ISSN No:-2456-2165

Forward Shift Heuristic for Minimization of Number of Early-Tardy Jobs in Just-In-Time Scheduling in a Flow Shop with Distinct Due Windows

Idowu, A. Gbolahan [1*], Adamu, O. Muminu [2], Mustapha, A. Rilwan[1], Sawyerr, A. Babatunde[3], Rahman, O. Idris[1]
[1.] Department of Mathematics, Lagos State University, Ojo, Lagos.
[2.] Department of Mathematics, University of Lagos, Akoka, Yaba, Lagos.
[3.] Department of Computer Science, University of Lagos, Akoka, Yaba, Lagos.

Abstract:- This paper addresses the problem of scheduling production to meet Just-in-Time requirement in a Flow Shop where jobs are to be completed within given due windows rather than single due dates. This problem has been proven to be NP-Complete in the strong sense and shown to be intractable as the number of jobs-and-machines combination increase. Consequently, a Forward Shift Search (FSS) algorithm is developed to solve large problem instances. To assess the performance of FSS, a benchmark of 1200 test problem were solved and compared with solution obtained by an exiting algorithm in the literature. FSS performs better in terms of quality of solution and computational time. Also, FSS obtained a relative deviation index of 4% when compared with a optimal solution obtained by a commercial solver within 600 seconds of computational time. Therefore, FSS can be deployed to obtain good approximation schedule for large scale production where desired error tolerance is not more than 0.004.

Keywords:- *Early/Tardy, Forward-Shift,Flow Shop, Heuristics, Just-In-Time.*

I. INTRODUCTION

Just-In-Time (JIT) scheduling is very important in large scale manufacturing where storage space in limited. Fabrication of large equipment, aircraft and production machinery does not encourage complete and store until demands by customers, it is required that products are made available as at when needed by clients. Therefore, Minimization of Number of Early/Tardy (NET) jobs are gaining attention recently.

Also, production strategies in large scale manufacturing sectors have shifted from massive production for a single client to JIT for many clients spread all around the world. To remain competitive, producers find themselves planning their daily productions according to customers' orders. Their objective is to satisfy target delivery date(s) of individual customer notwithstanding the size of order or distance of delivery destinations. Meeting delivery deadlines of customers translates to maximizing the number of on-time jobs among the orders, thereby minimizing the NET jobs.

At the commencement of research in JIT, the concept takes into consideration Earliness/Tardiness (ET) penalties as the objective to minimize. In JIT scheduling environment, jobs that are completed early must be kept in inventory until their due date(s), while jobs that are completed after their due date(s) may be rejected by customers or erodes customer's confidence. Therefore, an ideal schedule is one in which all jobs are completed exactly on their assigned due-date(s). This concept of penalizing earliness and tardiness simultaneously was introduced in Ohno (1988). JIT scheduling philosophy has motivated an on-going and rapidly developing trend in the field of scheduling. Since the use of combined ET penalties gave rise to a non-regular criterion, this has resulted in persistent methodological issues in the design of solution procedures. The cost function adds the penalties due to earliness or tardiness, and the resulting problem is usually referred to as the Multi-objective Earliness-Tardiness Scheduling Problem (METSP). Tardiness refers to the length an order is belated while earliness is the length in which an order remains in inventory before customer's pickup or delivery. However, Lann and Mosheiov (1996, 2003) introduced a different type of cost function that is related to the NET jobs rather than the ET values.

This research considered JIT problems of scheduling jobs on Permutation Flow Shop Scheduling Problem (JIT-PFSSP) with two distinct due dates (due windows) which are an emerging area of JIT research due to its application in the production of trendy, perishable and industrial products where commodity's shelf life is crucial. Cheng and Wang (2000) and Adamu, Budlender, and Idowu (2014) before now provided insights into the problem with due dates. The recent results on JIT scheduling models with due dates can be found, for instance, in Shabtay and Bensoussan (2012a), Shabtay and Bensoussan (2012b), Rasti-Barzoki and Hajazi (2013).

Prevalent JIT scheduling models require that jobs are completed on due dates as described in Rasti-Barzoki and Hajazi (2013) and Prot, Bellenguez-Morineau, and Lahlou (2002). However, in manufacturing industry, it is often expected that jobs are finished at certain intervals (due windows) rather than at single points in time (due dates). Scheduling problems with distinct due windows extends and generalizes that of the classical due dates. Numerous researches have dealt with JIT scheduling problems with due windows on single and parallel machines, providing various polynomial time algorithms for the case where machines are identical while that of problem on a set of unrelated machines in parallel is strongly NP-hard. Some of these works can be found in Arkin and Silverberg (1987), Bouzina and Emmons (1996), Hiraishi, Levner, and Vlach (2002), Cepek and Sung

(2002), Sung and Vlach (2005), Adamu and Abass (2010), Janiak, Janiak, and Januszkiewicz (2009) and Adamu et al. (2014).

There is a dearth of literature on JIT-PFSSP notwithstanding its importance in assembly and manufacturing problems. Yeung, Oğuz, and Cheng (2009) and Mosheiov and Sarig (2009) focused on models with window assignment and jobs-dependence where complexity proofs and solution algorithms were established. However, the problems with distinct/multiple due windows which have significant impact on future trend are yet to be widely explored, hence the motivation for this work.

Many studies have been conducted theoretically and experimentally on JIT scheduling by production engineers, operation managers, applied mathematicians and researchers. Study on minimization of NET jobs in JIT on a flow shop with distinct due windows has not been reported, as recent researches in this area stops at single due dates (to the best of the author's knowledge). Questions surrounding minimizing NET jobs on a flow shop with distinct due windows required answer in recent years. These questions among others include providing polynomial time algorithms. Most of the reviews on minimizing NET jobs in JIT scheduling have been limited to two-machine flow shop, single and parallel machines. However, some recent proposals to provide solution to minimizing NET jobs in JIT on flow shop using the advantage of forward shift heuristic have not been explored; among other areas include algorithms with exponential computational time for large problems; algorithm implementation and large deviation from optimal.

In respect of the foregoing, a forward shift search algorithm for minimizing NET jobs is proposed in this work with benefit of being less expensive with respect to computational time; effective in terms of deviation from optimal and ease of implementation which will improve the results obtained in Shabtay (2012c) and Adamu et al. (2014).

II. PROBLEM DESCRIPTION

For the problem of JIT-PFSSP of n jobs $(J_1, J_2, \dots, J_n)$ on m (mainly ordered) machines $(M_1, M_2, \dots, M_m)$ to minimize NET jobs objective, it is assumed that a machine can process at most one job at a time and that all jobs are ready for processing at time $t = 0$. Each job $j, j = 1, \dots, n$ is characterized by a set of m processing times $p_{ij}, i = 1, \dots, , m$ (that is, there are different processing times on different machines for a given job). For any given non - preemptive feasible schedule, π would be a sequence of permutation of n jobs $(\pi = \pi_{(1)}, \pi_{(2)}, \dots, \pi_{(n)})$ representing an order in which jobs are to be processed on all machines. Let C_{ij} be the completion time of job j on the ith machine, a_j be the beginning of the due window (earliest due date), d_j the end of the due window (latest due date) of job j and $D_j = d_j - a_j$ be the size of the due window. Job j is said to be early if $C_{ij} < a_j$, tardy if $C_{ij} > d_j$, and on time if $a_j \leq C_{ij} \leq d_j$. Let $\theta_j > 0$ and $\vartheta_j > 0$ be the weights/penalties for scheduling job j early and late respectively.

