5th African International Conference on Industrial Engineering and Operations Management, Johannesburg/Pretoria, South Africa, April 23 - 25, 2024
Publisher: IEOM Society International, USA
Published: April 23, 2024 DOI: 10.46254/AF05.20240071

Reentrant Permutation Flow Shop Scheduling with a Deteriorating Schedule

Makgoba Matsebe and Olufemi Adetunji
Department of Industrial and Systems Engineering,
University of Pretoria,
Pretoria, South Africa
makgobamatsebe@gmail.com, olufemi.adetunji@up.ac.za

Abstract

The classic flow shop problem assumes that jobs make only single passes through the processing machines and that the processing times are not affected by the length of the delay before jobs are processed. These assumptions are being relaxed in recent papers considering re-entrance problems and those with schedule deterioration. In this paper, these two assumptions are relaxed together, and a model of a re-entrance flow shop with a deteriorating schedule is considered. A linear programming formulation of the problem is first presented. Three solution heuristics are considered under different deterioration scenarios. It was observed that both the Nawaz Enscor and Ham (NEH) algorithm and Genetic Algorithm (GA) performed much better than the Campbell Dudek and Smith (CDS) algorithm. Overall, when considering both the quality of the solution and computational time together, the NEH algorithm seems to have performed much better than the others as the size of problems increases. This model would find useful applications in many metallurgical and manufacturing processes where such problems are usually encountered.

Keywords

Reentrant flow shop, deteriorating schedule, CDS algorithm, NEH Algorithm, Genetic Algorithm, makespan minimization

1. Introduction

The classical flow shop is characterized by jobs that are processed on each machine within the system only once. Chen %3et al. (2008), however, stated that in reality, this assumption is sometimes violated. The allowance of jobs to return to one or more machines for further processing is termed reentrance. For decades, the reentrant flow shop problem has attracted the attention of many researchers, and this can be attributed to the drive to reduce operating costs and increase profits by manufacturing facilities. The reentrant property is observed in processes such as semiconductor manufacturing, wherein components need to be processed more than once on the same machine before the final product is achieved, and cold drawing operations in steel tube manufacturing. Reentrance can also be used in tool machining shops, where a particular tool might require a polishing stage or heat treatment in between the machining stages to achieve the final tooling finish (Graves et al. 1983).

Deterioration of jobs, on the other hand, can be observed in processes where job processing time is increased as a result of queuing due to the unavailability of machines, deterioration in the state of the job, or deterioration caused by fatigue or tiredness of machine operators. In some instances jobs require a preparation step before processing; the cold-drawn tube manufacturing is a typical example. The input material may lose temperature while it waits to be processed and would need to be re-heated to ensure it is at the correct temperature before processing. Ingot and/or bloom rolling can also be affected by the deterioration of processing times. In these steel rolling operations, the input material is reheated to a predetermined temperature before rolling. Each steel grade has a critical temperature below which rolling becomes difficult and may cause machine damage. Below this temperature, the input ingots/blooms are said to have deteriorated and require a reheating cycle. In all cases of deteriorating jobs, the resultant effect is processing times that increase with an increase in waiting time before processing. A deteriorating job is, thus, referred to as a job that will take more time to be processed later than when it is processed first in the schedule.

This paper presents a permutation flow shop that considers a scheduling problem that is both reentrant and has a deteriorating schedule. The reentrant flow shop problem has been studied for various machine environments; however, work on deteriorating jobs has significantly been on one and two-machine environments. To date, there appears to be no evidence of work published that considers the reentrant flow shop with deteriorating jobs and this paper seeks to fill that gap. The aim of this study is to develop a mathematical programming model and utilize heuristics to solve the problem related to scheduling of N jobs on M machines in a reentrant permutation flow shop with deteriorating jobs with the objective being the minimization of the makespan. The remainder of the paper is organized in the following manner. Section 2 gives a literature review on the study of reentrance and deteriorating jobs. The model for the problem being studied and the proposed solution algorithms are presented in Section 3. Section 4 covers the experimental design for computations. The results of the computational experiments are presented and discussed in Section 5. The paper is concluded in Section 6.

2. Literature

The earliest work identified on reentrant flow shop scheduling was that of Graves et al. (1983). The manufacturing process that was used for their study is that of an integrated circuit fabrication facility. They solved the problem of minimizing the throughput time using a heuristic. It was initially suggested that the system could be operated as a job shop with the use of sequencing rules to determine an optimal sequence of jobs at each machine. This implied that a simple Gantt chart for sequencing jobs in this system could be used. However, for a large number of jobs, it would be nearly impossible to manage. A heuristic was, thus, developed to minimize the throughput time using a cyclic scheduling method at specified production rates. The heuristic was referred to as a cyclic Gantt chart. In the cyclic schedule, the chart is divided into manageable cycle times to enable the scheduler to know the number of cycles required to complete a single job. One of the unique features of the model is that the generated schedule could be adapted when conditions in the system such as shutdowns, machine breakdowns and operator unavailability arose. The downside, however, was that the model was developed as a computer program from which the results needed to be transferred onto a shop schedule. Any errors that take place during the transfer of the schedule from the program to the workshop can affect the entire process negatively.

Pan and Chen (2010) studied the minimization of makespan in a reentrant permutation flow shop. They proved that the reentrant flow shop problem is NP-hard, even for a two-machine shop. Heuristics were used to solve the problem. Chen (2006) extended the work done on the reentrant permutation flow shop by developing a branch and bound algorithm. For the algorithm to reach an optimal solution quickly, a branching rule, upper and lower bounds, and a fathoming rule were utilised. Chen et al. (2007) developed an integer programming model and used heuristics to find an initial solution. Tabu search was then applied to improve the initial solution in scheduling jobs in a reentrant N machine environment to minimize makespan.

