_j/m$.

The objective is to sequence all jobs to minimize the total tardiness (TT); i.e. $\sum_{j=1}^{n} T_j$.

3. Dispatching Rules

Dispatching rules are simple heuristics for building a schedule. Their popularity is due to their ability to rapidly provide good solution in practical production settings. They are also used as initial sequences for metaheuristic and improvement heuristics.

Let s denote the sequence of jobs which are scheduled so far and t denote the time at which jobs need to be selected. Moreover, let $C_j(s)$ denote the completion time of job j considered for scheduling (not in s yet) if it is scheduled as the last job in the sequence s. The following are five commonly used dispatching rules for the total tardiness m-machine regular flowshop problem (Vallada et al. 2008):

1. Earliest due date (EDD); where jobs are sequenced in non-decreasing order of their due dates.
2. Earliest due date with processing times (*EDDP*); where jobs are sequenced in non-decreasing order of $d_j/\sum_{i=1}^{m} P_{ij}$.
3. Modified due date (*MDD*); where jobs are sequenced in non-decreasing order of $max\{d_j , C_j(s)\}$.
4. *SLACK*; where jobs are sequenced in non-decreasing order of $(d_j - C_j(s))$.
5. Slack per remaining work (*SRMWK*); where jobs are sequenced in non-decreasing order of $(d_j - C_j(s))/\sum_{i=1}^{m} P_{ij}$.

We propose two new dispatching rules by first reducing the m-machine problem to a single-machine problem, for which an optimal algorithm exists; and then using the optimal algorithm to determine the solution sequence as follows:

1. Single-machine processing time (*SMPT*); where jobs are sequenced by reducing the problem to a single-machine problem with processing times $\acute{P}_j = \sum_{i=1}^{m} P_{ij} / m$ and due dates $\acute{d}_j = d_j / m$, and then finding the optimal job sequence that minimizes total tardiness [16].
2. Adjusted single-machine processing time (*ASMPT*); where the single-machine processing time $\acute{P}_j$ is adjusted to account for the variations of processing times among the m-machines as follows: $\acute{\acute{P}}_j = \frac{\sigma_j}{\min(\sigma_j)} * \acute{P}_j$ and $\sigma_j = \sqrt{\frac{\sum_{i=1}^{m}(P_{ij} - \acute{P}_j)^2}{m}}$. The optimal job sequence is then determined similar to *SMPT* but using the adjusted processing time $\acute{\acute{P}}_j$. The *ASMPT* normalizes the averages of the processing time of jobs across machines by multiplying by a factor reflecting the difference in variances of the processing times on machines for the considered jobs.

For the considered experimentation parameters for *n*, *m*, *E*, and *R*; and the relative deviation index (*RDI*) error performance measure (to be discussed later in the "Computational Analysis" section), Figure 1 shows the comparison among the five and the two proposed dispatching rules for different number of jobs, while Figure 2 provides the comparison for different number of machines; where $\bar{n}$, $\bar{m}$, $\bar{E}$, and $\bar{R}$ denote the averages. Note that E denotes the tardiness factor, R denotes the due date range factor, and RDI is defined as follows:

$RDI = \frac{Heuristic - Best}{Worst - Best} * 100$, where *Heuristic* is the solution obtained by a given heuristic, *Best* is the best solution obtained from among the compared heuristics, and *Worst* is the worst solution obtained from among the compared heuristics.