Adopting the classification of Graham, Lawler, Lenstra, and Rinnooy Kan (1979), the general JIT-PFSSP for minimizing NET jobs can be denoted as:

$$Fm|a_j, d_j|\sum_{j=1}^{n}(\theta_j U_j + \vartheta_j V_j) \quad (2.1)$$

where U_j and V_j are defined as follows:

$$U_j = \begin{cases} 1, & \text{if } C_{ij} < a_j \\ 0, & \text{otherwise.} \end{cases} \quad (2.2)$$

$$V_j = \begin{cases} 1, & \text{if } C_{ij} > d_j \\ 0, & \text{otherwise.} \end{cases} \quad (2.3)$$

Considering a special case where unit penalties are incurred on both early and tardy jobs $(\theta_j = \vartheta_j = 1)$. This special case is also referred to as unweighted represented as $Fm|a_j, d_j|\sum_{j=1}^{n}(U_j + V_j)$ for minimizing NET jobs where the objective function is: $Z = \sum_{j=1}^{n}(U_j + V_j)$.

III. MATERIAL AND METHODS

A. Proposed Algorithms for JIT-PFSSP

For minimizing the number of early-tardy jobs in JIT-PFSSP, this study recommends an incre- mentally built iterative local search algorithm based on Forward Shift Search (FS) of the solution space. This method is a non-population base searching technique adopted from Pinedo (2015) that improves a selected candidate solution until a given stop criterion is reached. Un- like the population-based which chooses the best solution among selected several candidate solutions using a prescribed procedure. The selection of a candidate solution(s) can either be through a dispatching, probabilistic, systematic rule or the combination of these rules.

Section 3.2 describes the representation of a candidate solution for minimizing NET jobs in JIT-PFSSP. How the initial candidate solution is chosen is presented in Section 3.3 while in Sections 3.5, the details of FSS algorithms is presented.

B. Solution Representation

Generally, a solution for JIT-PFSSP with n jobs is represented by a sequence π_i of length n. Each index $i = 1,2, \dots, n$ depicts the job to be executed at position i of π. For instance, in the sequence $\pi = (3,5,6,7,1,4,2)$ of 7 jobs, job 3 is the first to be executed and job 2 is the last to be executed on each of the machines.

C. Initial Solution

In non-population based heuristics, selection technique of the initial candidate solution is very important because it can influence the effectiveness of an algorithm to reach the best solution within a given stopping criterion, (see M'Hallah (2016)).

The initial solution of the permutation π for JIT-PFSSP is constructed by applying the Earliest Due Date dispatching (EDD) or probabilistic (RND) rule. EDD and RND rules are explained below:

EDD - This is a greedy constructive dispatching rule, often applied to scheduling problems with distinct due dates, as employed in Adamu et al. (2014); M'Hallah (2016); Rosa et al. (2017). The proposed construction begins with an empty sequence. The job with the least EDD is inserted first, followed by the next job with the least EDD and so on, with breaking ties broken arbitrarily. The construction procedure stopped when no more jobs lie outside of the execution sequence.

RND - The sequence with the least value of the objective function $\phi(\pi)$ out of twenty randomly generated permutations of the n jobs. This rule can only be applied to problems where $n \geq 4$ jobs. It is important to note that the least number of jobs considered in this work is 10.

D. Evaluation of a Candidate Solution

For any feasible job sequence (π), a solution to JIT-PFSSP will be obtained by means of computation of Boolean matrix $B^T = \{y_j : c_{m,\pi_j} < a_j \text{ or } c_{m,\pi_j} > d_j\}$ and Integral matrix $D^T = \{x_j : \max\left(0, c_{m,\pi_j} - d_j\right) + \max\left(0, a_j - c_{m,\pi_j}\right)\}$.

For matrices B^T and D^T, the completion times matrix (c_{i,π_j}) of the $j-th$ job in the sequence π on the ith machine can be computed as follows:

$$c_{i,\pi_j}(\pi) = c_{i,\pi_{j-1}}(\pi) + p_{i,\pi_j} \; i = 1, \dots, m, \; j = 1, \dots, n$$

Where

$$c_{i,\pi_0}(\pi) = 0 \; \forall \; i = 1, \dots, m$$

The feasible solution to NET (ϕ_1) objective is the independent sum of the elements of matrix B^T. That is, $\phi_1 = \sum_{j=1}^{n} B^T$. For instance, $B^T = \{1,0,0,0,1,0,1\}$ for the sequence $\pi = (3,5,6,7,1,4,2)$ of 7 jobs; jobs 3, 1 and 2 finished early-tardy then $\phi_1(\pi) = \sum_{j=1}^{7} B^T = 3$. Also, solution to ET (ϕ_2) objective is the sum of the values in matrix D^T and $\phi_2 = \sum_{j=1}^{n} D^T$.

E. Proposed Local Search Algorithm to Minimize NET Jobs

In this Section, a near optimal algorithm based on Forward Shift Search (FSS) is proposed to solve JIT-PFSSP with minimization of NET jobs objective. This algorithm is motivated by the assertion in Janiak, Janiak, Krysiak, and Kwiatkowski (2015)) that there is no specific exact optimization algorithm for solving JIT-PFSSP. The proposed algorithm (FSS) for solving the problem of minimizing NET in JIT-PFSP employed the breadth-first search technique of the branch and bound algorithm proposed in Pinedo (2015). However, at each breadth/level, iterative permutation of jobs was evaluated for its contribution to the objective value rather than phantom of decision variables. Then a sequence with the lowest objective value (tie is broken arbitrarily) moves to the next level for incremental build and further iteration.

In formulation of algorithm FSS shown in **Algorithm 1**, let X be the set of n jobs to be scheduled. Let UB be a known upper bound on the problem, which can be obtained by any of the two initial solution method in Section 3.3 with an associated sequence A. During the enumeration process, sequences S_i^k $(k = 1, \dots i)$ correspond to a structure that stores pairwise shift permutation of a subsequence of i jobs $(2 \leq i \leq n)$, S be Scheduled subsequence of A and U be a set of $n - i$ jobs that are outside of the subsequence. The sequence $\left(S_i^k\right)$ is evaluated by the function ϕ_1, as in section 3.4 at level of k enumeration. The level of enumeration is represented as the subscript i while the superscript k shows the position of the last job in S_i^k after each forward-shift (that is, pair wise interchange of the kth and $k-1$th jobs) of the last job. For instance, the sequence $S_3 = \{2,3,1\}$ of three jobs with job 1 being the last job, $S_3^1 = \{1,2,3\}$ indicates that job 1 is now shifted to the first position while in sequence $S_3^2 = \{2,1,3\}$ job 1 was shifted to position 2. The FSS algorithm is described as follows:

Algorithm 1: Details of Forward Shift Search - FSS(X)
Input: X
Output: ϕ_1
1 $A_n \leftarrow InitialSolution(X)$;
2 $UB \leftarrow \phi_1(A)$;
3 **if** $UB > 0$ **then**
4 Goto Step 9;
5 **else**
6 $\phi_1 \leftarrow 0$;
7 Step 23;
8 **end**
9 $S \leftarrow \cup\{A_1\} \cup \{A_2\}$;
10 $U \leftarrow A_n \setminus S$;
11 $i = 2$;
12 **while** $(U \neq \emptyset)$ **do**
13 $S^k \leftarrow S_i$;
14 **for** $k = i$ *to* 1 **do**
15 $S^k \leftarrow Swap(k, k-1)$;
16 **end**
17 $S_i = \min\{\phi_1(S^{\star,k})\}$;
18 $i \leftarrow i + 1$;
19 $S_i \cup \{A_i\}$;
20 $U \leftarrow A_n \setminus \{A_i\}$;
21 $\phi_1 \leftarrow S_i$;
22 **end**
23 **return** ϕ_1;
24 End;

Algorithm 1 illustrates the FSS procedure applied to minimize number of early-tardy jobs in JIT-PFSSP. In this procedure, the initial UB corresponds to a known upper bound for the problem. This value is updated using the FSS procedure, so that the final value of UB is its best solution.