Recent work includes scheduling a reentrant no-wait shop and systems with multiple objectives. One such problem was studied by Rifai et al. (2016) where three objective functions, namely maximum completion time, total production cost, and average tardiness, were integrated. A multi-objective adaptive large neighborhood search (MOALNS) algorithm was developed to find near-optimal solutions for the problem. TasoujiHassanpour et al. (2015) proved that the no-wait reentrant problem is NP-hard and utilized heuristic algorithms to solve the problem. Genetic algorithm, a bottleneck-based heuristic, and simulated annealing heuristics were used to solve the problem. The simulated annealing algorithm outperformed the genetic algorithm and the bottleneck-based algorithm in terms of finding the best solution within reasonable computation time. Amrouche et al. (2020) presented a two-machine chain-reentrant flowshop problem with no wait, which was solved in polynomial time based on dynamic programming formulation. Rifai et al (2021) proposed the multi-objective adaptive large neighborhood search (MOALNS) algorithm to solve the Multi-objective distributed reentrant permutation flow shop scheduling with sequence-dependent setup time. Zheng et al. (2023) investigated the optimal solution properties of a two-stage reentrant flexible flow shop scheduling problem using mixed integer linear programming and a greedy random constructive heuristic.

The job deterioration problem is relatively new, currently spanning only three decades of research. However, much like any area of scheduling, the job deterioration problem has had variants of the original problem since its inception. The first decade did not see much development. Studies were conducted with a focus on single-machine problems with the objective of optimizing makespan (Browne and Yechiali, 1990), Kubiak and van de Velde (1998) and minimization of flowtime Mosheiov (1991). Browne and Yechiali (1990) pioneered the study of deteriorating jobs, with a focus on a single machine problem with simple linear deterioration and an objective of makespan minimization. They proved that the processing time of a job increases linearly in relation to the waiting time, and they used scheduling policies to develop a solution approach. Meanwhile, Mosheiov (1991) studied the minimization of flowtime for N jobs on a single machine and proved that the optimal sequence for the problem is V-shaped. The V-shape property essentially means that the first set of jobs to be scheduled is arranged in decreasing order of the deterioration rate. In contrast, the remaining set of the jobs is scheduled in increasing order of their deterioration rate. A study proving the NP-hardness of a two- and three-machine scheduling problem for deteriorating jobs was conducted by Kononov and Gawiejnowicz (2001). The types of deteriorating jobs studied were simple linear and proportional deterioration with the objective of minimizing the makespan. Hindi and Mhlanga (2001) studied two variants of deterioration, simple linear deterioration and jobs with basic processing times. Simple linear deterioration assumes that jobs are only available at a positive time t_0. Jobs with basic processing time are said to have a scheduling horizon that starts at $t = 0$ for all jobs. The total processing time of jobs with basic processing is made up of two components; the deterioration-affected component and the actual processing time. Thus, the longer a job waits in the queue before processing, the longer it will take to be completed. Heuristics were used to schedule jobs for the parallel machine environment to minimise makespan. Their choice of the solution technique was based on the fact that scheduling problems with parallel machines is NP-hard, and introducing deterioration increases the complexity of the problem.

A lot of focus for the scheduling of deteriorating jobs has been on manufacturing facilities with machines that are less than or equal to three. Single-machine scheduling of deteriorating jobs has been studied extensively, for a range of variations (Browne and Yechiali (1990), Mosheiov (1991), Wand and Xia (2005), Ji and Cheng (2010), Wang and Wang (2011)). Shiau et al. (2007) studied deteriorating jobs for two machine manufacturing systems. Jafari et al. (2017) studied three machine problems. Wang et al. (2019), however, studied an M machine (m > 3) problem for scheduling deteriorating jobs. These were one of the few identified in the literature to have studied the problem for M machines. They utilized a metaheuristic called multi-verse optimizer (MVO) to find a solution to the problem.

3. Model development and solution algorithms

The following notations are adopted for model development:

Symbol	Description
$\boldsymbol{J}$	job index set; J: $\{j = 1,2, \dots, k\}$, where k is the number of jobs
$\boldsymbol{M}$	machine index set; M: $\{m = 1,2, \dots, n\}$, where n is the number of machines
$\boldsymbol{L}$	number of levels of job j; L: $\{l = 1,2, \dots, b\}$, where b is the number of levels
$\boldsymbol{i}$	position of job j in the sequence
$\boldsymbol{x_{ij}}$	1, if job j is scheduled in the ith position at each level; 0 otherwise
$\boldsymbol{s_{mli}}$	the starting time of a job scheduled in the ith position of level l on machine m
$\boldsymbol{a_{mli}}$	the deterioration rate of a job scheduled in the ith position of level l on machine m
$\boldsymbol{a_j}$	the deterioration rate of a job j
$\boldsymbol{p'_{jlm}}$	the normal processing time of the operation of job j on machine m at level l
$\boldsymbol{p_{jlm}}$	the actual processing time of the operation of job j on machine m at level l, where $p_{jlm} = p'_{jlm} + a_{mli} \cdot s_{mli}$
$\boldsymbol{C_{max}}$	completion time of the last job in the sequence

$\boldsymbol{C_j}$	completion time of job j

The following assumptions are also made for model development:

- All jobs are available from time zero (batch processing)
- Each machine can only process a single job at a time
- Jobs visit every machine in the same order $M_1, M_2, \dots M_n$ (permutation)
- No machine breakdowns
- All job processing times are known
- There is unlimited storage space for jobs waiting to be processed
- Pre-emption is not allowed
- Machine set-up times are included in the processing time

The model is presented as follows:

Minimise $\quad C_{max} \quad$ (1)

