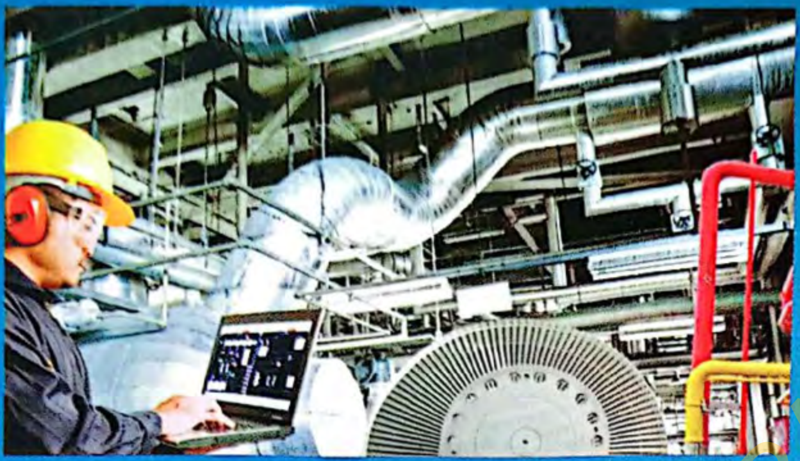


ADVANCING INDUSTRIAL ENGINEERING IN NIGERIA THROUGH TEACHING, RESEARCH AND INNOVATION

A BOOK OF READING

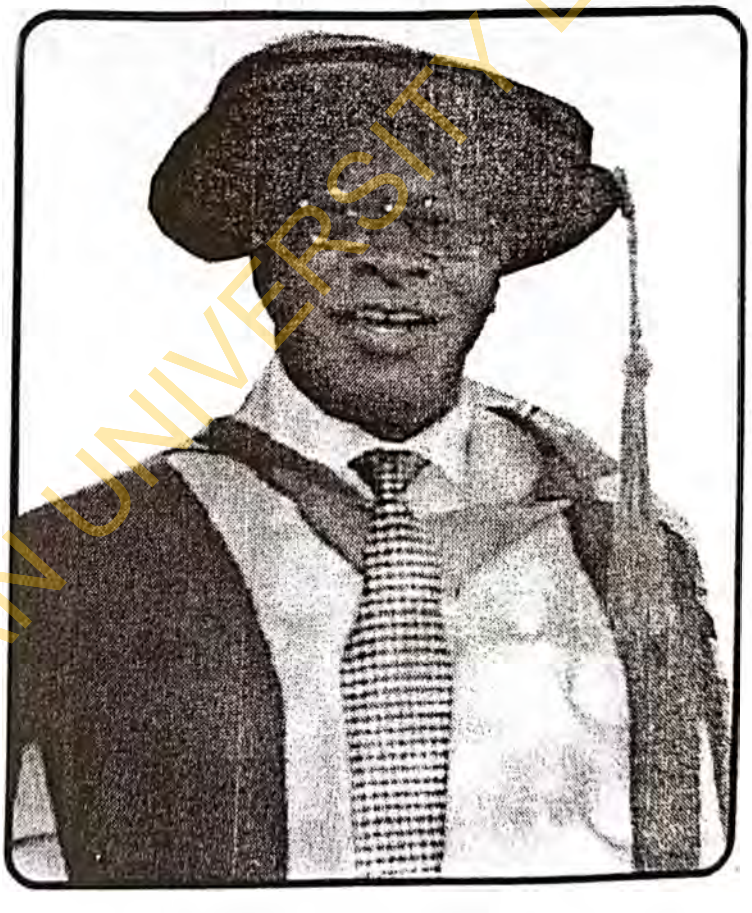
Edited By
Ayodeji E. Oluleye
Victor O. Oladokun
Olusegun G. Akanbi

ADVANCING INDUSTRIAL ENGINEERING IN NIGERIA THROUGH TEACHING, RESEARCH AND INNOVATION

Edited By
Ayodeji E. Oluleye
Victor O. Oladokun
Olusegun G. Akanbi

ADVANCING INDUSTRIAL ENGINEERING IN NIGERIA THROUGH TEACHING, RESEARCH AND INNOVATION

(A Festchrift in honour of Professor O. E Charles-Owaba)



Professor O. E. Charles-Owaba

Advancing Industrial Engineering in Nigeria through Teaching, Research and Innovation.

ISBN : 978-078-515-9

Published by

Department of Industrial and Production Engineering
University of Ibadan.

Printed by:
Leading Edge Printers & Publisher
Ibadan

FOREWORD

It gives me great pleasure writing the foreword to this book. The book was written in recognition of the immense contributions of one of Nigeria's foremost industrial engineers, respected teacher, mentor, and lover of youth – Professor Oliver Charles-Owaba.

His commitment to the teaching and learning process, passionate pursuit of research and demonstration of excellence has prompted his colleagues and mentees to write this book titled – Advancing Industrial Engineering in Nigeria through Teaching, Research and Innovation (A Festschrift in honour of Professor O. E Charles-Owaba) as a mark of honour, respect and recognition for his personality and achievements.

Professor Charles-Owaba has written scores of articles and books while also consulting for a medley of organisations. He has served as external examiner to various programmes in the tertiary educational system. The topics presented in the book cover the areas of Production/Manufacturing Engineering, Ergonomics/Human Factors Engineering, Systems Engineering, Engineering Management, Operations Research and Policy. They present the review of the literature, extension of theories and real-life applications. These should find good use in the drive for national development.

Based on the above, and the collection of expertise in the various fields, the book is a fitting contribution to the corpus of knowledge in industrial engineering. It is indeed a befitting gift in honour of erudite Professor Charles-Owaba.

I strongly recommend this book to everyone who is interested in how work systems can be made more productive and profitable. It represents a resourceful compilation to honour a man who has spent the last forty years building up several generations of industrial engineers who are part of the process to put Nigeria in the rightful seat in the comity of nations. Congratulations to Professor Charles-Owaba, his colleagues and mentees for this festschrift.

Professor Godwin Ovuworie
Department of Production Engineering
University of Benin

TABLE OF CONTENTS

CHAPTER 5

Some Developments In Scheduling Algorithms

*Ayodeji E. Oluleye[1], Elkanah O. Oyetunji[2] and Saheed Akande[3]

1. Department of Industrial and Production Engineering, University of Ibadan, Oyo State

2. Department of Mechanical Engineering, Lagos State University, Lagos State

2. Department of Mechanical and Mechatronics Engineering, Afe Babalola University, Ado Ekiti

* Corresponding author: ayodeji.oluleye@ui.edu.ng

1.0 Introduction

It is widely acknowledged that time is a resource. Particularly in today's world, it is a critical metric of competitiveness. For designers of products and services the first to the market is key. When establishing production plans; the first to deliver is also key. The sequence of tasks is important in optimizing time metrics. Also, for effective deployment of resources, the start and finish times of the tasks in sequence aid in determining the schedules. In many respects, sequencing and scheduling are sometimes used interchangeably. While sequence represents the ordering of tasks, timetabling the sequence results in schedules. The use of time as a surrogate cost factor is due to the varied nature. Costs influencers are many but time is considered at the top of the leader board.