➢ *Numerical Example of FSS Algorithm*

This section demonstrated with a numerical example, how FSS performs when applied to JIT- PFSSP to minimize NET jobs. A numerical problem solved in Adamu et al. (2014) was considered, re-evaluated using FSS and solution obtained compared with that obtained in the literature.

Considering a scheduling problem of seven jobs to be processed on three machines in series represented in Table 1, this table shows the processing times (p_{ij}), early due date (a_j) and late due date (d_j) of each job ($j = \{1,2,\dots,7\}$) to be processed on through all machines ($i = \{1,2,3\}$).

The algorithm F1 in Adamu et al. (2014) obtained an objective value of NET = 2 when applied to JIT-PFSSP to minimize NET jobs.

Table 1: Test problem of 7 jobs on 3 machines in

j	p_{ij} m			a_j	d_j
	1	2	3		
1	2	1	2	0	6
2	3	2	1	2	9
3	2	2	2	4	10
4	3	2	2	8	16
5	1	1	1	9	15
6	4	2	2	11	19
7	3	3	1	13	19

Let A be a sequence $[A = \{1,2,3,5,4,6,7\}]$ obtained as an initial solution when EDD rule on d_j is applied to the above problem. Given that all jobs are to start processing as soon as possible, as depicted in Figure 3.2(a), thus $UB = 3$ and $U = \{1,2,3,5,4,6,7\}$.

The second step evaluated the initial solution for optimality, and if not optimal, it will proceed to the next step where non optimal solution will be improved. In this example, the initial solution is not optimal, as the first two jobs in the sequence A will be assigned to the subsequence S_2 (i.e $S_2 = \{1,2,\}$), then removed from U (i.e $U = \{3,5,4,6,7\}$) and set the enumeration level k to $k = 2$.

The improvement process began with permutation of the sequence S_2 by changing the kth position of the last job in sequence S_2. This gave sequences $S_2^2 = \{1,2\}$ and $S_2^1 = \{2,1\}$ evaluated as $\phi_1(S_2^2) = 0$ and $\phi_1(S_2^1) = 1$, respectively.

The next iteration level is set to $k = 3$ with an associated sequence $S_3 = \{1,2,3\}$ because $\{3\}$ is the next job in sequence A after jobs $\{1\}$ and $\{2\}$ were assigned to S, and job $\{3\}$ is also removed from U. This process was repeated until the iteration level k attained $k = 7$ and sequence U is completely emptied. The detail of the numerical example of FSS procedure is as follows:

- Set $k = 3$
  $S_3 = \{1,2,3\}$ $U = \{5,4,6,7\}$
  $S_3^3 = \{1,2,3\};$ $S_3^2 = \{1,3,2\};$ $S_3^1 = \{3,1,2\};$
  $\min\{\phi_1(S_3^3), \phi_1(S_3^2), \phi_1(S_3^1)\} = \phi_1(S_3^2) = 0$

- Therefore, S_3^2 is selected for the next iteration.

- Set $k = 4$
  $S_4 = \{1,3,2,5\}$ $U = \{4,6,7\}$
  $S_4^4 = \{1,3,2,5\}$ $S_4^3 = \{1,3,5,2\};$ $S_4^2 = \{1,5,3,2\};$ $S_3^1 = \{5,1,3,2\};$
  $\min\{\phi_1(S_4^4), \phi_1(S_4^3), \phi_1(S_4^2), \phi_1(S_4^1)\} = \phi_1(S_4^3) = 1$

  Therefore, S_4^3 is advanced to the next iteration.

- Set $k = 5$
  $S_5 = \{1,3,5,2,4\}$ $U = \{6,7\}$
  $S_5^5 = \{1,3,5,2,4\}$ $S_5^4 = \{1,3,5,4,2\}$ $S_5^3 = \{1,3,4,5,2\};$
  $S_5^2 = \{1,4,3,5,2\};$ $S_3^1 = \{4,1,3,5,2\};$
  $\min\{\phi_1(S_5^5), \phi_1(S_5^4), \phi_1(S_5^3), \phi_1(S_5^2), \phi_1(S_5^1)\} = \phi_1(S_5^4) = 1$

  Therefore, S_5^4 is selected for the next iteration.

- Set $k = 6$
  $S_6 = \{1,3,5,4,2,6\}$ $U = \{7\}$
  $S_6^6 = \{1,3,5,4,2,6\}$ $S_6^5 = \{1,3,5,4,6,2\} S_6^4 = \{1,3,5,6,4,2\}$
  $S_6^3 = \{1,3,6,5,4,2\}$ $S_6^2 = \{1,6,3,5,4,2\}$ $S_6^1 = \{6,1,3,5,4,2\}$
  $\min\{\phi_1(S_6^6), \phi_1(S_6^5), \phi_1(S_6^4), \phi_1(S_6^3), \phi_1(S_6^2), \phi_1(S_6^1)\} = \phi_1(S_6^5) = 1$

  Therefore, S_6^5 is selected for the next iteration.

- Set $k = 7$
  $S_7 = \{1,3,5,4,6,2,7\}$ $U = \{\emptyset\}$
  $S_7^7 = \{1,3,5,4,6,2,7\}$ $S_7^6 = \{1,3,5,4,6,7,2\}$ $S_7^5 = \{1,3,5,4,7,6,2,\} S_7^4 = \{1,3,5,7,4,6,2\}$ $S_7^3 = \{1,3,7,5,4,6,2\}$
  $S_7^2 = \{1,7,3,5,4,6,2\}$ $S_7^1 = \{7,1,3,5,4,6,2\}$
  $\min\{\phi_1(S_7^7), \phi_1(S_7^6), \phi_1(S_7^5), \phi_1(S_7^4), \phi_1(S_7^3), \phi_1(S_7^2), \phi_1(S_7^1)\} = \phi_1(S_7^5) = 1$

Since $U = \{\emptyset\}$, then S_7^5 is taken as the final sequence which returned $UB = 1$ as the value of the objective function such that the best solution obtained with FSS is $\phi_1(\pi) = 1$.

The Gantt chart shown in Figure 3.2(b), depicts the objective solution obtained by FSS after improving the initial value derived with EDD. It is observed that algorithms presented in obtained the value of the objective function $\phi_1(\pi) = 2$; therefore; algorithm FSS showed an improved result.

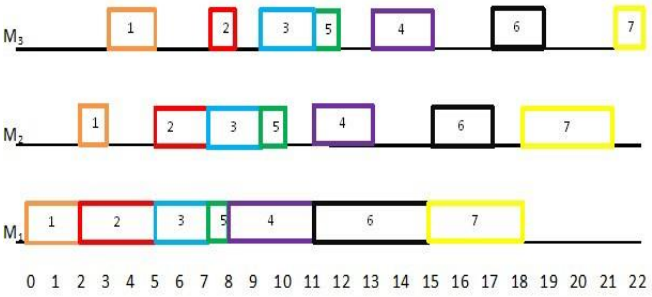
The above numerical example demonstrated that the algorithm FSS detailed in Section (3.5) can be easily followed at each computation stage. The maximum number of partial solutions evaluated were $\frac{n(n-1)-2}{2}$ and such that n candidate solutions are evaluated at the $k = n$ levels. The worst case computational complexity of algorithm FSS is of order $O(n^2)$.