Subject to $\quad \sum_{j=1}^{k} x_{ij} = 1 \quad i = 1,2,\dots,k \quad$ (2)

$$\sum_{i=1}^{q} x_{ij} = 1 \qquad j = 1,2,\dots,k \tag{3}$$

$$s_{111} = 0 \tag{4}$$

$$s_{1,1,i+1} = a_{11i}.s_{11i} + \sum_{j=1}^{k} x_{ij} p_{j11} \qquad i = 1,2,\dots,q-1 \tag{5}$$

$$s_{1,l,i+1} \geq a_{1li}.s_{1li} + \sum_{j=1}^{k} x_{ij}\, p_{111} \qquad l = 2,3,\dots,b; i = 1,2,\dots,q-1 \tag{6}$$

$$s_{m,l,i+1} \geq a_{mli}.s_{mli} + \sum_{j=1}^{k} x_{ij}\, p_{jlm} \qquad m = 2,3,\dots n; l = 1,2,\dots,b; i = 1,2,\dots,q-1 \tag{7}$$

$$s_{1,l+1,1} \geq a_{1lq}.s_{1lq} + \sum_{j=1}^{k} x_{qj}\, p_{jl1} \qquad l = 1,2,\dots,b-1 \tag{8}$$

$$s_{m,l+1,1} \geq a_{mlq}.s_{mlq} + \sum_{j=1}^{k} x_{qj}\, p_{jlm} \qquad m = 2,3,\dots n-1; l = 1,2,\dots,b-1 \tag{9}$$

$$s_{m+1,1,1} = a_{m11}.s_{m11} + \sum_{j=1}^{k} x_{1j}\, p_{j1m} \qquad m = 1,2,\dots,n-1 \tag{10}$$

$$s_{m+1,l+1,1} \geq a_{mlq}.s_{mlq} + \sum_{j=1}^{k} x_{qj}\, p_{jlm} \qquad m = 1,2,\dots,n-1; l = 1,2,\dots,b-1 \tag{11}$$

$$s_{m+1,l,i} \geq a_{mli}s_{mli} + \sum_{j=1}^{k} x_{ij}\, p_{jlm} \quad m = 1,2,3; l = 1,2,\dots,b; i = 1,2,\dots,q; (l,i) \notin \{(1,1)\} \tag{12}$$

$$s_{m,l+1,i} \geq a_{nli}.s_{nli} + \sum_{j=1}^{k} x_{ij}\, p_{jln} \qquad l = 1,2,\dots,b-1; i = 1,2,\dots,q \tag{13}$$

$$C_{max} = \sum_{j=1}^{k} p_{jbn} \prod_{r=j+1}^{k} (1 + a_r) \tag{14}$$

$$C_{max} \geq 0, s_{mli} \geq 0 \quad m = 1,2,3; l = 1,2,\dots,b; i = 1,2,\dots,q;$$

$$x_{ij} = 0\ or\ 1 \quad i = 1,2,\dots,q; j = 1,2,\dots,k \tag{15}$$

Equation (1) is the objective function. Constraints (2) and (3) describe the decision variables for the problem ensuring that each job is scheduled in only a single position and that a position has only a single job allocated to it, while constraint (4) defines the starting time of the first operation scheduled at the first level of the first machine. Constraints (5) and (10) define the starting time of any job, on any machine at the first level of operation. Constraints (6) to (9) and (11) to (13) describe the precedence of jobs in the process. Constraint (14) is the expression for calculating the makespan of the system, allowing deterioration of processing time, while constraint (15) enforces the non-negativity and binary restrictions.

Three solution algorithms; CDS, NEH, and GA, which were adapted for reentrance and deterioration, are presented as pseudo-codes 1, 2, and 3, respectively.

Pseudo-code 1: Modified CDS algorithm for reentrant flow shop problem

```
//Create an aggregate processing time for each job for each machine across all levels
For every job across all levels $l = 1 \ to \ b$
        $p'_{jm} = \sum_{l=1}^{b} p_{jlm}$
EndFor
//solve $n - 1$ surrogate two machine problems
Set makespan to a large number
For $z$ varying from $1$ to $n - 1$

For every job, $j$, on two pseudo machines, $M'_1$ and $M'_2$
                Create two surrogate processing times as follows
                $p'_{jm1} = \sum_{m=1}^{z} p'_{jm}$ as processing time of job $j$ on $M'_1$
                $p'_{jm2} = \sum_{m=n-z+1=1}^{n} p'_{jm}$ as processing time of job $j$ on $M'_2$
EndFor
        Solve sequencing problem for $M'_1$ and $M'_2$ using Johnson's Algorithm
        If makespan from sequence is better
                Update makespan
                Update optimum sequence
EndIf
EndFor

Johnson's Algorithm
Create two sets, $I$ and $II$ such that
        If $p'_{jm1} < p'_{jm2}$ allocate job to set $I$
        ElseIf $p'_{jm2} < p'_{jm1}$ allocate job to set $II$
        Else allocate to either set
        EndIf
Sort set $I$ in non-decreasing order of $p'_{jm1}$
Sort set $II$ in non-increasing order of $p'_{jm2}$
Append set $II$ to set $I$
```

Pseudo-code 2: NEH Algorithm

```
For each job, $j$,
Determine the work content, $p'_j$, from $p'_j = \sum_{l=1}^{b} \sum_{m=1}^{n} p_{jlm}$
EndFor
Sort jobs into set $I$ in decreasing order of work content, $p'_j$,
Take the first two jobs out of set $I$ and form two partial sequences with them
Retain the partial sequence with the minimum makespan of the two
While there exists an unscheduled job in set $I$
        Remove the next job (with largest work content)
Form a set of new partial sequences by inserting the job in all possible positions in the currently retained partial sequence
```

```
        Retain one of the new partial sequence which has minimum makespan
EndWhile
```

Pseudo-code 3: GA algorithm

```
Set current iteration to 1
Generate initial population
Evaluate fitness and rank chromosomes based on makespan
Store the best chromosome
While current iteration is less than the required iterations
        Evaluate fitness and rank chromosomes based on makespan
        Update the best chromosome found
        Retain top performers and discard the remaining chromosomes
        Cross breed top performers to create new chromosomes make up population
        Mutate the top chromosomes
        Increment current iteration

EndWhile
```

The various components of GA, including the crossover and mutation operators that were modified to incorporate reentrance and deterioration are discussed next.

