COMPARING THE PERFORMANCE OF DIFFERENT META-HEURISTICS FOR UNWEIGHTED PARALLEL MACHINE SCHEDULING

M.O. Adamu[1]* & A. Adewumi[2]

[1]Department of Mathematics
University of Lagos, Nigeria.
madamu@unilag.edu.ng

[2]School of Mathematics, Statistics & Computer Science
University of Kwazulu-Natal, Durban, South Africa
adewumia@ukzn.ac.za

ABSTRACT

This article considers the due window scheduling problem to minimise the number of early and tardy jobs on identical parallel machines. This problem is known to be NP complete and thus finding an optimal solution is unlikely. Three meta-heuristics and their hybrids are proposed and extensive computational experiments are conducted. The purpose of this paper is to compare the performance of these meta-heuristics and their hybrids and to determine the best among them. Detailed comparative tests have also been conducted to analyse the different heuristics with the simulated annealing hybrid giving the best result.

OPSOMMING

Die minimering van vroeë en trae take op identiese parallelle masjien met behulp van die gepaste gleufskeduleringsprobleem word oorweeg. Die probleem is nie-deterministies polinomiese tyd volledig en die vind van 'n optimale oplossing is dus onwaarskynlik. Drie meta-heuristieke en hul hibriede word voorgestel en uitgebreide berekeningseksperimente word uitgevoer. Die doel van hierdie artikel is om die vertoning van hierdie meta-heuristieke en hul hibriede met mekaar te vergelyk om sodoende die beste te identifiseer. Gedetaileerde vergelykende toetse is ook uitgevoer om die verskillende heuristieke te ondersoek; daar is gevind dat die gesimuleerde uitgloei hibried die beste resultaat lewer.

* Corresponding author

1 INTRODUCTION

Due to an increased emphasis on satisfying customers in service provision, due-date related objectives are becoming more important in scheduling. In this article, the objective of minimising the number of early and tardy jobs is considered – a situation where more jobs are completed within their due windows (due-dates). This objective is practical in real-world situations, in order to attain a better customer service rating. To the best of our knowledge, no other article has considered this type of problem. Therefore, the purpose of this study is to compare the performance of several meta-heuristics and to determine a good meta-heuristic to solve this problem. In the identical parallel machine problem, *n* jobs are to be processed on *m* machines, assuming the following facts:

- A job is completed when processed by any of the machines;
- A machine can process only one job at a time;
- Once a job is being processed, it cannot be interrupted;
- All jobs are available from time zero;
- The weights (penalties) of the jobs are equal (unweighted); and
- All processed jobs are completed within their due windows.

During the past few decades, a considerable amount of work has been conducted on scheduling on multiple machines [1] and single machine [2] in order to minimize the number of tardy jobs. When jobs in a schedule have equal important, they are referred to as 'unweighted' jobs, whereas 'weighted' jobs are cases where jobs' contributions are not the same. Garey and Johnson [3] have shown the problem considered in this work is NP-complete; they argue that finding an optimal solution appears unlikely.

Using the three-field notation of Graham et al. [4], the unweighted problem considered in this paper can be represented as $Pm|\ |\Sigma(U_j + V_j),$), where P describes the shop (machine) environment for identical parallel machines, and m describes the number of machines. The space between the bars is for possible constraints on the jobs such as pre-emption, release time, setup, batching precedence, etc. This current work considers job scheduling case on parallel machine with no explicit constraint. The symbols U_j and V_j are binary (0 or 1) that indicates whether a job is scheduled early or tardy respectively, that is, 0 is used when the job is scheduled on time, and 1 is used if it is not.

Scheduling to minimise the (weighted) number of tardy jobs has been considered by Ho and Chang [5], Süer et al. [6], Süer [7], Süer et al. [8], Van der Akker [9], Chen and Powell [10], Liu and Wu [11], and M'Hallah and Bulfin [12]. Sevaux and Thomin [13] addressed the NP-hard problem to minimise the weighted number of late jobs with release time ($P|r_j|\Sigma w_j U_j$). They presented several approaches to the problem, including two MILP formulations for exact resolution, and various heuristics and meta-heuristics to solve large size instances. They compared their results with those of Baptiste et al. [14], who performed better on average. Baptiste et al. [14] used a constraint-based method to explore the solution space and give good results on small problems ($n < 50$).

Dauzère-Pérès and Sevaux [15] determined conditions that must be satisfied by at least one optimal sequence for the problem of minimising the weighted number of late jobs on a single machine. Sevaux and Sörensen [16] proposed a variable neighbourhood search (VNS) algorithm in which a 'tabu' search algorithm was embedded as a local search operator. The approach was compared with an exact method by Baptiste et al. [14]. Li [17] addressed the $P|\text{agreeable due dates}|\Sigma U_j$ problem, where the due dates and release times were assumed to be in agreement. A heuristic algorithm was presented and a dynamic programming lower bounding procedure developed. Hiraishi et al. [18] addressed the non-pre-emptive scheduling of *n* jobs that are completed exactly at their due dates. They showed that this problem is polynomially solvable, even if positive set-up is allowed.

Sung and Vlach [19] showed that when the number of machines is fixed, the weighted problem considered by Hirashi et al. [18] is solvable in polynomial time (exponential in the number of machines), no matter whether the parallel machines are identical, uniform, or unrelated. However, when the number of machines is part of the input, the unrelated parallel machine case of the problem becomes strongly NP-hard. Lann and Mosheiov [20] provided a simple greedy O (n log n) algorithm to solve the problem of Hiraishi et al. [18], thus greatly improving the time complexity. Čepek and Sung [21] considered the same problem of Hiraishi et al. [18], where they corrected the greedy algorithm of Lann and Mosheiov [20], which they argued was wrong; Hiraishi et al. [18] presented a new quadratic time algorithm that solved the problem.