Work systems are replete with sequencing and scheduling examples. They are encountered in transport, computer, manufacturing, aviation, and banking systems among others. Even individual tasks and chores require that forethought be given to effective time management.

Deciding sequences and schedules to adopt can be confounding given the many combinations possible. The challenge of scheduling is how to sift through the many feasible schedules and make good choices in good time since time is a resource and metric of competitiveness.

Determining optimal sequences and hence schedules have preoccupied many researchers for decades. The challenge is in the manner in which real-life problems present. The problems may be encountered in varied settings such as :

a. Optimising single criterion
b. Optimising multiple criteria
c. Flow operations structures
d. Job shop operational structures
e. Single-channel inputs
f. Multi-channel inputs

These represent just a few of the settings, which sometimes could be a combination (hybrid).

With the COVID-19 pandemic and subsequent disruptions to supply chains, scheduling systems need re-examination to enable a reconfiguration for organisations to remain going concerns. This work is an attempt at reviewing the evolution of some scheduling solution approaches.

1.1 Exact and enumerative algorithms

Generally, solution methods for scheduling problems may be classified into two: Exact methods and approximation methods. In this section, the exact methods are discussed.
Exact methods are solution methods that can find an optimal solution to an optimization problem (of which scheduling problem is one). Exact methods have been found to always solve an optimization problem to optimality. The following solution methods may be classified under the exact methods:

1.1.1 Enumeration methods: Enumeration methods involve the complete listing of all the items in a collection. Also, it involves the listing of all of the elements of a given set. Enumeration methods are often used to solve combinatorial optimization problems such as scheduling of machines in production planning, aircraft rotation/crew

scheduling in airlines as well as transport routing/scheduling in logistics. Enumeration methods lead to lots of possible solutions with the difficulty of selecting and finding optimal solutions. The number of outputs of an enumeration method may be exponential in the size of the input.
Generally, enumeration methods may be classified into two: complete enumeration methods and incomplete enumeration methods.

1.1.1.1 Complete enumeration methods: Complete enumeration methods systematically consider all possible solutions. They are also called total or explicit enumeration methods. This method involves enumeration of all possible alternatives and a comparison of all of them to pick the best solution. Complete enumeration methods can be very expensive or even impossible for more complicated problems.

1.1.1.2 Incomplete enumeration methods: This can also be called an implicit or partial enumeration method. This method involves excluding parts of the solution space that are known to be sub-optimal.The method also involves the selection of alternatives by only considering parts of the solution space.This leads to a reduction in computation efforts because only the most promising solutions are often considered. Methods such as Branch & Bound (BB), and Dynamic Programming (DP) can be classified under implicit/incomplete enumeration methods.

1.1.2.2 Dynamic Programming method

Dynamic programming (DP) is a mathematical optimization method which breaks problems into smaller parts. It uses recursion to break and assemble them. The focus is mainly on simplification to enable traction. The method was developed by Bellman Richard around the 1950s. Although similar to divide and conquer in terms of the breakdown of the problem into smaller sub-problems; however in the DP method, the resulting sub-problems are not solved independently. The DP method remembers the results of the smaller sub-problems and then used the same for similar sub-problems. The dynamic programming approach is used to solve problems that can easily be divided into similar sub-problemsto re-usethe results obtained from these sub-problems.

In the dynamic programming method, referring to the output of the previous solution is cheaper (concerning CPU cycles) than re-computing it.The DPmethod avoids repeated work by remembering previous partial results.The DP approach trades space for time. This means that instead of calculating all the states thereby taking a lot of time but no space, space is taken up to store the results of all the sub-problems to save time later.

1.1.2.3 Branch and Bounds method

The branch and bound (BB) method is an enumeration technique in which schedules are discarded because they are worse off than established lower bounds. These could be single schedules or set of schedules. There are two important elements in the use of the branch and bound procedure. These are the search (branching) procedure and the bounding at nodes. The BB procedure, if well implemented, assures optimality (Oyetunji and Oluleye, 2008).

i. Search Procedures

There are two types of search procedures. These are depth-first and frontier search methods (French, 1982). The depth-first search procedure starts at the root node and explores the branches as far down as possible. They backtrack when better schedules are not feasible down the line. Essentially, it traverses the depths of the branch and uses a stack to determine the next vertex to begin a search, when improvements are infeasible (Fig. 1).

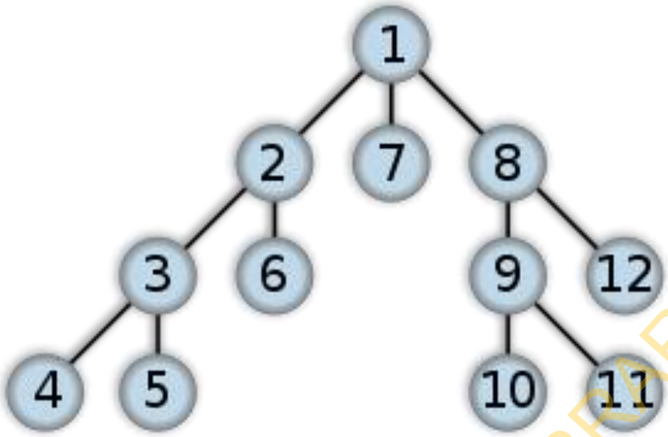


Fig. 1. Hypothetical search tree

On the other hand, the frontier search procedure is a novel approach applicable to wide classes of trade-offs between runtime and program size. Frontier search reduces the memory requirement by storing only the Open nodes while deleting closed nodes once they are expanded.
The depth-first procedure has the advantage of working with fewer variables at each node and thus requiring less storage while the frontier search procedure requires less calculations thereby obtaining solution quickly.

ii. Bounding
At every node, lower bounds must be computed for the objective function. The way the lower bound is obtained determines the efficiency of the branch and bound procedure. The lower bound gives an idea of what the value of the objective function is likely to be at the node. The typical way to develop a bound is to relax the original problem to an easily solvable problem. The relaxed problem solution bounds the original problem (Ólafsson, 2002). The lowestbound nodedetermines branches to be explored. When all the jobs have been assigned, the solution is the node with the lowest value.