Chromosome representation

To represent a chromosome, a vector of the same length as the number of jobs to be sequenced is created and populated with a random number between 1 and the number of jobs in such a way as to guarantee that every job is placed in exactly one position only in the vector.

Crossover Implementation

The crossover algorithm makes use of a logic termed herein as autogamy, meaning a chromosome is crossed with itself. The advantages of this procedure are twofold. The first is that it ensures that all jobs are placed exactly in one position after crossover. This is important because it is easy for some jobs to be missing while others are present in more than a single position after the crossover process. The second advantage is that it preserves some partial sequences of schedules that have performed well so that the advantage of promising sequences is not lost while trying to explore other areas during search. The procedure randomly selects, through a probability mechanism, a breeding chromosome. It then randomly generates the crossover point. From the crossover point, two partial sequences (the head and the tail) are created. The tail partial sequence is swapped to the head position as a block while the head sequence is also swapped to the tail position as a block. This way, the promising partial sequences are preserved with minimum disturbance. If further mix in the sequence is desired, more than one crossover procedures may be implemented in a single reproduction process. How the partial sequences are repositioned may also alter the new sequences formed.

4. Mutation implementation

In implementing the mutation process, there is also the need to ensure that all jobs are sequenced in exactly one position. To do this, a swap procedure is implemented. Two random positions are generated within the sequence and the jobs in these two positions are swapped. This procedure is repeated for the number of mutations required per chromosome.

4.1 Experimental design

The experimental design procedure involves the generation of data, development of test instances, and implementation of the experiments.

4.2 Data generation for processing times

Test data sizes and the method of data generation approach used by Chen et al. [3] for minimizing makespan in reentrant flow shops were adopted. The method involves the use of a wide range of data sizes; it tackles the scheduling of jobs in manufacturing facilities with as few as two operational levels to as many as ten operational levels. Widening the test data range allows for observations to be made on the effect that manufacturing facility configurations have on the makespan.

Parameters used in the experimental environment are described as $k\ x\ n\ x\ b$; where k represents the number of jobs, n the number of machines, and b the number of operational levels. The test data is divided into three categories: small, medium, and large problem sizes. The small problems are composed of eight matrix sizes: $3x3x3$, $3x3x4$, $3x4x2$, $4x3x3$, $4x4x3$, $4x5x3$, $4x4x4$, and $4x5x4$. Medium problems are also composed of eight matrix sizes: $6x6x2$, $6x8x5$, $6x9x3$, $7x7x5$, $7x8x4$, $8x8x3$, $9x9x2$, and $10x10x2$. The large problems include five matrix sizes: $12x12x10$, $15x15x5$, $20x20x4$, $25x25x8$, and $30x30x5$. Data was generated randomly as no benchmark data was available. Two types of data sets were generated. The first set of processing times was generated in the range [1, 100] on all machines since most benchmark data is generated within this range (Chen et al., 2007). The set was then termed the same data range (SDR) set. The second set was generated by first setting unique upper and lower bounds for the processing times of each machine in the first level within the range [1, 100]. These unique data ranges were then applied to all jobs on all other operational levels, and was termed the unique data range (UDR) set. The unique processing time ranges per machine were introduced to mimic typical manufacturing. In manufacturing, machines often have specific processing times based on their function. A set of five instances of processing time matrices was generated for each of the data set types (i.e. SDR and UDR) for each of the matrix sizes mentioned.

4.3 Data generation for penalty matrices

Penalty matrices were generated to evaluate the effect of deterioration of processing times within the range [0, 0.1]. This penalty range was classified as small by Ng et al. (2010) in their study of a two-machine flow shop problem with deteriorating jobs. Four scenarios were created within this range.

Table 1. Scenarios for delay penalty

Scenario #	Scenario	Details
1	No penalty	To observe the effect of job reentrance system without deterioration (delay penalty).
2	Penalty of 0.1 only on the first machine in the first operational level for jobs not scheduled first	To determine the effect of penalizing delay only on the first machine.
3	0.1 penalty on the first machine and 0.05 penalty for all other jobs not scheduled first	To determine the effect of uniformly penalising all delays on all machines, with the delay on the first machine incurring higher penalty
4	Random penalty in the range [0.01, 0.1] on all jobs not scheduled first	To determine the effect of random deterioration rates on all machines.

Development of test instances

A test instance refers to a computation conducted on a particular processing time matrix instance. Computations were run for all four penalty scenarios on CDS, NEH and GA algorithms for the three test categories (i.e. small, medium and large problems). Unlike the CDS and NEH heuristics, the GA algorithm requires more information in addition to the processing time and deterioration rate. The additional GA parameters are discussed next.

GA parameter setting

Parameters required for the execution of the GA include population size, mutation rate, retention rate and the number of iterations. The same mutation and retention rates were used for all problem sizes. The retention rate used was 0.5. A rate of 0.5 was selected for the mutation rate after initially testing with 0.3, 0.5, and 0.7. Figures 1 and 2 are a representation of the makespan convergence for the tested instances of mutation rates. The population sizes and the number of iterations were varied for the different matrix sizes. The population sizes were selected in relation to the number of machines for the matrix. For the small problems fewer population sizes were used. Problems with fewer machines have a tendency to repeat chromosomes due to the limited number of possible combinations.

