Nigerian Journal of Mathematics and Applications
Volume **23**, (2014), 77 − 95
©*Nig. J. Math. Appl.* *http : //www.kwsman.com*

JUST-IN-TIME SCHEDULING PROBLEMS ON IDENTICAL PARALLEL MACHINES

[1]Adamu M.O. and [2]Abass O.

Abstract

Problems involving Just-In-Time (JIT) scheduling provide an interesting and difficult challenge, which is addressed critically within this paper. This study considers the scheduling of parallel identical machines to maximize the (weighted) number of on-time jobs. This problem is known to be NP-complete. Three problems were dealt with in this paper. Two greedy heuristics with time complexity $O(n \log n)$ are provided for maximizing the weighted number of on-time jobs with equal processing times. Two greedy heuristic algorithms are proposed for solving the unweighted number of on-time jobs on m parallel identical machines using two different approaches. It is shown by computational and worst-case analysis that these algorithms with time complexity $O(n^{m+1} \log n)$ will give results very close to the optimal solutions. Lastly, an optimal greedy heuristic solution is provided for solving the problem of maximizing the number of on-time agreeable jobs with equal processing time with a running time given as $O(n \log n)$. A proof for feasibility of the algorithm is presented.

Received December 22, 2014. * Corresponding author.
2010 *Mathematics Subject Classification.* 49Nxx & 00Axx.
Key words and phrases. Just-In-Time, Parallel machines, NP Complete, Scheduling, Heuristics

[1]Department of Mathematics, University of Lagos
[2] Department of Computer Science, University of Lagos; e-mail: adamumus@yahoo.com

1. Introduction

The recently developed area of Just-In-Time (JIT) scheduling is concerned with scheduling jobs to minimize the total cost associated with both early and tardy completion. Most studies in this area have considered minsum objectives (in which the schedule sought minimizes the total sum of earliness/tardiness cost incurred for all jobs), or minmax objectives (in which the schedule sought minimizes the earliness/tardiness cost of the worst scheduled job ([12]), [3] gave a comprehensive review on the earliness and tardiness scheduling problems. A complete review of literature can be found in [1]. A complete review of literature on single machine is in [2]. Some papers that have considered this area of study, are [8], [9], [7] and [15]. The objective is to minimize the (weighted) number of early/tardy jobs or maximize the (weighted) number of on-time jobs on parallel identical machines. [11] and [5] have considered problems with due window on parallel machines where the objective is to find an optimal schedule with minimum earliness-tardiness penalty. The other two that have considered a type of our problem are [10] and [13]. Though in their case, the jobs are completed exactly at their due dates, that is the due windows for the jobs are zeros. They have shown the problem to be NP-hard. Though [12] and [15] and [17] are some few works that have considered a type of our problem on a single machine. This problem was shown to be NP-complete by [15] and [17]. The objective of this work is to find a schedule that minimizes the (weighted) number of early and tardy jobs or, in general to maximize the (weighted) number of on-time jobs on m parallel identical machines. [4] described how some classical scheduling criteria have been integrated into a constraint-based scheduling model. Their study focused on minimization of the weighted number of late jobs. New lower bounding technique and new constraint propagation schemes were proposed. They allowed them to optimally solve instances consisting of up to 50 jobs of the problem of minimizing the weighted number of late jobs on parallel machines with release dates ($P|ri|\sum w_iU_i$). They also showed how the bicriteria problem $P|r_i|Fh(Sw_iU_i, T_{\max})$ can be solved.

[16] also presented a branch and bound to minimize the weighted number of tardy jobs on either identical or non-identical processors. Bounds came from a surrogate relaxation resulting in a multiple-choice knapsack. Extensive computational experiments indicated problems with 400 jobs and several machines can be solved quickly. The results also indicated what parameters affect solution difficulty for this algorithm approach.

Consider an n job parallel identical machines scheduling problem. Let each job have a processing time, earliest start time and latest due date, an earliness weight, and a tardiness weight (penalty, cost). In a given schedule, let each job be penalized by the fixed individual earliness (tardiness) weight, if it is completed prior to (after) its due dates. A schedule is sought that will minimize the total cost incurred for all penalized jobs. As mentioned, contrary to previously studied models ([11] and [5]), here the penalty is not a function of the size of deviation from the due dates.

In the setting studied here, exact completion time is crucial and an early or tardy job is equally penalized, however early/tardy it is (In the general case the earliness cost may be different from the tardiness cost). Thus, it would cost just as much to miss a due date by a short period of time as by a long period of time. Cases of problems of this sort is significant and has many practical applications, e.g. in chemical or hi-tech industries, where often parts need to be ready at specific times in order to meet certain required conditions (arrival of other parts, specific temperature, pressure, e.t.c.). Production of perishable items (e.g. food, blood, drugs, photographic film) under deterministic demand has a similar cost structure. The remaining parts of the paper are as follows: Section two considers the problem formulation. The problem to maximize the weighted number of on-time jobs with equal processing time is outlined in section three. The problem to maximize the number of on-time jobs is presented in section four. In section five, the problem to maximize the number of on-time agreeable jobs with equal processing time is considered and finally, in section six, the conclusion.