IV. IMPLEMENTATION AND COMPUTATIONAL EXPERIMENTATION

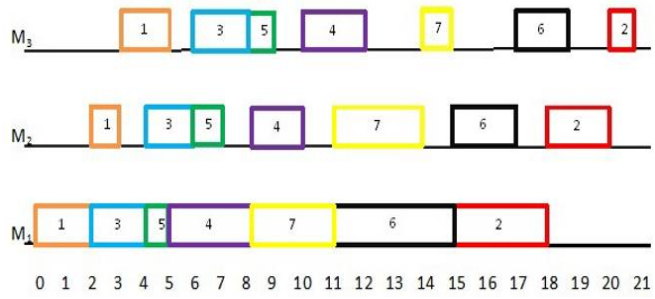
The algorithms FSS is presented in Sections 3.5, Shabtay (2012c) and Adamu et al. (2014) were implemented using java programing language with Netbeans 8.2© as Integrated Development Environment (IDE). Also, a mathematical model (MBILP-1) in Idowu, Adamu, Sawyerr, Mustapha, and Rahman (2022) was implemented using $XPRESS - MP$ ©, a commercial solver with $IV\ E8.1$© interface provided by $FICO$© optimization. Therefore, the solutions obtained when (MBILP-1) is applied to any instance is referred to as FICO in the rest of this work.

These simulations were performed on the same computer with *Intel*© *Core*™ 2 Duo CPU P8800 running at 2.66Ghz of processor speed, 4GB of RAM, and the *Windows*® 10pro 64-bit operating system. This low grade computer was adopted to observe the worst-case experimental characteristics of these methods.

- *Gantt chart of sequence obtained using EDD*



- *Gantt chart obtained 7after initial sequence is improved by FSS*



The test-data used to assess FSS, Adamu et al. (2014), Shabtay (2012c) and MBILP-1 are described in Section 4.1. Section 6.1 explains how the initialization method employed affects the efficiency of FSS. In Section 4.2, statistical test employed to compare implemented algorithms are presented.

A. Instance/Test-Data Generation

In order to evaluate the algorithm FSS, test-data were generated using the methodology of Idowu et al. (2022) Bulfin and M'Hallah (2003) and Adamu et al. (2014). This methodology is shown in Table 2 and described below.

For each job j, the processing time $p_{i,j}$ on machine i is randomly generated integers from the uniform distribution within the interval [1,99]. The time windows of job j, that is, the earliest due date a_j and latest due dates d_j are also random integer numbers from the same distribution within the interval $\left[\left(1-\theta-\frac{\vartheta}{2}\right)\sum_i p_{i,j}, \left(1-\theta+\frac{\vartheta}{2}\right)\sum_i p_{i,j}\right]$, where: $\sum_i p_{i,j}$ is the total processing time of each job j on machine i; θ is the tardiness factor and ϑ is the relative range of the time windows. Sets of instance are generated for small and large problems with their parameters as shown in Table 3 using $\theta = \{0.2, 0.4, 0.6, 0.8\}$ and $\vartheta = \{0.4, 0.6, 0.8, 1.0\}$, where $\theta < \vartheta$.

Table 2: Proposed instance/data generation scheme

Parameter	Distribution function	Indices (independent)
Processing time	$p_{i,j} \approx U[1,99]$	$i = 1,2,\dots m$
		$j = 1,2,\dots n$
Due dates	$a_j \approx U\left(1-\theta-\frac{\vartheta}{2}\right)\sum_i p_{i,j}$	$j = 1,2,\dots n$
	$d_j \approx U\left(1-\theta+\frac{\vartheta}{2}\right)\sum_i p_{i,j}$	$j = 1,2,\dots n$

For each problem combination 50 independent test-data were generated, thus, a total of 17,250 problem instances were solved.

Table 3: Parameters of test problems

Parameter		Problem type
Number of	Small	Large
Machines	3, 5, 7, 9, 10	3, 5, 7, 9, 10, 15, 20, 25, 30, 40, 50,
		60, 70, 80, 100
Jobs	10, 20, 30, 40, 50	90, 100, 200, 300, 500, 900, 1000,
		1500, 2000, 2500
Replications	50	50
Total replication in each	$45 \times 50 = 2250$	$300 \times 50 = 15000$
Total replications	2250 + 15000 = 17250	

B. Statistical Analysis

For each instance, the result obtained by the algorithms in this study is compared with existing algorithms in the literature. These comparisons are performed by using one-way hypothesis test for three independent samples with significant level of $\alpha = 0.05$. In order to determine an appropriate statistical test for the results obtained, a Kolmogorov-Smirnov (KS) test was performed to examine

the normality of the data set. The KS test of the results set is shown in Table 4. It was observed that the data set for FSS, SHAB6 and F1 are not normally distributed at 5% level of significance. Also a general observation of the result indicates that they are ordered prior to analysis. Consequently, a well-known non-parametric Jonckhere-Terpstra (JT) test is more appropriate procedure for this result.

For each set of three algorithms (FSS, SHAB6, F1), the following hypotheses are formulated to compare the means of their solutions/CPU run-times on a given instance:

Null Hypothesis (H_0): The mean of the solutions obtained using algorithms are equal. Alternative Hypothesis (H_1): The mean of the solutions obtained using algorithms are not equal.

Table 4: Kolmogorov-Smirnov (Normality) Test of Solution Quality and CPU Run-time

Algorithm	Statistic									
	Objective					Runtime				
	Mean	S.D.	Min.	Max.	Sig.	Mean	S.D.	Min.	Max.	Sig.
SHAB6	66.78	85.83	0.79	364.02	.000	32.93	60.92	0.04	370.68	.000
F1	61.87	79.58	0.72	340.18	.000	20.94	39.05	0.03	253.15	.000
FSS	46.68	59.94	0.55	252.38	.000	4.49	8.37	0.01	52.89	.000

Also, similar hypothesis are followed for the comparison of the average CPU times for the algorithms.

Consequently, the null hypothesis implies that there is no statistical significance that the mean of the solutions obtained by using algorithm FSS is lower than the mean of the solution obtained by using the other two algorithms. However, accepting the alternative hypothesis and rejecting the null hypothesis suggests that at the level of significance adopted, there is statistically significant proof that the solution obtained by using a particular method is better on the average than others.

V. RESULTS AND DISCUSSION

In this section, the results obtained from the computational experiment performed in this study are discussed. Section 6 discusses the performance of a heuristic FSS in Section 6.1, when compared with known heuristics in the literature (that is, F1 in Adamu et al. (2014) and Algorithm 6 in Shabtay (2012c)). Also, Section 6.3 discusses the effectiveness of applying FSS and FICO on generated instances of JIT-PFSSP that are not more than 10 machines and 100 jobs.

VI. PERFORMANCE OF FSS TO MINIMIZE NET JOBS

A. Effect of Initial Solution on FSS

This section studies the sensitivity of FSS to its starting solution. FSS is initiated with sequence π of the n jobs obtained by the EDD sequence and the least $\phi(\pi)$ sequence among 20 randomly generated ones (RND). For each of the 50 instances of small and large problems, the CPU time in seconds obtained by applying each of the two sequences to FSS as initial solution was recorded in Table 5. An appropriate Wilcoxon (W) tests showed that the mean CPU time of the EDD is on average less than that of the RND, and is statistically different at 5% significant level.

Also, Figure 2 displays the difference in average CPU runtimes of 10 to 2500 jobs across all machines. The result shows that there is no obvious difference when the number of job is less than 100 with significant differences thereafter. In order words, the figure shows clearly that RND performs poorly relatively to EDD when the problem size is large. This effect could be due to the time taken to generate 20 random permutation of candidate solutions and evaluation of each solution to obtain the result with the least $\phi(\pi)$. As the diagrams above clearly shows, EDD provides a better initial solution.

B. Comparison of Performance of FSS with Other Algorithms

This section examines the effectiveness and efficiency associated with the use of FSS for minimizing the NET jobs in JIT-PFSSP over two known algorithms in the literature. The first is a greedy choice algorithm proposed and implemented in Adamu et al. (2014) while the second is a constructive algorithm proposed by Shabtay (2012c).