The single-machine scheduling problem to minimize the total job tardiness was considered by Cheng et al. [22] using the ant colony optimization (ACO) meta-heuristic. [23] compared scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. Janiak *et al.* [24] studied the problem of scheduling *n* jobs on *m* identical parallel machines, where for each job a distinct due window is given and the processing time in unit time to minimize the weighted number of early and tardy jobs. They gave an $O(n^5)$ complexity for solving the problem $(Pm|p_j = 1\ |\Sigma w_j(U_j + V_j)$. They also considered a special case with agreeable earliness and tardiness weights where they gave on $O(n^3)$ complexity $(Pm|p_j = 1, r_j$, agreeable ET weights$|\Sigma w_j(U_j + V_j))$. Adamu and Abass [25] proposed four greedy heuristics for the $Pm||\Sigma w_j\ (U_j + V_j)$ problem, and performed extensive computational experiments. Adamu and Adewumi [26] proposed some meta-heuristics and their hybrids to solve the problem considered by Adamu and Abass [25]; they found them performing better.

2 PROBLEM FORMULATION

Let there be an independent set, $N = \{1,2, . . . , n\}$ of jobs that are to be scheduled on *m* parallel identical machines that are immediately available from time zero, each having an interval rather than a point in time, which is called the due window of the job. The left end and the right end of the window are called the earliest due date (i.e., the instant at which a job becomes available for delivery) and the latest due date (i.e., the instant by which processing or delivery of a job must be completed) respectively. There is no penalty when a job is completed within the due window, but for earliness or tardiness, a penalty is incurred when a job is completed before the earliest due date or after the latest due date. Each job $j\epsilon$ N has a processing time p_j, earliest due date a_j, and latest due date d_j; it is assumed that there are no pre-emptions and only one job may be processed on a given machine at any given time. For any schedule S, let t_{ij} and $C_{ij}(S) = t_{ij} + p_j$ represent the 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 $a_j \leq C_{ij}(S) \leq d_j$. For any job k(ij), where k(ij) stands for the jth processed job on machine i, the number of early and tardy jobs [11] can be calculated by

$$U_{k(ij)} = \mathrm{int}\left\{\frac{1}{2} sign[C_{k(ij)}(S) - p_j] + \frac{1}{2}\right\} \tag{1}$$

where we define that

$$sign[C_{k(ij)}(S) - p_j] = \begin{cases} 1, & if\ a_j > C_{k(ij)}(S) \quad or \quad C_{k(ij)}(S) > d_j \\ -1, & a_j \leq C_{k(ij)}(S) \leq d_j \end{cases}$$

and that *int* is the operation for making an integer. Obviously,

$$U_{k(ij)} = \begin{cases} 1, & if\ a_j > C_{k(ij)}(S) \quad or \quad C_{k(ij)}(S) > d_j \\ 0, & a_j \leq C_{k(ij)}(S) \leq d_j \end{cases}$$

Therefore, the problem of scheduling to minimise the number of tardy jobs on identical parallel machines can be formulated as

$$W = \sum_{i=1}^{m} \sum_{j=1}^{n} U_{k(ij)} = \sum_{i=1}^{m} \sum_{j=1}^{n} \operatorname{int}\left\{ \frac{1}{2} sign[C_{k(ij)}(S) - p_j] + \frac{1}{2} \right\} \quad (2)$$

$$\text{Min W} = \sum_{i=1}^{m} \sum_{j=1}^{n} U_{k(ij)} = \min \sum_{i=1}^{m} \sum_{j=1}^{n} \operatorname{int}\left\{ \frac{1}{2} sign[C_{k(ij)}(S) - p_j] + \frac{1}{2} \right\} \quad (3)$$

3 HEURISTICS AND META-HEURISTICS

3.1 Greedy heuristic

Adamu and Abass [25] proposed four greedy heuristics that attempt to provide near-optimal solutions to the parallel machine scheduling problem. In this paper, the fourth heuristic (DO2) is used. It entails sorting the jobs according to their latest due date (i.e., latest due time - processing time) and the ties broken by the highest inverse ratio of processing time (i.e., 1 / processing time).

The results of these greedy heuristics are encouraging; however, whether using meta-heuristics and their hybrids can achieve better results will be investigated further. Similar codes used in Adamu and Adewumi [26] will be used to solve the problem of minimising the number of early and tardy jobs on parallel machines.

3.2 Genetic algorithm

Genetic algorithm (GA) is one of the best known meta-heuristics for solving optimisation problems [27,28,29]. The technique is loosely based on evolution in nature, and uses strategies such as survival of the fittest, genetic crossover, and mutation. It has proved very useful in handling many discrete optimisation problems [30-32]; hence the decision to test its performance on the current problem and to compare it with the performance of greedy heuristics. An overview of the GA implemented for the current problem is presented as follows:

1. Problem representation: Deciding on a suitable representation is one of the most important aspects of a GA. An integer string representation is adopted in this study. Each job is fixed to a gene in the chromosome; this implies that the chromosome has length n (where n is the number of jobs). Each position in the chromosome therefore represents a job. Each gene contains two integer numbers representing the number of the machine to which the job will be assigned, and an order, respectively. The order number is a value between 1 and n, representing the order in which jobs assigned to the same machine will be executed. Genetic operators are then applied to both the machine number and the order. Figure 1 presents a typical representation for a case of five jobs (n = 5) to be processed on three machines. In this case, Jobs 1 and 4 should both be processed on Machine 2, but with Job 4 having priority over Job 1.
2.

