Journal of Quality Engineering and Production Optimization

Vol. 6, No. 2, Summer & Autumn 2021

http://jqepo.shahed.ac.ir

Research Paper

DOI: 10.22070/JQEPO.2016.348

A new algorithm for solving the parallel machine scheduling problem to maximize benefit and the number of jobs processed

Mohammad Taghi Rezvan [1*], Hadi Gholami [2], Reza Zakerian [3]

[1] *Department of Industrial Engineering, Faculty of Engineering, University of Kashan, Kashan, Iran*

[2] *Department of Computer Engineering, Ayatollah Amoli Branch, Islamic Azad University, Amol, Iran*

[3] *Department of Computer Engineering, Faculty of Mahmoudabad, Technical and Vocational University, Mazandaran, Iran*

**** Corresponding Author:*** *Mohammad Taghi Rezvan (Email: rezvan@kashanu.ac.ir)*

Abstract *– This paper provides a mathematical model and a bi-phase heuristic algorithm for the uniform parallel machines scheduling problem to maximize benefits and the number of jobs processed before their due dates as the weighted objective function. In the first phase of this heuristic, named "the neighborhood combined dispatching rules algorithm" (NCDRA), an initial sequence by the segmentation of the dispatching rules (DRs) is generated. Then, the output sequence is segmented, and required efforts are made to derive a sequence combined with these rules to improve the objective. The second phase involves a local search in which operators such as swapping, insertion, and reversion are concurrently implemented there on. The proposed algorithm is examined on four classes of problems with 50, 100, and 1000 jobs on 5, 10, and 50 machines, respectively. Results obtained by NCDRA and a Simulated Annealing (SA) algorithm developed on problem instances indicate that the NCDRA provides high-quality results on objective function for solving problems in different scales.*

Keywords*– Uniform Parallel machines, Benefit, Number of jobs processed, Heuristics.*

I. INTRODUCTION

Manufacturing resources are limited in capacity. In the modern competitive world, effective scheduling is of prime importance for achieving desirable production capacities because a proper production schedule will ensure rapid response to customer demands, timely supply of plant raw material and spare part requirements, reduced machine idle times, and decreased waste, all of which ultimately lead to improved plant productivity (Phanden et al. 2013).

The general parallel machine scheduling problem is essential because each workstation in this problem might involve several identical machines. It involves a set of independent jobs on parallel machines or processors that operate toward a certain objective of job completion time. The set of jobs are independent of each other, and each is to be processed on one of the machines available. Furthermore, each machine is capable of performing one job at a time, and each job can be processed only on one machine. Most research on parallel machine scheduling is concentrated on the optimization of a single criterion such as total/mean completion time, total weighted completion time, makespan and total tardiness penalties, the total number of tardy jobs, the number of jobs completed, and total revenue obtained from

Manuscript Received:17- August -2021 & Revised: 3- Nov -2021 & Accepted: 5- December -2021
ISSN: 2423-3781

completed jobs (Pinedo, 2012). To reflect the needs of the competitive environment, need to be considered more than one criterion since they are generally contradictory.

The problem addressed in this paper involves a set of uniform parallel machines to process a set of jobs. The attributes of each job are the release time, due date, and value/revenue called benefit in this paper. The processing time of the job is the time interval between its release time and due date. The processing of each job on a machine can be started at its release time and completed until its due date. In this case, the benefit of a job is obtained.

The motivation of this paper is to provide a practical foundation in permutation-based problems for researchers to improve the upper bound of the problems in a simple way. In cloud computing, the number of completed jobs is the desire of the client and affects the service provider's brand at long-term objective while the benefit is fairly a short-term objective of a service provider. Since increasing benefits is not necessarily in line with enhancing the number of jobs completed, service providers must balance between two objectives to survive in a competitive environment (Huang et al. 2020). The berth allocation and quay cranes scheduling problems in container terminals are the most important issues that should be considered by port managers (Hoseini et al., 2018). Optimizing the allocation of berths and quay cranes to incoming ships, in addition to the satisfaction of shipping liners, will also lead to more revenue for container terminals.

Previous studies on the parallel machine scheduling problem with objectives of revenue and the number of jobs processed as separate and simultaneous are limited to several below researches. Rojanasoonthon et al. (2003) proposed a heuristic based on dynamic programming (DP) and a greedy randomized adaptive search (GRAS) procedure for the uniform parallel machines scheduling and the objective of maximizing the weighted number of jobs processed with strict priority enforcement. Bard and Rojanasoonthon (2006) developed a branch-and-price algorithm for the uniform parallel machines scheduling problem with multiple time windows and job priorities to maximize the weighted number of jobs completed. Islam et al. (2008) proposed a heuristic algorithm based on urgency and processing times of jobs for the parallel jobs scheduling problem in a multi supercomputer centers system to maximize revenue gained by the resource provider. Juraszek et al. (2009) modeled the problem of scheduling identical parallel machines to maximize revenues and solved it using a simulated annealing (SA) algorithm. They also compared their results with those obtained from the branch & bound algorithm and the list scheduling approach. Chung et al. (2009) proposed mathematical programming and two network algorithms for the identical parallel machines scheduling problem with sequence dependence setup time, product-type-dependent processing time, machine capacity, and multiple product profit to maximize the total profit in the thin film transistor liquid crystal display manufacturing industry. Gholami et al. (2019) solved the problem of scheduling uniform parallel machines to maximize revenues and the number of jobs processed using such metaheuristics as genetic algorithm (GA), Tabu Search, SA. They tested their algorithms on large-size problems with up to 50 machines and 500 jobs. Croce et al. (2021) analyzed Parallel machine scheduling with the minimum number of tardy jobs and solved it by exponential-time approximation algorithms and fixed-parameter tractable exact algorithms.

Some algorithms proposed for parallel machine scheduling problems use dispatching rules (DRs). To schedule jobs on identical parallel machines to minimize the makespan, Chen and Vestjens (1997) proposed the longest process time (LPT) rule, and Laha and Gupta (2018) applied the LPT rule long with the job-interchange mechanism to generate the initial population of an improved cuckoo search algorithm (ICSA). Yang-Kuei and Chi-Wei (2013) employed DRs for unrelated parallel machine problems with release times to obtain rapid optimum solutions. These rules had suitable performance for problem instances of all sizes. Joo and Kim (2015) used three DRs in hybrid GAs for the unrelated parallel machine scheduling problem with sequence-and machine-dependent setup time to minimize the total completion time. Lee (2018) developed a DR and an iterative greedy meta-heuristic for scheduling identical parallel machines to minimize total weighted completion time. Due to the proper performance of DRs, this paper considers a set of well-known DRs and puts together different DRs based on their efficiency at each segment solution.

For parallel machine scheduling problems, local search (LS) techniques are sometimes used combined with other

algorithms to improve their effectiveness. Fanjul-Peyro and Ruiz (2010) proposed two consecutive simple LS named insertion and interchange to enhance the objective function. Mensendiek et al. (2015) introduced a new mechanism for LS by combining shift and swap moves and a one-point crossover operator. Their algorithm examines the neighborhood of a solution until no further improvement is achieved. Bitar et al. (2016) developed the mutation operator and the selection of jobs randomly based on uniform law. Kowalczyk and Leus (2017) presented a scheme based on the swap procedure. This scheme keeps the jobs processed on each machine in separate lists. It then puts the job of each list in another list until the objective function is improved. Nattaf et al. (2019) proposed a neighborhood structure based on two different ways of intra-change insertion and inter-change insertion that selection and insertion of jobs are random. Thus, the use of random-based methods for selecting and moving jobs and machines is presented in the literature. The use of three co-operative operators that randomly create neighbors is a scheme used in this paper to avoid a local optimum. The literature reviewed in this study is summarized in Table I concerning the type of objective functions parallel machines, scheduling constraints, the solution method, and the scale of the solved problem.

Based on the first section of Table I, the research on parallel machines scheduling problem with the objectives of maximizing benefits and the number of jobs processed is still rare, and also their performance on different problems show the proposed algorithms are not effective on large scale problems. Thus, the development of algorithm outperformed needs to be further studied. Based on the second section of Table I, DRs and LS are widely used in researches. This paper presents an innovative algorithm by combining DRs and exploiting LS as well as considering subtleties such as filling in the gaps between jobs. This algorithm is a heuristic algorithm which, unlike meta-heuristic algorithm such as SA, yield more accurate solutions and also it generates one specific solution in each run.

This paper proposes a mathematical model and a heuristic bi-phase algorithm to maximize the benefits and the number of jobs processed simultaneously that exhibits an acceptable efficiency despite its simplicity. The algorithm in its first phase employs well-known DRs, adopts a particular method for segmentation each DR, selects the segment of the highest quality from each rule that suits the objective function, and sequences the segments in order to derive a strategy that leads to solutions superior to those obtained from the DRs. The second phase involves the concurrent implementation of three cooperate operators and their contribution to improving the solution. To assign the jobs to the machines, it also applies policies to improve the objective function.

The main contribution is to propose a heuristic algorithm that differs significantly from the literature methods by not repeating the phases, and each of them is executed only once. In this way, in addition to providing competitive results, it can also save run-time. This work presents an idea to check the sequences of schedules and separate the best sub-sequences at different time intervals as an optimal local solution and combine them as a new schedule. Furthermore, a procedure is developed for online filling of the gap between jobs in order to improve criteria in multi-objective problems.