Shabtay's Algorithm 6 (SHAB6) and Adamu (F1) are two-phased algorithms which has similar characteristic with FSS. They start with an initial solution (sequence) at the first phase then the solution is improved on at the second phase. In order to have a fair comparison of FSS with Shabtay6 and F1, the three algorithms were implemented using the same experimental environment.

In Sections 6.2.1 and 6.2.2, the quality of solutions and CPU run-time obtained by these algorithms are compared respectively. Furthermore, Section 6.3 discusses the Relative Deviation Index (RDI) of FSS from known optimal results.

C. Comparison of quality of Solution of FSS with SHAB6 and F1

Each problem of the 50 replications of all instances described in Table 3 is solved using FSS, SHAB6 and F1. Table 6, depicts the average quality of solutions returned by each algorithm when applied to all set of instances. The first column shows number of machine, the second column is the number of jobs while the rest of the columns represent the

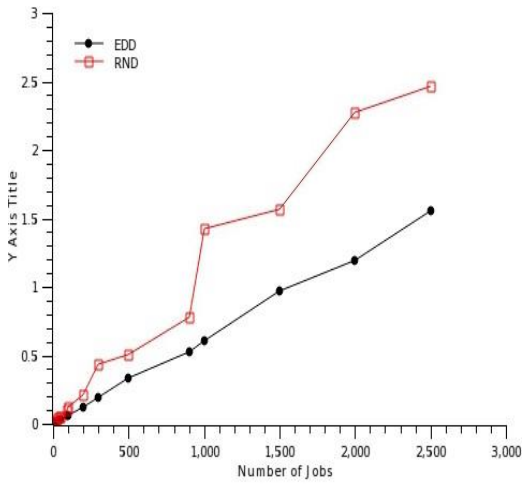
results obtained by SHAB6, F1 and FSS, respectively.

A JT statistical analysis confirms that for a fixed level of *m* and *n* the three algorithms are statistically significant at 5% level which explains why the null hypothesis is rejected as shown in Table **4**. Also, Figure 3 shows that there is evidence to assert that the mean solution of FSS is less than that of SHAB6 and F1 in many of the instances while F1 is, in turn, less than that of SHAB6.

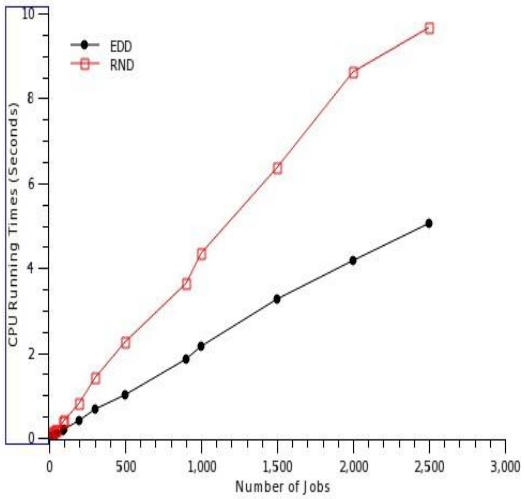
D. Comparison of Computational Times of FSS with SHAB6 and F1

This section discusses the comparison of CPU running time in seconds of FSS with SHAB6 and F1 algorithms when applied to solve test beds of JIT-PFSSP to minimize the number of early- tardy jobs. Table 7, shows the mean running time obtained when 50 instances of machines- Jobs combination were solved with the three named algorithms. The first and second column represent the number of machines and jobs respectively while other columns depict running time of SHAB6 followed by F1 and FSS.

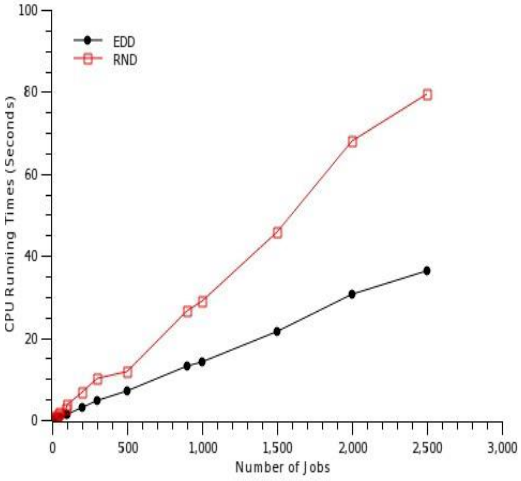
Figure 4 illustrates the mean runtime of SHAB6, F1 and FSS at $m = 3, 10, 80, 100$ as a function of n. It shows that FSS algorithm is efficient than both F1 and SHAB6 in all of the stacked diagrams. The mean runtime of both SHAB6 and F1 infers a rapid growth than FSS as the number of job increases. A JT analysis shows that the runtimes are significant at $\alpha = 0.05\%$.



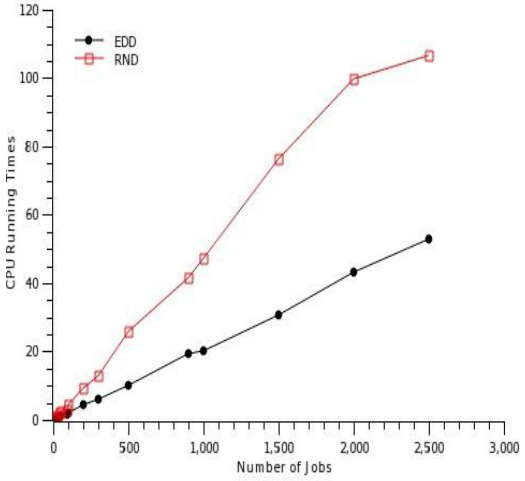
(a) CPU Running Times in Seconds when m=3



(b) CPU Running Times in Seconds when m=10

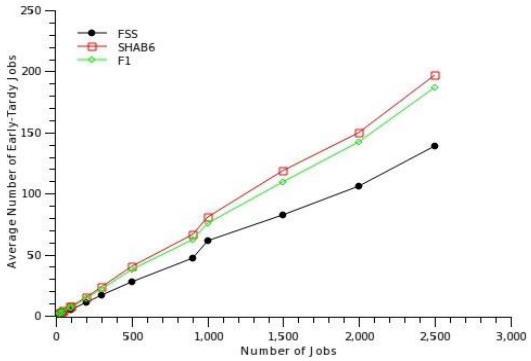


(c) CPU Running Times in Seconds when m=70

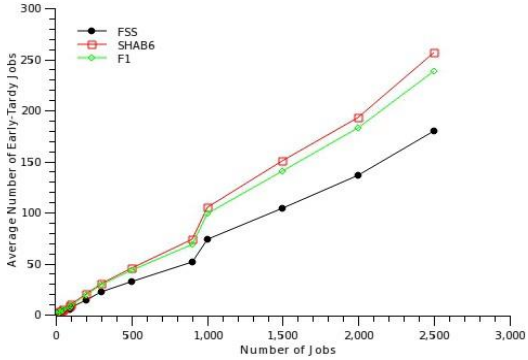


(d) CPU Running Times in Seconds when m=100

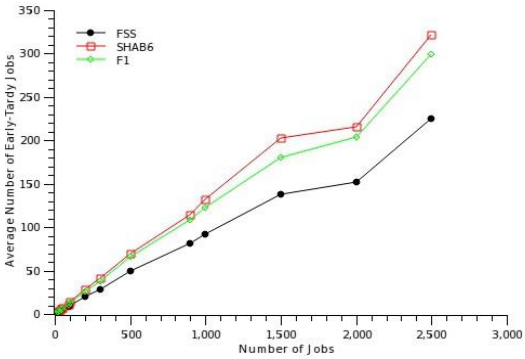
Fig 2: Effect of initial solutions on FSS