Job 1	Job 2	Job 3	Job 4	Job 5
2 (2)	3 (1)	1 (1)	2 (1)	1 (2)

Figure 1: A sample representation for GA

3. Algorithm: The pseudo code of the GA implemented is presented on the next page:

```
Generate a population of randomly initialised individuals.
iterations← 0
repeat
fori = 1 →popSizedo
     Perform crosssover with probability crossoverRate.
end for
fori = 1 →popSizedo
forj = 1 →numJobsdo
           Mutate machine with probability mutationRate.
end for
end for
fori = 1 →popSizedo
forj = 1 →numJobsdo
          Mutate order with probability mutationRate.
end for
end for
   Use selection to form a new population of individuals.
iterations←iterations + 1
untiliterations ≥ numIterations
   Return the fitness of the best individual.
```

4. Fitness function: The fitness function calculates the total number of jobs that could not be assigned to any of the machines so that they would finish between the earliest due date and the latest due date. For each machine, jobs that are assigned to it are placed in a priority queue (based on their respective order). Each job is then removed from the queue and placed on the machine. If the job were to finish early it would be scheduled to begin later (at earliest due date - processing time) in order to avoid the earliness penalty. However, if the job were to finish past the end time, it would not be scheduled at all; instead the job would be added to the total penalty (fitness). Since the fitness calculates the penalty incurred, it then implies that the lower the fitness function, the better the performance of the algorithms or the result generated.
5. Genetic operators: The choice of basic operators of selection, crossover, and mutation influences the behaviours of the GA [29]. In the current study, the operators are applied separately to the machine and the order number. For both, the tournament selection was used to select the two parents for crossover. For the machine, the 1-point crossover and conventional mutation was adopted. The mutation operator chooses a random machine from 0 to m-1 inclusive, and also changes the machine number randomly. Since the order number (the order in which jobs are to be processed on a machine) is permutation-based, swap mutation for the execution order was used. This involves randomly selecting two jobs to be processed on the same machine, and swapping the order in which they were originally to be processed. However, since there were no guarantees that these operators would allow for the best performance, further experimentation with variations of these operators was performed. More details will be given in a later section. Detail and basic descriptions of these operators can be found in Goldberg [27] and Mitchell [29].

3.3 Particle swarm optimisation

Particle swarm optimisation (PSO) is regarded as another efficient optimisation technique [33,34], hence its selection for the current parallel machine scheduling problem. It is a population-based technique derived from the flocking behaviour of birds, and relies on both the particle's best position found so far and the entire population's best position, in order to get out of local optimums to approach the global optimum. PSO is appropriate to use for parallel machine scheduling since not much is known about the solution landscape. A description of the PSO, as implemented for this study, is presented on the next page:

1. Problem representation: The PSO algorithm requires that a representation of the solution (or encoding of the solution) is chosen. Each particle will be an instance of the chosen representation. A complication is that PSO works in the continuous space,

whereas the scheduling problem is a discrete problem. Thus a method is needed to convert from the continuous space to the discrete space. The representation is as follows:

- Each particle is represented by a pair of two digits.
- The first digit is a number in the range [0,m), where m represents the number of machines on which the particle is scheduled. Note that 0 is inclusive and m is exclusive in the range. The number is simply truncated to convert to the discrete space.
- Similarly, the second digit, also in the range [0,m) represents the order of scheduling of the particle relative to other particles on the same machine; a lower number indicates that the job will be scheduled before jobs with higher numbers.

2. Algorithm: Below is a basic pseudo code of the PSO that was used.

```
for i = 1 →PopSize do
    Construct particle with randomly initialised machine number and order.
end for
repeat
    pbest← 2^31
    for i = 1 →PopSize do
        fitness←calcfitness(pop[i])
        if fitness <pbest then
            pbest← fitness
        end if
        if fitness< fitness(gbest) then
            gbest← pop[i]
        end if
    end for
    for i = 1 →PopSize do
        v[i + 1] ←wv[i] + r1c1(pbest – pos[i]) + r2c2(gbest – pos[i])
        pos[i + 1] ←pos[i] + v[i] (ensuring to clamp the position within range)
    end for
    iterations← iterations + 1
until iterations ≥ numIterations
```

3. Fitness function: Finally, a method is needed to convert the encoding into a valid schedule (performed when calculating the fitness). This is performed by separating the jobs into groups based on the machine to which they are assigned. Within a group, the jobs are sorted by their order parameter and organised into a queue. The schedule for a particular machine is then formed by removing jobs from the queue and scheduling them as early as possible without breaking the earliness constraint. The fitness of a solution is computed as the penalty incurred; that is, the total number of jobs that cannot be scheduled, which ideally should be as small as possible.

3.4 Simulated annealing

Simulated annealing (SA) has also been shown to be highly effective for discrete problems [35,36], hence its selection for the current problem. SA is based on real-life annealing, where the heating of metals allows for atoms to move from their initial position, and cooling allows for the atoms to settle in new optimal positions. SA is not a population-based heuristic; thus only one solution is kept at any one stage. Since SA should result in fewer operations being performed than by a population-based technique, execution times may be quicker. It is for this reason that SA was chosen for investigation.

It should also be noted that SA will in all likelihood achieve better results than a simple hill-climbing technique. This is because SA can take downward steps (i.e., accept worse solutions) in order to obtain greater exploration. Thus, it is less likely to become stuck in a local minimum (a very real problem, given the complex solution space).