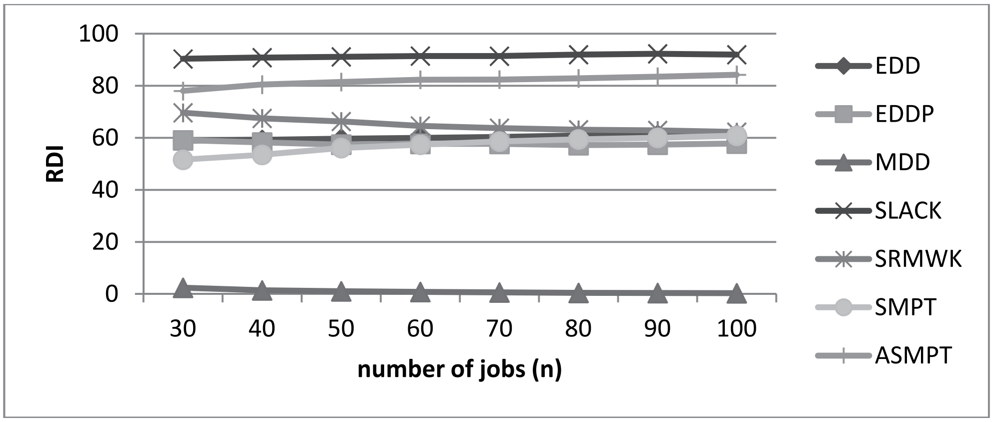

Figure 1. Comparison of dispatching rules for RDI criterion vs. n; for $\bar{E}$, $\bar{R}$ and $\bar{m}$

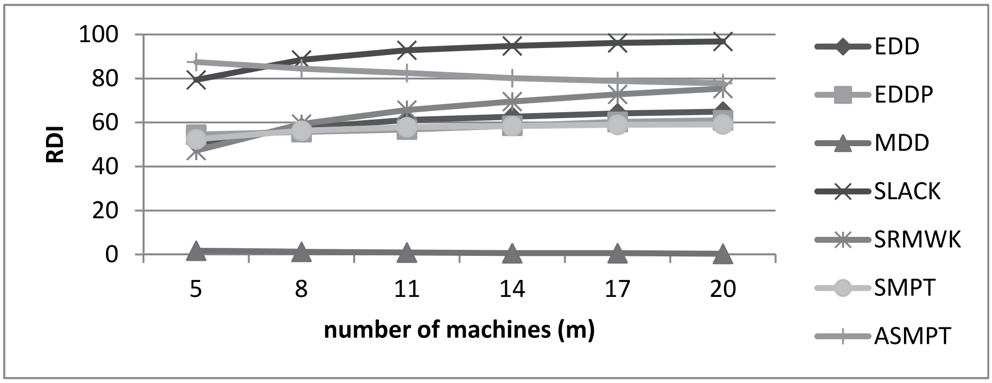

Figure 2. Comparison of dispatching rules for RDI criterion vs. m; for $\bar{E}$, $\bar{R}$ and $\bar{n}$

The proposed *ASMPT* didn't perform well; it only outperformed the *SLACK* dispatching rule. However, *SMPT* did well against all considered dispatching rules except *MDD* and for smaller number of jobs against *EDDP*. As for the number of machines, the proposed *SMPT* performance improves with the increase in the number of machines to outperform all other rules at about m=14; however, *MDD* is still the superior performing dispatching rule.

In the next section, we use *MDD* as an initial sequence to develop an improved simulated annealing heuristic; and use the three best performing dispatching rules *MDD*, *SMPT*, and *EDDP* as part of the initial generation population to develop an improved genetic algorithm heuristic.

4. Metaheuristics

Metaheuristics are computational methods that seek based on a preset objective criterion to search for a good heuristic in a large solution space till a certain stopping criterion is met; e.g. time, number of iterations, improvement threshold value, etc. We develop two improved metaheuristic methods that are based on simulated annealing and genetic algorithm.

4.1 ISA Heuristic

Simulated Annealing (*SA*) is a generic probabilistic metaheuristic for global optimization; locating a good approximation to the global maximum/minimum of a given function in a large search space. It is usually much more efficient than exhaustive brute-force search for such search spaces. Commonly *SA* is applied to an initial random sequence. In this paper, however, we propose an improved simulated annealing (*ISA*) heuristic using the best performing dispatching rule *MDD* as the initial sequence.

During the execution of the simulated annealing, two sequences *S* and *B* are maintained. *S* is the current sequence obtained so far and to be manipulated throughout the algorithm execution. Initially, *S* is set to be the *MDD* sequence. *B* is the best sequence that was obtained so far.

One of the main attributes of any simulated annealing implementations is the ‘temperature’ variable (*T*) which indicates the amount of energy that the system has. Higher *T* means that the system is unstable and the current solution could jump to a distant solution and vice-versa; it is used to determine the probability of accepting an inferior sequence during the search process. The initial temperature is set to T_0. A higher initial temperature would lead to the acceptance of very inferior sequences. For example, setting $T_0 = 500$ leads to accepting a sequence that is inferior by 50% relative to the original sequence, with a probability of 0.905 when the simulated annealing algorithm starts; whereas for $T_0 = 400$, the probability becomes 0.882, and as *T* decreases, so does this probability.