2. Problem Formulation

There are n independent jobs, scheduled on m machines, which are simultaneously available from time zero, each having an interval rather than a point in time, called *due window* of the job, and the left end and the right end of the window are called, respectively, *the earliest start time* $a_i \geq 0$ and the *latest due date* $d_i \geq 0$. There is no penalty when a job is completed within its due window, but earliness (tardiness) penalty is incurred if a job is completed before its earliest start time (after its latest due date).
For any given schedule S, let p_j, t_{ij} and $C_j(S) = t_{ij} + p_j$ represent the processing time, actual start time on a given machine and completion time of job j on machine i, respectively. Job j is said to be early if $C_{ij}(S) < a_j$, tardy if $C_{ij}(S) > d_j$ and on-time if $a_j \leq t_{ij} + p_j \leq d_j$. In addition, let x_{ij} for j be defined as follows:

$$x_{ij} = \begin{cases} 1, & \text{if } a_j \leq C_{ij}(S) \leq d_j, \ (on - time) \\ 0, & \text{otherwise} \end{cases}$$

Furthermore, let $w_j \geq 0$ be the weights for the early/tardy jobs. The mathematical model of this problem is given as follows:

$$P : \max \sum_{i=1}^{m} \sum_{j=1}^{n} w_j x_{ij}$$

Such that

$$a_j \leq \left\{ \max_{k=1}^{j} \{C_{ik-1}(S), a_j p_j\} + p_j \right\} x_{ij},\ i = 1, \ldots, m; j = 1, \ldots, n, \tag{1}$$

$$\left\{ \max_{k=1}^{j} \{C_{ik-1}(S), a_j p_j\} + p_j \right\} x_{ij} \leq d_j,\ i = 1, \ldots, m; j = 1, \ldots, n, \tag{2}$$

$$\sum_{i=1,m} x_{ij} \leq 1 j = 1, \ldots, n \tag{3}$$

$$x_{ij} \in \{0, 1\}, i = 1, \ldots, m; j = 1, \ldots, n \tag{4}$$

Constraint (1) insures job j is not finished before its earliest start time a_j. Constraint (2) is the completion time of job j if it is no greater than its latest due date d_j. Constraint (3) insures that a job is assigned to at most one machine, and constraint (4) forces a job to be either on-time or early/tardy; 1 if on-time and zero otherwise.

3. Maximizing Weighted Number of On-Time Jobs with Equal Processing Time

In this section, the problem with equal processing time, $P_m|p_j = p|\sum\sum w_j x_{ij}$ is studied. This problem is strongly NP-complete. If we assume that the jobs are numbered in Earliest Due Date (EDD) order so that $a_1 \leq a_2 \leq \ldots \leq a_n$. Two heuristics WOP and DOP are proposed for solving this problem. In this section, a job is assigned to the machine with the smallest completion time.

3.1. Heuristics for Maximizing the Weighted Number On-Time Jobs with Equal Processing Time.

First Greedy Heuristic for Maximizing Weighted Number of On-Time Jobs with Equal Processing Time (WOP)

```
• Step 1 : Arrange the earliest start time in ascending order
• Step 2 : Assign the first m jobs on the m machines, A job is selected
  according to EDD and a tie is broken by highest weighted job (HWJ)
  $j \leftarrow m$
• Step 3 : If $j > n$, Goto Step 5 Else Select a machine with the smallest
  completion time and assign a job to it according to EDD and break tie
  by HWJ
• Step 4 : If $\max\{C_{ik}(S), a_j p_j\} + p_j \leq dj$
  Then $C_i(S) = \max\{C_{ik}(S), a_j p_j\} + p_j$
  $j \leftarrow j + 1$,
  Goto Step 3
  Else
  Let job $j$ be moved to $Q$ (tardy jobs),
    $j \leftarrow j + 1$
    Goto Step 3
• Step 5 : Then append the jobs in $Q$ to any of the $m$ machines and stop.
```

Second Greedy Heuristic for Maximizing Weighted Number of On-Time Jobs with Equal Processing Time (DOP)

```
• Step 1 : Arrange $E_j$ of all jobs in ascending order
• Step 2 : Assign the first $m$ jobs on the $m$ machines,
    A job is selected in ascending order of $E_j$ and a tie is broken by highest weighted job (HWJ)
    $j \leftarrow m$
• Step 3 : If $j > n$, Goto Step 5
    Else Select a machine with the smallest completion time and assign a job in ascending order of $E_j (E_j = a_j - p_j)$ and break tie by HWJ
• Step 4 : If $\max\{C_{ik}(S), a_j p_j\} + p_j \leq d_j$
    Then $C_i(S) = \max\{C_{ik}(S), a_j p_j\} + p_j$
    $j \leftarrow j + 1$,
    Goto Step 3
    Else
    Let job $j$ be moved to $Q$ (tardy jobs),
    $j \leftarrow j + 1$
    Goto Step 3 Step 5 : Then append the jobs in $Q$ to any of the $m$ machines and stop.
```

Running time: Step 1 has to do with sorting and that requires $\log n$. Step 2 is repeated m times and requires $O(m)$ time. Step 3 and 4 are repeated at most nm times and requires $O(n)$ time. Therefore, the time completion, for each of the heuristics is at most $O(n \log n)$.

3.2. **Computational Studies for Weighted Number of On-Time Jobs on Parallel Machines with Equal Processing Time. Problem Generation**

These experiments were designed to test the most effective of the proposed algorithms WOP and DOP. Problems with 500, 1,000, 1,500, 2,000 and 2,500 jobs were generated to test these heuristics. The number of machines is set at five levels: 2, 5, 10, 15 and 20 . The processing time p_j for all set to 40. The integer earliest due date a_j is randomly generated in the interval $[0, n/mk_1]$ and the integer latest due date d_j is randomly generated in the interval $[a_j + p_j, a_j + p_j + n/mk_2]$. Two parameters k_1 and k_2 are used and selected from the set $\{1, 5, 10, 20\}$. For each combination of n, k_1 and k_2 10 instances are generated, i.e. for each value of n, 160 instances are generated with a weight randomly chosen in $[1, 10]$ and with a total of 8,000 problems (50 replications). The number of machines, m is fixed at 20. Each problem set was solved on a Pentium 233 MHz processor with a 32 MB memory.