1. Problem representation: The representation is remarkably similar to that used in the GA. A solution consists of n elements (where n is the number of jobs). Each element has a specific job, as well as the machine to which it will be assigned and the order of assignment. Perhaps the major difference between SA and GA is that the GA has a population of solutions (chromosomes), whereas SA focuses on a single solution.
2. Algorithm: This is the basic algorithm used in the SA technique:

```
Generate a randomly initialised solution sol.
repeat
   fori→ 10 do
         find a neighbor of sol and call it solPrime.
         if fitness(solPrime) < fitness(sol) then
               sol←solPrime
         else
               if e^(fitness(sol) - fitness(solPrime)) > rand(0..1) then
                     sol←solPrime
               end if
         end if
   end for
   temp← temp*beta
until temp ≤ templ
```

3. Fitness function: Since the solution is represented in virtually the same manner as a chromosome in the GA and a particle in PSO, the fitness function is calculated in the same way. That is, jobs pertaining to a particular machine are placed in a priority queue before being assigned to the machine. Those that cannot be assigned contribute towards the penalty.
4. Operators: Although SA does not really have operators (in the sense of a GA having genetic operators), the SA algorithm does have to select a neighbour. The particular neighbour selection strategy that is used updates only a single element of the solution. The element is given a new randomly-chosen machine and a new order (done by swapping with the order of another randomly chosen element). By allowing for a high level of randomness when selecting the neighbour, it is ensured that good exploration will be achieved and that a local best is not found too early.

4 COMPUTATIONAL ANALYSIS AND RESULTS

4.1 Data generation

The program was written in Java using Eclipse. It actually consists of a number of programs, each one implementing a different type of solution. The output of each of these programs gives the final fitness after the algorithm has been performed, and the time in milliseconds that the algorithm took to run.

The heuristics were tested on problems generated with 100, 200, 300, and 400 jobs, similar to Adamu and Abass [25], Adamu and Adewumi [26], Ho and Chang [5], Baptiste et al. [14], and M'Hallah and Bulfin [12]. The number of machines was set at levels of 2, 5, 10, 15, and 20. For each job j, an integer processing time p_j was randomly generated in the interval (1, 99). Two parameters, k1 and k2 (levels of traffic congestion ratio) were taken from the set {1, 5, 10, 20}. For the data to depend on the number of jobs *n*, the integer's earliest due date (a_j) was randomly generated in the interval (0, n / (m * k1)), and the integer's latest due date (d_j) was randomly generated in the interval ($a_j + p_j$, $a_j + p_j$+ (2 * n * p) / (m * k2)).

For each combination of n, k1, and k2, ten instances were generated; i.e., for each value of n, 160 instances were generated for 8,000 problems of 50 replications. The meta-heuristics were implemented on a Pentium Dual 1.86 GHz, 782 MHz, and 1.99 GB of RAM. The following meta-heuristics were analysed: GA, PSO, SA, GA Hybrid, PSO Hybrid, and SA Hybrid.

4.2 Improvements

The use of genetic operators of crossover and mutation for both exploration and exploitation of solution space gives the GA a unique advantage over some other meta-heuristics. In this work, the original GA that was tested used 1-point crossover, random mutation for machines, swap mutation for order, and tournament selection. Other combinations of operators were also tested to check which ones improved the performance of the algorithm. This initial experiment showed that roulette-wheel selection, uniform crossover, and insert mutation (for order) are better for the problem at hand. A user can choose any combination of these operators to use to run the algorithm. More information on the optimal combination of genetic operators will be mentioned in Section 4.4.

4.3 Greedy hybrids

This work seeks to improve the performance of the underlying meta-heuristics (GA, PSO, and SA) by potentially hybridising them with some features of the greedy heuristic proposed by Adamu and Abass [25]. The key feature of the greedy heuristics in that work essentially lies in the order in which jobs were assigned to machines. So the mechanisms of ordering in DO2 [25] are incorporated in the meta-heuristics (GA, PSO, SA).

To implement the hybrid in the three meta-heuristics, the order field was removed from Gene, Dimension, and Element respectively. Also, any code in Chromosome, Particle, and Solution, which dealt with the order (e.g., swap mutation in Chromosome), was removed.

4.4 Parameter settings

For each solutions strategy, there are a number of different parameters that affect the performance of the algorithm, such as population size, mutation rate, initial temperature, etc. These parameters were determined experimentally by running the algorithms on a subset of all the testing data, in order to determine the optimal parameters. This involved experimenting with the full range of each parameter and recording and tabulating the results achieved. The combination of parameters that gave the best performance was selected as the optimal combination. After this initial experiment, the optimal parameters for the GA were found to be as follows:

- A population size of 10.
- Random mutation (for machines) used at a rate of 0.01.
- Swap mutation (for order) used at a rate of 0.01.
- Uniform crossover at a rate of 0.5.
- Tournament selection with a k set at 40 per cent of the population size.
- The number of iterations of the algorithm was set at 2,000.

The best performance with a population size of 10 for this initial experience is likely due to the inherently parallel nature of the GA; hence lower population size might give better runtime and fitness for problems of this nature where time is a critical factor.

Further to the above parameters, the GA hybrid achieved best results when hybridised with the DO2 greedy heuristic.

The optimal parameters for PSO are:

- A population size of 50.
- A w (momentum value) of 0.3.
- A c1 of 2.
- A c2 of 2.
- The number of iterations of the algorithm was set at 2,000.

Further to the above parameters, the particle swarm optimisation hybrid achieved best results when hybridised with the DO2 greedy heuristic.

The optimal parameters for SA are:

- An initial temperature of 25.
- A final temperature of 0.01.
- A geometrical decreasing factor (beta) of 0.999.

Further to the above parameters, the SA hybrid achieved best results when hybridised with the DO2 greedy heuristic.

5 DISCUSSION OF RESULTS

In this section, the results of the algorithms are shown, including the results of the hybridisations. In the four tables shown in Table 1 (a and b), each cell consists of two numbers. The top number is the weight of the schedule that is produced, averaged over 50 runs. The bottom number is the average time in milliseconds that the algorithm takes to complete.

Also included are four charts, each for the performance of the meta-heuristics in relation to the penalty (see Figure 2) and time (see Figure 3) for n= 100, 200, 300, and 400. Figure 2 compares the relative performance (penalty) of each of the six algorithms with the number of machines used. Again, four charts are given to show the computational times of the meta-heuristics for various values of N.