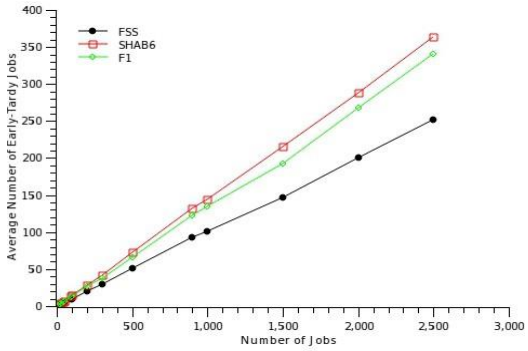
(a) Average NET of Jobs m=3



(b) Average NET of Jobs m=10



(c) Average NET of Jobs m=70



(d) Average NET of Jobs m=100

Fig 3: Comparison of Average NET values FSS with SHAB6 and F1 Algorithms

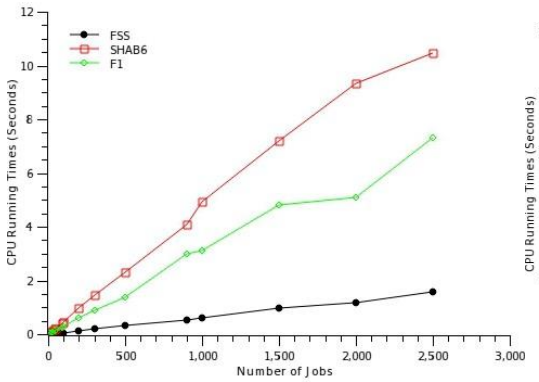
E. Analysis of Relative Deviation Index

This section studies the quality of solution obtained by an heuristic with respect to known op- tima solutions. To determine if an algorithm provides good solutions, analysis of Relative De- viation Index (RDI) was adopted as a standard performance indicator for measuring quality of approximate solutions in scheduling problems. Also, RDI was used extensively in scheduling problems where due dates are involved. (see for example; Adamu and Idowu (2017), Rosa et al. (2017), Framinan, Perez-Gonzalez, and Fernandez-Viagas (2019) and Idowu et al. (2022)).

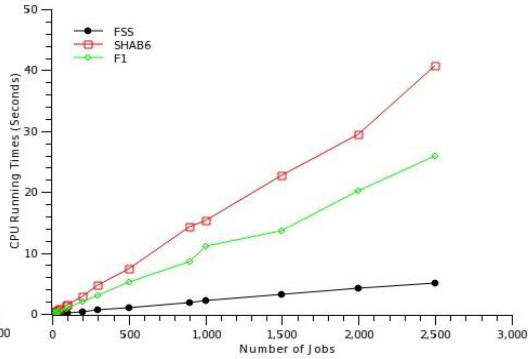
Testing a set of algorithm (Heuristics / Meta-heuristics) H, the RDI obtained by an algorithm $s \in H$ when applied to an instance t is defined as follows:

$$RDI = \begin{cases} 0, & \text{if } \min\limits_{s\in H}\{\phi_{st}\} = \max\limits_{s\in H}\{\phi_{st}\} \\ \dfrac{\phi_{st} - \min\limits_{s\in H}\{\phi_{st}\}}{\max\limits_{s\in H}\{\phi_{st}\} - \min\limits_{s\in H}\{\phi_{st}\}} \times 100 & \text{otherwise} \end{cases}$$

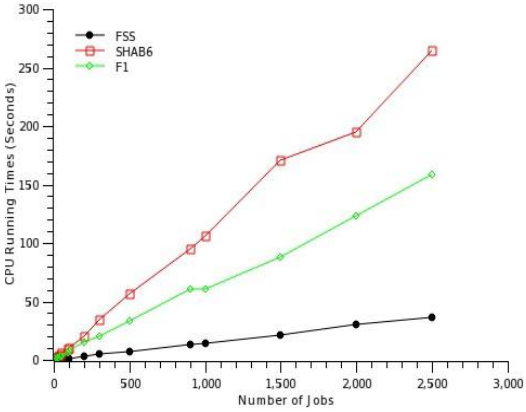
where ϕ_{st} is the objective value obtained by algorithm FSS on any of the instances described earlier in section 2. In this work, $\min\limits_{s\in H}\{\phi_{st}\}$ is the optimal value obtained if MILBPNW model is able to find optimal solution within an assigned computational time.



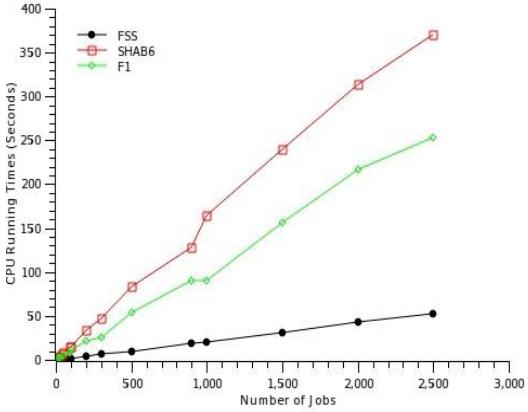
(a) CPU Running Times when m=3



(b) CPU Running Times when m=10



(c) CPU Running Times when m=70



(d) CPU Running Times when m=100

Fig 4: Comparison of Average CPU Running Times FSS with SHAB6 and F1 Algorithms

Table 8: Average Relative Deviation of FSS from optimal

m	n	OBJECTIVE		RDI	m	n	OBJECTIVE		RDI	m	n	OBJECTIVE		RDI
		MBILP-1	FSS				MBILP-1	FSS				MBILP-1	FSS	
3	10	0.54	0.55	2.00	7	10	0.64	0.65	2.00	10	10	0.71	0.72	2.00
	20	1.06	1.09	2.86		20	1.11	1.14	2.86		20	1.22	1.25	2.86
	30	1.82	1.84	1.23		30	2.01	2.03	1.23		30	2.29	2.32	1.23
	40	2.16	2.22	2.60		40	2.59	2.66	2.60		40	2.79	2.86	2.60
	50	3.00	3.09	3.06		50	3.44	3.55	3.06		50	3.72	3.83	3.06
	90	5.00	5.08	1.53		90	5.16	5.24	1.53		90	*	5.44	*
	100	5.13	5.24	2.15		100	5.54	5.66	2.15		100	*	7.23	*
15	10	0.72	0.74	2.42	20	10	0.73	0.75	2.16	25	10	0.75	0.76	1.51
	20	1.27	1.29	1.65		20	1.28	1.31	2.03		20	1.32	1.36	2.91
	30	2.32	2.39	2.95		30	*	2.40	*		30	*	2.43	*
	40	*	2.87	*		40	*	2.87	*		40	*	2.97	*
	50	*	3.90	*		50	*	3.96	*		50	*	4.00	*
	90	*	5.46	*		90	*	5.66	*		90	*	5.70	*
	100	*	7.82	*		100	*	8.13	*		100	*	8.62	*

* Problem instances where XPRESS-MP did not return optimal values after 600seconds

Table 8 shows the RDI computed for FSS with MILBP-1 using test instances described in Section 2 where optimal solutions were obtained within 600 seconds CPU time. As shown Table 8, FSS attained an overall average of 3% deviation from the optimum. This implies that a solution obtained with FSS can be a substitute for an optima schedule if the desired error tolerance is not more than 0.03. Also, it was observed that the maximum RDI occurred when the number of machines is at levels 3, 7 and 10 as well as when the number of Jobs is at 50; while it was at the lowest when the number of machines is at 3 and jobs at 30. Therefore, the number of solutions nearly optimal (that is 99% close to optimal) was obtained when the number of jobs is 30.