Computational Results

The overall results of the 8,000 test problems are summarized in Table 1. Figures 1 to 10 reveal some important facts about these two algorithms WOP and DOP. Our simulation results reveal that heuristic DOP performs slightly better than algorithm WOP. It is also observed that algorithm DOP keeps improving faster than algorithm WOP as the value of m increases. So, for large $m > 20$, the gap between algorithm DOP and WOP will be widening. The only noticed improvement of algorithm WOP over algorithm DOP is at $n = 2000$ and the number of machines is 5 (Figure 4).

Table 1: Results for maximizing the weighted number of on-time jobs on identical parallel machines with Equal Processing Time

Number of Jobs	Heuristics	Number of Machines				
		2	5	10	15	20
	WOP	713.1	550.36	408.68	311.7	209.9
500		73.6	110.6	131.2	163.6	203.6
	DOP	703.66	521.78	351.16	215.32	118.92
		64.6	87	126.6	154.8	182.6
	WOP	856.4	785.34	674.84	583.88	498.32
1000		69.4	107.6	135	164	191
	DOP	844.28	757.78	615.84	497.04	385
		89.2	119.2	151.6	185.4	215
	WOP	827.62	748.08	626.3	519.34	428
1500		68	102.2	133.6	167.4	202.6
	DOP	815	719.48	571.18	434.94	315.92
		75.6	110.4	135.6	166.2	195
	WOP	793	611.3	542.38	439.92	355.46
2000		61.8	91.6	124.2	156	182.4
	DOP	788.96	664.31	486.81	355.88	243.64
		73.8	110.4	134.8	165.2	196
	WOP	792.44	682.08	532.14	429.4	346.76
2500		60.6	88.4	122.6	148.4	179.6
	DOP	781.26	653.94	477.66	348.32	276.7
		71.2	107.8	132	159.8	191.4

The performances of both algorithms at $m = 2$ and $m = 5$ are very close for all N. The average CPU time of each of the Heuristics is less than a second. Moreover, it was observed that the time taken to run heuristics increases with the increase in the numbers of machines.

4. Maximizing the Number of On-Time Jobs

This section continues with a special case (where the weights of the jobs are equal) of the general problem $P_m||\sum\sum w_j x_{ij}$. The problem of maximizing the number of on-time jobs on m parallel machines with due window ($P_m||\sum\sum x_{ij}$) will be considered in this section. This problem is also known to be strongly NP-complete and finding an algorithm to optimally solve the problem is very unlikely.