Table I. Comparisons among the related literature and this paper

	Publications (year)	***Type Objective(s)***	***Parallel Machine types***	***Constraints***	***Algorithms***	***Performance on Problem Scale***
Problem viewpoint	Rojanasoonthon et al.(2003)	Max Weighted number of jobs processed	Uniform	Strict priority enforcement	DP & Greedy Randomized Adaptive Search	Up to 6 machines and 418 jobs
	Bard and Rojanasoonthon (2006)	Max Weighted number of jobs processed	Uniform	Multiple time windows and job priorities	Branch & Price	Up to 6 machines and 100 jobs
	Islam et al. (2008)	Max Revenue	Identical	Dynamic arrival time of jobs	Heuristic	Up to 10000 jobs
	Juraszek et al. (2009)	Max Revenue	Identical	Release time and due date of jobs	MIP and SA	Up to 20 machines and 300 jobs

Continue Table I. Comparisons among the related literature and this paper

	Publications (year)	***Type Objective(s)***	***Parallel Machine types***	***Constraints***	***Algorithms***	***Performance on Problem Scale***
Problem viewpoint	Chung et al. (2009)	Max Profit	Identical	Sequence dependence setup time, product-type-dependent processing time, machine capacity, and multiple product profit	MIP and two network algorithms	Up to 120 jobs
	Gholami et al. (2019)	Max Revenues & The number of jobs processed	Uniform	Release time and due date of jobs	GA, TS, & SA	Up to 50 machines and 500 jobs
	Croce et al. (2021)	Min Number of tardy jobs	Identical	Due date of jobs and non-preemptively	Approximation and exponential algorithms	-
Solution methodology viewpoint	Chen and Vestjens (1997)	Min Makespan	Identical	Release times	Online LPT rule	The number of machines and jobs is unknown
	Fanjul-Peyro, Ruiz (2010)	Min Makespan	Unrelated	Dynamic arrival time	Iterated greedy LS	Up to 50 machines and 1000 jobs
	Yang-Kuei and Chi-Wei (2013)	Min Makespan & Total weighted tardiness	Unrelated	Release times	DRs	Instances of all sizes
	Joo and Kim (2015)	Min Total completion time	Unrelated	Sequence-and machine-dependent setup time	3 DRs in hybrid GAs	Up to 10 machines and 100 jobs
	Mensendiek et al. (2015)	Min Total tardiness	Identical	Fixed delivery dates	Hybrid GA and TS	Up to 5 machines and 50 jobs
	Bitar et al. (2016)	Min The weighted flow time & Max the number of jobs processed	Unrelated	Auxiliary resources	LS in Memetic algorithm	Up to 8 machines and 200 jobs
	Kowalczyk & Leus (2017)	Min Makespan	Identical	Conflicting jobs	Exact algorithm	Up to 20 machines and 100 jobs
	Laha and Gupta (2018)	Min Makespan	Identical	Setup times of jobs	Improved cuckoo search	Up to 10 machines and 100 jobs
	Lee (2018)	Min Total weighted completion time	Identical	Setup adjustment on each machine and due date of jobs	DR and an iterative greedy	Up to 15 machines and 300 jobs
	Nattaf et al. (2019)	Min completion times & The number of machine disqualifications	non-identical	Time constraints on machine qualifications	MIP and two improved heuristics	Up to 5 machines and 70 jobs
Our Paper		Max Revenues & The number of jobs processed	Identical & Uniform	Release time and due date of jobs	Heuristic based on DR, segmentation, and LS	Up to 50 machines and 1000 jobs

The rest of the paper is organized as follows. Section 2 presents the problem formulation with a mixed integer programming (MIP) model. Section 3 develops the algorithm for solving the problem and presents its specifications to be supplemented with small illustrative examples that provide a better understanding of the proposed algorithm. Section 4 describes the specifications of the problem instances, tunes the algorithm's control parameters, and examines the performance of the components of the algorithm on problem instances. Moreover, the computational results obtained from the algorithm developed in this study are compared with those obtained from SA. Finally, conclusions will be provided in Section 5.

II. PROBLEM FORMULATION

Scheduling problems are designated by the standard three-part naming system as in $\alpha/\beta/\gamma$, in which the initial α refers to the type of scheduling problem, β denotes the scheduling constraints and details, and γ represents the objective function (Graham et al. 1979). The problem investigated in this paper is denoted by $Q/r_j, d_j/w_1 * \sum b_j + w_2 * |J(S)|$ where r_j and d_j are release time and due date, respectively. $\sum b_j$ denotes the total benefit for processing jobs and w_1 is its weight. $|J(S)|$ denotes the number of jobs, $J_j \in J(S)$ processed within their intervals $[r_j, d_j]$ in schedule S and also w_2 is its weight. In fact, this problem involves a set of m parallel uniform machines $M = \{M_1, M_2, \dots, M_m\}$ that differ in their speeds so that the speed of M_i is denoted by s_i. Let us assume that there is a set of n independent jobs, $J = \{J_1, J_2, \dots, J_n\}$, and that each job has a processing time of $p_j > 0$. If J_j is processed on M_i, it takes $p_{i,j} = p_j/s_i$ actual time units to complete the process. In fact, $p_{i,j}$ is the actual processing time of job j if it is assigned to machine i. Each job j also requires a single operation with a release time of $r_j \geq 0$ and a strict due date of $d_j > r_j$. If job $J_j \in J$ is processed within the closed interval $[r_j, d_j]$, then a benefit b_j is earned. Since no preemption is allowed in the processing of a job, the completion time, $c_{i,j}$, of job J_j on machine M_i may be calculated as follows: $c_{i,j} = st_{i,j} + p_{i,j}$; where, $st_{i,j}$ represents the start time of job j on machine i.In this paper, jobs cannot be interrupted during their processing and $C(S)$ denotes the makespan of schedule S.

Assume that $M = \{M_1, M_2\}$ and $CS_1(S)$ and $CS_2(S)$ denote the set of all jobs processed on M_1 and M_2, based on their completion times, respectively. If $c_1 = c_{1,1} \leq c_{1,3} \leq \cdots \leq c_{1,n-1}$ and $c_2 = c_{2,2} \leq c_{2,4} \leq \cdots \leq c_{2,n}$, then the critical sequence of schedule S can be shown as follows: $CS(S) = \{J_2, J_1, J_3, J_4, J_n, J_{n-1}\}$, where $c = c_{2,2} \leq c_{1,1} \leq c_{1,3} \leq c_{2,4} \leq \cdots \leq c_{2,n} \leq c_{1,n-1}$, and all the jobs $J_j \in J$ are processed within their intervals $[r_j, d_j]$ in schedule S. In M_1, given that $c_{1,1} \leq st_{1,3}$, it is said that J_1 is in the first position and J_3 is in the second position. According to this, C_j and $C_{i,k}$ represent the completion time of job j and that of the one on the k^{th} position of machine i, respectively. $|J(S)|$ indicates the number of jobs performed in schedule S.

In the following, a mixed-integer programing (MIP) model is presented for solving small-sized instances of the problem. In this model, binary variables x_{ijk}, represent the fact of assigning job j to machine i on k^{th} position. st_{ik} and PT_{ik} are the starting times and the processing time of k^{th} position on machine i, respectively. Binary variables u_j show the tardy jobs which are not run on any machine. Besides, n_i is the number of jobs assigned to machine i and M is a large number.

$$Maximize\ \ obj1 = \sum_{j=1}^{n} b_j(1 - u_j) \tag{1}$$

$$Maximize\ \ obj2 = \sum_{j=1}^{n} (1 - u_j) \tag{2}$$

$$\sum_{i=1}^{m}\sum_{k=1}^{n_i} x_{ijk} \le 1 \qquad \forall j = 1, \dots, n \tag{3}$$

$$\sum_{j=1}^{n} x_{ijk} \le 1 \qquad \forall i = 1, \dots, m\,;\ k = 1, \dots, n_i \tag{4}$$

$$\sum_{j=1}^{n} r_j * x_{ijk} \le \acute{s}t_{ik} \qquad \forall i = 1, \dots, m\,;\ k = 1, \dots, n_i \tag{5}$$

$$\acute{s}t_{ik} \le \sum_{j=1}^{n} x_{ijk} * \left(d_j - p_{ij}\right) \qquad \forall i = 1, \dots, m\,;\ k = 1, \dots, n_i \tag{6}$$

$$PT_{ik} = \sum_{j=1}^{n} p_{ij} * x_{ijk} \qquad \forall i = 1, \dots, m;\ k = 1, \dots, n_i \tag{7}$$

$$\acute{s}t_{ik} + PT_{ik} \le \acute{s}t_{i(k+1)} \qquad \forall i = 1, \dots, m;\ k = 1, \dots, n_i \tag{8}$$

$$\sum_{i=1}^{m}\sum_{k=1}^{n_i} x_{ijk} + u_j = 1 \qquad \forall j = 1, \dots, n \tag{9}$$

$$C_j \ge \acute{s}t_{ik} + PT_{ik} + M * (x_{ijk} - 1) \qquad \forall i = 1, \dots, m\,;\ j = 1, \dots, n;\ k = 1, \dots, n_i \tag{10}$$

$$C_j - d_j \le M * u_j \qquad \forall j = 1, \dots, n \tag{11}$$

$$\begin{aligned} &x_{ijk} \in \{0,1\} && \forall i = 1, \dots, m\,;\ j = 1, \dots, n;\ k = 1, \dots, n_i \\ &\acute{s}t_{ik}, \quad PT_{ik} \ge 0 && \forall i = 1, \dots, m\,;\ k = 1, \dots, n_i \\ &C_j \ge 0 && \forall j = 1, \dots, n \\ &u_j \in \{0,1\} && \forall j = 1, \dots, n \end{aligned} \tag{12}$$

The objective functions (1) and (2) maximize the total benefit for processing jobs ($\sum_{J_j \in \mathcal{J}(\mathcal{S})} b_j$) and the number of jobs processed ($|\mathcal{J}(\mathcal{S})|$), respectively. Constraints (3) guarantee each job is not assigned to more than one position on each machine. Constraints (4) ensure that each position can be occupied on each machine by at most one job. Constraints (5) and (6) define the processing times for each job is started after its release time and finished before its deadline. Constraints (7) describe the processing time of each position on each machine. Constraints (8) ensure that two consecutive jobs do not overlap. Constraint (9) determines the job is tardy that it does not have a position on one machine. The completion time of each job is computed by constraints (10). Constraints (11) specify if each job is a tardy job or not. Finally, constraints (12) establish the binary restrictions x_{ijk} and u_j and also determine the non-

negativity of $\acute{st}_{ik}$, PT_{ik} and C_j.

Maximizing total revenue at identical parallel machine scheduling problem is NP-hard (Juraszek et al. 2009); thus, the investigated problem is also NP-hard. Using this formulation, it could be solved using CPLEX of GAMS software for very small-sized instances.

III.THE PROPOSED ALGORITHM

In this section, an algorithm called the neighborhood combined dispatching rules algorithm (NCDRA) with the general structure shown in Figure 1 is proposed for solving the problem $Q/r_j, d_j/w_1 * \sum b_j + w_2 * |J(S)|$.

Phase I:
Apply the dispatching rules as described in Section 3.1.1
Apply the best segment selector procedure as described in Section 3.1.3
Phase II:
While termination condition has not been satisfied **Do**
Apply the neighborhood search as described in Section 3.2
End-While

Figure 1. The general structure of NCDRA

NCDRA consists of two phases. The first focuses on determining an initial sequence of jobs, hereafter called "the sequence search (SS) phase", which itself comprises two stages. The first stage involves segmentation of jobs; in each segment, the DRs are used to generate an initial scheduling sequence for each DR. In the second stage, a new sequence is generated by selection and aggregation of the best segments by applying the best segment selector (BSS) procedure on sequences obtained in the first stage.