It should be clear from both Table 1 and the charts (Figures 2 and 3) that the SA Hybrid (SAH) outperformed the other meta-heuristics in almost all points. Its average performance time is 0.5 seconds. It was observed that the various hybrids performed better than their meta-heuristic without hybridisation. It further proves the effectiveness of hybridisation on the meta-heuristics.

The GA performed worse than other meta-heuristics in all of the categories considered for all N jobs and M machines. The GA time is on the average less than two seconds, far slower than the SAH – notably because it keeps track of a population of individual solutions. Results show it to being in the region of four times slower than SAH.

The GA that is hybridised with DO2 (GAH) achieves better results (see Table 1 and Figure 2) on all of the test cases than the simple GA. In all cases considered, the GAH outperforms the ordinary GA, and as the value of N increases, the performance rate of the GAH over the GA widens. For larger values of N, the performance of the GAH is almost equivalent to, if not better, than the SAH. The GAH takes on average about 2.55 seconds. The GAH would be ideal for larger values of N where an optimal solution is not readily feasible. It is observed that, on average, the GA takes less time to run than the GAH.

The PSO and the hybrid PSO (PSOH) produce a lower number of early and tardy jobs than the GA. Furthermore, they are far slower than all the meta-heuristics considered (over 33.1 times slower for PSO and 22.9 for PSOH in relation to the SA). This is understandable since PSO is a population-based algorithm, so a lot of work is done at each step. Hybridising PSO with the DO2 greedy heuristic produces results that are better than PSO for all cases. The PSOH is also about 1.45 times faster than PSO. While it is observed for all other meta-heuristics that, as the number of machines increases, their corresponding penalties reduce, the reverse is the case for PSO and PSOH.

The results for SA are far better on average than those for GA, PSO, AND PSOH, in performance of both penalty and time (see Tables 1 and 2 and Figures 2 and 3). On average, SA takes 0.45 seconds to run. However, it is about 4.41, 5.72, 33.1, and 22.9 times

Table 1a: Performance of meta-heuristics for different values of N (100 and 200)

		m=2			m=5			m=10			m=15			m=20		
		MIN	AVE	MAX	MIN	AVE	MAX	MIN	AVE	MAX	MIN	AVE	MAX	MIN	AVE	MAX
n=100	GA	113	119.38	126	105	110.84	116	96	103.14	109	92	98.68	106	88	93.26	101
		1937	2006.86	2122	1891	1961.52	2266	1828	1909.34	1985	1890	1928.72	2000	1922	1987.5	2094
	GAH	63	76.04	85	59	69.52	59	52	64.02	76	51	59.96	67	41	51.8	62
		2781	2904.34	3063	2641	2770.62	2938	2484	2628.76	2782	2485	61	2750	2531	2648.48	2813
	PSO	80	90.42	104	80	93.44	103	83	92.08	99	82	90.54	95	80	88	95
		13734	14069.74	14453	13359	13795.38	14250	13406	13703.4	14141	13468	14622.4	20687	14187	15614.72	18344
	PSOH	65	74.96	85	68	82.88	90	64	85.5	95	74	83.82	93	72	80.48	87
		10093	10771.24	11765	9625	10251.9	11172	9375	9993.76	10922	9234	10029.34	10687	9531	10251.54	11109
	SA	74	81.68	91	69	75.84	81	59	67.46	76	54	62.06	69	47	54.36	64
		422	482.36	609	391	439.5	516	375	425.06	485	390	435.64	516	406	459.08	531
	SAH	67	78.58	93	56	66.7	75	47	58.32	71	41	53.54	61	35	46.28	57
		516	564.34	641	484	506.38	562	437	470.62	500	437	468.14	515	453	476.62	532

		m=2			m=5			m=10			m=15			m=20		
		MIN	AVE	MAX	MIN	AVE	MAX	MIN	AVE	MAX	MIN	AVE	MAX	MIN	AVE	MAX
n=200	GA	105	112.74	119	94	100.46	111	78	89.28	96	75	82.8	92	72	77.14	85
		1922	2014.68	2640	1844	1948.48	2453	1844	1915.86	2297	1875	1922.2	1985	1890	1986.2	2625
	GAH	32	43	53	25	38.1	45	22	35.68	43	22	31.94	39	18	27.86	36
		2797	2889.16	3031	2625	2751.26	2938	2484	2591.58	2719	2454	2579.58	2734	2484	2606.54	2766
	PSO	57	69.68	82	54	71.4	87	65	73.6	83	62	72.36	81	62	72.02	79
		14187	15639.1	17656	13953	15253.88	16884	13750	15079.68	16750	13875	15277.78	17157	14218	15587.76	17687
	PSOH	33	42.38	54	31	52.34	63	46	57.24	65	49	57.46	65	48	56.64	67
		9953	10685.86	11672	9500	10410.12	20562	9359	10099.74	18812	9343	10094.66	18422	9484	10341.26	19719
	SA	49	57.2	66	36	50.02	58	31	43.64	51	31	38.02	45	24	33.68	41
		421	470	532	406	436.52	485	375	424.3	500	390	434.74	500	406	457.3	547
	SAH	32	44.12	52	24	35.96	49	18	31.92	42	17	26.88	34	15	23.02	34
		531	553.74	610	468	500.9	547	437	464.44	515	437	466.26	437	453	479.92	547