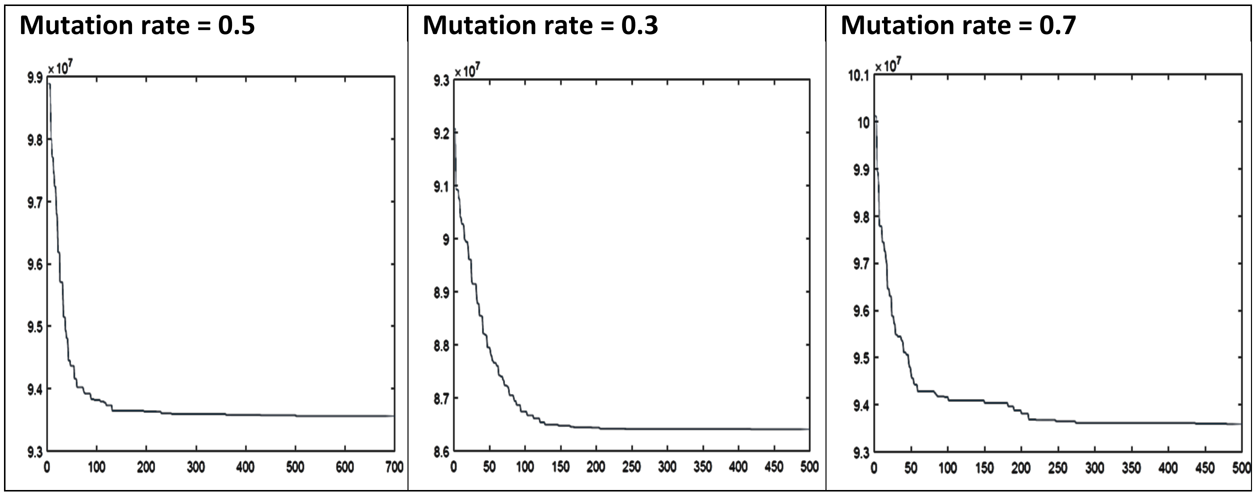

Figure 1. GA solution convergence for a 25x25x8 matrix for various mutation rates with scenario 3

Limiting the population size to a small number eliminates this challenge. The number of iterations for each matrix size was selected if the minimum makespan values appear to always converge (or stabilise) within the selected number of runs.

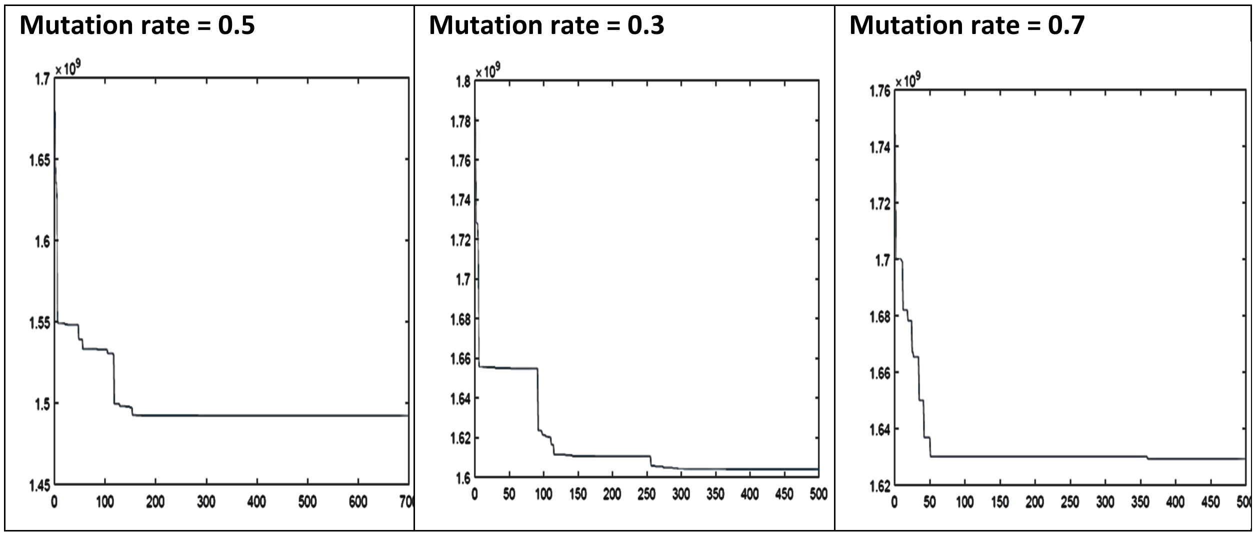

Figure 2. GA solution convergence for a 25x25x8 matrix with scenario 4 for various mutation rates

5. Implementation

The three algorithms, CDS, NEH, and GA, were coded in MATLAB R2019a, and all computational experiments were performed with this code. The experiment was run on all categories of test instances. For each category, computations for makespan, subject to the varying degrees of penalty were performed using the three algorithms.

During experimentation, the minimum makespan values were achieved and their corresponding computation times were recorded. The makespan output analysis and statistical testing were performed in MS Excel. The output analysis was conducted by applying two measures; the count of the number of times that any given algorithm returned the lowest makespan value and the average proportion above the minimum makespan when an algorithm did not return the lowest makespan value. The procedure followed to execute the counting exercise involved several steps. The five makespan values returned for each matrix size were evaluated. A value of one was allocated for the test instances with the lowest makespan value, and zero otherwise. The count of instances is not mutually exclusive; i.e. a test instance of one algorithm is counted for as long as it returned the lowest value, regardless of another algorithm being already counted for the same test instance.

The procedure for determining the average value above the minimum makespan was also stepwise. The minimum makespan value among the three algorithms for each of the matrix sizes was determined. This was then termed the global minimum. The average of the five instances of makespan values was calculated for each matrix size for the three algorithms. The proportional value above the global minimum was then calculated by subtracting the global minimum from the average makespan and then dividing by the global minimum.

Statistical testing was applied on all instances of makespan values for each of the matrix sizes. The t-test was selected as the test of choice due to the random nature of the data and the small sample sizes being evaluated. Two sets of t-tests were performed; one for the count of minimum makespan values and the other one for the proportion of values above the global minimum makespan. The paired t-test for sample means was executed for the counts of minimum makespan and the proportion above minimum makespan.

6. Discussion of results

The results are divided into three sections, makespan analysis, statistical test results, and computation time's discussion.