The sequence of jobs resulting by Phase I probably corresponds to a local optimum. In the second phase, it is supposed to find a better sequence in the neighborhood of this optimum. This way, the algorithm tries to achieve a global solution in the second phase. Hence, phase II is called the "local search (LS) phase" whose general goal is to drive the local optimum obtained in Phase I toward a global optimum. The limitation of the proposed algorithm is that it requires information that is not available in dynamic schedulers. In fact, the proposed algorithm is a static scheduler.

A. Sequence search phase

The initial sequence plays a vital role in accelerating the convergence of the algorithms used in solving scheduling problems because it saves the time spent in searching problem spaces that do not ultimately lead to improved solutions (Nattaf et al., 2019). From their entry, jobs may be classified according to such specifications as release time, due date, benefit, and process time. Once the non-identical DRs are combined, a proper sequence may appear at a point in the problem space different from previously detected ones. It follows then that phase I reduces the search space for the optimal solution and considerably improves the algorithm's effectiveness by finding the upper bound.

1. Dispatching Rules

Dispatching Rules (DRs) are highly desirable in the field of scheduling for allocating unscheduled jobs to machines. The different types of these rules are based on processing time (PT), due date (DD), neither DD nor PT, and combinations thereof. There exist 11 DRs, alternatively called 'schedulers', as listed below, out of which Nos. 9 through 11 are dynamic schedulers (Raghu and Rajendran, 1993) and the rest are static schedulers (Kaban et al., 2012):

1. Earliest Start Time (EST), according to which jobs are sorted in ascending order of their release times, r_j.
2. Earliest Finish Time (EFT), according to which jobs are sorted in ascending order of their processing times, p_j.
3. Longest Benefit First (LBF), according to which jobs are sorted in descending order of their benefits, b_j.
4. Longest Processing Time First (LPF), according to which jobs are sorted in descending order of their processing times, p_j.
5. Earliest Due Date First (EDF), according to which jobs are sorted in ascending order of their due dates, d_j.
6. Ascending order of Due Date Minus Release Time (ADMR), according to which jobs are sorted in ascending order of the result of their due dates minus their release times, $d_j - r_j$.
7. Descending order of Due date Minus Release time (DDMR), according to which jobs are sorted in descending order of the result of their due dates minus their release times, $d_j - r_j$.
8. Maximum Benefit and Minimum of Release time minus Processing time (MBMRP), according to which jobs are sorted in descending order of $\max\{b_j\}$ && $min\{r_j - p_j\}$.
9. Largest Benefit Minus Processing time (LBMP), according to which jobs are sorted in descending order of the results of their benefits minus their processing times $b_j - p_j$.
10. Min Slack, according to which jobs are sorted in ascending order of $d_j - C_{i,n_i} - p_j$.
11. Max Slack, according to which jobs are sorted in descending order of $d_j - C_{i,n_i} - p_j$.

These rules are executed to select jobs for processing. In case a rule assigns the same priority to two jobs, then selection between the two competing jobs will be based on either minimum due date or maximum benefit. The segmentation method (SM) was used to improve most of the DRs in terms of their performance toward the maximized benefit and the number of jobs processed. In fact, SM inspired by the divide-and-conquer method shows its effectiveness. However, the advantages of segmentation in DRs can be exploited in cases with not too few jobs. Since the number of jobs and their specifications are given in advance of the scheduling process, SM places jobs in segments according to their release times, and the segments are submitted to machines according to the current DR. The number of jobs in each segment is designated by n_{jb}.

As an illustrative example, let us assume that our proposed algorithm is to schedule 50 jobs in a system. If further SM is to place them in 10 segments (n_{jb} =10), there will then exist five segments. It might, however, be the case that each segment length fails to accommodate exactly 10 jobs since batching jobs are based on their release times so that if two jobs have the same release times, then they cannot be placed in two distinct segments.

2. Machine Selection

Once jobs have been sequenced, they are allocated according to their sequence to a machine $\mathcal{M}$ from among those available. Clearly, machine speed will affect processing time due to differences in machine speeds; hence, machine selection will have other significant impacts on increases in both criteria of the objectives function (Cao et al., 2005). It is, therefore, evident that these criteria can be improved if priorities are considered that will lead to the selection of the machine most appropriate for processing the job in question. For this purpose, three policies are used as machine selection policies (MSPs):

Policy A: According to this policy, machines are selected according to the ascending order of their processing speed. For a given job J_j, the processing time is computed for the machine with the least speed; i.e., $p_{i,j}$. If $\max\{r_j, C_{i,n_i}\} + p_{i,j} \leq d_j$, the job will be processed on M_i; otherwise, the same computations will be performed for M_{i+1}. This procedure will be continued for all machines up to M_m.

Policy B: The processing times for a selected job J_j on all the machines in the set $\mathcal{M}$ are computed to be added to $\max\{r_j, C_{i,n_i}\}$ and d_i is subtracted from the sum obtained – that is, $d_j - (\max\{r_j, C_{i,n_i}\} + p_{i,j})$. Then, the machine with

the least value of $d_j - (\max\{r_j, C_{i,n_i}\} + p_{i,j})$ is selected for processing the job. It must, however, be noted that the result of the subtraction should be a non-negative value; otherwise, the job will expire.

Policy C: The machine with the shortest processing time will be selected based on the sum of $\max\{r_j, C_{i,n_i}\}$ and job J_j processing time computed for all the machines in the set $\mathcal{M}$.

Example 1 below may be used for a better illustration of the SM described in Section (3.1.1.) and the machine selection policies outlined above.

Example 1: The specifications for 24 jobs $\mathcal{J} = \{J_1, J_2, \dots, J_{24}\}$ to be processed on two uniform parallel machines $\mathcal{M} = \{M_1, M_2\}$ operating at speeds $s_1 = 1$ and $s_2 = 1.1$ are reported in Table II. For instance, job j_8 enters the system when the time unit is 5 ($r_8 = 5$) and requires a machine for four units of time if it is to be processed on M_1 ($i.e., p_{1,8} = 4$). If, further, the processing is completed by the time unit 21 ($d_8 = 21$), then the benefit earned will be equal to 15 ($b_8 = 15$).

Let us now assume that LBF is adopted as the scheduler and policy B as MSP while segment length accommodates four jobs (i.e., $n_{jb} = 4$), then the jobs are grouped into the classes of A, as the class of very small problem that includes $\mathcal{J} = \{J_1, J_2, \dots, J_{12}\}$, and B, as a small class that includes $\mathcal{J} = \{J_1, J_2, \dots, J_{24}\}$. Class A is expected to have three segments, but this will be the maximum number of segments. Job processing in Class A shows that segmentation of jobs yields three segments, including the set $\{J_1, J_2, J_3, J_4, J_5\}$ in Segment 1, $\{J_6, J_7, J_8, J_8, J_9\}$ in Segment 2, and $\{J_{10}, J_{11}, J_{12}\}$ in Segment 3. Evidently, there is a change in segment lengths since segment length must be 4. A job batching is based on their release time. So, segment lengths change if jobs at the boundary between consecutive segments have equal release times. It is obvious that J_4 in Segment 1 is located at a boundary position while $r_4 = r_5 = 2$. Thus, J_5 is located in Segment 1, as is J_4. Changes in the length of the last segment due to the inadequate number of jobs relative to segment length is quite obvious. If jobs J_{10}, J_{10}, J_{11}, and J_{12}were to have equal processing times, then the number of segments would reduce to two, and the change in segment length would occur in this case, too, so that Segment 1 would have a length of 5 and Segment 2 a length of 7.

Table II. Specifications of the jobs in Examples 1

j	p_{1j}	r_j	d_j	b_j	j	p_{1j}	r_j	d_j	b_j
1	6	0	10	19	13	5	11	19	4
2	9	0	20	7	14	7	14	23	10
3	9	1	29	16	15	3	18	33	1
4	3	2	10	18	16	4	19	33	13
5	6	2	18	9	17	4	20	32	14
6	7	4	27	2	18	9	21	30	3
7	6	4	20	15	19	5	21	31	9
8	4	5	21	15	20	4	22	36	16
9	8	7	23	14	21	5	25	36	15
10	7	10	25	14	22	9	25	42	9
11	6	10	32	4	23	6	30	37	15
12	10	11	34	5	24	2	31	49	19

If the segments are submitted to machines one after the other, then $\sum b_j(LBF) = 104$, $|\mathcal{J}(LBF)| = 9$, and $CS(LBF) = \{J_1, J_5, J_4, J_2, J_3, J_8, J_{10}, J_6, J_{11}\}$. This is while failure to employ SM would yield $\sum b_j(LBF) = 120$, $|\mathcal{J}(LBF)| = 9$, and $CS(LBF) = \{J_1, J_4, J_7, J_8, J_3, J_9, J_{10}, J_{12}, J_{11}\}$.

Using the other MSPs in the absence of SM would lead to the same results as obtained under policy B; employing SM, however, would lead to a number of jobs completed equal to 9 under either policy but with a benefit equal to 95. The Gantt charts represent this situation in Figure 2.

In Figure (2a), $n_1 = 4$, $n_2 = 5$, $C_{1.4} = 25$, and $C_{2.5} = 31.07$ and in Figure (2b), $n_1 = 5$, $n_2 = 4$, $C_{1.5} = 31$, and $C_{2.4} = 29.43$.

Let us assume that $MSP = B$ and $n_{jb} = 8$ when processing job in Class B. Employing SM leads to sustained benefit and, thereby, to an increased number of jobs completed; in other words, $\sum b_j(LBF) = 162$ and $|\mathcal{J}(LBF)| = 13$. Numerical results showed that the SM exhibited a satisfactory performance when employed for problems with very small numbers of jobs (say, 12) but that it was not cost-effective with a small problem size (say, 24).

It is clear that by segmentation of jobs into several segments before implementing a DR on each, the SM assumes that the whole problem is the segment for which it is trying to sequence jobs.