VII. CONCLUSION

This work addressed the problems of minimizing NET jobs on flow shop. A two-stage optimal solution seeking procedure is proposed for minimizing NET jobs on permutation flow shop problem where a near optimal solution obtained at first stage is improved by a FSS at the second stage. The proposed algorithm is shown to be superior in execution time as well as in quality of solutions to existing methods. In addition, the FSS algorithm was compared using small problem instances with an optimal solver. The mean optimal gap between the two of them did not exceed 4% in all cases considered.

REFERENCES

[1]. Adamu, M. O., & Abass, O. (2010). Parallel machine scheduling to maximize the weighted number of just-in-time jobs. *Journal of Applied Science and Technology*, *15*(1 and 2), 27-34.

[2]. Adamu, M. O., Budlender, N., & Idowu, G. A. (2014). A note on just in time scheduling on flow shop machines. *Journal of the Nigerian Mathematical Society*, *33*, 321-331.

[3]. Adamu, M. O., & Idowu, G. A. (2017). Meta-heuristics for unrelated machine scheduling problems. *Nigerian Journal of Scientific Research*, *16*(1), 8-17.

[4]. Arkin, E., & Silverberg, E. (1987). Scheduling jobs with fixed start and finish times. *Discrete Applied Mathematics*, *18*, 1-8.
[5]. Bouzina, K. I., & Emmons, H. (1996). Interval scheduling on identical machines. *Journal of Global Optimization*, *9*(3-4), 379-393.
[6]. Bulfin, R. L., & M'Hallah, R. (2003). Minimizing the weighted number of tardy jobs on a two-machine flow shop. *Computers and Operations Research*, *30*(12), 1887–1900.
[7]. Cepek, O., & Sung, S. C. (2002). A quadratic time algorithm to maximize the number of just- in-time jobs on identical parallel machine. *Computers and operations research*, *32*(12), 3265-3271.
[8]. Cheng, J., T.C.E.and Gupta, & Wang, G. (2000). A review of flow shop scheduling research with setup times. *Production and Operations Management*, *9*, 262-282.
[9]. Framinan, J. M., Perez-Gonzalez, P., & Fernandez-Viagas, V. (2019). Deterministic assembly scheduling problems: A review and classification of concurrent-type scheduling models and solution procedures. *European Journal of Operation Research*, *373*(2), 401-417.
[10]. Graham, R., Lawler, E., Lenstra, T., & Rinnooy Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathe- matics*, *5*, 287-326.
[11]. Hiraishi, K., Levner, E., & Vlach, M. (2002). Scheduling of parallel identical machines to max- imize the weighted number of just-in-time jobs. *Computers and Operations Research*, *29*(7), 841-848.
[12]. Idowu, G. A., Adamu, O. M., Sawyerr, A. B., Mustapha, A. R., & Rahman, O. I. (2022). Complexity and mathematical programming models for minimization of number of early- tardy jobs in just-in-time scheduling in flow shop. *Journal of Engineering, Science and Technology*, *XX*(XX), In Press.
[13]. Janiak, A., Janiak, W., & Januszkiewicz, R. (2009). Algorithms for parallel processor scheduling with distinct due windows and unit-time jobs. *Bulletin of the Polish Academy of Sciences Technical Sciences*, *57*(3), 209-215.
[14]. Janiak, A., Janiak, W. A., Krysiak, T., & Kwiatkowski, T. (2015). A survey on scheduling problems with due windows. *European Journal of Operational Research*, *242*(2), 347– 357.
[15]. Lann, A., & Mosheiov, G. (1996). A single machine scheduling to minimize the number of early and tardy jobs. *Computers and Operations Research*, *23*, 765-781.
[16]. Lann, A., & Mosheiov, G. (2003). A note on the maximum number of on-time jobs on parallel identical machines. *Computers and Operations Research*, *30*, 1745-1749.
[17]. M'Hallah, R. (2016). Minimizing total earliness and tardiness on permutation flow shop using vns and mip. *Computers and Industrial Engineering*, *75*, 142-156.
[18]. Mosheiov, G., & Sarig, A. (2009). Minmax scheduling problems with a common due-window.
[19]. *Computers and Operations Research*, *36*(6), 1886–1892.
[20]. Ohno, T. (1988). *Toyota production system: beyond large-scale production*. crc Press.
[21]. Pinedo, M. (2015). Scheduling. *Springer*.
[22]. Prot, D., Bellenguez-Morineau, O., & Lahlou, C. (2002). New complexity results for parallel identical machines scheduling problems with preemption, release dates and regular criterion. *European Journal of operational Research*, *231*, 282-287.
[23]. Rasti-Barzoki, M., & Hajazi, S. (2013). Minimizing the weighted number of tardy jobs with due date assignment and capacity deliveries for multiple custormers in supply chains. *European Journal of operational Research*, *228*, 345-357.
[24]. Rosa, B. F., Souza, M. J. F., De Souza, S. R., Filho, M. F., Ales, Z., & Michelon, P. Y. P. (2017). Algorithms for job scheduling problems with distinct time windows and general earliness/tardiness penaltiese. *Computers and Operations Research*, *81*, 203–215.
[25]. Shabtay, D., & Bensoussan, Y. (2012a). Maximizing the weighted number of just-in-time jobs in several two machine scheduling systems. *Journal of Scheduling*, *15*(1), 39–47.
[26]. Shabtay, D., & Bensoussan, Y. (2012b). Maximizing the weighted number of just-in-time jobs in several two-machine scheduling systems. *Journal of Scheduling*, *15*(1), 39–47.
[27]. Shabtay, D. (2012c). The just-in-time scheduling problem in a flow shop scheduling system.
[28]. *European Journal of Operational Research*, *216*, 521–532.
[29]. Sung, S., & Vlach, M. (2005). Maximizing weighted number of just-in-time jobs on unrelated parallel machines. *Journal of Scheduling*, *8*(5), 453-460.
[30]. Yeung, W. K., Oğuz, C., & Cheng, T. C. E. (2009). Two-machine flow shop scheduling with common due window to minimize weighted number of early and tardy jobs. *Naval Research Logistics*, *5*, 593-599.

m	n	RND	EDD
3	10	0.014	0.006
	20	0.023	0.013
	50	0.051	0.03
	100	0.119	0.062
	200	0.206	0.122
	500	0.505	0.325
	1000	1.573	0.607
	1500	1.42	0.97
	2000	2.464	1.191
	2500	2.267	1.563
40	10	0.166	0.081
	20	0.444	0.161
	50	0.757	0.416
	100	1.53	0.798
	200	3.32	1.558
	500	7.547	4.421
	1000	17.717	8.851
	1500	27.08	11.813
	2000	26.739	16.51
	2500	50.26	21.347

m	n	RND	EDD
7	10	0.026	0.015
	20	0.053	0.031
	50	0.137	0.074
	100	0.202	0.152
	200	0.673	0.294
	500	1.258	0.738
	1000	2.164	1.493
	1500	5.536	2.191
	2000	6.182	2.797
	2500	9.113	3.784
70	10	0.346	0.141
	20	0.547	0.289
	50	1.652	0.769
	100	3.675	1.465
	200	6.625	2.883
	500	11.818	7.032
	1000	28.859	14.132
	1500	45.915	21.477
	2000	67.922	30.63
	2500	79.543	36.41

m	n	RND	EDD
9	10	0.042	0.019
	20	0.096	0.037
	50	0.165	0.094
	100	0.279	0.191
	200	0.683	0.397
	500	1.783	0.898
	1000	2.81	1.973
	1500	5.989	2.994
	2000	7.903	3.741
	2500	9.263	4.686
80	10	0.397	0.169
	20	0.551	0.319
	50	1.654	0.835
	100	3.706	1.674
	200	6.701	3.2
	500	17.344	7.796
	1000	31.538	16.989
	1500	46.597	26.362
	2000	73.671	34.676
	2500	85.702	39.928

m	n	RND	EDD
10	10	0.05	0.02
	20	0.103	0.039
	50	0.182	0.101
	100	0.369	0.196
	200	0.811	0.415
	500	2.263	1.011
	1000	4.356	2.147
	1500	6.373	3.267
	2000	8.626	4.171
	2500	9.666	5.06
100	10	0.445	0.2
	20	0.769	0.44
	50	2.241	1.032
	100	4.249	2.013
	200	9.384	4.257
	500	25.897	10.034
	1000	47.217	20.023
	1500	76.459	30.846
	2000	106.773	43.155
	2500	99.649	52.894