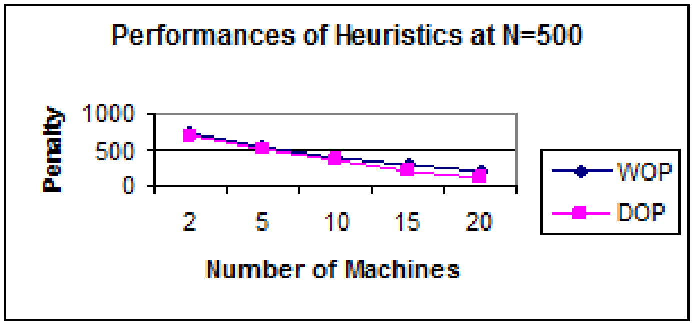

Figure 1: Performances of Heuristics at N=500

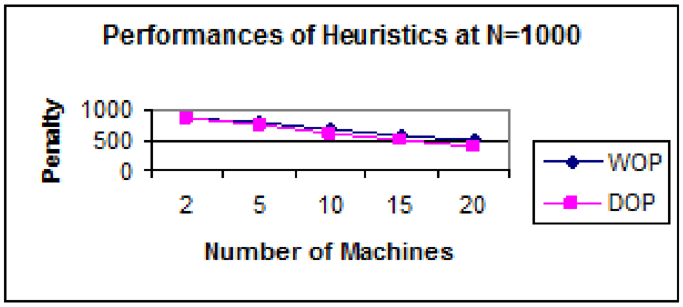

Figure 2: Performances of Heuristics at N=1000

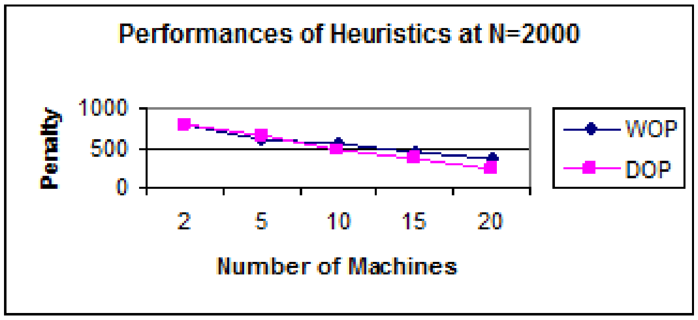

Figure 3: Performances of Heuristics at N=1500

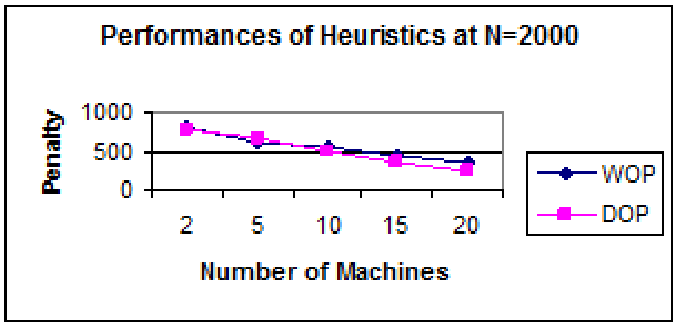

Figure 4 : Performances of Heuristics at N=2000

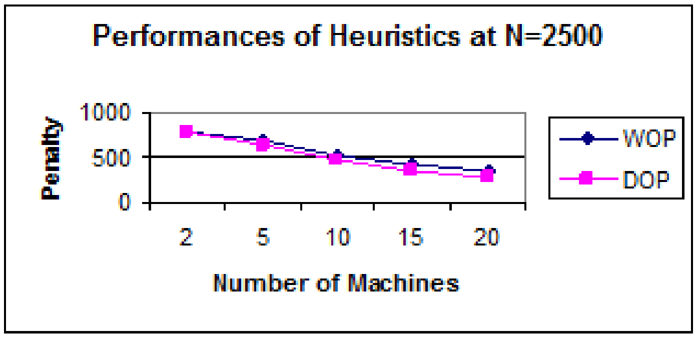

Figure 5: Performances of Heuristics at N=2500

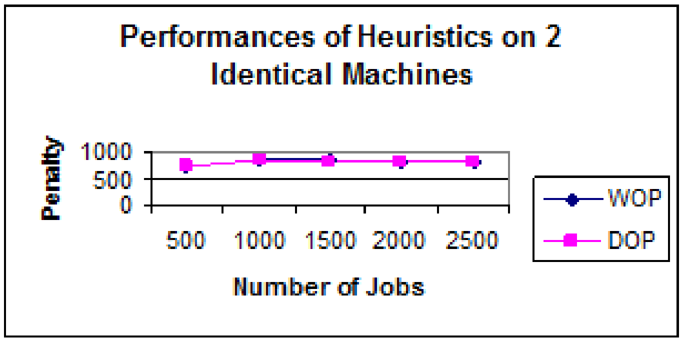

Figure 6: Performances of Heuristics on 2 Machines

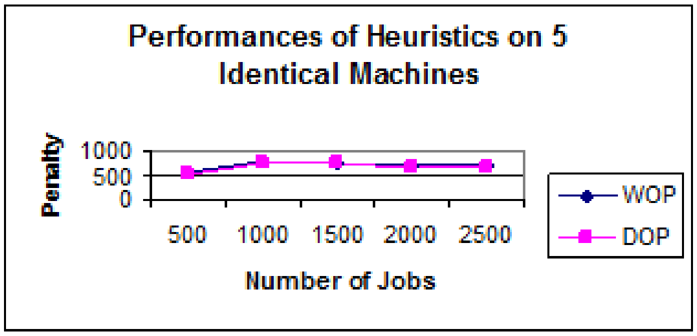

Figure 7: Performances of Heuristics on 5 Machines

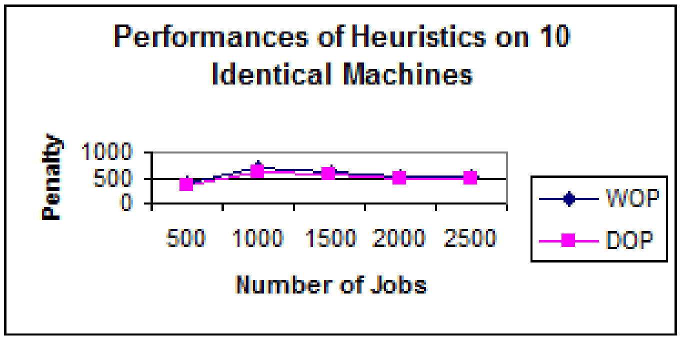

Figure 8: Performances of Heuristics on 10 Machines

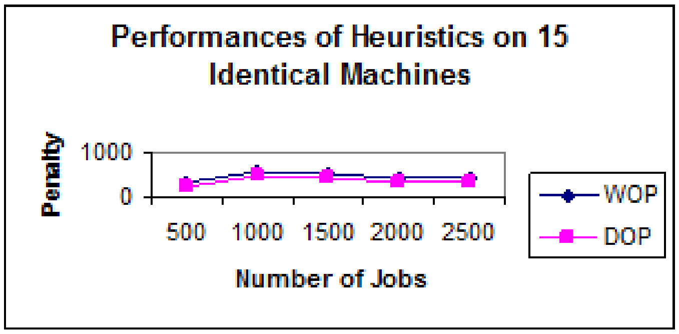

Figure 9: Performances of Heuristics on 15 Machines

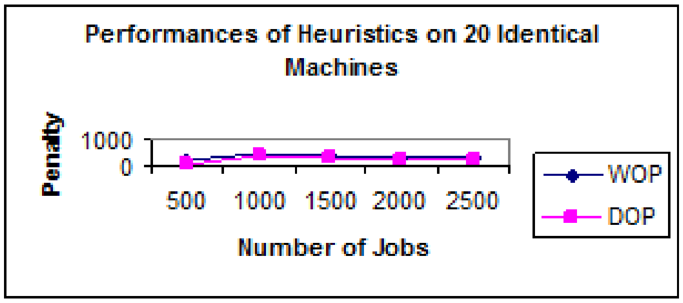

Figure 10 : Performances of Heuristics on 20 Machines

It is clearly better to schedule a penalized job late rather than early, since a tardy job can be scheduled sufficiently late (with no additional cost) to guarantee that it will have no effect on the other jobs. Based on this observation, we propose two heuristic algorithms containing on time and tardy jobs only. Jobs are assigned machine by machine. In both algorithms the shortest processing time (SPT) rule is applied to the arrangement of the jobs before scheduling.