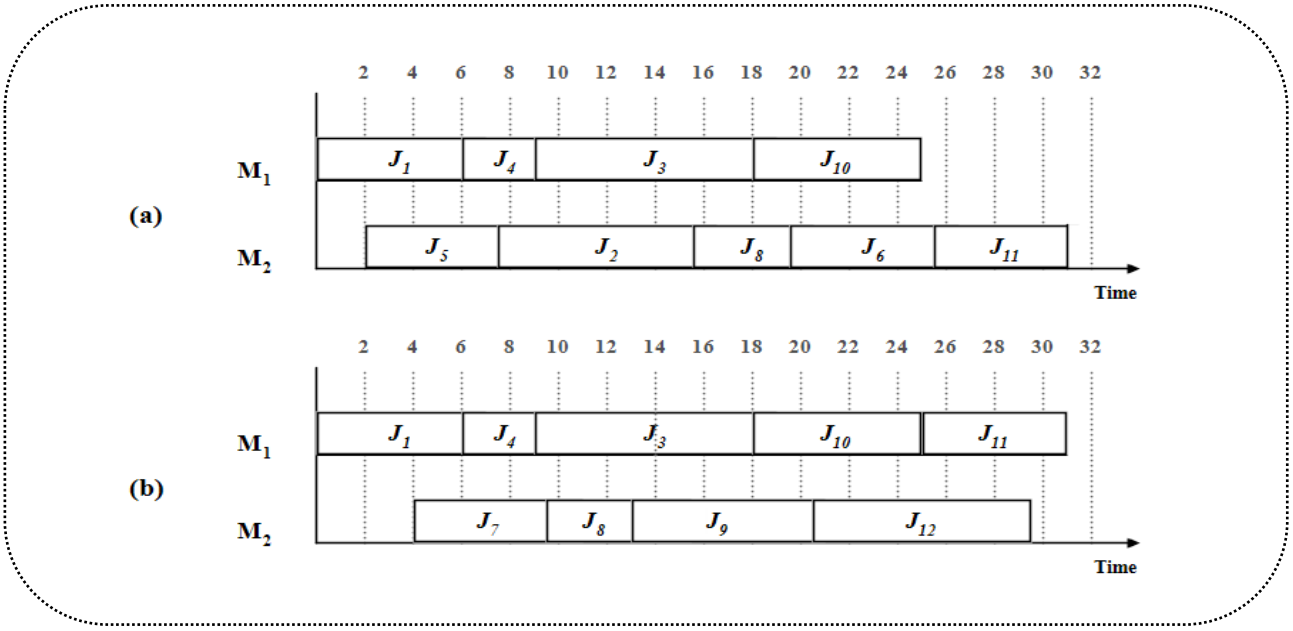

Figure 2. Gantt charts for Example 1; a): LBF with segmentation, and b): LBF without segmentation

3. Best Segment Selector procedure

Each DR implemented in a segment seems to have a desirable performance only in a specific time period in terms of improving the objective function. It follows then that if policies are adopted that could segment together good quality segments of sequences of jobs that have been allocated to machines by each DR, the outcome would guarantee a desirably optimized objective function. This idea is realized by the BSS procedure that selects the best segments in each schedule and recombines them into a unified schedule to be fed into the system. This will lead to a desirable search divergence so that the search procedure is relaxed from the local space to detect other possible solutions in the neighborhood as well. Indeed, the BSS initiates from different points to explore possible solutions to the problem and scan enough of the problem search space. Moreover, this procedure creates an upper bound for the problem that

prepares the grounds for the second phase of the NCDRA to get closer to the lower bound. The BSS procedure is accomplished through the following steps:

Step 1: The jobs scheduled by each DR are sorted in an ascending order of their start times to process by the set of machines $\mathcal{M}$ and placed in $st(\mathcal{S})$.

Step 2: The sum of process end times under all the DRs is calculated, and the average end time is obtained to be divided by the number of segments, rounded to the next higher unit, and used as the timespan for each segment. The boundary between two consecutive segments in each rule is obtained based on the start time to process jobs in $st(\mathcal{S})$; the number of segments is designated by n_p.

Step 3: The timespans obtained from Step 2 are overlaid on the sequence obtained from Step 1. The corresponding segments (i.e., timespans) in each scheduler are compared to select the best segment in a sequence that suits improving the objective function and then designated as *BSSseq*.

For illustration, if n_p = 3 in Step 3, the first segments from all the schedulers are compared to find the one that leads to the best value of the objective function; this is then transferred to *BSSseq*. It will then go on to compare the second segments from the schedulers in search of the best to be transferred to *BSSseq*. Prior to the comparisons, a function named *edt()* is initially implemented for all the second segments from all the schedulers in order to remove repeated jobs in each scheduler (that is, the second segment) already transferred to *BSSseq* (that is, the first segment). The same procedure is performed on the third segments, the best of which is ultimately transferred to *BSSseq*. Clearly, the function *edt()* avoids repeated occurrences of jobs in the *BSSseq*. The jobs not assigned to the *BSSseq* after Step 3 will be transferred to a sequence called *UnAssSeq*; in other words, $UnAssSeq = \mathcal{J} - BPPseq$.

Step 4: The jobs in *BSSseq* are assigned to machines according to one of the MSPs.

Step 5: If gaps appear on machines, jobs are selected from the *UnAssSeq* to fill the gap. A gap on M_i may be defined as an idle time between the end process time of one job and the start time of the next on M_i. Job J_i from *UnAssSeq* is transferred to the gap on M_i if $d_j \leq GET$ and $p_{i,j} \leq GET - GST$, where GST and GET represent the gap start time and end time, respectively.

Example 2: Based on the data of Example 1, assume $\mathcal{M} = \{M_1, M_2\}$, $s_1 = 1$, $s_2 = 1.1$, $MSP = B$, $n_p = 5$, and $n_{jb} = 8$. When following the step-by-step procedure of BSS, the five DRs of LBF, EST, MBMRP, and Min Slack are used.

In Step 1, the jobs scheduled by each DR on $\mathcal{M}$ are sorted in an ascending order of their start process times to produce the sorting $st(\mathcal{S})$. In the case of jobs with identical start times, the one job is prioritized that is to be processed on a machine with a smaller subscript. In other words, if $st_{1,1} = 0$ and $st_{2,2} = 0$, then J_1 has priority over J_2. Thus, the five rules above schedule jobs as follows:

$st(LBF) = \{J_1, J_7, J_4, J_3, J_8, J_9, J_6, J_{12}, J_{16}, J_{20}, J_{15}, J_{24}, J_{22}\}, \sum b_j(LBF) = 162, |\mathcal{J}(LBF)| = 13$

$st(EST) = \{J_1, J_4, J_5, J_2, J_7, J_3, J_6, J_{12}, J_{11}, J_{15}, J_{20}, J_{22}, J_{24}\}, \sum b_j(EST) = 140, |\mathcal{J}(EST)| = 13$

$st(MBMRP) = \{J_1, J_5, J_4, J_7, J_2, J_8, J_6, J_3, J_{12}, J_{16}, J_{23}, J_{20}, J_{24}\}, \sum b_j(MBMRP) = 169, |\mathcal{J}(MBMRP)| = 13$

$st(Min\ Slack) = \{J_1, J_9, J_{13}, J_{15}, J_{19}, J_{18}, J_{23}\}, \sum b_j(Min\ Slack) = 65, |\mathcal{J}(Min\ Slack)| = 7$

$st(LBMP) = \{J_1, J_9, J_{13}, J_{15}, J_{18}, J_{19}, J_{23}\}, \sum b_j(Min\ Slack) = 65, |\mathcal{J}(Min\ Slack)| = 7$

It is clear that MBMRP yields the highest benefit, but the number of jobs completed will be equal to those scheduled by EST and LBF.

In Step 2, $st(\mathcal{S})$ is broken up into five segments, and each is compared with its corresponding one based on the benefits earned and the number of jobs completed. In order to determine the appropriate timespan for each segment, the end times obtained for the five DRs (schedulers) are averaged, and the average value thus obtained is divided by n_p. Given that $C(\mathcal{S}) = \max\{C_{1,n_1}, C_{2,n_2}\}$ must be used to compute the end time obtained from each rule when two competing machines are available; we will have:

$C(EST) = 44$, $(LBF) = 42$, $C(MBMRP) = 39$, $C(LBMP) = 36$, and $C(Min\ Slack) = 36$

Thus, the sum of the end times of all rules will be equal to 197; dividing this quantity by the number of DRs (5 in this example) yields 39.4. Subsequently, the length of each segment on $st(\mathcal{S})$ is equal to 8 ($8 = \lceil 39.35 \div n_p \rceil$). The timespans for segments will be as follows: [0-8], (8-16], (16-24], (24-32], and (32-$C(\mathcal{S})$]. According to Step 3, the jobs scheduled by each rule are allocated to determining timespans. In Figure 3, the corresponding segments under the different rules are shown with the same color. For LBMP, for instance, $\{J_1, J_9\}$ are placed in the first segment, $\{J_{13}\}$ in the second, $\{J_{15}, J_{18}, J_{19}\}$ in the third, and $\{J_{23}\}$ in the fourth. Clearly, the fifth segments of both LBMP and Min Slack rules are void, indicating that no job processing is initiated in their fifth timespans.

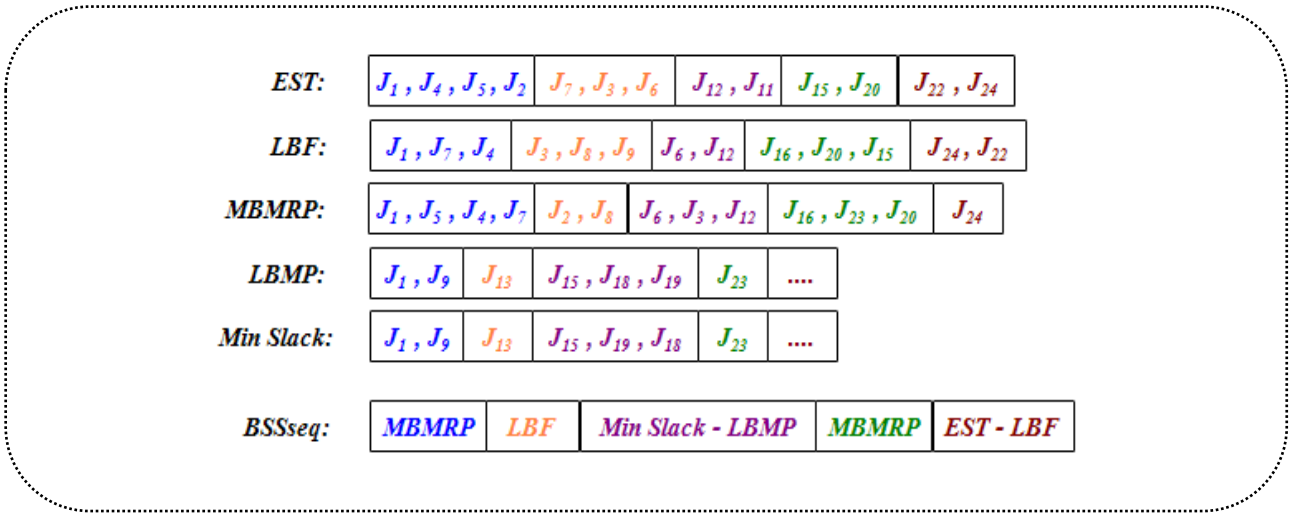

Figure 3. Segmentation of the DRs and the BSSseq in Example 2.