Table 1b: Performance of meta-heuristics for different values of N (300 and 400)

		m=2			m=5			m=10			m=15			m=20		
		MIN	AVE	MAX	MIN	AVE	MAX	MIN	AVE	MAX	MIN	AVE	MAX	MIN	AVE	MAX
	GA	96	109.28	127	85	94.9	105	71	82.52	95	67	74.3	83	60	68.26	76
		1938	2031.28	3266	1844	1945.92	2422	1844	1918.78	2391	1828	1962.22	2766	1891	2003.44	2734
	GAH	13	21.06	29	12	18.72	30	8	15.94	24	8	14.94	23	3	11.48	21
		2765	2868.62	3047	2609	2745.68	2938	2500	2579.72	2656	2500	2561.46	2687	2485	2592.92	2688
	PSO	44	57.7	69	42	57.38	75	50	63.2	72	41	61.44	70	56	61.96	72
n=300		14093	15610.6	18469	13890	15334.38	17375	13765	15153.82	17062	13906	15309.38	17969	14218	15566.66	18500
	PSOH	13	21	32	15	30.74	47	18	35.98	45	12	38.34	51	10	37.78	48
		10125	10898.8	21672	9610	10250.84	14594	9266	9850.64	10750	9094	9849.82	10610	9078	10079.08	11203
	SA	32	42.26	56	28	34.84	44	19	28.36	36	18	24.88	35	13	20.1	29
		422	479.12	563	391	448.86	547	375	432.72	516	390	435.64	485	406	451.26	516
	SAH	14	21.6	32	13	18.04	29	7	13.86	21	4	11.96	19	2	8.4	16
		516	551.54	609	468	501.16	578	437	459.1	500	437	461.88	500	437	463.8	516

		m=2			m=5			m=10			m=15			m=20		
		MIN	AVE	MAX	MIN	AVE	MAX	MIN	AVE	MAX	MIN	AVE	MAX	MIN	AVE	MAX
	GA	94	106.38	118	81	93.22	107	63	79.1	93	62	71.3	83	57	64.7	74
		1953	2021.86	2312	1844	1974.82	3141	1875	1914.62	169	1844	1974.86	3016	1938	1993.8	2625
	GAH	3	8.9	15	2	8.32	15	0	6.6	18	0	4.84	12	0	3.16	13
		2766	2861	2938	2640	2716.32	2782	2500	2566.86	2672	2484	2560.86	2860	2515	2576.74	2641
	PSO	42	54.72	66	37	52.88	71	36	56.24	65	30	54.64	62	49	54.62	61
n=400		14157	15572.48	17468	13110	14119.14	16953	12906	13153.44	13516	12953	13333.5	13641	13265	13559.38	13906
	PSOH	3	9.46	16	10	18.26	29	11	23.76	37	12	27.26	39	1	25.84	40
		9921	10640.96	11672	9437	10122.72	10922	9297	9868.22	10579	9016	9887.2	11188	9437	10066.84	13297
	SA	23	43.2	42	18	25.88	33	14	20.28	33	8	15.92	24	6	12.58	22
		422	468.78	532	406	442.44	500	375	417.8	484	390	437.8	390	406	446.32	515
	SAH	3	9.32	16	1	8.22	16	0	5.74	15	0	3.64	10	0	2	9
		515	549.7	609	468	496.56	532	437	460.06	516	437	466.54	532	437	472.78	547

quicker than the GA, GAH, PSO, and PSOH, respectively. SA has the best overall time performance of all the meta-heuristics.

Table 2: Homogeneous subsets

PENALTY

Scheffe[a]

HEURISTICS	N	Subset for alpha = 0.05		
		1	2	3
SAH	20	28.4050		
GAH	20	30.5940		
SA	20	41.6130		
PSOH	20	47.1060		
PSO	20		69.4160	
GA	20			91.5840
Sig.		.147	1.000	1.000

Means for groups in homogeneous subsets are displayed.

a. Uses harmonic mean sample size = 20.000.