[6] considered a similar case of our problem. The input to their problem consists of n jobs and k machines. Each of the jobs is associated with a release time, a weight, a deadline, and a processing time on each of the machines. The goal is to find a non preemptive schedule that maximizes the (weight) cardinal of jobs that meet their respective deadlines. Our case is an extension of their problem for the unweighted model. Firstly, our jobs are simultaneously available from time zero and secondly, our jobs can be scheduled before their earliest start times but must finish on or after their earliest start times. We are adopting the method of [6] for solving our problem. The greedy algorithms proposed schedule the jobs machine by machine, updating the set of jobs to be scheduled on each machine to include only jobs that have not been scheduled on previous machines. [6] greedy algorithm schedules the job instance that finishes first among all jobs that can be scheduled at time t or later and does not take into consideration the deadlines of the jobs, except for determining whether the jobs are eligible for scheduling.

For our heuristic PO1, we scheduled the jobs in ascending orders of their earliest start times while for heuristic PO2, the method of [6] is adopted to schedule the jobs.

4.1. **Greedy Heuristics.** *First Greedy Heuristic Maximizing the Number of On-Time Jobs (PO1)*

- Step 1: Arrange the jobs in ascending orders of their earliest start time i.e., $a_1 \leq a_2 \leq \ldots \leq a_n$ and the shortest processing time (SPT) is applied if the earliest start times are equal
- Step 2: Do $I = 1, m$
  Assign the jobs machine by machine

- Step 3: $j \leftarrow j+1$, if $j > n$, Goto step 5
  Else select a job and assign job j to machine i
  according to EDD and break tie by SPT
- Step 4: If $\max\{C_{ik}(S), a_j - p_j\} + p_j \leq d_j$ Then assign job j on machine i Goto Step 3
  Else Job j is retained to be scheduled on next machine
  Goto step 3
- Step 5: If $I < m$ goto step 2
- Step 6: Then append the jobs tardy to any of the m machines and stop.

Second Greedy Heuristic for Maximizing the Number of On-Time Jobs (PO2)

- Step 1: Arrange all the jobs in ascending orders of their completion times
- Step 2: Do $I = 1, m$
  Assign the jobs machine by machine
- Step 3: $j \leftarrow j+1$, if $j > n$, Goto step 5
  Else select a job and assign job j to machine i
  In ascending order of the E_j and break tie by SPT
- Step 4: If $\max\{C_{ik}(S), a_j - p_j\} + p_j \leq d_j$ Then assign job j on machine i Goto Step 3
  Else
  Job j is retained to be scheduled on next machine Goto step 3
- Step 5: If $I < m$ goto step 2
- Step 6: Then append the jobs tardy to any of the m machines and stop.

Running Time: Step 1 is done in $O(n \log n)$ time, steps 2, 3 and 4 are done in at most $O(nm)$ time. Therefore, the overall time complexity of each if the heuristic is at most $O(nm + 1 \log n)$.

4.2. **Computational Studies for Maximizing Number of On-Time Jobs on Parallel. Problem Generation**

A simulation experiment is employed to test the efficiency of the proposed heuristics. The number of machines (m) is set at five levels : 2, 5, 10, 15 and 20. For each j, an integer processing time p_j is randomly generated in $[1, 99]$. In this case, all the jobs have equal weights. Two parameters k_1 and k_2 are used, and selected from the set $\{1, 5, 10, 20\}$. Due to the fact that we want the data to depend on the number of jobs n, the integers earliest start times a_j are randomly generated in the interval $[0, n/mk_1]$ and the integer latest due dates d_j are randomly generated in the interval $[a_j + p_j, a_j + p_j + n/mk_2]$. For each combination of n, k_1 and k_2, 10 instance are generated, i.e., for each value of n, 160. The heuristics are implemented in Fortran on a Pentium 233 MHz processor with a 32 MB memory

Computational Results

The overall results of the 4,000 test problems are summarized in Table 2 while figures 11-20 show the performances of the heuristics. The values in Table 2 represent the average number of on-time jobs. Figures 11-15 reveal the performances of the heuristics on different numbers of machines at 5 levels of $N = 500, 1,000, 1,500, 2,000$ and $2,500$. Also, figures 16-20 show the performances of the heuristics at different levels of n on m identical machines (where $m = 2, 5, 10, 15$ and 20).

It is interesting to note that heuristic PO1 performs better in all cases considered than PO2 as a result of having higher number of on-time jobs. It is also observed that heuristic PO1 improves over heuristic PO2 as the number of jobs and number of machines increase.
The average timing for both algorithms is very similar and is less than a second.

We proceed to give the constant factor approximation for our algorithms with guaranteed performance (approximation factor or worst case). We say that an algorithm has a worst-case factor for a maximization problem if the weight of its solution is at least $\frac{1}{\rho}$. OPT, where OPT is the weight of an optimal solution. (Note that we defined the worst case factor so that it would always be at least 1.)