After segmentation, the corresponding segments must be compared to find the best segments to be transferred to *BSSseq*. Comparison of the first segments from all the DRs shows the MBMRP rule is the one with the highest benefit of 61 and the greatest number of jobs processed (i.e., 4) while its jobs are also placed in the first segment of *BSSseq*. To examine the second segment, the function *edt()* is first implemented for it to find and remove not only identical jobs in these segments of the DRs but also those in the *BSSseq*. The jobs compared with each other from the second segments of the EST, LBF, MBMRP, LBMP, and Min Slack rules are $\{J_7, J_3, J_6\}$, $\{J_3, J_8, J_9\}$, $\{J_2, J_8\}$, $\{J_{13}\}$, and $\{J_{13}\}$. The highest benefit of 45 and the greatest number 3 of jobs processed in the second segment of the *BSSseq* were recorded for the LBF rule. This is while the Min Slack and LBMP rules both had the best third segment in the *BSSseq*. Compared to the other rules, the MBMRP rule recorded the best results, and its jobs were placed in the fourth segment of the *BSSseq*. Finally, $\{J_{22}, J_{24}\}$ from the EST and LBF rules are the ones selected for the fifth segment. Thus, the *BSSseq* comprises 15 jobs arranged in this sequence: $BSSseq = \{J_1, J_5, J_4, J_7, J_3, J_8, J_9, J_{15}, J_{19}, J_{18}, J_{16}, J_{23}, J_{20}, J_{22}, J_{24}\}$. When the *BSSseq* is submitted to the machines in Step 4, then $\sum b_j(BSSseq) = 188$, $|\mathcal{J}(BSSseq)| = 14$, and $\mathrm{CS(BSSseq)} = \{J_4, J_1, J_8, J_5, J_9, J_7, J_{19}, J_3, J_{16}, J_{15}, J_{20}, J_{23}, J_{22}, J_{24}\}$ is obtained. This is while J_{18} was part of the *BSSseq* but failed to be allocated to a machine within the timespan $[r_{18}, d_{18}]$. It follows then that it is possible for jobs in the *BSSseq*

to fail to find themselves a machine at the right time due to coincidence with other jobs within their process time. Thus, if J_j is a job in the *BSSseq*, and if it fails to be submitted to a machine within the timespan $[r_j, d_j]$, then it will be removed from the *BSSseq* and transferred to *UnAssSeq*.

In Step 5, the jobs on machines are controlled for any gaps so that the gaps are filled with jobs from the *UnAssSeq*. Examination of the sequence on M_2 in Figure 4 reveals that $C_{2,4} = 4.72$ and $st_{2,8} = 5$ and that the first gap is found in the void between these two jobs: $GST_1 = 4.72$ and $GET_1 = 5$. It is also observed that $C_{2,9} = 15.9$ and $st_{2,19} = 21$. Thus, the start and end times of the gap on M_2 will be as follows: $GST_2 = 15.9$ and $GET_2 = 21$. It is now time to fill the gaps with jobs from the *UnAssSeq*. For Gap 1, its length is initially computed: $lenGap_1 = GET_1 - GST_1 = 5 - 4.72$.

The intervals $[r_2, d_2]$ and $[r_6, d_6]$ for the two jobs J_2 and J_6 are such that the jobs can be placed in the timespan $[GST_1, GET_1]$; however, since $p_{2,2}, p_{2,6} > lenGap_1$, they can be transferred into the first gap. Examination of the jobs $\{J_2, J_6, J_{10}, J_{11}, J_{12}, J_{13}, J_{14}\}$ also shows that they do not have the requirements for being transferred into the second gap. Hence, Step 5 in this example failed to improve the results obtained in Step 4.

Based on the results obtained from implementing the BSS, it may be claimed that both the efficiency criteria in the objective function were improved. It should also be noted in Figure 4 that $C_{1,23} = 36$ and $C_{2,24} = 42.78$ whereby $C(BSS) = \max\{C_{1,23}, C_{2,24}\} = 42.78$.

It is, of course, possible that the BSS fails to improve the objective function by providing results superior to those yielded by the DRs. Specifically, this might happen if every one of the jobs in the *BSSseq* fails to get on a machine by the end of their allocated process times, in which case the jobs will be deleted; or, if Step 5 fails to add jobs to the set of processed jobs in case a gap appears. In this situation, the benefits and the number of jobs completed under the BSS will be lower than those under any of the DRs.

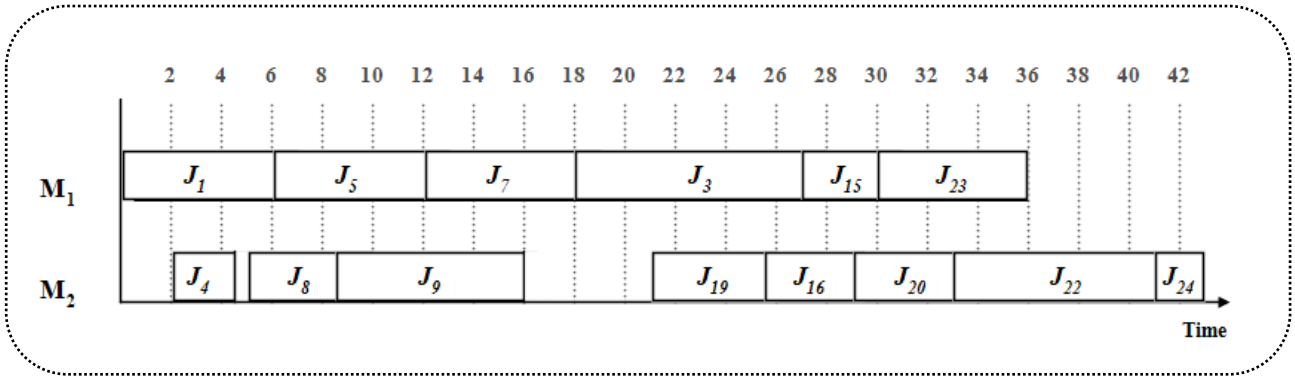

Figure 4. Gantt chart for BSSseq in Step 4 of the BSS.

B. Local search phase

A neighborhood search structure for the LS phase of NCDRA is equipped with swapping, insertion, and reversion operators that serve as exploration operators. To enhance the algorithm's efficiency, three operators are used concurrently and interconnected through a shared memory both to avoid trapping in a local optimum and to increase the algorithm's speed in obtaining improved solutions. Moreover, these operators can be exploited during the updating process to create perturbation in order to enhance diversity in the sequences generated. This way, operators attempt to explore and exploit in addition to cooperation in order to achieve the global optimal solution.

Due to limitations on r_j and d_j, each exploration operator is customized for each job. For this purpose, a frequency table of r_j is created for each instance of the problem in which jobs are sorted in ascending order of r_j. Once the range is

computed ($R = r_{J_n} - r_{J_1}$), class distances (CD) are determined using $CD = \lceil {}^{R}/_{NF} \rceil$, in which NF represents the number of classes. It is after this stage that the exploration operator is able to create, quite randomly, a neighborhood for the jobs in each class of each scheduler. For instance, in a scheduler with $NF = 5$, if a random number is observed, then the neighborhood is created for the second class.

Exploration in the solution space will be accomplished in the following three steps:

Step 1: The four features of $\sum b_j(exSeq)$, $|\mathcal{J}(exSeq)|$, *exUnAss*, and *exAss* in the shared memory are assigned the initial values of $\sum b_j(BSSseq)$, $|\mathcal{J}(BSSseq)|$, *UnAssSeq*, and *BSSseq* as the output values of Phase I of the proposed algorithm.

Step 2: Each of the operators starts the search operation in accordance with their nature in the neighborhood of the best sequence thus far found (*exAss*) and stored in the shared memory.

Step 3: Once an operator succeeds in improving each of the objective function criteria (i.e., $\sum b_i$ and $|\mathcal{J}(\mathcal{S})|$), the four features in the shared memory are updated and replaced with the new values.

Steps 2 and 3 are repeated until the termination condition is met. Exploration operators continue unless the user terminates them at a specific point. For the purposes of this study, the number of iterations (n_{itr}) is used as the termination condition. However, the proposed algorithm stops if $|\mathcal{J}(\mathcal{S})|$ becomes equal to the number of problem jobs before the given number of iterations is reached.

```
For each Exploration Operator Do
  Apply the selected Exploration Operator on the best Critical Sequence to find a neighbor sequence
  For each job in the neighbor sequence Do
    Allocate the job to a processor based on the MSP
    If there is a Gap Then
      Create a ready job list; select jobs from exUnAss where r_j <= GST && d_j >= GET
      Find a job with maximum benefit from ready job list and name it sJob
      Sort the ready job list in an ascending order of processing time
      Calculate maximum number of jobs to fill in the Gap and name it maxJobInSlot
      Make collections of 1 to maxJobInSlot and name it mJobList
      Calculate the benefit of each member in mJobList
      Sort the mJobList in an ascending order of item length
      For each member in the mJobList Do
        If (the member's benefit × α ) >= s Job's benefit Then
          Allocate the member to the Gap
        End-If
      End-For
    End-If
  End-For
  Calculate the objective function of the neighbor sequence
  If the neighbor sequence's objective function is better than that of the Critical Sequence Then
    Update the Critical Sequence and its elements based on the neighbor sequence
    Update the Critical Sequence in other Exploration Operators based on the new Critical Sequence
  End-If;
End-For
```

Figure 5. Online gap filler procedure.

In the second step, it is possible to occur a gap by assigning J_j to the selected machine. In this case, job or jobs that meet the requirements for filling the gap is/are selected from *exUnAss* and assigned to the machine before assigning the job J_j. The condition necessary for filling a gap with a job or jobs from the *exUnAss* is that the replacement must not lead to the expiry of J_j. It is evident that filling a gap in this step of Phase II of the heuristic algorithm is at the time that jobs from *exAss* are allocated to machines, which is slightly different from what happens in Step 5 of the BSS procedure. Moreover, filling a gap in this step might lead to the expiry of a job or jobs from the *exAss* that come after J_j. Allocation of *exAss* to machines and estimation of the associated benefits and number of jobs completed is accomplished using the online gap filler procedure shown in Figure 5.