A variable Temperature Reduction Factor (*TRF*) is used to gradually cool down the system by geometrically decreasing *T*. Both *T* and T_0 allow for a reasonable number of iterations to be carried out before the algorithm freezes (i.e. stops).

Another parameter of simulated annealing is the neighbor function which finds a neighbor of a given state. We implement Adjacent Interchange Scheme (*AIS*) to generate a new sequence *S’* by perturbing the current sequence *S*. The *AIS* algorithm chooses a job at random and then interchanges it with its left or right neighbor according to whether a randomly generated number (between 0 and 1) is less than, equal to, or higher than 0.5. When the selected job is either the first or last in the sequence, it should be interchanged with its neighbor.

S is replaced by *S'* in two cases; when *S'* is superior to *S*, or if otherwise, with a probability $P=exp(-\Delta/T)$, where Δ is the relative deterioration of *S’* with respect to *S*, and *T* is the actual temperature. If the replacement was done, a variable *ACC*, used to keep track of the number of accepted moves at a particular temperature, is incremented by one. If *S’* is better than *S*, the best sequence *B* is also checked against the obtained sequence *S’*. If *S’* turns out to be better, *B* is also updated by *S’*.

The total number of moves at some temperature *T* is kept track of with the variable *TOT*. Initially *TOT* and *ACC* are set to 0. They are also set to 0 at the start of every new temperature. When either *TOT* or *ACC* exceed some predefined limits (TOT_{max} and ACC_{max} respectively), the algorithm assumes that at that particular temperature, the chances of obtaining a better sequence is getting very low. In such occasions, the algorithm cools down the temperature *T* by the value *TRF* to obtain a new temperature.

The algorithm freezes when a counter freezing number *FN* reaches a maximum limit of FN_{max}. *FN* is initialized to zero and incremented by one when the percentage of accepted moves at a particular temperature is lower than a preset value $PCNT_{max}$. It is reinitialized to zero each time the best sequence is updated. A high *FN* indicates that the chances of finding better sequences at lower temperatures are diminishing to the point that it is not worth the search effort. *FN* is reset to 0 whenever the best sequence *B* is updated. In addition to FN_{max}, the algorithm will also stop if the actual temperature drops below the freezing temperature *FT*. In other words, the search will continue till either the maximum freeze number FN_{max} is reached or the freezing temperature *FT* is attained. The latter condition is used to terminate the search when the freeze counter is repeatedly reset to 0 which requires a large computational effort.

Many parameters affect the performance of simulated annealing when implemented for optimizing the total tardiness of the no-wait flowshop problem. For every parameter of the considered six parameters a range of values was suggested. Our approach for coming up with the best parameter values was to run the simulated annealing on all the different parameter value combinations, analyze the results, and study the behavior for increasing or decreasing each parameter value.

The maximum accepted moves ACC_{max} is one of the typical parameters considered when studying the behavior of simulated annealing. The values of $n/2$, n, $2n$, $3n$ and $4n$ were tested where n is the number of jobs. The analysis showed that increasing ACC_{max} up to $2n$ results in improved total tardiness while no improvement is gained for higher values. The same was done for maximum total moves TOT_{max}. For that parameter, higher values yielded better total tardiness; however the execution time increased as well. Therefore $5n$ was chosen to balance between the execution time and the quality of the resulting solution.

The same has been done for maximum freezing number FN_{max} and freezing temperature FT. For FN_{max} and FT the values 10 and 20 were selected, respectively.

For the temperature reduction factor TRF, the higher the value is, the longer algorithm takes to complete. It was found that 0.9 and 0.95 gave similar results. 0.90 was chosen for its execution time advantage. Finally, for the maximum accepted moves percentage $PCNT_{max}$ the experiment showed that lower values yielded better results. The value 0.10 was selected since it gave the best average total tardiness. The tested and selected values for the parameters of the improved simulated annealing are given in Table 1.