Let the job system be n jobs $J = (J_1, \dots, J_n)$ and m machines $M = (M_1, \dots, M_m)$. Each job J_j is characterized by the triplet (a_j, d_j, p_j). The interpretation is that job J_j can be ready from time a_j, the earliest start time, not later than d_j, the

Table.2 : Results for Maximizing the number of on-time jobs on identical parallel machines

Number of Jobs	Heuristics	Number of Machines				
		2	5	10	15	20
500	PO1	18.4	39	58.8	69.8	81.4
	PO2	14.6	34.6	52.4	62.6	71.6
	PO1	5.1	12.3	24.7	35.9	46.9
1000	PO2	4	11.2	20.1	30.7	38.4
	PO1	6.6	16.4	30.6	44.2	57
1500	PO2	5.4	14	26	35.4	42.6
	PO1	10	24.2	43.4	56.8	66.8
2000	PO2	7.6	16.8	31.4	39.6	42.8
	PO1	10	24.4	44.4	57.8	67.8
2500	PO2	7.8	16.6	31.8	40	43.2

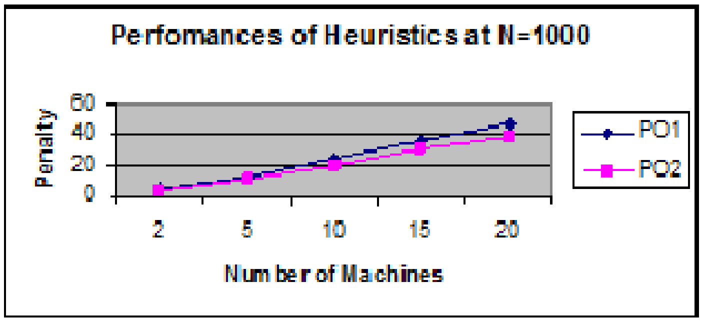

Figure 11: Performances of Heuristics at N=500

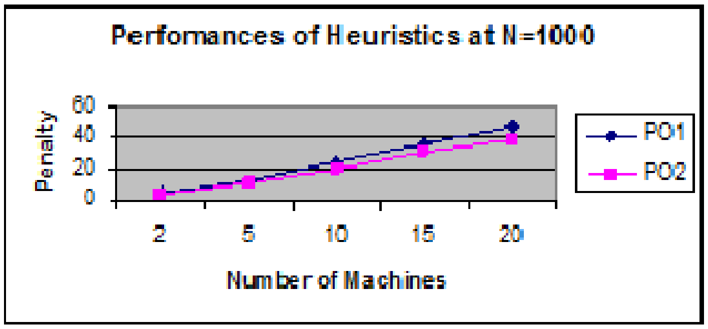

Figure 12: Performances of Heuristics at N=1000

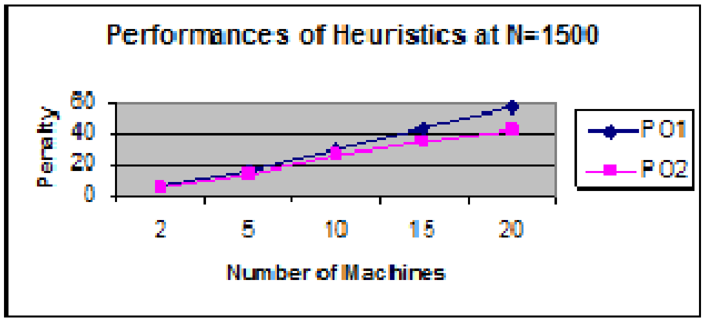

Figure 13: Performances of Heuristics at N=1500

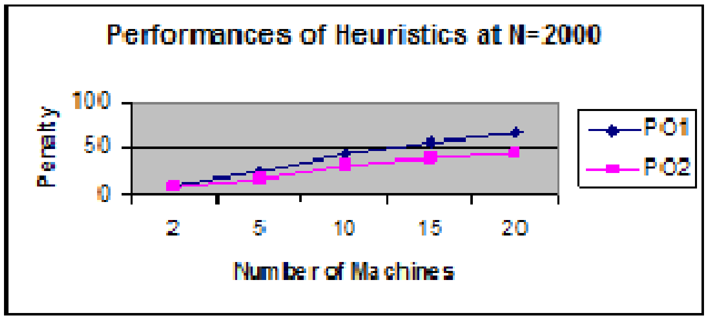

Figure 14 : Performances of Heuristics at N=2000

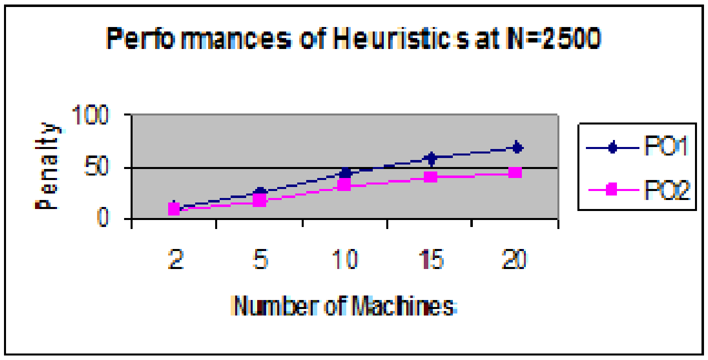

Figure 15: Performances of Heuristics at N=2500

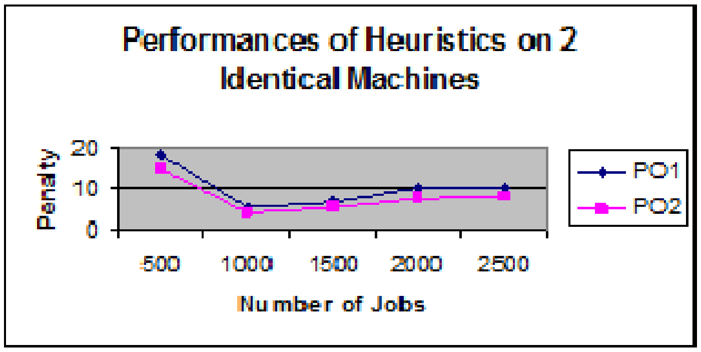

Figure 16: Performances of Heuristics on 2 Machines

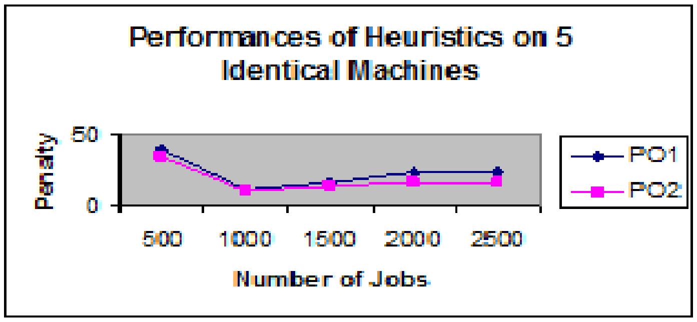

Figure 17: Performances of Heuristics on 5 Machines

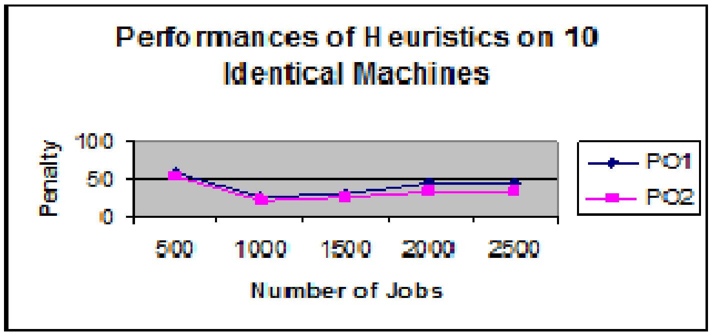

Figure 18: Performances of Heuristics on 10 Machines

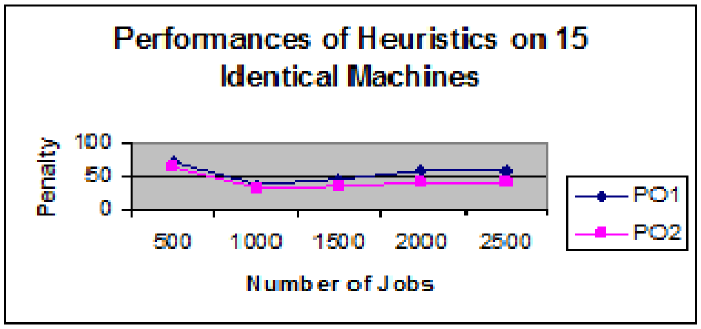

Figure 19: Performances of Heuristics on 15 Machines

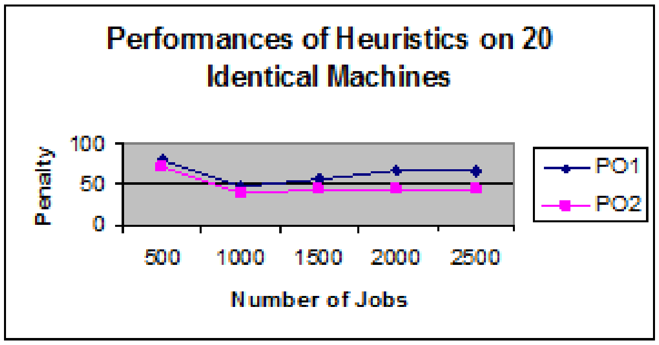

Figure 20 : Performances of Heuristics on 20 Machines

latest due date, its processing time on any of the machine is p_j. In these heuristics, PO1 and PO2, our goal is to maximize the number (cardinality) of the set of scheduled jobs. The analysis of the heuristic PO2 is similar to the one described by [6] where the worst-case factor is$\rho(m) = \dfrac{(m+1)^m - m^m}{(m+1)^m}$ and from the results of the numerical analysis heuristic PO1 cannot perform worst than the worst-case of heuristic PO2.