To illustrate this procedure, let us assume that $\mathcal{M} = \{M_1, M_2\}$, $s_1 = 1$, $s_2 = 1.1$, $MSP = B$, $C_{1,n_1} = 2$, and $C_{2,n_2} = 4$. If J_j with the specifications $p_j = 3$, $r_j = 0$ and $d_j = 5$ is nominated for processing, then it will occupy M_1; hence, $C_j = 5$ because $r_j \leq \min\{C_{1,n_1}, C_{2,n_2}\}$ and $\min\{C_{1,n_1}, C_{2,n_2}\} + p_{1,j} \leq d_j$. In this situation, no gap is created when J_j is added on M_1 since $C_{1,n1} = st_{1,n_1+1} = 3$. If $J_{j\prime}$, instead of J_j, with the specifications $p_{j\prime} = 2$, $r_{j\prime} = 5$, and $d_{j\prime} = 9$ occupies the machine, given that $r_{j\prime} > \min\{C_{1,n_1}, C_{2,n_2}\}$, the release time of $J_{j\prime}$ will be subtracted from the idle time of the machines so that the machine with the lowest remaining value of the subtraction process will be nominated for processing $J_{j\prime}$; in other words, $\min\{r_{j\prime} - C_{1,n_1}, r_{j\prime} - C_{2,n_2}\}$. If $J_{j\prime}$ is allocated to M_2, a gap of the one-time unit will be created between the timespans 4 and 5 because $st_{2,n_2} = 5$ and $C_{2,n_2-1} = 4$. In this situation, given the specifications of $J_{j\prime}$, lines 5 to 18 of the algorithm will try to select a job or jobs from the *exUnAss* to fill the gap in order to improve the objective function. Finally, $J_{j\prime}$ should not expire during the gap-filling process (i.e., selecting jobs from the *exUnAss* and allocating them to the machine in question).

In this procedure via job selection from the *exUnAss*, maximum care must be taken to strike a balance between the number of jobs selected and the benefit earned. This is reflected in a coefficient α that expresses the weight of each of the objective function criteria (w_1, w_2) to ensure a minimum score has been earned when a job is selected to fill a gap. It is noteworthy that in computational results, a value of 0.5 was assigned to the coefficient α. Assume three jobs with benefits and processing times of $b_1 = 20$, $b_2 = 9$, $b_3 = 10$, $p_1 = 2$, $p_2 = 1$ and $p_3 = 1$, respectively, and a gap of 2-time units. Under these conditions, the selection of J_1 will yield a benefit of 20; however, then jobs other than J_1 will be selected if they can be processed simultaneously and if they earn at the same time 95% of the benefit earned by J_1. In other words, J_2 and J_3 will be selected for α=0.95. Clearly, the sum of benefits earned from both J_2 and J_3 is lower than that of the selected job (J_1), but these two jobs are selected instead of J_1 to fill the gap because they yield a total benefit equal to α% that of the selected J_1 job.

Example 2 continued: The output from SS phase in Example 2 is updated using the output from the LS phase of the proposed algorithm. The exploration operators can improve both the number of processed jobs and the benefit earned via swapping the jobs and reducing the gaps. Implementation of LS phase yields $C(exOp) = \max\{C_{1,n_1}, C_{2,n_2}\} = 40.87$. This yields $\sum b_j(exOp) = 217$, $|\mathcal{J}(exOp)| = 15$. It may be noted that the scheduler in LS phase is denoted by *exOp*. Figure 6 depicts the job scheduling on two processors in which $C(exOp) < C(BSS)$.

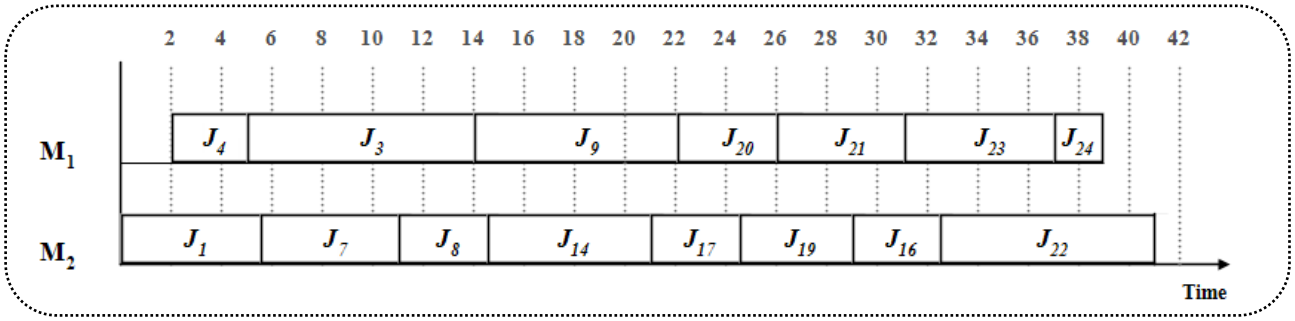

Figure 6. Gantt chart after the implementation of the LS phase in Example 2

IV. COMPUTATIONAL RESULTS

This section uses extensive experiments to investigate the performance of NCDRA. Initially, a description is provided of the specifications of the problems studied. This is followed by a detailed explanation of parameter tuning in NCDRA to improve its performance. Finally, NCDRA and SA are compared in terms of their performance in dealing with the problem instances.

The proposed algorithms are implemented in Java 9, and the experiments are conducted on the computer with the following specifications: Processor, Intel(R) Core^{TM} i5-3330; CPU, 3.00 GHz; RAM, 4 GB; and System type, 64-bit Windows 10 operating system.

A. Specifications of the problems studied

The test data are generated in such a way that the effectiveness of the algorithm is reflected in the different parameters in terms of test instance size and characteristics. The three main groups of problem instances studied are i) Small size ones including 20 instances designated by numbers 1 through 20, 50 jobs need to be processed on five uniform parallel machines, machine speeds are randomly selected from $s_i = \{0.6, 0.8, 1, 1.1, 1.3\}$. ii) Medium size problem instances including eight problems designated by numbers from 21 through 28, there are 100 jobs to be processed on ten uniform parallel machines, and the machine speeds are to be randomly selected from among $s_i = \{0.5, 0.6, 0.7, 0.8, 0.9, 1, 1, 1.1, 1.2, 1.3\}$. iii) Large size problem instances, including eight problems designated by numbers from 29 through 36, 1000 jobs are planned to run on 50 uniform parallel machines, and machine speeds are randomly selected from the range of [0.4-1.5] at steps of 0.01 or 0.02. Three small, moderate, and large size problem instances are grouped into four classes of problems according to the characters representing their processing times, release times, and due dates. Table III reports the specifications of four classes. Processing time and release time are randomly generated within a specified range. Finally, the due date is obtained as the sum of release time, processing time, and a random number from the specified range; for instance, $d_j = p_j + r_j + randInt(range)$.

Table III. Specifications of the input data for the four classes of small, moderate, and large size problems

Problem class	p_j	r_j	d_j	b_j
1	[1-10]	[0-40]	[1-10]	[1-20]
2	[1-10]	[0-70]	[1-10]	[1-20]
3	[1-10]	[0-40]	[1-20]	[1-20]
4	[1-10]	[0-70]	[1-20]	[1-20]

The illustrative examples below are meant to provide a better understanding of the four problem classes. Class 1 problems (i.e., problem instances 1 through 5, and also 21 and 22 as well as 29 and 30) include those with tight release times and due dates. This means that job availabilities are concentrated at the beginning of the schedule-timeline and that their time windows between release time and due date are as short as possible. Class 2 problems (i.e., problem instances 6 through 10, and also 23 and 24 as well as 31 and 32) are those with loose release times but tight due dates. This means that job availabilities are scattered throughout the schedule-timeline and that the time window between release time and due date is as short as possible. Class 3 problems (i.e., problem instance 11 through 15, and also 25, and 26 as well as 33 and 34) are those with tight release times but lose due dates. This means that jobs are available at the beginning of the schedule-timeline and that the time window between their release times and due dates is broad such that processing jobs begin a long time after their release times. Class 4 problems (i.e., problem instances 16 to 20, and also 27 and 28 as well as 35 and 36) are those with loose release times and due dates. This means that job availabilities are scattered throughout the schedule and that the time window between their release times and due dates are broader such that processing jobs will become possible long after their release times.

B. Parametric tuning of NCDRA

It is essential to tune the parameters of NCDRA, including n_{jb}, n_p, n_{itr}, and MSP. These parameters are tuned through preliminary computational experiments so that NCDRA will have the best performance.

1. Number of jobs completed in each segment (n_{jb})

Tuning n_{jb} in the proposed algorithm for the small size problem instances was performed by assigning even numbers of 4 to 50 at steps of 2, and the effects were investigated on $(Obj1 = w_1 \sum b_i)$ and $(Obj2 = w_2|\mathcal{J}(\mathcal{S})|)$. Implementation of the DRs on the problem instances showed that n_{jb} in the range [4–14] led to better results. Figure 7 shows the results obtained from effecting average values of the components $\sum b_i$ and $|\mathcal{J}(\mathcal{S})|$ by implementing 11 DRs on n_{jb} in the range [4–30] in the problem instance No. 14. The values for the two components (i.e., $\sum b_i$ and $|\mathcal{J}(\mathcal{S})|$) of the objective function for n_{jb} in the range of [4–14] will be equal to 446.4 to 459.9 and 37.4 to 38.2, respectively. The best values of 439.5 and 36.4 for $\sum b_i$ and $|\mathcal{J}(\mathcal{S})|$, respectively, were obtained for n_{jb}>14. Investigation of moderate and large size instances revealed that n_{jb} in the range of [20–56] and [50-350] assigned at steps of 4 and 50 generated desirable values for the objective function, respectively.