Table 1. Selected values for the parameters of ISA

Parameter	Description	Tested Values	Selected Values
ACC_{max}	Max Accepted Moves	$n/2$, n, $2n$, $3n$, $4n$	$2n$
TOT_{max}	Max Total Moves	$n/2$, n, $2n$, $3n$, $4n$, $5n$	$5n$
FN_{max}	Max Freezing Number	2, 5, 10, 15, 20	10
FT	Freezing Temperature	5 ,10, 20, 30, 150	20
TRF	Temperature Reduction Factor	0.60, 0.70, 0.80, 0.80, 0.90, 0.95	0.90
$PCNT_{max}$	Max Accepted Move Percentage	0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35	0.10

4.2 IGA Heuristic

A genetic algorithm (*GA*) is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms, which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

An initial generation of structures or solutions (job sequences in our case) is randomly selected and new generations are generated by applying simple genetic operations on the current population structures. However, in this paper an improved genetic algorithm (*IGA*) is developed using the best performing three dispatching heuristics *MDD*, *SMPT*, and *EDDP* as part of the initial population; and the remaining initial population members are generated randomly.

The structures from the current population with higher fitness values are given higher chance to be coupled together and thus increase the chance of getting offspring of high fitness. This process is repeated many times leading to continuous improvement of the solutions fitness.

The scheduling problem is a possible application of the *IGA* provided that a set of genetic operations is defined between two job sequences and a job sequence fitness evaluation procedure is specified. The *IGA* algorithm is next described, where *P(t)* denotes the population at the t^{th} generation, $s_i(t)$ represents the i^{th} job sequence in *P(t)* and $f(s_i(t))$ is the fitness value of $s_i(t)$.

It starts by first specifying the population size *POPSIZE*, the number of generations *NGEN* and setting *t* to 0. The initial generation *P(0)* is generated randomly along with the three best performing dispatching solutions. Starting with good sequences speeds up the process of fitness improvement. Next, the fitness value $f(s_i(t))$ of each job sequence in the population P(*t*) is calculated as follows:

$$f\big(s_i(t)\big) = \frac{\max\limits_j\,(Obj(s_j(t))) - Obj(s_i(t))}{\sum_k(\max\limits_j(Obj(s_j(t))) - Obj(s_k(t)))}$$

Where $Obj(s_i(t))$ is the objective function value of the sequence $s_i(t)$ to be minimized; i.e. the total tardiness in our case. After that, the selection probability $p_i(t)$ of each job sequence $s_i(t)$ in *P(t)* is calculates as follows:

$$p_i(t) = f\big(s_i(t)\big)/\sum_j f(s_j(t))$$

Two job sequences (the parents) are picked from the population based on their selection probability for reproduction (crossover and mutation). Sequences with higher probability are favored over the others. At most ten selection trials are made to ensure that the two selected sequences are different. The parents are submitted to crossover and mutation with probabilities *PCROSS* and *PMUTE* respectively before adding *P(t+1).* Crossover uses Goldberg's *PMX* operator while mutation consists of swapping two randomly selected jobs in the job sequence. The process of selecting the parents and applying the mutation and the crossover is repeated until the new generation is constructed. This way, it is guaranteed that sequences with highest fitness value are always passed to the children populations. Once *P(t+1)* is complete it becomes the new parent population and *t* is incremented by 1. The whole process is repeated until *t* reaches *NGEN* or the fitted percent of the new generation reaches *POPFIT*; where *POPFIT* denotes population fitted percentage and it is assigned a value of 60%. The i^{th} job sequence is considered fitted if:

$$\max_j\,(f(s_j(t)) - f(s_i(t))) \le FITTOLER$$

where, *FITTOLER* denotes fitted tolerance and it has a value of 0.10.

Finally, the job sequence in the most recent generation with the highest fitness value is selected as the solution of the scheduling problem.

Several parameters affect the performance of genetic algorithm when implemented for optimizing the total tardiness of the no-wait flowshop problem. For every parameter a range of values was suggested. Similar to the *ISA*, our approach for determining the best parameter values was to run the *IGA* on all the different parameter value combinations, analyze the results, and study the behavior for increasing or decreasing each parameter value.

Probability of crossover *PCROSS* and probability of mutation *PMUTE* are important to improve the existing population and to maintain diversity in order not to trap at a local minimum. The values of 0.70, 0.75, 0.80, 0.85, and 0.90 were tested to decide the value for *PCROSS*. Experimental tests were conducted, and the results indicated that the performance of *GA* improved as *PCROSS* value was increased up to 0.75, and remained almost the same for higher values. In order to decide the *PMUTE* value, 0.00, 0.01, 0.02, 0.03, 0.04, 0.05 and 0.10 were considered and 0.05 was selected after the pilot runs.