Makespan Analysis

The total number of counts when a solution method results in the lowest makespan value for the various test instances is presented in Table 2. The asterisks beside the counts indicate the algorithm(s) with the most number of lowest makespan for each test category. It should be noted that the maximum count obtainable for small and medium sized problems is 40, while for large sized problems is 25.

The CDS algorithm reported several test instances that had the lowest makespan values for the small problems of scenario 1 (i.e. no deterioration) and scenario 2 (i.e. deterioration only on the first machine) for the SDR data set. The CDS algorithm also returned a few instances with the lowest makespan values for the small and medium problems of scenarios 1 and 2 for the UDR data set. Overall, the NEH and GA algorithms returned the most number of the lowest makespan values. For the SDR data set, GA returned more instances of the lowest makespan values than NEH. On the other hand, the NEH algorithm returned the most instances for the UDR data set, while dominating scenario 3 (i.e. uniform deterioration on all machines, with the first machine incurring the highest penalty) and scenario 4 (i.e. random deterioration on all machines).

Table 2. Count of minimum makespan attainment

	Problem size	Same data range (SDR)			Unique data range (UDR)		
		CDS	NEH	GA	CDS	NEH	GA
Scenario 1	Small	14	36	39*	8	34	35*
	Medium	0	20	34*	1	11	39*
	Large	0	15*	10	0	19*	7
Scenario 2	Small	11	35*	29	5	32*	30
	Medium	0	16	32*	1	16	33*
	Large	0	4	21*	0	13*	13*
Scenario 3	Small	0	33*	10	0	40*	3
	Medium	0	27*	13	0	32*	9

	Large	0	8	17*	0	18*	7
Scenario 4	Small	0	33*	8	0	33*	8
	Medium	0	10	30*	0	23*	17
	Large	0	10	15*	0	15*	10

The analysis of the average proportion above the global minimum makespan is presented in Table 3, and the asterisks beside the values indicate the lowest proportional value.

The proportional values above the global minimum for the CDS algorithm got significantly large for the medium and large problem sizes of scenarios 3 and 4 as compared to scenarios 1 and 2 for both the SDR and UDR data sets. The NEH and GA returned the lowest proportional values, the same number of instances for the SDR data set. The NEH, however, had the lowest proportional values mostly for scenarios 3 and 4. For the UDR data set, the NEH returned the most instances of the lowest proportional values. As with the SDR data set, NEH dominated scenarios 3 and 4

Table 3. Average proportion above global minimum makespan

	Problem size	Same data range (SDR)			Unique data range (UDR)		
		CDS	NEH	GA	CDS	NEH	GA
Scenario 1	Small	0.157	0.114	0.113*	0.116	0.089*	0.089*
	Medium	0.121	0.073	0.067*	0.038	0.023	0.019*
	Large	0.076	0.021*	0.024	0.036	0.010*	0.011
Scenario 2	Small	0.159	0.119	0.117*	0.122	0.097*	0.097*
	Medium	0.124	0.076	0.068*	0.041	0.028	0.024*
	Large	0.109	0.058	0.031*	0.077	0.042	0.029*
Scenario 3	Small	0.893	0.178*	0.269	0.738	0.088*	0.406
	Medium	2.189	0.070*	0.085	2.124	0.055*	0.075
	Large	6264	1.029*	1.605	3264	0.050*	0.069
Scenario 4	Small	1.209	0.184*	0.247	1.013	0.105*	0.174
	Medium	4.097	0.125*	0.317	4.320	0.093*	0.109
	Large	99723	0.195	0.100*	100254	0.197	0.182*

.

Statistical test of significance of differences

Statistical testing was performed to determine if the difference in the count of the number of instances in which the minimum makespans were found by each of the algorithms was significant or not. The null hypothesis, H_0, is that two algorithms being compared have no significant difference of the makespan values. The alternative hypothesis, H_1, is that the algorithms have a significant difference in makespan values. This is important because the data were randomly generated. The statistical test values achieved are presented in Tables 6 and 8 for the count of the minimum makespan and proportion above the minimum makespan value respectively. The absolute values of the t statistic are of interest in determining the algorithm(s) resulting in the minimum makespan. The evaluation of the significance of the t stat values returned against the t critical value is presented in Tables 7 and 9. Instances for which there is no significant difference between two algorithms are denoted by N and S denotes results with a significant difference. Additionally, the algorithm that performed better for a particular problem size or penalty scenario is indicated in brackets for instances with a significant difference.

Count of minimum makespan

The NEH-GA comparison generally returned instances with the lowest t-stat values, however, with a few exceptions. The CDS-NEH comparison returned the lowest t stat values for scenario 2 (large problem sizes) and scenario 4 (medium problem sizes) for the SDR data set, and scenario 1 (medium problem sizes) for the UDR data set. The CDS-

GA comparison returned the lowest t stat values for scenario 3 (small problem sizes) and scenario 4 (small problem sizes) for the SDR data set, and scenario 4 (small problem sizes) for the UDR data set. GA was the better-performing algorithm for cases where there was a significant difference for the SDR data set. The NEH algorithm returned the majority of instances with the minimum makespan counts for the UDR data set. The difference in makespans was significant for any algorithm compared to CDS, with the CDS algorithm being the worst performing. It is for this reason that there is no indication of the algorithm which performs better for CDS-NEH and CDS-GA comparisons.