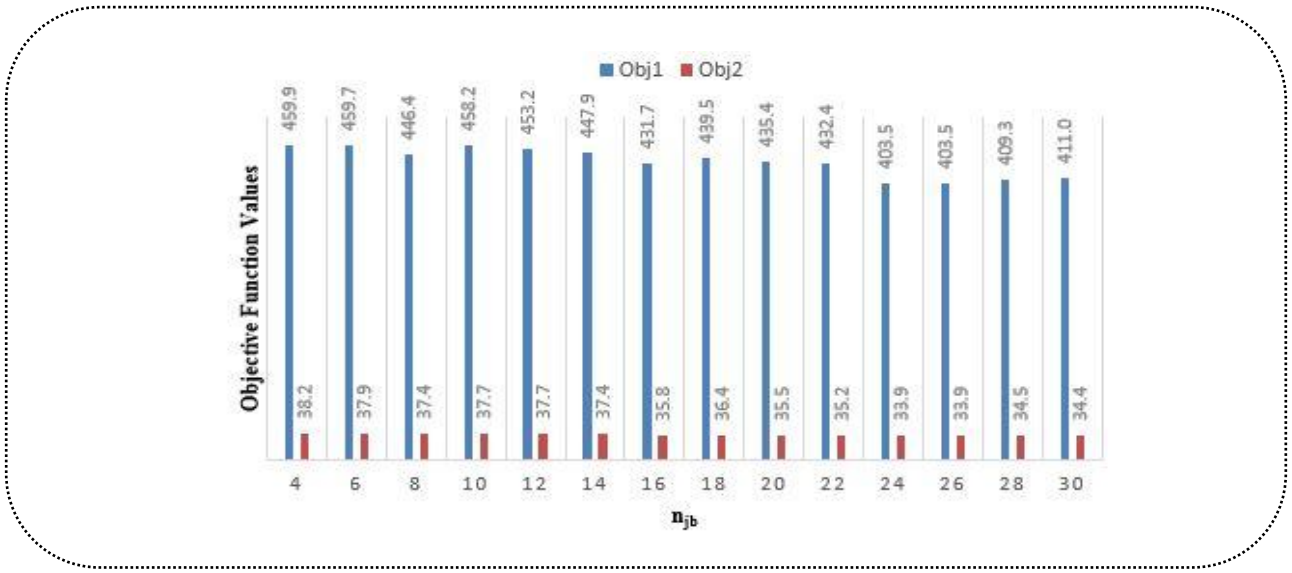

Figure 7. Comparison of the effects of the parameter n_{jb} on both criteria of the objective function

2. The parameter n_p

The BSS procedure not only produces good results in SS phase of the NCDRA but also plays an important role in generating results of good quality in LS phase. The parameter affecting the performance of BSS is n_p, whose value had to be determined and for which values from 3 to 10 were tested. The BSS in small, moderate, and large size problem instances revealed that n_p values of 3, 4, and 5 produced the best results, indicating that increasing values of n_p have no effects on BSS quality.

3. The parameter n_{itr}

One factor involved in the best-constructed scheduler is the maximum n_{itr} of the allowed exploration operator iterations. The computations were started with a n_{itr} number of 700 and the variation in the performance of the objective function was measured after each iteration. The different iterations of the algorithm on a moderate size problem instance revealed only slight changes in each of the objective function criteria after around 200 iterations,

thereby only wasting CPU time. Experiments were also performed with large size instances to tune the value for n_{itr} and the best value was found to be 500. The trend in the objective function improvement is depicted in Figure 8 for problem instance No. 14. Its left and right-hand chart display the trend of benefit and the number of jobs processed, respectively. Clearly, increased benefit in some iterations did not lead to an increased number of jobs processed, neither did the reverse. It may thus be concluded that although the number of jobs processed might have remained unchanged in certain iterations, swapping led to the replacement of a job with a higher benefit for one with a lower benefit.

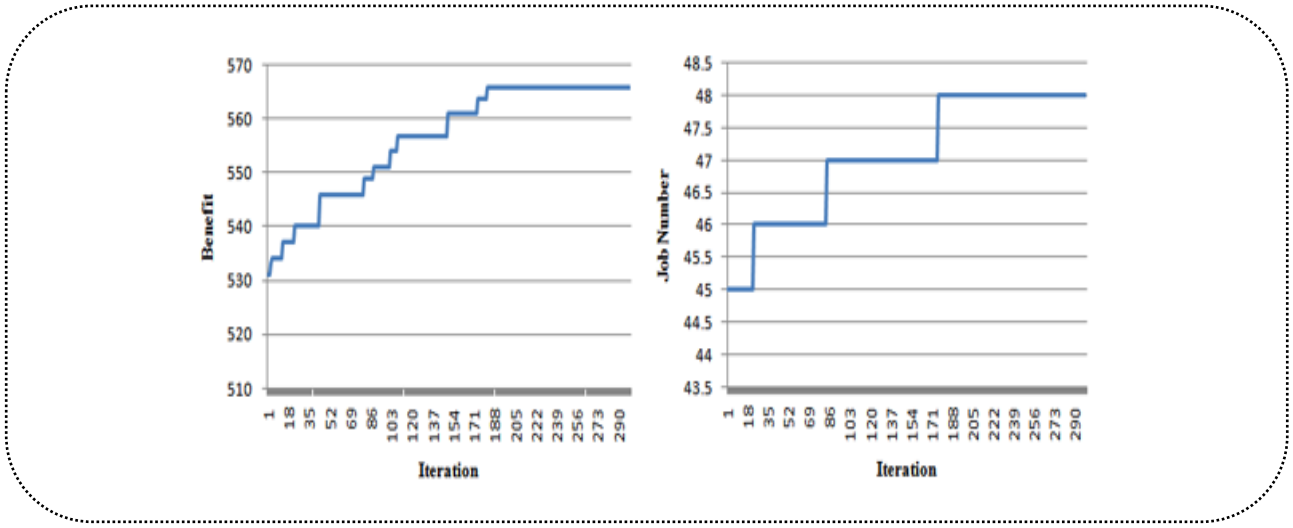

Figure 8. Quality of extrapolation of the NCRDA results after each iteration for problem instance No. 14.

4. Machine selection policy

The quality of results as affected by MSP was also investigated. In this process, the DRs and the LS phase were tested under different policies. Table IV reports the average values obtained from 10 iterations of the proposed algorithm on instances 1 to 5. Comparison of the results obtained from effecting policies A, B, and C reveals the relative efficacy of all the three policies in some of the instances solved using the DRs. In the LS phase of the algorithm, policy B proved the best in four instances in terms of the number of jobs processed and in two instances with regard to the benefit; hence, it was adopted as the policy for the LS phase. Policy A led to the greatest benefit earned in two problem instances but only yielded the greatest number of jobs completed in one instance. Policy C led to the greatest benefit in one instance.

Table IV. Comparison of machine selection policies in the LS phase

Problem class	*Instance*	*Machine Selection Policy*					
		A		*B*		*C*	
		$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$
1	1	485.9	41.9	483.4	42.2	476.1	41.4
	2	529.3	41.6	530.1	41.9	521.7	40.8
	3	489.8	38.6	489.1	38.8	491.0	37.8
	4	536.4	42.8	534.2	42.7	525.7	42.0
	5	526.0	42.5	527.3	42.7	515.2	41.8

Investigation of the tuning MSPs also revealed that no specific policy led to the best results for either all or even most of the DRs. For example, implementation of EFT in problem instances showed that policy B yielded satisfactory results or that policy C yielded better results under the EDF rule. Therefore, all three policies were used in the computational experiments to select the one policy that led to the best schedule.

C. Investigation of NCDRA components

This subsection investigates the different components of the proposed algorithm to show how the different stages of the two phases of the algorithm improve its performance.

1. Investigation of the performance of DRs

The DRs were compared in terms of their performance to create an upper bound in the computational experiments. The results are reported in Table V. It is seen that the MBMRP rule from among 11 others was able to produce the best results while the rules EDF, LBF, and EST produced better results in 7, 4, and 4 instances, respectively. This table also provides the proper value for n_{jb} that led to the improvement of the objective function by one DR. In problem instance 1, for example, all the possible values in the range [4–14] were assigned to n_{jb} at steps of 2 under the ADMR rule. For n_{jb}= 8, the values obtained were $|\mathcal{J}(\mathcal{S})| = 40$ and $\sum b_i = 434$, which led to the best improvement in the objective function. It may be noted that the same problem instance was terminated at a time unit of 50 under ADMR (i.e., $C(ADMR) = 50$). The most frequent value for n_{jb} with 8 iterations for 50 jobs was 12. This is while its least value was n_{jb}=4 with five iterations. An interesting point to note about EST is that it outperformed the other rules in four problem instances of small size. This is while none of the values in the range [4–124] assigned to n_{jb} exhibited any significantly different effects on either $|\mathcal{J}(\mathcal{S})|$ or $\sum b_i$; hence, the word 'all' used in the n_{jb} column in all cases in which EST was the rule selected.

2. Investigation of the BSS procedure

The next component investigated to evaluate the quality and performance of the NCDRA algorithm is the improvement of the upper bound gained by the BSS procedure used by DRs. Table VI reports the computational results obtained from the BSS procedure for the selected parameter n_p (i.e., 3, 4, and 5) were considered for all problem instances. Comparison of the results reveals that, from among the 72 values obtained for $\sum b_i$ and $|\mathcal{J}(\mathcal{S})|$ in 36 problem instances of the four classes, $n_p = 3$ yielded better results, especially for instances in classes 3 and 4.

The performance made by DRs to the construction of the BSS segments was measured. In problem instance 1, for example, the LBF was used twice, and MBMRP was used once for constructing *BSSseq* when $n_p = 3$. LBF with 35 and 30 segments in classes 3 and 4, respectively, had a great contribution to the *BSSseq*, while EFT and EST had the greatest contribution to *BSSseq* in classes 1 and 2 with 27 and 26 segments, respectively.

The best results due to the BSS may also be compared with those due to DR. The comparison reveals that neither is superior to the other with regards to the improvement they made in the objective function for class 1 as both yielded the best results for this class in 9 cases. For classes 2, 3, and however, BSS procedure outperformed DR in 13, 12, and 14 cases while DR produced better results in only 5, 6, and 4 cases, respectively. Another criterion that can be exploited to show the difference between BSS and DR as a good scheduler is $C(\mathcal{S})$. In the problem instance 20 from class 4, both BSS and DR equally recorded benefit scores of 564 and jobs completed scores of 50 for the best-constructed scheduler, but they differed in their C values: $C(BSS) = 81.91$ and $C(DR) = 94.5$. These observations confirm the requirement for using the BSS procedure as part of the NCDRA algorithm.