Finally, for population size *POPSIZE* and number of generations *NGEN* parameters, first we set a fixed value for the problems with different job sizes, *n*. We tested varying values for these two parameters. The tested values were *n/2*, *n*, 2*n*, and 3*n*. It was observed that the computation time increased significantly as *n* increases, thus we chose *n* for *POPSIZE* and *2n* for *NGEN*, which yielded the best total tardiness while considering the least possible execution time. The tested and selected values for the IGA parameters are given in Table 2.

Table 2. Selected values for the parameters of IGA

Parameter	Description	Tested Values	Selected Values
PCROSS	Probability of Crossover	0.70, 0.75, 0.80, 0.85, 0.90	0.75
PMUTE	Probability of Mutation	0.00, 0.01 0.02, 0.03, 0.04, 0.05, 0.10	0.05
POPSIZE	Population Size	n/2, n, 2n, 3n	n
NGEN	Number of Generations	n/2, n, 2n, 3n	2n

4.3 FISA and FIGA Heuristics

Using the heuristic of *PAAH* of Allahverdi and Aldowaisan [5], where the *ISA* sequence is used as the initial sequence π in *PAAH* of Allahverdi and Aldowaisan, and where block size in *PAAH* is chosen as one. The resulting sequence becomes the further enhanced *ISA* solution (i.e. *FISA*). Similarly, we apply the heuristic *PAAH* to the *IGA* sequence to obtain a further enhanced *IGA* solution (i.e. *FIGA*.) In the following section, the PAAH heuristic is adapted to our problem.

The PAAH heuristic consists of three main parts; obtaining an initial sequence, improving the initial sequence using an insertion procedure, and further improving the solution using an exchange procedure. First we explain the insertion procedure that will be used in PAAH.

Step 1: Use *ISA* as an initial sequence π.

Step 2: Set k=1. Pick the first job from π and sequence it in order to minimize the partial objective function value of TT. Select the best sequence as a current solution.

Step 3: Set k=k+1. Generate candidate sequences by selecting the next job from π and inserting this job into each position or slot of the current solution. Among these candidates, select the best one with the least partial TT value. Set the best one as a current solution.

Step 4: Repeat step 3 until all the jobs in π have been considered.

Next, the solution is further improved using the following exchange procedure.

Step 1: Set $k=u=d=2$, $p=c=g=1$, $\pi_1=\{1, ..., n\}$, $\pi_2=\phi$. Set the number of repetition of insertions and exchanges to be number of repetition of insertions (NRI) and number of repetition of exchanges (NRE). The values of NRI and NRE are set to 10 and 3 respectively as experimentally was determined by Allahverdi and Aldowaisan [5].

Step 2: Choose job i such that

$$\sum_{j=1}^{m} t_{i,j} \leq \sum_{j=1}^{m} t_{r,j} \text{ for all r in } \pi_1.$$

Remove job i from π_1 and place it in the first position of π_2.

Step 3: Calculate the partial objective function of TT for each job $i \in \pi_1$ after inserting it in position k of π_2 and assign the job with the minimum partial objective function of TT in position k in π_2 and remove it from π_1

Let $k=k+1$. Set $\pi=\pi_2$.

Step 4: Go to Step 3 if $k<n$, otherwise go to Step 5.

Step 5: Apply the proposed insertion procedure to the sequence π_d. Let $d=d+1$ and let the sequence obtained after the insertion be called π_d. If TT(π_d)<TT(π), then set $\pi=\pi_d$.

Step 6: Apply the Nawaz et al. [14] insertion procedure to the sequence π_d. Let $d=d+1$ and let the sequence be π_d. If TT(π_d)<TT(π), then set $\pi=\pi_d$. Let $g=g+1$.

Step 7: If g<NRI, go to Step 5

Step 8: Exchange the jobs in positions c and u of the sequence π. Let the new sequence be π. If TT(π_d)>TT(π), then set $\pi_d=\pi$.

Step 9: Set $u=u+1$. If $u\leq n$ go to Step 8.

Step 10: Set $c=c+1$ and $u=c+1$. If $c<n$ go to Step 8.

Step 11: Set $p=p+1$. If $p\leq$NRE, then set $c=1$, $u=2$ and go to Step 8.