Table 4. t Stat values for the count of minimum makespan

	Problem size	Same data range (SDR)			Unique data range (UDR)		
		CDS - NEH	NEH - GA	CDS - GA	CDS - NEH	NEH - GA	CDS - GA
Scenario 1	Small	-6.2973	-1.3556	-8.0623	-8.5105	-0.2981	-8.1219
	Medium	-6.245	-3.0095	-14.8661	-3.6056	-8.5732	-27.2213
	Large	-6	1	-4	-8.7178	2.753	-3.0551
Scenario 2	Small	-6.958	1.5246	-5.1523	-8.1219	0.4953	-7.3193
	Medium	-5.099	-3.1225	-12.49	-4.8374	-3.4426	-12.49
	Large	-2.1381	-4.5434	-11.225	-5.099	0	-5.099
Scenario 3	Small	-13.5594	4.6577	-3.6056	UNDEF	21.9317	-1.7782
	Medium	-9	2.3333	-4.3333	-12.49	4.4733	-3.3649
	Large	-3.3607	-1.8904	-7.1414	-7.8558	2.4004	-3.0551
Scenario 4	Small	-13.5594	5.1058	-3.1225	-13.5594	5.1058	-3.1225
	Medium	-3.6056	-3.6056	-10.8167	-7.2639	0.9475	-5.369
	Large	-4	-1	-6	-6	1	-4

Table 5. Evaluation of the significance of T values for a count of minimum makespan

	Problem size	Same data range (SDR)			Unique data range (UDR)		
		CDS - NEH	NEH - GA	CDS - GA	CDS - NEH	NEH - GA	CDS - GA
Scenario 1	Small	S	N	S	S	N	S
	Medium	S	S (GA)	S	S	S (GA)	S
	Large	S	N	S	S	S (NEH)	S
Scenario 2	Small	S	N	S	S	N	S
	Medium	S	S (GA)	S	S	S (GA)	S
	Large	S	S (GA)	S	S	N	S
Scenario 3	Small	S	S (NEH)	S	N	S (NEH)	N
	Medium	S	S (NEH)	S	S	S (NEH)	S
	Large	S	N	S	S	S (NEH)	S
Scenario 4	Small	S	S (NEH)	S	S	S (NEH)	S
	Medium	S	S (GA)	S	S	N	S
	Large	S	N	S	S	N	S

Proportion above minimum makespan

In the test for a proportion of values above the minimum makespan, the same observation was made for all penalty scenarios and problem sizes. The NEH-GA comparison resulted in the lowest t-stat values. GA performed better for cases where there was a significant difference for the SDR data set with penalty scenarios 1 and 2. Penalty scenarios 3 and 4 of the SDR data set were dominated by NEH. The NEH algorithm was the overall better-performing method for the UDR data set. The paired test of CDS with any method resulted in higher t-stat values, and thus a significant difference from t-critical was observed. The CDS algorithm was the worst performing for both data sets.

Table 6. t Stat values for proportion falling within the minimum makespan

	Problem size	Same data range			Unique data range		
		CDS - NEH	NEH - GA	CDS - GA	CDS - NEH	NEH - GA	CDS - GA
Scenario 1	Small	5.2017	0.9105	5.2271	3.9067	0.3697	4.0086
	Medium	9.3686	3.1116	10.2402	6.774	4.1266	9.8536
	Large	21.2569	-1.7356	20.4309	24.4516	-3.0656	21.3496
Scenario 2	Small	4.9676	0.8628	5.1592	3.5914	-0.4565	3.5985
	Medium	9.1832	3.3348	10.6583	5.5364	3.0074	9.4545
	Large	6.9113	3.4371	6.3809	4.7638	1.3925	6.0476
Scenario 3	Small	15.3478	-3.5605	10.9191	17.5279	-2.9098	3.4483
	Medium	12.9587	-2.2846	12.997	12.4692	-5.3536	12.4092
	Large	3.3186	-1.3395	3.3186	2.9278	-2.2007	2.9278
Scenario 4	Small	11.6613	-5.1845	11.3359	12.7631	-5.3719	11.8116
	Medium	12.2037	-0.7797	8.344	12.2398	-1.4284	12.2818
	Large	2.7558	2.4267	2.7558	2.7092	0.3481	2.7091

Table 7. Evaluation of the significance of T values for proportion falling above the minimum makespan

	Problem size	Same data range			Unique data range		
		CDS - NEH	NEH - GA	CDS - GA	CDS - NEH	NEH - GA	CDS - GA
Scenario 1	Small	S	N	S	S	N	S
	Medium	S	S (GA)	S	S	S	S
	Large	S	N	S	S	S (NEH)	S
Scenario 2	Small	S	N	S	S	N	S
	Medium	S	S (GA)	S	S	S (GA)	S
	Large	S	S (GA)	S	S	N	S
Scenario 3	Small	S	S (NEH)	S	S	S (NEH)	S
	Medium	S	S (NEH)	S	S	S (NEH)	S
	Large	S	N	S	S	S (NEH)	S
Scenario 4	Small	S	S (NEH)	S	S	S (NEH)	S
	Medium	S	N	S	S	N	S
	Large	S	S (GA)	S	S	N	S

Computation times

The average computation times are presented in Table 8. The CDS algorithm had the shortest computation times, followed by NEH. The computation times increased slightly from the small problem size to the large problems for both the CDS and NEH algorithms for all penalty scenarios. The GA algorithm had the longest computation times for the large problem size. The time complexity for GA is influenced by the large population sizes and the high number of iterations related to the problem sizes.

Table 8. Average computation times

Penalty type	Solution method	Small problems		Medium problems		Large problems	
		Min computation time	Max computation time	Min computation time	Max computation time	Min computation time	Max computation time
Scenario 1	CDS	0.0003	0.0393	0.0005	0.1402	0.0043	0.1017
	NEH	0.0002	0.0153	0.0006	0.0075	0.0157	0.5409
	GA	0.0252	0.2008	0.0993	0.8794	38	872
Scenario 2	CDS	0.0003	0.0890	0.0005	0.0650	0.0044	0.0993
	NEH	0.0003	0.0282	0.0009	0.0234	0.0166	0.2481
	GA	0.0251	0.3108	0.1589	7	42	915
Scenario 3	CDS	0.0003	0.0789	0.0005	0.0495	0.0051	0.0828
	NEH	0.0003	0.0558	0.0008	0.0995	0.0178	0.2942
	GA	0.0242	0.1700	0.1348	6	40	970
Scenario 4	CDS	0.0003	0.0468	0.0005	0.1554	0.0051	0.1385
	NEH	0.0003	0.0073	0.0007	0.0059	0.0174	0.3294
	GA	0.0313	0.1935	0.1735	7	46	874