Table 5: Comparison of average CPU runtimes of initializing FSS with EDD and RND in seconds

m	n	ALGORITHMS		
		SHAB6	F1	FSS
3	10	0.79	0.72	0.55
	20	1.54	1.45	1.09
	50	4.11	4.03	3.09
	100	7.42	6.92	5.24
	200	15.25	14.31	10.78
	500	40.13	37.52	28.03
	1000	80.77	75.34	61.58
	1500	119	109.17	82.65
	2000	150.16	142.49	106.23
	2500	197.36	186.6	138.63
25	10	1.07	0.99	0.76
	20	1.94	1.84	1.36
	50	5.55	5.31	4
	100	12.51	11.52	8.62
	200	24.06	22.41	17.15
	500	59.62	56.3	41.53
	1000	111	102.53	78.34
	1500	171.36	154.94	115.48
	2000	197.59	195.51	147.82
	2500	278.54	256.4	193.22

m	n	ALGORITHMS		
		SHAB6	F1	FSS
7	10	0.91	0.86	0.65
	20	1.6	1.53	1.14
	50	4.47	4.73	3.55
	100	8.01	7.61	5.66
	200	17.69	16.36	12.4
	500	41.96	39.02	29.7
	1000	95.9	88.45	72.9
	1500	133.85	127.47	94.8
	2000	179.96	159.79	136
	2500	207.03	195.43	147
50	10	1.17	1.07	0.81
	20	2.2	2.02	1.54
	50	6.5	6	4.61
	100	12.95	12.23	9.04
	200	27.53	25.22	19.1
	500	64.23	58.97	45.1
	1000	114.64	107.67	80.6
	1500	192.63	172.5	129
	2000	213.4	199.45	150
	2500	315.69	289.8	218

m	n	ALGORITHMS		
		SHAB6	F1	FSS
15	10	1.06	0.98	0.74
	20	1.82	1.69	1.29
	50	5.45	5.19	3.9
	100	11.11	10.25	7.82
	200	20.68	19.28	14.4
	500	46.62	43.37	32.9
	1000	107.38	99.93	74.8
	1500	151.18	141.38	105
	2000	193.91	182.72	137
	2500	257.48	240.94	181
70	10	1.22	1.12	0.85
	20	2.63	2.26	1.71
	50	6.67	6.22	4.72
	100	13.66	12.83	9.62
	200	27.83	25.77	19.6
	500	70.08	65.72	49.5
	1000	132.41	122.22	92
	1500	202.99	180.01	137
	2000	215.47	204.23	152
	2500	321.29	298.84	225

m	n	ALGORITHMS		
		SHAB6	F1	FSS
20	10	1.07	0.98	0.75
	20	1.86	1.73	1.31
	50	5.53	5.28	3.96
	100	11.68	10.84	8.13
	200	21.04	19.6	15.02
	500	57.48	55.23	41.13
	1000	110.51	101.43	77.42
	1500	165.4	144.54	107.85
	2000	196.4	187.83	140.01
	2500	276.3	253.12	191.83
100	10	1.46	1.35	1.02
	20	2.92	2.68	2
	50	7.14	6.67	5.05
	100	14.25	13.15	9.84
	200	28.86	26.63	20.1
	500	72.19	66.51	50.51
	1000	144.05	134.69	100.96
	1500	214.92	191.94	146.21
	2000	288.77	268	200.39
	2500	364.02	340.18	252.38

Table 6: Average Number of Early-Tardy jobs of SHAB6, F1 and FSS

m	n	ALGORITHMS		
		SHAB6	F1	FSS
3	10	0.042	0.03	0.01
	20	0.091	0.06	0.01
	50	0.218	0.15	0.03
	100	0.461	0.29	0.06
	200	0.96	0.62	0.12
	500	2.311	1.36	0.33
	1000	4.913	3.11	0.61
	1500	7.179	4.79	0.97
	2000	9.354	5.1	1.19
	2500	10.48	7.31	1.56
25	10	0.362	0.26	0.05
	20	0.72	0.49	0.11
	50	1.827	1.09	0.28
	100	4.134	2.28	0.49
	200	7.24	4.4	1.01
	500	20.274	11.6	2.67
	1000	34.846	23.7	5.2
	1500	58.065	33.9	7.92
	2000	82.642	45	10
	2500	86.907	63.2	12.5

m	n	ALGORITHMS		
		SHAB6	F1	FSS
7	10	0.103	0.067	0.02
	20	0.223	0.131	0.03
	50	0.569	0.358	0.07
	100	0.993	0.778	0.15
	200	2.181	1.434	0.29
	500	5.594	3.586	0.74
	1000	11.45	5.98	1.49
	1500	17.22	10.96	2.19
	2000	21.2	11.69	2.8
	2500	24.54	15.64	3.78
50	10	0.806	0.525	0.1
	20	1.46	0.863	0.2
	50	3.778	2.535	0.51
	100	7.215	4.516	1.04
	200	15.07	9.276	2.02
	500	41.2	26.15	5.32
	1000	79.85	43.16	10.4
	1500	123.5	79.89	15.3
	2000	165.1	103.7	21.8
	2500	207.1	106.5	24.5

m	n	ALGORITHMS		
		SHAB6	F1	FSS
15	10	0.22	0.151	0.03
	20	0.453	0.261	0.06
	50	1.198	0.828	0.16
	100	2.484	1.657	0.29
	200	4.311	2.649	0.6
	500	10.857	7.46	1.48
	1000	23.388	14.97	3.14
	1500	37.008	24.7	4.44
	2000	48.683	29.1	6.56
	2500	53.697	39.79	7.53
70	10	1.016	0.68	0.14
	20	2.158	1.37	0.29
	50	5.812	3.289	0.77
	100	10.047	7.577	1.47
	200	20.212	15.44	2.88
	500	56.58	32.84	7.03
	1000	106.152	60.77	14.1
	1500	171.193	87.52	21.5
	2000	195.038	123.4	30.6
	2500	264.607	158.9	36.4

m	n	ALGORITHMS		
		SHAB6	F1	FSS
20	10	0.311	0.178	0.04
	20	0.662	0.341	0.09
	50	1.539	1.067	0.2
	100	3.172	1.913	0.42
	200	5.922	4.206	0.81
	500	13.911	10.4	2.01
	1000	29.732	18.24	4.37
	1500	44.517	32.39	5.88
	2000	61.118	37.09	8.59
	2500	79.171	53.48	11
100	10	1.658	0.871	0.2
	20	3.003	1.75	0.44
	50	7.544	5.554	1.03
	100	15.371	11.03	2.01
	200	33.17	21.73	4.26
	500	83.317	53.66	10
	1000	164.713	89.57	20
	1500	239.731	156.4	30.8
	2000	313.81	216.4	43.2
	2500	370.677	253.2	52.9

Table 7: Average CPU runtimes in seconds of FSS, SHAB6 and F1