5. Maximizing the Number of On-Time Agreeable Jobs with Equal Processing Time.

In this section, we present an efficient $O(n \log n)$ solution algorithm for the problem of maximizing the number of on-time agreeable jobs with equal processing time, i.e. minimizing the number of early and tardy agreeable jobs with equal processing time $(Pm|p_j = p|\sum\sum x_{ij})$. We consider the problem when the earliest due dates and the latest due dates are agreeable, that is, the earliest due dates must increase in the same sequence as the latest due dates. We present an optimal polynomial time algorithm for this problem below

5.1. **Heuristic for Maximizing the Number of On-Time Agreeable Jobs with Equal Processing Time (POP).**

- Step 1: Arrange the jobs in ascending orders of their earliest due date. $a_1 \leq a_2 \leq \ldots \leq a_n$
- Step 2: $j \leftarrow j + 1$, if $j > n$, Goto step 4
  Else select the machine with the smallest completion time and assign a job to it according to EDD order
- Step 3: If $\max\{C_{ik-l}(S), a_jp_j\} + p_j \leq d_j$ Then $C_i(S) \leftarrow \max\{C_{ik-1}(S), a_jp_j\} + p_j$. Goto step 2 Else
  Let job j be moved to Q (tardy job) . Goto step 2.
- Step 4: Then append the jobs in Q to any of the m machines and stop.

Running Time: Step 1 is done in $O(\log n)$ time. Step 2 and 3 are done at most $O(n)$ time. The overall complexity of this heuristic is $O(n \log n)$.

Algorithm POP will always produce an optional Solution of this problem considered. It is observed that the two sorting procedure used in section 4 will amount to the same thing due to the equal processing time (i.e. sorting according to EDD and ascending order of E_j will give the same result) The proof of optimality of POP is presented below:

Proposition 5.1 Algorithm POP produces an optimal schedule.