1.1.3 Johnsons’ 2-machines algorithm

Johnson's 2-machines algorithm is a method of scheduling jobs in two work stations or machines. Johnson’s 2-machines algorithm seeks to find an optimal sequence of jobs to minimize makespan (the completion time of the last scheduled job). The rule also reduces the amount of idletime between two workstations. To apply Johnson’s rule, the following conditions must be met:

i. Processing time ofeach job must be constant.
ii. Processing times of jobs must be mutually exclusive of the job sequence.
iii. All the jobs must be processed through the first work station before going through the second work station.
iv. All the jobs have the same priority.

Johnson's rule for the 2-machine problem can be described as follows:

Step 1: List all the jobs and their processing times on each machine.

Step 2: Select the job having the shortest processing time. If that processing time is on for the first machine, then schedule the job first. If that processing time is on the second machine then schedule the job last. Break the ties arbitrarily.

Step 3: Remove the scheduled job from the list of unscheduled jobs.

Step 4: Repeat steps 2 & 3until all the jobs have been scheduled.Johnson (1954)

Illustrating Johnson’s 2-machine algorithm

Suppose we have a five-jobs two machines scheduling problem as shown in Table 1. Each of the five jobs needs to go through machine 1 and 2. We are required to find the optimum schedule of the jobs using Johnson's rule for a 2-machine problem.

Job Processing times (mins)		
Job	**Machine 1**	**Machine 2**
1	3.20	4.20
2	4.70	1.50
3	2.20	5.00
4	5.80	4.00
5	3.10	2.80

Solution

1. Since Job 2 has the smallest processing time (1.5 mins) and it is on machine 2, schedule job2 last.Remove Job 2 from the set of unscheduled jobs.

X	X	X	X	2

2. Job 3 has the next smallest processing time (2.20mins) and it is on machine 1, therefore schedule job 3 first.Remove Job 3 from the set of unscheduled jobs.

3	X	X	X	2

3. Job 5 has the next smallest processing time (2.80mins) and it is on machine 2, schedule job 5 last.Remove Job 5 from the set of unscheduled jobs.

3	X	X	5	2

4. Job 1 has the next smallest processing time (3.20 mins) and it is on machine 1, schedule job 1 first. Remove Job 1 from the set of unscheduled jobs.

3	1	X	5	2

5. Schedule the only remaining job (4) to the only available space.

3	1	4	5	2

So, the jobs must be processed in the order 3 → 1 → 4 → 5 → 2 and must be processed in the same order on the two machines.

1.1.4 Johnson's 3-machine algorithm

Johnson's 3-machine algorithm is similar to his 2-machine algorithm. Johnson extended his algorithm for the 2-machine problem to solve a variant of the 3-machine problem. Johnson (1954) considered a special structure case (i.e. problems in which the minimum processing time on the first or third machines is greater than or equal to the maximum processing time on the second machine).

Mathematically,

- The smallest processing time on machine 1 is greater than or equal to the largest processing time on machine 2, i.e.,

$$\text{Min } P_{i1} \geq \text{Max } P_{j2}, \forall i, j,$$

- The smallest processing time on machine 3 is greater than or equal to the largest processing time on machine 2, i.e.,

$$\text{Max } P_{j2} \leq \text{Min } P_{k3}, \forall j, k$$

At least one of the above two conditions must be met.

Where P_{tm}= processing time of t^{th} job on m^{th} machine.

When the above conditions hold, Johnson (1954) forms an artificial (machines a and b) 2-machine problem by letting

$P_{ia} = P_{i1} + P_{i2}$

$P_{ib} = P_{i2} + P_{i3}$

The artificial (2-machine) is solved by applying Johnson’s 2-machinealgorithm.

1.2 Approximation algorithms

These are efficient algorithms that yield effective solutions to problems with provable guarantees on its closeness to the optimal one. The concept of complexity gives perspective on solution techniques concerning the computation requirements (Baker and Tritsch, 2013). The order-of-magnitude notation is used to measure the computational effort required by a solution method. For example, a scheduling problem of size ***m*** (*m*represents a quantum of information needed to specify the problem), the number of computations required by the algorithm is bounded by a function of ***m***. For example, if the function has an order of magnitude m^2, represented by $O(m^2)$, then the algorithm is polynomial. However, if the function is $O(2^{\mathbf{m}})$, the algorithm is nonpolynomial. For a function of the form,$O(2^{\mathbf{m}})$, it is called an exponential algorithm. Polynomial-time algorithms are more efficient than exponential-time algorithms given that the execution times grow rapidly with problem size.

Numerous scheduling problems in practice belong to a class of optimization problems called Non deterministic Polynomial time-complete *(NP-complete)* problems. For these classes of problems, no efficient solution has been established *(Weisstein, 2015).* A problem is NP if its solution can be estimatedand verified in polynomial time. Nondeterministic implies that no specific rule is adoptedfor the estimation. If two or more problems of the same class are NP and are polynomial-time reducible to each other, such problems are called NP-complete problems. Therefore, finding an efficient algorithm for a given NP-complete problem implies that an efficient algorithm can also be found for all other problems belonging to the same class since the problem can be restructured or modified to yield one another. However, many years of research in optimization has not yielded a single polynomial-time algorithm for problems in this class, and the surmise is that no such algorithm exists. This class of problem is thus called *NP-hard (*Non-deterministic Polynomial-time hardness) problems. Therefore, it is unlikely to obtain optimal solutions to NP-hard problems efficiently, i.e. by polynomial-time algorithms. An optimal solution can be found for an NP-hard problem either by complete enumeration or implicit enumeration techniques. In both cases, for real-life problems of practical interest, only small-sized problems can be solved due to the time complexity (exponential increased) involved. However, in real-life like industrial setting, production workshop, hospital, school among others where scheduling problems is a challenge, there is always the need to solve large problem-sized NP-hard problems. This practical importance necessitates relaxations to achieve tractability. A very fruitful approach has been to relax the notion of optimality and settle for an efficient and effective (or near-optimal) solution. It is desired that solutions be within a small multiplicative factor of the optimal value (approximation ratio). Approximation algorithms provide a provably good approximation ratioto the optimal. While there are numerous (good) approximation algorithms for several NP-Hard problems in the literature, scheduling problems of certain classes remain indistinguishablein the theory of NP-Completeness. They behave very differently when subjected to approximation algorithms (Brucker, 2007).

Furthermore, there exist numerous NP-hard scheduling problems thatrequire lesser time to execute the work in the workshop using approximation algorithms than to solve the problem optimally using the fastest available computing machine. Therefore, the reliance on approximation algorithms is often the rule in practice.Furthermore, the closeness of the generated solution (approximation ratio) of an approximation algorithm to the optimum is usually established analytically either in the worst-case or on the average (Akande, 2017).