Step 12: The heuristic solution is π_d.

5. Computational Analysis

The experiments were performed on a PC running Windows 7 32-bits, with an Intel Dual Core CPU 2.26 GHz and 3GB RAM. The data generation and testing application were developed using the C# programming language which runs on the top of Microsoft's .NET Framework 3.5. The Experimentation parameters are as follows:

- n: 30, 40, …, 100.
- m: 5, 8, 11, 14, 17, 20.
- E: 0.2, 0.4, 0.6.
- R: 0.2, 0.6, 1.0 .

The above mentioned E and R combinations are recommended by Vallada et al. [18] based on the literature review. Also, as often used in the literature, the processing times of jobs P_{ij} are generated with a uniform distribution between 1 and 99 and due dates of jobs are generated using due date parameters E and R with a uniform distribution between $B(1-E-R/2)$ and $B(1-E+R/2)$, where B is a tight lower bound of the makespan.

$$B = \max_{1\leq j\leq m} \left\{ \sum_{i=1}^{n} p_{ij} + \min_{1\leq i\leq n} \sum_{k=1}^{j-1} p_{ik} + \min_{1\leq i\leq n} \sum_{k=j+1}^{m} p_{ik} \right\}$$

Where, $\sum_{k=1}^{0} p_{ik} = 0 \quad and \quad \sum_{k=m+1}^{m} p_{ik} = 0$

Based on the n, m, E, and R values, the total number of combinations is 432; where each combination is replicated 30 times.

The performance measure used for evaluating the heuristics is the relative deviation index (*RDI*); calculated as follows:

$RDI = \frac{Heuristic-Best}{Worst-Best} * 100$, where *Heuristic* is the solution obtained by a given heuristic, *Best* is the best solution obtained from among the compared heuristics, and *Worst* is the worst solution obtained from among the compared heuristics. The *RDI* will produce a result between 0 and 100; where good solutions will have an *RDI* closer to 0.

Using the *RDI* evaluation criteria, we compare the regular simulated annealing (*SA*) and genetic algorithm (*GA*) where random initial solutions are used; the proposed improved simulated annealing (*ISA*), improved genetic algorithm (*IGA*); and the further enhanced simulated annealing (*FISA*) and genetic algorithm (*FIGA*).

Figure 3 shows the results of the *RDI* comparison for different number of jobs, while Figure 4 shows the results for different number of machines. On the other hand, Figure 5 shows the results for different *E*/*R* combinations. It should be noted that the heuristic performances in general did not change for different values of m, n, and *E*/*R* combinations. The only noted exception is the slight improvement in ISA and IGA performances for high values of E and R.

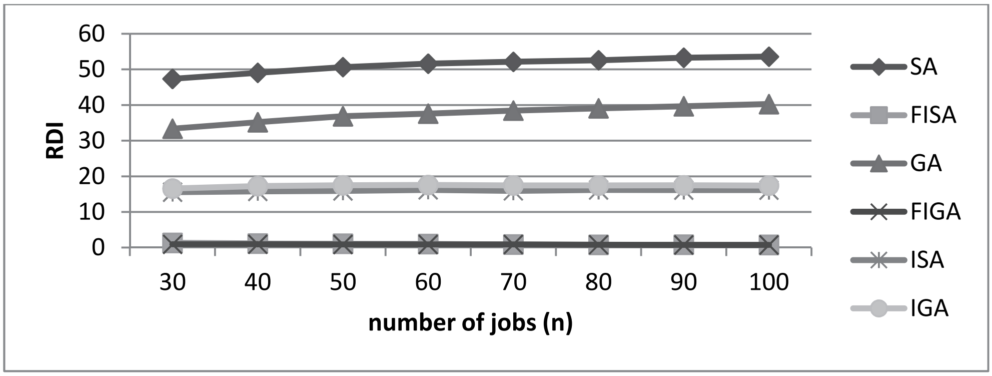

Figure 3. Comparison of metaheuristics for RDI criterion vs. n; for $\bar{E}$, $\bar{R}$ and $\bar{m}$