Proof

Let the first m jobs be scheduled on-time on the M identical parallel machines. Let H denote the set of on-time jobs. We prove by showing that removal of any given job from the set H enables a maximum of one alternative job to be included in H. Assume the next job j cannot be scheduled on-time on any of the M machines. Excluding any job i from any of the first M jobs from H, and replacing it with job j (to be completed on time on the same machine is feasible). Since the due dates of the jobs are agreeable, then $a_i \leq a_j$, $d_i \leq d_j$ and $p_i = p_j$, this implies that $C_i \leq C_j$. This increases the completion time of job j on the machine. From this, it is clear that set H cannot be expanded by this replacement. Therefore, Algorithm POP indeed gives an optional solution.

6. Conclusion

The parallel machine heuristics that have been developed or outlined in this paper have practical relevance not only in scheduling machines on a detailed, day to day basis. At least two important and interesting practical applications for these heuristics can be proposed. The ability to compare schedule performance with different number of identical parallel machines with consideration of the jobs meeting their time window provides a critical tool in strategic planning decisions involving the purchase or allocation of equipment in high variety, medium-volume manufacturing. Using $Pm|p_j = p|\sum\sum w_jx_{ij}$, $Pm||\sum\sum x_{ij}$ or $Pm|p_j = p|\sum\sum x_{ij}$ scheduling models as a basis, engineering managers may compare average waiting times, approximate inventory levels, make spans and hence production capacities for differing number of machines operating in parallel and for a wide range of product assignments. Informed decisions can then be made based on quantitative comparisons of the costs and benefits associated with each possible configuration. The problem of scheduling to maximize the weighted number of on-time job where the jobs have equal processing times was considered. The time complexity for the greedy heuristics is $O(n \log n)$. It was revealed that algorithm DOP performs slightly well than algorithm WOP. It has been shown by computational study and worst-case analysis given by [6] that the algorithms presented with time complexity $O(n^{m+1} \log n)$ will give results very close to their optimal solutions for the problem to maximize the number of on-time jobs. An optimal solution greedy heuristic is provided for solving the problem of maximizing the number of on-time agreeable jobs with equal processing time. The running (complexity) time is given as $O(n \log n)$ and a proof for feasibility of the algorithm is also presented. Further research should seek to improve on these results by using meta-heuristic methods, find exact solutions for small samples where possible, evolve approximation and pseudo-polynomial algorithms.

References

[1] Adamu, M.O. and Adewunmi, A. (2013), Minimizing the Weighted Number of Tardy Jobs on Multiple Machines: A Review, Asian Pacific Journal of Operations Research. Submitted for publication.

[2] Adamu, M.O. and Adewunmi, A. (2014), A Survey of Single machine Scheduling to Minimize Weighted Number of Tardy Jobs, Journal of Industrial and Management Optimization. 10(3) 219-241.

[3] Baker, K.S. and Scudder, G.D. (1990), Sequencing with Earliness and Tardiness Penalties:A Review. Operations Research, 38, 22-36.

[4] Baptiste,P.;Jouglet,A.; Le Pape.C. and Wim,N.(2004) A Constraint-Based Approach to Minimize the Weighted Number of Late jobs on Parallel Machine. Technical Report, UTC.

[5] Chen, Z.-L. and Lee, C.Y.(2002), Parallel Machine Scheduling with a Common Due Window . European Journal of Operational Research,136, 512-527

[6] Bar-noy,A.;Guha,S.;Naor,J. and Schieber,B.(2001) Approximating the Throughput of Multiple Machines in Real-Time Scheduling. SIAM Journal of Computing,31(2), 331-352.

[7] Cheng,T.C.E. and Chen,Z.-L.(1994), Parallel Machine Scheduling Problems with Earliness and Tardiness Penalties. Journal of the Operational Research Society; 45, 685-695.

[8] Federgruen, A., Mosheiov, G., (1996), Heuristics for Multi-Machine Scheduling Problems with Earliness and Tardiness Costs. Management Science, 42, 1544-1556.

[9] Federgruen, A., Mosheiov, G., (1997), Heuristics for Multi-Machine Min-Max Scheduling Problems with General Earliness and Tardiness Costs. Management Science, 42, 1544-1556.

[10] Hiraishi, K., Levner, E. and Vlacti, M. (2002), Scheduling of Parallel Identical Machines to Minimize the weighted Number of Just-In-Time Jobs. Computers and Operations Research, 29, 841-848.

[11] Kramer, F.-J., Lee, C.-Y. (1994), Due Window Scheduling for Parallel Machines. Mathematical and Computer Modeling, 20, 69-89.

[12] Lann, A. and Mosheiov, G. (1996), Single Machine Scheduling to Minimize the Number of Early and Tardy Jobs. Computers and Operations Research, 30(8), 769-781.

[13] Lann, A. and Mosheiov, G. (2003), A Note on the Maximum Number of On-Time Jobs On Parallel Identical Machines. Computers and Operations Research, 30, 1745-1749.

[14] Li, C.-L., Cheng, T.C.E., (1994),The Parallel Machine Min-Max Weighted Absolute Lateness Scheduling Problem. Naval Research Logistics 41, 33-46.

[15] Li, C.-L., Cheng, T.C.E. and Chen, Z.-L.(1995), Single Machine Scheduling to Minimize the Weighted Number of Early and Tardy Agreeable Jobs. Computers and Operations Research; 22(2), 205 -219.

[16] Rym MHallah and Bulfin, R.L.(2005) Minimizing the Weighted Number Tardy Jobs on Parallel Processors. European Journal of Operational Research,160(2), 471-484.

[17] Yeung, W.K., Oguz, C. and Cheng, T.C.E.(2001), Minimizing Weighted Number of Early and Tardy Jobs with a Common due Window Involving Location Penalty. Annals of Operations Research; 108: 33-54.