Experimental analyses of heuristicsare usually through several runs (via simulation) against benchmarks.

Examples of approximation algorithms are dispatching rules, heuristics, and metaheuristics or evolution Algorithms.

Heuristics are constructive approximation algorithms that start with no jobs scheduled and gradually construct schedules by adding jobssystematically.Over the last four decades, sequencing and scheduling problems have been solved using heuristics in the form of dispatching or priority rules. The priority or position of a job in the schedule is determined by the job or machine parameter as well as the shop characteristics. Al- Harkan (2013) classified scheduling dispatching rules into local rules, global rules, static rules, dynamic rules among others. Several Dispatching rules have been developed, investigated, and implemented by researchers and practitioners. These include; The Shortest Processing Time (SPT) rule (Smith, 1956; Bansal and Kulkarni, 2015).Modified Due Date (MDD) rule (Baker and Bertrand, 1982; Naidu, 2002), HR9 and HR10 (Oyetunji and Oluleye, 2010), Heuristic AA (Akande, 2018) among others. Special structure problems have also been explored with the aim of using the features to converge to good solutions (Oluleye and Charles-Owaba (1999), Oluleye and Jolayemi, (2000)).

Furthermore, approximation algorithms that are initialized with a complete schedule with the exploration of systematic improvementsachieved by manipulating the current schedule are called metaheuristics or evolution algorithms. Some authors use heuristics and metaheuristics interchangeably.

1.3 Evolutionary Algorithms

Evolutionary Algorithms are based on computational intelligence. They are also called metaheuristics. Metaheuristics are designed to provide good solutions to optimization problems with limited computation capacity (Bianchi, *et al.*, 2009). Methods explorean existing schedule making improvements through manipulations of the optimization problem being solved (Blum and Roli, 2003). Much like heuristics, metaheuristics do not guarantee optimality though they yield better solutions over and above the initially selected schedule (seed).

Evolutionary Algorithms explore local search procedures. The desire is to find a better schedule in the neighborhood.. Two schedules are neighbours, if one can be obtained by modifying the other. The method is performed through iteration. Many neighborhood solutions are generated by modification of the current solution by iteration. The method of modifying the current solutions to form a new neighbor, the acceptance-rejection criterion as well as the termination of the iterations are the basis for the classification of metaheuristics. Examples of metaheuristics include Tabu Search (TS), Simulated Annealing (SA), Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Water Waves Optimization (WWO), League Championship Algorithm (LCA), and Variable Neighborhood Search (VNS) among others. (Interested readers can read more about different heuristics and metaheuristics approaches from the literature)

1.4 Some illustrative examples

Some selected algorithms are to be applied to some real-life/random problems

In this section, two scheduling problems; Single Machine Total Tardiness Problem (SMTTP) with zero release date and the Single Machine Total Flowtime Problem (SMTFP) with non-zero release date are considered. Some existing problems and the corresponding proposed heuristics found in the literature were explored.

1. Single Machine Total Flowtime Problem (SMTFP) with a non-zero release date

Problem Definition: Given a single machine scheduling problem with a set of *n* jobs with minimize of the total flowtime as a performance measure. It is assumed that the problem is deterministic and only one job can be processed at a time.For every job *Ji*, *parameters like the*release time, r_i the processing time p_i are known. Also,*the* start time denoted as $S_i$is given by:

$S_i \geq \quad r_i$ (1)

Also, the completion time of each job (C_i) is defined as:

$C_i = S_i + p_i$ (2)

The flow time of each job defined as the time the job spent in the shop is given as the difference between the completion time and the release date:

$F_i = C_i - r_i$ (3)

Oyetunji (2009) defined the total flow time (F_{tot}) as

$F_{tot} = \sum_{i=1}^{n} F_I = F_1 + F_2 + F_3 + . . . + F_n$ (4)

For the problem of minimizing the sum of flowtime on a single machine with releasedatesOyetunji*et al*, (2012), proposed the KSA 1 heuristic. The steps are as follows:

KSA1 Algorithm Steps

STEP 1: Initialization

Job_Set_A = [$J_1 J_2 J_3$ . . . J_n], set of given jobs

Job_Set_B = [0], set of schedules job

Job_Set_C = [$J'_1 J'_2 J'_3$ . . . J'_n], set of unscheduled jobs,$J'_i = J_i$

n = number of jobs

STEP 2: Find index = $p_i + r_i$ for all jobs in Job_Set_A, i = 1, .., n

STEP 3: List jobs in the Job_Set_A in increasing index order and put the jobs in Job_Set_C. To break ties, select first the job with the lowest r_i, else break tie arbitrarily.STEP 4: Add the first job in Job_Set_A, to Job_Set_B and

remove it from Job_Set_C.

STEP 5: Compute the Completion time, (C_i) of the job scheduled in step 4

STEP 6: Compute $\Delta W_j = |R_j - C_j|$ for all the remaining jobs in the

Job_Set_D. Where $R_j$is the
release date of each of the remaining jobs in Job_Set_D and (C_j is the completion time of jobs scheduled prior to the next target position (i-1) in Job_Set_D, j= 2, 3, … n-1, I = 1, 2, 3, …, n.
STEP 7: Re-arrange the remaining jobs in Job_Set_D in the order of their increasing ΔW_jcomputed in Step 6 and schedule the job with the lowest ΔW_j in the next unscheduled position

STEP 8: Repeat step 6and 7 until all the jobs have been scheduled
STEP 9: Append Job_Set_D to Job_Set_B
STEP 10: Stop
Application

Consider a 4 x1 scheduling problem with the problem parameter in Table 1. Determine the total flowtime using the KSA 2 algorithm. Compare the results to the optimal value.

Table 1: A 4x1 Scheduling Problem

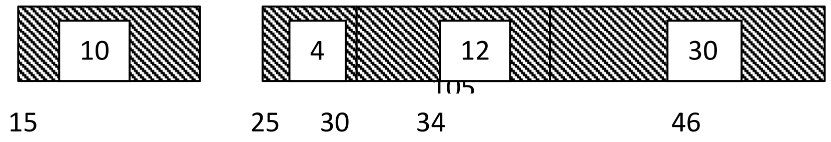
S/N	R	p
1	15	10
2	30	4
3	4	12
4	20	30

Solution.
Optimal value can be obtained by complete enumeration or implicit enumeration. In this case, we want to explore complete enumeration.

Number of Feasible Schedule = 4! = 4 × 3 × 2 = 24

The 24 schedule will be analyzed using the Gantt chart.