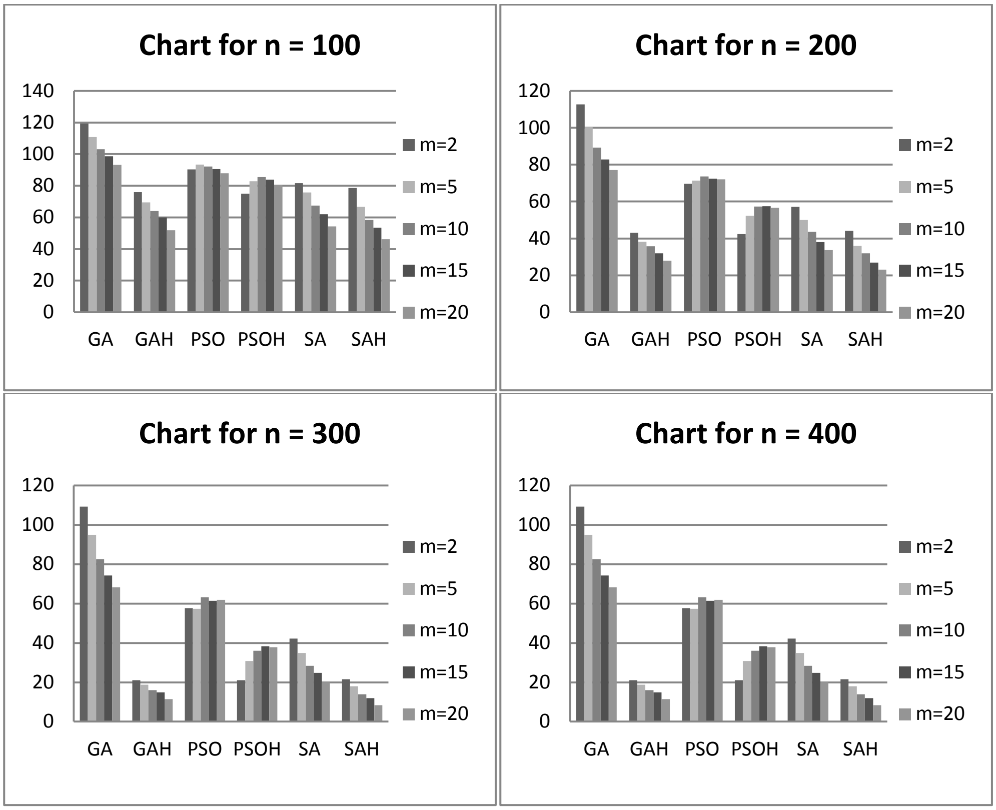

Figure 2: Meta-heuristics performance in relation to penalty

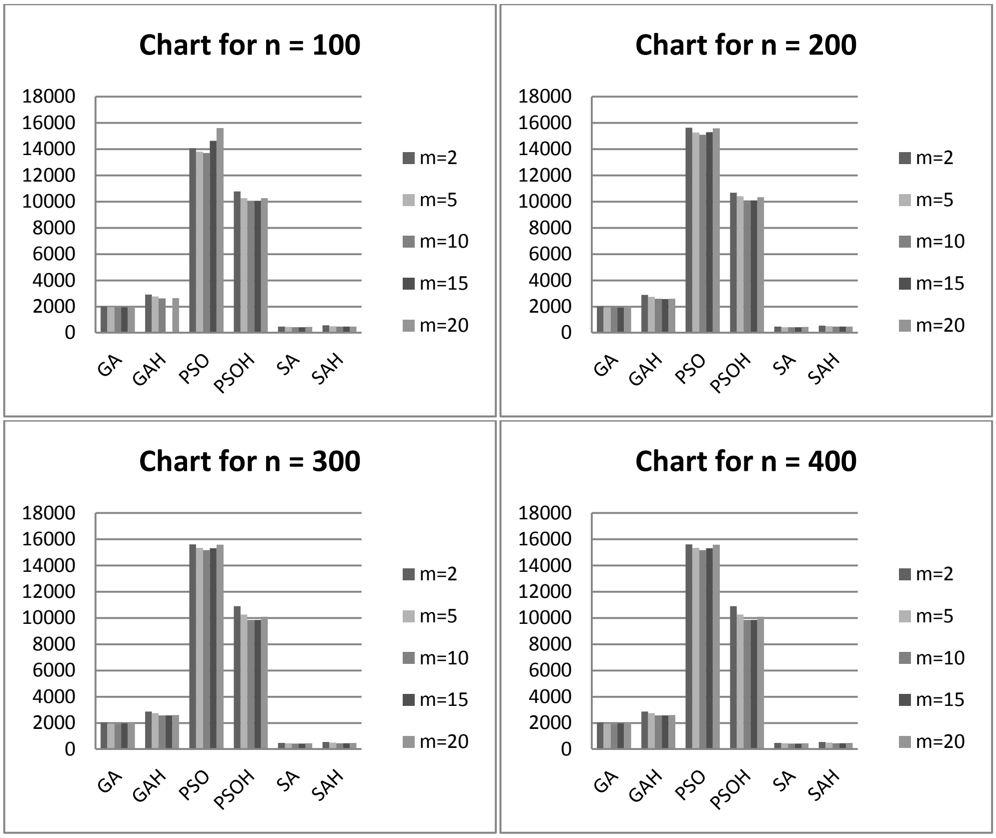

Figure 3: Performance time of the meta-heuristics

Hybridising SA with the DO2 greedy heuristic (SAH) produces results that are slightly better than the SA solution for all cases considered. It produces the overall best results among the meta-heuristics in terms of performance in relation to penalty. The average timing is a little less than 0.5 seconds.

Due to the equality of their variances, subsets of homogeneous groups are displayed in Table 2 using Scheffé's method. The Scheffé test is designed to allow all possible linear combinations of group means to be tested. That is, pair-wise multiple comparisons are done to determine which means differ. Three groups are obtained: Group 1 - SAH, GAH, SA, and PSOH; Group 2 - PSO; and Group 3 – GA. These groups are arranged in decreasing order of their effectiveness. The worst among them is the GA.

6 CONCLUSION

We considered an identical machine problem with the objective of minimising the number of early and tardy jobs. The purpose of this study was to compare the performance of several meta-heuristics and determine a good meta-heuristics to solve this problem. Six meta-heuristics that incorporate a fast greedy heuristic were suggested because they gave promising results. Computational experiments and statistical analyses were performed to compare these algorithms. The SAH was the best among the various meta-heuristics. This research can be extended in several directions. First, these results could be compared with an optimal solution. Second, the environment with uniform or unrelated parallel machines could be a practical extension.

ACKNOWLEDGEMENTS

The authors are grateful to the referee(s) for their useful comments that improved the quality of this paper.

REFERENCES

[1] **Adamu, M.O. & Adewumi, A.O.** 2015. Minimizing the weighted number of tardy jobs on multiple machines: A review. *Asian Pacific Journal of Operations Research*. In press.

[2] **Adamu, M.O. & Adewumi, A.O**. 2014. Single machine review to minimize weighted number of tardy jobs. *Journal of Industrial and Management Optimization,* 10(1), pp. 219–241.

[3] **Garey, M.R. & Johnson, D.S.** 1979. *Computers and intractability: A guide to the theory of NP completeness*. San Francisco: Freeman.

[4] **Graham, R.L., Lawler, E.L., Lenstra, T.K. & RinnooyKan, A.H.G.** 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5, pp. 287-326.

[5] **Ho, J.C. & Chang Y.L.** 1995. Minimizing the number of tardy jobs for *m* parallel machines. *European Journal of Operational Research*, 84, pp. 343-355.

[6] **Süer, G.A., Baez, E. & Czajkiewicz, Z.** 1993. Minimizing the number of tardy jobs in identical machine scheduling. *Computers & Industrial Engineering*, 25(1-4), pp. 243-246.