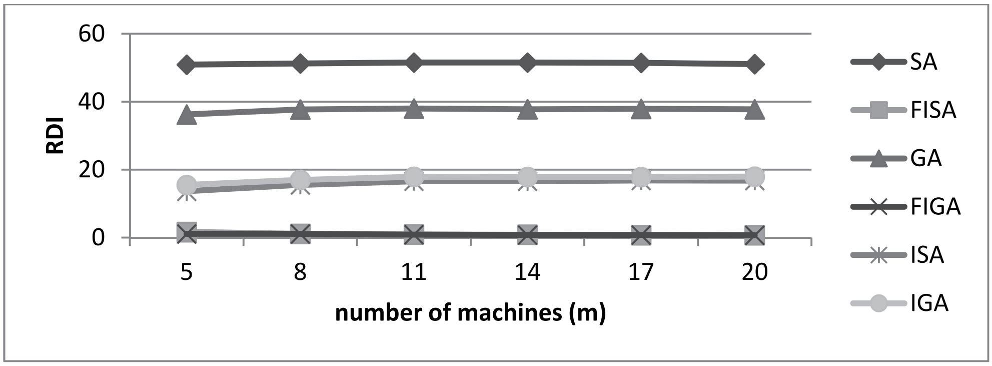

Figure 4. Comparison of metaheuristics for RDI criterion vs. m; for $\bar{E}$, $\bar{R}$ and $\bar{n}$

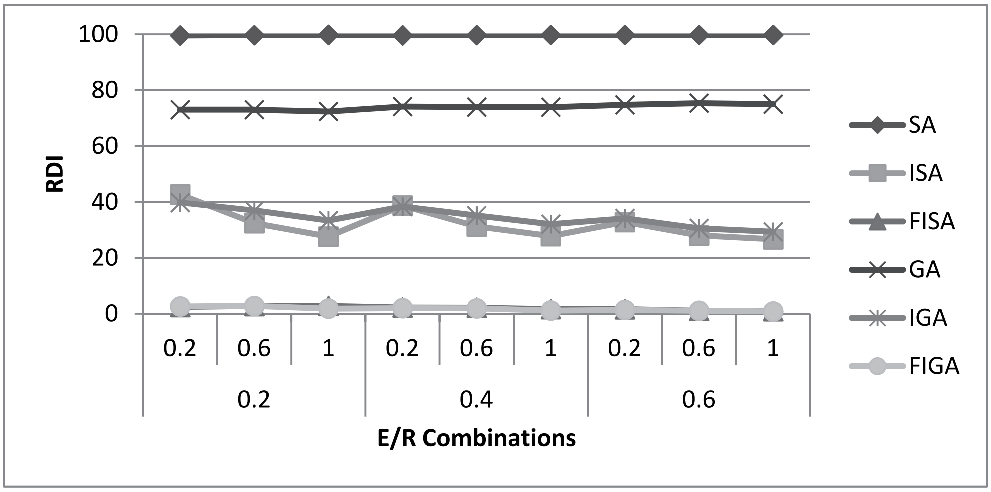

Figure 5. Comparison of metaheuristics for RDI criterion for different E/R combinations

Figure 3 shows that *GA* outperforms *SA* by about 33%; both of their performances deteriorate slightly with the increase in the number of jobs. *ISA* outperforms *SA* by about 70%; while *IGA* outperforms *GA* by about 55%. Both of their performances do not seem to change with the increase in number of jobs. It should be noted that *ISA* slightly outperforms *IGA*. This is expected as *ISA* takes the initial seed as *MDD* only, while *IGA* takes additional random seeds for its initial population. The two other proposed methods *FISA* and *FIGA* have very close *RDI* performance that is less than 1.0 with a slight advantage of *FIGA* that amounts to about 10%. Both *FISA* and *FIGA* outperforms *ISA* and *IGA*, respectively, by about 94%. Another observation is that the performances of *FISA* and *FIGA* slightly improve with the increase in the number of jobs. Figure 4, which shows the performance with respect to *m*, the relative performance of the considered heuristics is similar to that of Figure 3.

The computational time for all heuristics was reasonable. For the largest number of jobs case (n = 100), the computational time for *FIGA* is about 90 seconds; which is 10 times more than the computational time of *FISA*. Since both *FISA* and *FIGA* performance is less than 1.0 *RDI*, and *FIGA* has 10 times more computational time, we recommend FISA as the superior heuristic.