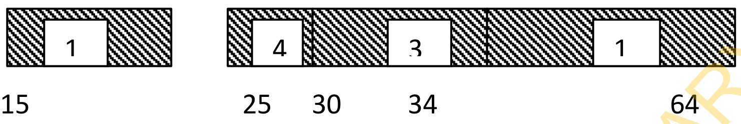
[1 2 3 4]



The Sum of flow time = 10 + 4 + 42 + 56 = 112

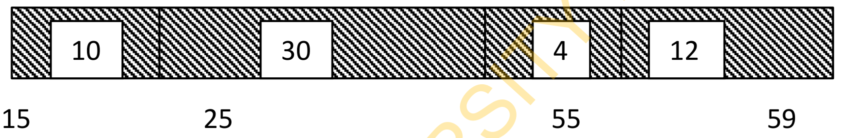
(NOTE:$F_i = C_i - r_i$)

[1 2 43]



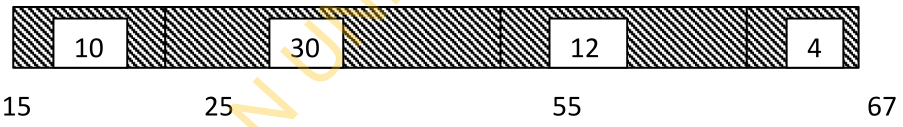
The Sum of flow time = 10 + 4 + 44 + 76 = 134

[1 4 2 3]



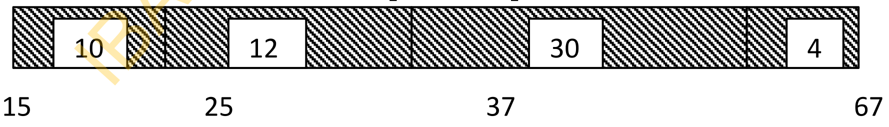
The Sum of flow time = 10 + 35 + 29 + 67 = 141

[1 4 3 2]



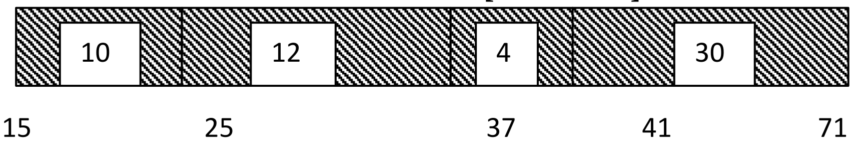
The Sum of flow time = 10 + 35 + 63 + 41 = 149

[1 3 4 2]



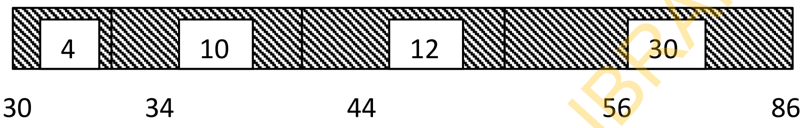
The Sum of flow time = 10 + 33 + 47 + 41 = 131

[1 3 2   4]



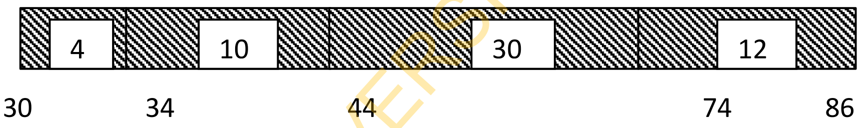
The Sum of flow time = 10 + 33 + 11 + 51 = 105

[2 1 3 4]



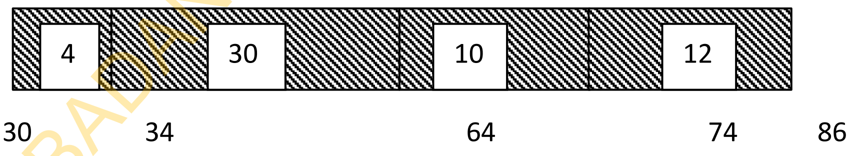
The Sum of flow time = 4 + 29 + 52 + 66 = 151

[2 1 4 3]



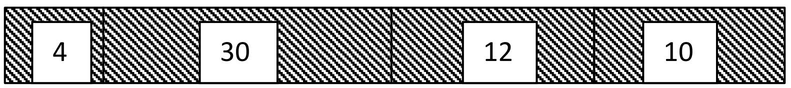
The Sum of flow time = 4 + 29 + 54 + 82 = 169

[2 4 1 3]



The Sum of flow time = 4 + 44 + 69 + 82 = 199

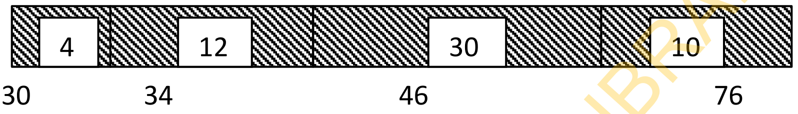
[2 4 3 1]



30 34 6 4 76
86

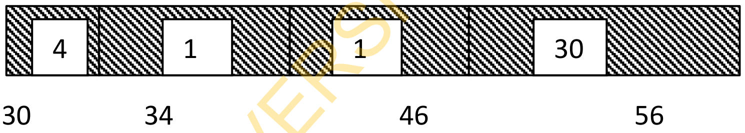
The Sum of flow time = 4 + 44 + 72 + 71 = 191

[2 3 4 1]



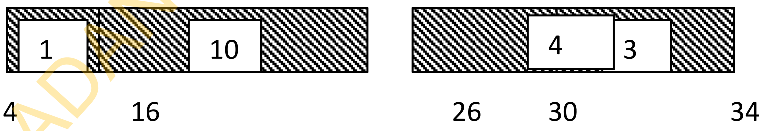
The Sum of flow time = 4 + 42 + 56 + 71 = 173

[2 3 1 4]



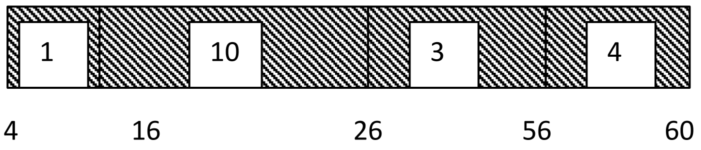
The Sum of flow time = 4 + 42 + 41 + 66 = 153

3 1 2 4]



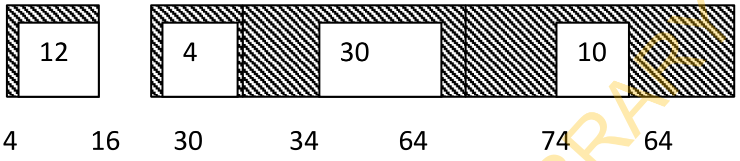
The Sum of flow time = 12 + 11 + 4 + 44 = 71

[3 1 4 2]