7. Conclusions and Recommendations

The problem of scheduling a reentrant permutation flow shop with deteriorating jobs with the objective of minimizing makespan was studied. The CDS, NEH and GA algorithms were utilized to model the solution. The study involved test problems classified as small, medium and large. Simulations for various deterioration rates were conducted.
The algorithms performed similarly for small problems that are not exposed to deterioration of processing time. The GA and NEH algorithms performed better than the CDS algorithm as the problem sizes increased in size (i.e. an increase in the number of jobs and machines). The NEH achieved the lowest makespan within reasonable computation times as problem sizes got bigger, and the complexity of deterioration of processing times increased. Essentially, the NEH algorithm is capable of handling changes that are introduced into the system being scheduled. In instances where the NEH achieved minimum makespan values above the global minimum, the proportional difference was small. The NEH algorithm, thus, appears to be the overall best-performing algorithm. For future studies, the NEH algorithm can be compared with other solution methods such as the branch and bound.

Conflict of Interest

The authors have no conflict of interest.

References

Amrouche K., Boudhar N. and Sami N., Two-machine chain-reentrant flow shop with the no-wait constraint, *European Journal of Industrial Engineering*,14(4), pp. 573-597, 2020.

Browne S, Yechiali U. Scheduling deteriorating jobs on a single processor. *Operations Research,* 1990; 38(3): pp. 495-498, 1990.

Chen J-S. A branch and bound procedure for the reentrant permutation flow-shop scheduling problem. *Int J Adv Manuf Technol*,; 29: 1186–1193, 2006.

Chen J-S, Pan J.C-H, Wu C-K. Minimizing makespan in reentrant flow-shops using hybrid tabu search. *Int J Adv Manuf Technol,* 2007; 34, pp. 353-361, 2007.

Chen J-S, Pan J.C-H, Lin C-M. A hybrid genetic algorithm for the re-entrant flow-shop scheduling problem. *Expert Systems with Applications* 2008; 34: pp. 570–577, 2008.

Graves S.C, Meal H.C, Stefek D, Zeghmi A.H. Scheduling of re-entrant flow shops. *Journal of Operations Management* 1983; 3(4): pp. 197-207, 1983.

Hindi K.S, Mhlanga S. Scheduling deteriorating jobs on parallel machines: a simulated annealing approach. *Production Planning & Control* 2001; 12(1): pp. 76-80, 2001.

Jafari A-A, Khademi-Zare H, Lotfi M.M, Tavakkoli-Moghaddam R. A note on "On three-machine flow shop scheduling with deteriorating jobs". *International Journal of Production Economics* 2017; 191: pp. 250–252, 2017.

Ji M, Cheng T.C.E. Scheduling resumable simple linear deteriorating jobs on a single machine with an availability constraint to minimize makespan. *Computers & Industrial Engineering* 2010; 59: pp. 794–798,2010.

Kononov A., and Gawiejnowicz S. NP-hard cases in scheduling deteriorating jobs on dedicated machines. *Journal of the Operational Research Society* 2001; 52: pp. 708-718.

Kubiak W, van de Velde S. Scheduling deteriorating jobs to minimize makespan. *Naval Research Logistics* 1998; 45: pp. 511-523, 1998.

Mosheiov G. V-shaped policies for scheduling deteriorating jobs. *Operations Research* 1991; 39(6): pp. 979-991, 1991.

Ng C.T, Wang J.-B, Cheng T.C.E, Liu L.L. A branch-and-bound algorithm for solving a two-machine flow shop problem with deteriorating jobs. *Computers &Operations* Research 2010; 37: pp. 83-90, 2010.

Pan JC-H, Chen J-S. Minimizing makespan in re-entrant permutation flow-shops. *Journal of the Operational Research Society* 2003; 54: pp. 642-653, 2003.

Rifai A.P., Mara S.T.W., and, Sudiarso A., Multi-objective distributed reentrant permutation flow shop scheduling with sequence-dependent setup time, *Expert Systems with Applications*, 183, 115339

Rifai A.P., Nguyen H-T, Dawal S.Z.M. Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling. *Applied Soft Computing*; 40: pp. 42–57, 2016.

Shiau Y-R., Lee W-C, Wu C-C and Chang C-M. Two-machine flowshop scheduling to minimize mean flow time under simple linear deterioration. *Int J Adv Manuf* 2007; 34: pp. 774–782, 2007.

TasoujiHassanpour S, Amin-Naseri M. R, Nahavandi N. Solving re-entrant no-wait flowshop scheduling problem. *IJE TRANSACTIONS C*: Aspects 2015; 28(6): pp. 903-912, 2015.

Wang H, Huang M, Wang J. An effective metaheuristic algorithm for flowshop scheduling with deteriorating jobs. *Journal of Intelligent Manufacturing* 2019; 30: pp. 2733–2742, 2019.

Wang J-B., Wang J-J, Ji P. Scheduling jobs with chain precedence constraints and deteriorating jobs. Journal of the *Operational Research Society* 2011; 62: pp. 1765–1770.

Wang J-B, Xia Z-Q. Scheduling jobs under decreasing linear deterioration. *Information Processing Letters* 2005; 94: pp. 63–69, 2005.

Zheng S., He Z., Yang Z., Chu C., and Wang N., Effective upper and lower bounds for a two-stage reentrant flexible flow shop scheduling problem, *Computers & Operations Research*, 2023, 153: 106183, 2023.

Biographies

Koketso Mbewe is a practicing metallurgical engineering practitioner who completed a Master's studies in Industrial Engineering in the University of Pretoria. She has interest in the applications of operations research techniques in diverse areas of manufacturing, especially in metal processing.

Olufemi Adetunji is a professor of Industrial and Systems Engineering in the University of Pretoria. His interest is in the planning and control of manufacturing systems.