6. Conclusion

The m-machine no-wait flowshop scheduling problem with the objective of minimizing total tardiness has been considered. Several dispatching rules and enhanced simulated annealing and genetic algorithms have been proposed. Computational analysis showed the dispatching rule *MDD* outperforms all other dispatching rules. Using the *MDD* as initial sequence in *GA* and *SA* has improved their performances more than 50%. Moreover, further enhancements on *GA* and *SA* resulted in more than 90% further improvement. The performance of the enhanced version of genetic algorithm *FIGA* was slightly better than that of the enhanced version of simulated annealing *FISA*; however, the computation time of *FIGA* was about 10 times that of *FISA*. Therefore, we conclude that *FISA* the superior heuristic.

In this paper, setup times are ignored or assumed to be included in the processing times. This assumption is valid for some scheduling environments. However, the assumption may not be valid for some other scheduling environments. The importance of setup times, treated as separate from processing times, has been addressed by many researchers and the work of those researchers has been reviewed by Allahverdi et al. [6, 7]. Therefore, a possible extension is to consider the problem addressed in this paper where setup times are treated as separate from processing times.

Acknowledgment

This research is supported by Kuwait University Research Administration project number EI02/09.

References

[1] Aldowaisan, T., and Allahverdi, A. No-wait flowshop scheduling problem to minimize the number of tardy Jobs. *International Journal of Advanced Manufacturing Technology* 2012 (to appear).

[2] Aldowaisan T, Allahverdi A. A new heuristic for m-machine no-wait flowshop to minimize total completion time. *OMEGA International Journal of Management Sciences* 2004; 32: 345-352.

[3] Aldowaisan T, Allahverdi A. New heuristics for no-wait flowshops to minimize makespan. *Computers & Operations Research* 2003; 30: 1219-1231.

[4] Allahverdi A, Aldowaisan T. No-wait flowshops with bicriteria of makespan and maximum lateness. *European Journal of Operational Research* 2004; 152: 132-147.

[5] Allahverdi A, Aldowaisan T. No-wait flowshops with bicriteria of makespan and total completion time. *Journal of the Operational Research Society* 2002; 53: 1004-1015.

[6] Allahverdi, A., Gupta, J.N.D., and Aldowaisan, T. A review of scheduling research involving setup considerations. *OMEGA International Journal of Management Sciences* 1999; 27: 219-239.

[7] Allahverdi, A., Ng, C.T., Cheng, T.C.E., and Kovalyov, M.Y. A survey of scheduling problems with setup times or costs. *European Journal of Operational Researc*h 2008; 187: 985-1032.

[8] Chang JL, Gong DW, Ma XP. A heuristic genetic algorithm for no-wait flowshop scheduling problem. *Journal of China University of Mining and Technology* 2007; 17: 582-586.

[9] Chen CL, Neppalli RV, Aljaber N. Genetic Algorithms Applied to the Continuous Flow Shop Problem. *Computers & Industrial Engineering* 1996; 30: 919-929.

[10] Framinan JM, Nagano MS. Evaluating the performance for makespan minimization in no-wait flowshop sequencing. Journal *of Materials Processing Technology* 2008; 197: 1-9.

[11] Framinan, J.M., Leisten, R. Total tardiness minimization in permutation flow shops: a simple approach based on a variable greedy algorithm. International *Journal of Production Research* 2008; 46: 6479-6498.

[12] Hall NG, Sriskandarajah C. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research* 1996; 44: 510-525.

[13] Lenstra JK, Rinnooy Kan, AHG., Brucker P. Complexity of machine scheduling problems. *Annals of Discrete Mathematics* 1977; 1: 343-362.

[14] Nawaz M, Enscore E, Ham I. A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *OMEGA International Journal of Management Science* 1983; 11: 91-95.

[15] Pan QK, Tasgetiren MF, Liang YC. A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research* 2008; 35: 2807-2839.

[16] Pinedo M (1995) *Scheduling; theory, algorithms, and systems*. Prentice Hall, Englewood Cliffs, New Jersey.

[17] Vallada, E., Ruiz, R. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *OMEGA International Journal of Management Science* 2010; 38: 57-67.

[18] Vallada, E., Ruiz, R., Minella, G. Minimizing total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers and Operations Research* 2008; 35: 1350-1373.

[19] Warburton RDH. EOQ extensions exploiting the Lambert W function. *European Journal of Industrial Engineering* 2009; 3: 45-69.

Received January, 2012