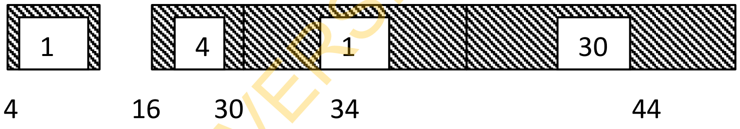
The Sum of flow time = 12 + 11 + 36 + 30 = 89

[3 2 4 1]



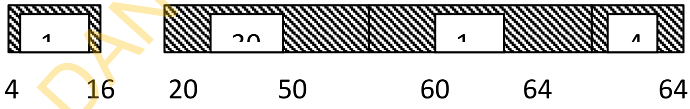
The Sum of flow time = 12 + 4 + 44 + 59 = 119

[3 2 1 4]



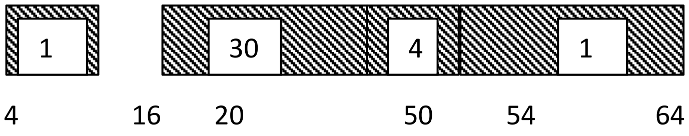
The Sum of flow time = 12 + 4 + 29 + 54 = 101

[3 4 1 2]



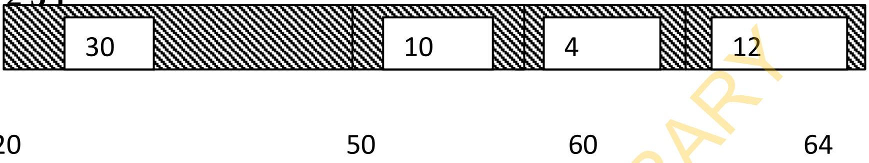
The Sum of flow time = 12 + 30 + 45 + 34 = 131

[3 4 2 1]



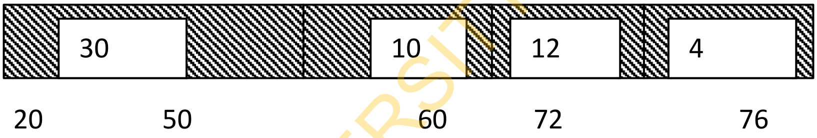
The Sum of flow time = 12 + 30 + 24 + 49 = 111

[4 1 2 3]



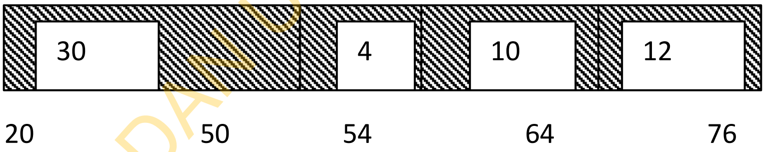
The Sum of flow time = 30 + 45 + 34 + 72 = 181

[4 1 3 2   ]



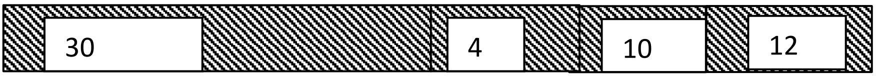
The Sum of flow time = 30 + 45 + 68 + 46 = 189

[4 2 1 3]



The Sum of flow time = 30 + 24 +49 + 72 = 175

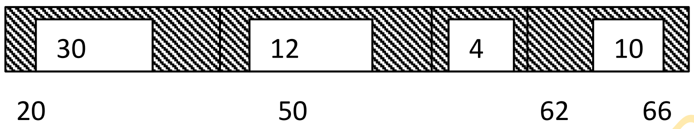
[4 2 3 1]



20 50 54 64 76

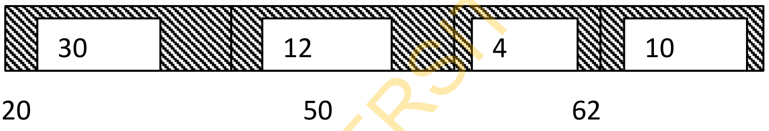
The Sum of flow time = 30 + 24+60 + 61 = 175

[4 3 2 1]



The Sum of flow time = 30 + 58 + 36 + 61 = 185

[4 3 1 2]



The Sum of flow time = 30 + 58 + 51 + 46= 185

From the complete enumeration, the optimal schedule is [3 1 2 4] with the optimal value (total flowtime) of 71

The Proposed Approximation Algorithm: K.S.A1 ALGORITHM

STEP 1: Initialization
Job_Set_A = [1 2 3 4], set of given jobs
Job_Set_B = [0], set of schedules job
Job_Set_C = [1 2 3 4], set of unscheduled jobs,$J'_i = J_i$
n = 4

STEP 2: Compute the index = $p_i + r_i$ for each of the jobs in Job_Set_A, i = 1, .., n

S/N	R	P	P+r
1	15	10	25
2	30	4	34
3	4	12	16
4	20	30	50

STEP 3: Job_Set_C = [3 2 1 4]

STEP 4: Job_Set_C = [2 1 4]

Job_Set_B = [3$- - -$]

STEP 5: Completion time of job 3, (C_i) = (4+12) = 16

STEP 6 : (Job_Set_C = [2 1 4] explore)

: For job 2, ΔW j = | 16–30| = 14

For Job 1 , ΔWi = | 16–15| = 1------------**Minimum (Selected)**

i+3 = Job 4, ΔW j = | 16–20| = 4

Then;

Job_Set_C = [2 4]

Job_Set_B = [3 1 $- -$]

Completion time of job 1, (C_i)= (16 + 10) = 26

STEP 6: (Job_Set_C = [2 4] explore)

: For job 2, ΔW j = | 26–30| = 4 ------------**Minimum (Selected)**

For Job 4, ΔW j = | 26–20| = 6

Job_Set_C = [4]

Job_Set_B = [3 1 2 4]

KSA 1 gives = [3 1 2 4] the optimal schedule.

1. Single Machine Total Tardiness Problem (SMTTP) with zero release date

Problem Definition: Given a single processor scheduling problem, where a set of n jobs have to be sequenced on a processor to minimise the total tardiness. Taking into consideration the following assumption;

i. only one job can be processed at a time

ii. the problem is deterministic that is the processing time (p_i) and the due dates (d_i)

iii. the release dates (r_i) is zero

A job is said to be late or tardy if it is completed after its due date.

The tardiness of job$i$ is given by: T_i = max (0, $C_I - D_i$)
The total tardiness is $T_{tot} = \sum_{i=1}^{n} T_i$

The SMTTP has been established to be NP-hard. One of the most tested effective and efficient heuristics for the problem is the Modified Due Date (MDD) algorithm.