[7] **Süer, G.A**. 1997. Minimizing the number of tardy jobs in multi-period cell loading problems. *Computers and Industrial Engineering,* 33(3,4), pp. 721-724.

[8] **Süer, G.A., Pico, F. & Santiago, A**. 1997. Identical machine scheduling to minimize the number of tardy jobs when lost-splitting is allowed. *Computers and Industrial Engineering*, 33 (1,2), pp. 271-280.

[9] **Van Den Akker, J.M., Hoogeveen, J.A. & Van De Velde, S.L.** 1999. Parallel machine scheduling by column generation. *Operations Research*, 47(6), pp. 862-872.

[10] **Chen, Z. & Powel, W.B.** 1999. Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing*, 11(1), pp. 78-94.

[11] **Liu, M. & Wu, C.** 2003. Scheduling algorithm based on evolutionary computing in identical parallel machine production line. *Robotics and Computer Integrated Manufacturing*, 19, pp. 401-407.

[12] **M'Hallah, R. & Bulfin, R.L.** 2005. Minimizing the weighted number of tardy jobs on parallel processors. *European Journal of Operational Research*, 160, pp. 471-484.

[13] **Sevaux, M. & Thomin, P.** 2001. Heuristics and metaheuristics for a parallel machine scheduling problem: A computational evaluation. *Proceedings of the 4th Metaheuristics International Conference*, pp. 411-415.

[14] **Baptiste, P., Jouglet, A., Pape, C.L. & Nuijten, W.** 2000. *A constraint based approach to minimize the weighted number of late jobs on parallel machines.* Technical Report 2000/228, UMR, CNRS 6599, Heudiasyc, France.

[15] **Dauzère-Pérès, S. & Sevaux, M.** 1999. *Using lagrangean relation to minimize the (weighted) number of late jobs on a single machine*. National Contribution IFORS 1999, Beijing, P.R. of China (Technical Report 99/8 Ecole des Minesdes Nantes, France).

[16] **Sevaux, M. & Sörensen, K.** 2005. *VNS/TS for a parallel machine scheduling problem*. MEC-VNS: 18th Mini Euro Conference on VNS.

[17] **Li., C.L**. 1995. A heuristic for parallel machine scheduling with agreeable due dates to minimize the number of late jobs. *Computers and Operations Research*, 22(3), pp. 277-283.

[18] **Hiraishi, K., Levner, E. & Vlach, M.** 2002. Scheduling of parallel identical machines to maximize the weighted number of just-in-time jobs. *Computers and Operations Research,* 29, pp. 841-848.

[19] **Sung, S.C. & Vlach, M.** 2001. *Just-in-time scheduling on parallel machines.* The European Operational Research Conference, Rotterdam, Netherlands.

[20] **Lann, A. & Mosheiov, G**. 2003. A note on the maximum number of on-time jobs on parallel identical machines. *Computers and Operations Research*, 30, pp. 1745-1749.

[21] **Čepek, O. & Sung, S.C**. 2005. A quadratic time algorithm to maximize the number of just-in-time jobs on identical parallel machines. *Computers and Operational Research*, 32, pp. 3265-3271.

[22] **Cheng, T., Lazarev, A. & Gafarov, E.** 2009. A hybrid algorithm for the single-machine total tardiness problem. *Computers & Operations Research,* 36(2), pp. 308–315.

[23] **Jungwattanakit, J., Reodecha, M., Chaovalitwongse, P. & Werner, F.** 2009. A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. *Computers & Operations Research,* 36(2), pp. 358–378.

[24] **Janiak, A., Janiak, W.A. & 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), pp. 209-215.

[25] **Adamu, M. & Abass, O.** 2010. Parallel machine scheduling to maximize the weighted number of just-in-time jobs. *Journal of Applied Science and Technology*, 15(1-2), pp. 27–34.
[26] **Adamu, M.O. & Adewumi, A.O.** 2012. Metaheuristics for scheduling on parallel machines to minimize the weighted number of early and tardy jobs. *International Journal of Physical Sciences,* 7(10), pp. 1641-1652.
[27] **Goldberg, D.E**. 1989. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.
[28] **Holland, J.H.** 1975. *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.
[29] **Mitchell, M.** 1998. *An introduction to genetic algorithms*. The MIT Press.
[30] **Adewumi, A.O. & Ali, M**. 2010. A multi-level genetic algorithm for a multi-stage space allocation problem. *Mathematical and Computer Modeling,* 51(1-2), pp. 109-126.
[31] **Adewumi, A.O., Sawyerr, B.A. & Ali, M.M.** 2009. A heuristic solution to the university timetabling problem. *Engineering Computations*, 26(8), pp. 972-984.
[32] **Naso, D., Surico, M., Turchiano, B. & Kaymak, U.** 2007. Genetic algorithms for supply-chain scheduling: A case study in the distribution of ready-mixed concrete. *European Journal of Operational Research,* 177, pp. 2069–2099.
[33] **Arasomwan, A. M. & Adewumi, A. O**.,2013. On the Performance of Linear Decreasing Inertia Weight Particle Swarm Optimization for Global Optimization, The Scientific World Journal, 2013, Article ID 860289, 12 pages. doi:10.1155/2013/860289.
[34] **Poli, R., Kennedy, J. & Blackwell, T.** 2007. Particle swarm optimization: An overview. *Swarm Intelligence*, 1, pp. 33–57.
[35] **Kirkpatrick, S., Gellat, C. & Vecchi, M.** 1983. Optimization by simulated annealing. *Science,* 220, pp. 671–680.
[36] **Chetty, S. & Adewumi, A.O.**, 2013. Three New Stochastic Local Search Metaheuristics for the Annual Crop Planning Problem Based on a New Irrigation Scheme. Journal of Applied Mathematics, 2013, Article ID 158538, 14 pages. http://dx.doi.org/10.1155/2013/158538.