Consider the five-job problem of minimizing total tardiness. Use the MDD solution method and compare the result to the optimal. (Source: Baker and Trietch, 2013)

Job i	1	2	3	4	5
Pi	4	3	7	2	2
Di	5	6	8	8	17

SOLUTION

In this case, the number of feasible schedules is = 5! = 5 × 4 × 3 × 2 = 120

This will take a prohibitive computation time using the complete enumeration. Thus, Branch and Bound implicit enumeration Techniques will be employed to find the optimal.The branching tree is as in Figure 2

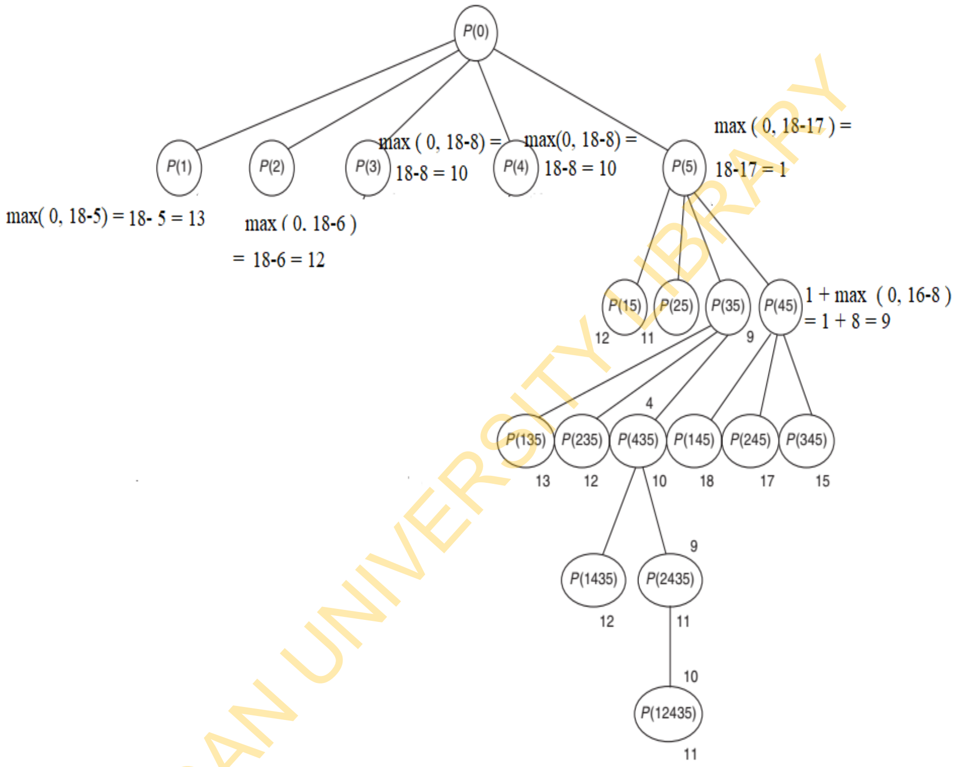


Fig. 2: The branching Tree

Analysis of the branching tree

At the step 1, the tree consists of *P*(0), with no job schedule.

At the step 2, the problem p(0) was partitioned into *n* subproblems, p(1), p(2), . p(3), p(4), p(5), by assigning the last position in the sequence to each of the nodes in the first level of the branching tree. For the subproblems, put each associated job in the last position sequentially. That is, for p(1), put job 1 last, for p(2) put job 2 last, etc. The tardiness for each job at the last position is computed as follows:

For p [- - - - 1],: T_i = max (0, $C_I - D_i$) = max (0, $18 - 5$) = 13
For p[- - - -2] : T_i = max (0, $C_I - D_i$) = max (0, $18 - 6$) = 13

For p [- - - - 3],: T_i = max (0, $C_I - D_i$) = max (0, $18 - 8$) = 10

For p[- - - - 4],: T_i = max (0, $C_I - D_i$) = max (0, $18 - 8$) = 10

For p [- - - - 5],: T_i = max (0, $C_I - D_i$) = max (0, $18 - 17$) = 1

NOTE (The completion time will be the summation of all the processing time since the last position of the job is assigned)

To eliminate some redundant branches, only the branch with the minimum tardiness value is explored further.At the next stage, the remaining jobs (1, 2, 3, 4) are assigned the position,$n - 1$. The tardiness of each of the sub-problem is computed as follows

For p[- - - 45],: T_i = 1 + max (0, $C_I - D_i$) = 1 + max (0, $16 - 8$) = 9

For p[- - - 35],: T_i = 1 + max (0, $C_I - D_i$) = 1 + max (0, $16 - 8$) = 9

For p[- - - 25],: T_i = 1 + max (0, $C_I - D_i$) = 1 + max (0, $16 - 6$) = 11

For p[- - - 15],: T_i = 1 + max (0, $C_I - D_i$) = 1 + max (0, $16 - 5$) = 12

Also, the p[- - - 45] and the p[- - - 35] are explored further. This process continues until all the explored branches were explored as illustrated in Fig. 1. The optimal solution from the Branch and Bound is **11** and the schedule is (12435).

The Modified Due Date (MDD) heuristics for the Problem

The Modified Due Date (MDD) Rule: MDD schedules the next job from unscheduled jobs set 'U' with the smallest priority index (Πi). The priority index is given by:

Π_i = {max$\{t_i + \ p_i \ , d_i\}$}

where:
t_i is the starting time of the next unscheduled job $i(i \in U)$ which can either
be the completion time of the job in position i-1 or the release date of job i,
$p_i$is the processing time, and
d_i is the due date.
If two jobs j and k compete to be scheduled at time t, then, job j will precede
job k if $\Pi_j \leq \Pi_k$

However, the MDD rule does not consider two jobs at a time when there are more than
two unscheduled jobs. It considers all the available jobs, computes their priority indices
(Π_i) and chooses the job with the least priority index.

STEP 1: Initialization
Job_Set_A = [1 2 3 4], set of given jobs
Job_Set_B = [0], set of schedules job
Job_Set_C = [1 2 3 4], set of unscheduled jobs, $J'_i = J_i$
n = 5

For i = 1, t = 0, JobSET B = {}

Job *i*	1	2	3	4	5
Pi	4	3	7	2	2
Di	5	6	8	8	17

$t_i + p_i$	4	3	7	2	2
Π_i = {max{$t_i + p_i$, d_i}}	5	6	8	8	17

The minimum Π_i = 5, Thus, JobSET B = {1}

For i = 2, t = 5, JobSET B = {1}

Job *i*	2	3	4	5
Pi	3	7	2	2
Di	6	8	8	17
$t_i + p_i$	8	12	7	7
Π_i = {max{$t_i + p_i$, d_i}}	8	12	8	17

The minimum Π_i = 8, Thus, JobSET B = {1 2} or JobSET B = {1 4}

Though, the MDD does not specify how the tie should be broken. The common approach is to break the tie with the due date (by assigning jobs with lower date) as explored by Akande (2017).

Thus, job 2 is scheduled.

For i = 3, t = 8, JobSET B = {1 2}

Job *i*	3	4	5
Pi	7	2	2
Di	8	8	17
$t_i + p_i$	15	10	10
Π_i = {max{$t_i + p_i$, d_i}}	15	10	17

The minimum Π_i = 15, Thus, JobSET B = {1 2 4}

For i = 4, t = 10, JobSET B = {1 2 4}

Job i	3	5
Pi	7	2
Di	8	17
$t_i + p_i$	17	12
Π_i = {max$\{t_i + p_i \ , d_i\}$}	17	17

The minimum Π_i= 17, (break the tie by assigned the job with the lower due date) Thus, JobSET B = {1 2 4 3}
For i = 5, JobSETB = [1 2 4 35]   MDD heuristic schedule is [1 2 4 35]

Job i	1	2	4	3	5
P_I	4	3	2	7	2
C_I	4	7	9	16	18
Di	5	6	8	8	17
T_i = max (0, $C_I - D_i$)	0	1	1	8	1

$$T_{tot} = \sum_{i=1}^{n} \max (0, C_I - D_i) = 11$$

1.5 Future directions

After about seven decades of active researches on scheduling which have resulted in the development of many scheduling algorithms, the followings are some of the areas where future research efforts are of utmost importance.

i. **Quality of solutions (Effectiveness):** Owing to advancements in the information communication technology over the years, there have been tremendous improvements in the computing speed. Thus, it is of utmost importance for researchers to concentrate more efforts on the developments of algorithms that are capable of generating solutions that are extremely close to (if not) the optimal. It is believed that we can always leverage improvements in computing power/speed.

ii. **Execution time of solutions (Efficiency):** Even though there have been improvements in the computing speed/power over the years, there is a need for researchers to continue the search for faster/shorter methods of solving combinatorial/optimization problems. The world itself is not static, hence researchers should be encouraged to continue to explore the development of fast (efficient) algorithms that can produce results if possible at the speed of light.

iii. **Multi-criteria/Multi-objective problems:**Since most combinatorial/scheduling problems are mostly multi-criteria in nature, research efforts should be majorly focused on developments of algorithms that can be applied to multi-criteria problems. Today, a number of the algorithms purportedly developed for multi-criteria scheduling problems reduce the original problems into single criterion problems. There is the need to further develop algorithms that will explore multi-criteria problems in a multi-criteria manner and not pseudo-multi-criteria manner.

iv. **Real-life scheduling problems:** Although many researchers have explored multi-criteria scheduling problems, some of these have been limited to hypothetical problems, with only a few exploring real-life problems. Since real-life scheduling problems have their own unique characteristics, future research efforts should therefore be directed at solving real-life problems that are of practical importance to the society at large.

1.6 Conclusion

In this work, sequences and scheduling have been introduced with time being used as an underlying surrogate measure for cost factors.

Algorithms that enable the solution of scheduling problems have been classified andmethodologies espoused. Exact methods have been distinguished from approximation methods. This enables a tradeoff between accuracy and timeliness. Examples have been given to enable readers to have traction with implementing some selected solution methods. This should generate more interest in applying scheduling approaches to improve the performance of individual and organisational work systems.

References

1. Akande, S. (2018). New solution methods for single machine bicriteria scheduling problems: Minimizing the total flowtime and maximum tardiness with zero release dates.*International Journal of Engineering Research in Africa*, 38, 144-153.
2. Akande, S. (2017). Development of heuristics for single processor scheduling problems with tardiness-related performance measures. *Statistics*, 1.
3. Al-harkan, I. *Algorithms for Sequencing and Scheduling.* Available from: faculty.ksu.edu.sablog(http://http://faculty.ksu.edu.sa/ialharkan/IE428) retrieved 15 September, 2020.
4. Baker, K.R and Trietsch, D. (2013). Principles of sequencing and scheduling.Canada: John Wiley & Sons
5. Bansal, N. and Kulkarni, J. (2015). Minimizing flow-time on unrelated machines. *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*. Portland. 92-102.
6. Bellman, R, (1957). Dynamic Programming. Princeton : University Press.
7. Bianchi, L., Marco, D., Luca, M.G., and Walter, J. G. (2009). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing: an International Journal*, 8, 239–287.

8. Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization:
Overview and conceptual comparison. *ACM Computing Surveys*, 35, 268–308.
9. Brucker, P. (2007) . *Scheduling Algorithms*, 5th ed. Springer: Heidelberg
Publisher.
10. French, S. (1982). Sequencing and Scheduling. United Kingdom : Ellis Horwood Limited.
11. Johnson, S.M. (1954) . Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics*, 1 : 61-68.
12. Naidu J.T. (2003). A Note on a Well-Known Dispatching Rule to Minimise Total Tardiness. *International Journal of Management Science* 31, 137–140.

13. Oluleye, A E and Charles-Owaba, O E, (1999). Optimality Conditions for Some Special structure Scheduling Problems, *Proceedings of the International Conference on Production Research (ICPR-15), Ireland, UK* 333 - 336.
14. Oluleye, A E and Jolayemi J K, 2000, Performance Evaluation of Some Index Based Flowshop Heuristics. *Nigerian Journal of Engineering Management* 44 – 48.
15. Oyetunji, E. O., Oluleye, A. E. and Akande, S.A. (2012) . Approximation
algorithms for minimizing sum of flow time on single machine with release
dates. *International Journal of Modern Engineering Research*,2, 687-696.
16. Oyetunji, E.O., (2009) . Some common performance measures in scheduling
problems. *Research Journal of Applied Science, Engineering and Technology*,2, 6-9.
17. Oyetunji, E.O. and Oluleye, A.E. (2010). Hierarchical minimization of total

completion time and number of tardy jobs criteria. *Asian Journal of Information Technology*,74, 157-161

18. Oyetunji, E. O. and Oluleye, A.E. (2008) . Heuristics for minimizing total completion time and number of tardy jobs simultaneously on single machine with release time. *Research Journal of Applied Sciences*, 3, 147-152.
19. Ólafsson, S. (2002), IE 514 Lecture notes, Industrial Engineering Department, Iowa State University http://www.public.iastate.edu/%7Eolafsson/ie514_2001.html
20. Smith, W. E., (1956). Various Optimisers for single-stage production. *Naval Research Logistic Quarter*,31, 59-66.
21. Weisstein, E W. Complexity Theory. A Wolfram Web Resource. http://mathworld.wolfram.com/ComplexityTheory.html [Accessed: June, 21st 2015].