Table V. Computational results of the DR used to determine the best DR and n_{jb}

Problem class	*Instance*	*DR*	$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	n_{jb}	$C(\mathcal{S})$
1	1	ADMR	434	40	8	50
	2	LBF	481	38	6	51
	3	MBMRP	447	37	12	52.12
	4	MBMRP	505	39	10	50.37
	5	EDF	500	43	12	47.76
	21	EDF	859	79	44	54
	22	EDF	970	85	52, 56	52.33
	29	EFD	7268	698	350	56
	30	EFT	7203	688	150	55
2	6	MBMRP	538	47	14	78.33
	7	EDF	473	46	14	85.25
	8	EST	412	49	All	75.05
	9	MBMRP	506	50	8	79
	10	ADMR	547	49	10	76.71
	23	MBMRP	971	91	28, 56	83
	24	MBMRP	867	91	20	76
	31	LBF	8852	767	300	81.55
	32	LBF	8599	783	250	87
3	11	EST	470	41	All	55.67
	12	LBF	500	39	10	61
	13	EFT	424	41	12	51.25
	14	EFT	531	44	6	56
	15	LBF	396	38	12	60.63
	25	LBF	838	77	24	54.65
	26	MBMRP	805	72	40	54
	33	EDF	7468	718	200	62.61
	34	EDF	7280	709	300	64.93

Continue Table V. Computational results of the DR used to determine the best DR and n_{jb}

Problem class	*Instance*	*DR*	$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	n_{jb}	$C(\mathcal{S})$
4	16	EDF	540	50	6	89.41
	17	EST	515	50	All	80
	18	DDMR	510	50	4	79
	19	EDF	514	50	14	87.3
	20	EST	564	50	All	94.5
	27	MBMRP	974	96	56	82.65
	28	EDF	1019	99	40	95
	35	EFT	8926	916	300	92.3
	36	LBF	9280	888	150	90.94

D. Results obtained from implementing NCDRA and SA on different problems

Table VI reports the best results obtained of the ten times run of the NCDRA on all problem instances. As expected, the LS phase of the proposed algorithm improved the components of the objective function. In the LS phase, both DR and BSS exhibited equal capacity in finding the best values for 14 of the 72 components of the objective ($\sum b_i$ and $|\mathcal{J}(\mathcal{S})|$). These components belonged to problem instance 9 of class 2 as well as instances 16 through 20 and 28 of class 4, in which all the jobs were submitted to machines and were completely processed. In the remaining instances, it was the LS phase that improved the components of the objective when compared with either DR or BSS. This is confirmed by a comparison of the two columns under NCDRA in Table VII.

The best scheduler derived from the BSS procedure serves as an input string for the LS phase. Now assume a case in which the best scheduler is one of the DRs to be used as an input string for the LS phase. See the results reported under the two columns under the heading NCDRA-without-BSS in Table VII. It is observed that, in this situation, NCDRA and NCDRA-without-BSS both produced identical results for 14 parameters of the instances in class 4, and in the other four parameters, NCDRA performed better. In class 3 instances, the NCDRA-without-BSS outperformed NCDRA in only two problem instances, NCDRA outperformed NCDRA-without-BSS in 15 instances, and both performed equally well in one problem instance. In class 2 instances, the NCDRA-without-BSS yielded only one superior result; indeed, both produced similar results in three problem instances while NCDRA underperformed the NCDRA-without-BSS in the other 14 instances. Finally, better results were recorded for NCDRA-without-BSS in class 1 instances when compared with the other problem classes. Despite this, the NCDRA-without-BSS outperformed NCDRA in three problem instances in this class while the reverse was true in 10 problem instances, and both performed equally well in 5. These observations provide further confirmation of the efficiency of the BSS procedure and, ultimately, that of NCDRA. Further support for the efficiency of NCDRA is provided by comparing this algorithm with the SA developed in Gholami et al. (2019). Comparison of results reveals an optimum value of $|\mathcal{J}(\mathcal{S})| = 50$ in small instances and $|\mathcal{J}(\mathcal{S})| = 100$ in moderate instances recorded by both algorithms in 12 problem instances. NCDRA outperform SA in 17 instances with regards to improvements it made in the objective parameters of $|\mathcal{J}(\mathcal{S})|$ and $\sum b_i$ while the reverse was true in nine instances, it may thus be claimed that the NCDRA outperformed SA in this regard. Moreover, NCDRA proved in most class 3 problems more efficient than its rival SA while both proved efficient in 14 class 4 problems.

Table VI. Computational results obtained using the BSS procedure for $n_p = \{3, 4, 5\}$

Problem class	*Instance*	$n_p = 3$				$n_p = 4$				$n_p = 5$			
		$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	$BSSseq$	$C(\mathcal{S})$	$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	$BSSseq$	$C(\mathcal{S})$	$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	$BSSseq$	$C(\mathcal{S})$
1	1	415	35	11-6-6	48	425	38	11-6-4-8	47	426	39	6-11-6-8-3	47.27
	2	452	39	10-11-8	48.63	450	34	6-6-2-4	46	471	37	6-10-1-10-3	45.53
	3	425	37	4-4-8	50	443	37	4-11-11-8	52.13	415	38	1-4-4-4-4	51.75
	4	488	40	11-11-11	49	477	39	10-11-11-4	50.45	480	38	10-4-1-6-3	50.43
	5	504	41	10-8-3	46.58	509	41	10-6-6-10	46.12	488	40	3-10-6-6-9	47
	21	888	78	8-6-4	53.5	899	74	6-6-6-6	51.84	882	80	11-4-4-4-8	51.33
	22	964	81	10-6-4	51	967	81	10-10-6-4	51	975	81	10-10-10-6-3	50.27
	29	7168	676	5-4-4	55.47	7193	689	5-4-4-4	55.87	7156	688	5-4-4-10-10	49.94
	30	7274	678	5-10-10	50.58	7553	696	5-4-4-4	55	7108	666	5-5-10-10-10	50.25
2	6	536	47	11-11-3	77.33	537	45	3-11-8-1	75.25	547	46	8-6-6-4-1	75.25
	7	468	45	10-10-6	83	473	45	3-4-6-1	77	474	43	3-2-11-11-1	74.92
	8	390	48	3-10-6	75.68	389	47	3-3-6-3	73.22	397	47	3-11-3-6-3	73.53
	9	506	50	3-3-8	77.07	506	50	3-3-11-6	78.09	506	50	3-3-3-8-4	78.08
	10	533	48	3-10-8	76.33	544	49	10-10-10-4	74.53	536	47	1-3-3-3-8	73
	23	970	93	11-10-10	82.75	967	90	10-6-6-3	81.69	974	91	10-6-11-11-2	80.7
	24	864	93	11-11-10	75.51	862	93	11-11-11-4	76.26	863	92	11-11-11-11-7	80.26
	31	8662	845	4-4-10	74.62	8715	817	6-4-11-3	84	8950	810	6-6-6-11-3	74.83
	32	8087	808	11-11-4	74	8699	801	6-6-6-6	87	8655	810	6-6-6-11-3	74.62
3	11	477	39	1-6-6	53	488	39	1-6-6-3	53	452	37	1-6-1-11-3	50.12
	12	501	41	11-6-44	55.71	514	40	10-6-6-3	58	506	40	11-2-10-6-6	54.62
	13	445	41	11-11-8	51.96	420	37	6-2-10-3	48.59	437	41	3-4-10-6-6	50.45
	14	531	45	3-10-8	57.42	528	42	10-10-10-6	53.79	526	42	11-11-11-7-3	53
	15	406	37	1-11-4	55	396	37	4-11-4-3	51.64	412	37	1-11-11-11-6	53.16
	25	853	80	11-6-6	57	849	78	6-6-6-6	53.3	854	79	6-4-6-4-4	56.32
	26	787	76	3-3-10	53.61	818	72	3-6-6-6	52.15	838	74	3-3-6-11-11	52.98
	33	7415	709	5-4-4	57.69	7439	681	5-4-6-6	63	7088	651	5-5-6-6-6	63
	34	7136	649	5-6-10	65	6788	643	5-5-8-8	65	6496	629	5-5-5-6-6	56.19

Continue Table VI. Computational results obtained using the BSS procedure for $n_p = \{3, 4, 5\}$

Problem class	*Instance*	$n_p = 3$				$n_p = 4$				$n_p = 5$			
		$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	$BSSseq$	$C(\mathcal{S})$	$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	$BSSseq$	$C(\mathcal{S})$	$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	$BSSseq$	$C(\mathcal{S})$
4	16	540	50	10-9-11	85.25	540	50	4-4-4-11	84.08	540	50	4-4-4-10-11	85.66
	17	515	50	9-9-4	74.72	515	50	3-6-7-3	75	512	49	11-11-11-6-6	76
	18	510	50	3-3-8	75.92	510	50	11-11-11-8	75.92	501	49	11-11-11-9-4	75.92
	19	512	49	10-11-10	82.99	512	49	3-3-3-3	79.25	512	49	2-3-3-3-3	77.67
	20	564	50	3-9-4	82.9	564	50	3-11-11-11	81.91	561	49	3-4-9-6-6	81.74
	27	950	96	3-3-6	85	962	97	3-3-11-4	90	972	98	3-3-11-11-4	88.58
	28	1022	100	8-3-10	94	1022	100	3-11-6-6	92	1021	99	11-3-6-10-4	94
	35	8921	914	4-4-4	92.93	9029	890	6-6-6-6	78.31	9022	889	6-6-6-6-6	78.13
	36	9485	902	6-6-6	78.31	9387	896	6-6-6-6	90.98	9401	898	6-6-6-6-6	90.89

* BSSseq column numbers: 1: Slack Min, 2: Slack Max, 3: EST, 4: EFT, 5: LBMP, 6: LBF, 7: LPF, 8: ADMR, 9: DDMR, 10: EDF, 11: MBMRP.

Table VII. Results obtained from the algorithms NCDRA, NCDRA-without-BSS, and SA

Problem class	*Instance*	*NCDRA*		*NCDRA-without-BSS*		*SA*	
		$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$
1	1	**492**	**43**	483	42	**492**	**43**
	2	**534**	**42**	**534**	**42**	**534**	**42**
	3	500	39	**507**	**40**	**507**	**40**
	4	**543**	**43**	537	**43**	**543**	**43**
	5	**541**	**43**	**541**	**43**	531	**43**
	21	**994**	**81**	989	79	**994**	**81**
	22	1032	**86**	1023	85	**1033**	84
	29	**7721**	688	7252	**709**	7256	**709**
	30	**8135**	**738**	7483	712	7364	706
2	6	**560**	**50**	558	48	**560**	**50**
	7	**502**	**46**	495	45	**502**	**46**
	8	**413**	**50**	412	49	**413**	**50**
	9	**506**	**50**	**506**	**50**	**506**	**50**

Continue Table VII. Results obtained from the algorithms NCDRA, NCDRA-without-BSS, and SA

Problem class	*Instance*	*NCDRA*		*NCDRA-without-BSS*		*SA*	
		$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$	$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$
2	10	**552**	**50**	551	49	**552**	**50**
	23	**998**	**95**	997	**95**	**998**	**95**
	24	**890**	**100**	888	98	**890**	**100**
	31	**8827**	826	8609	836	8614	**841**
	32	**8631**	848	8253	845	8310	**850**
3	11	**523**	**45**	518	43	**523**	**45**
	12	**567**	**44**	561	43	564	**44**
	13	**497**	**44**	490	42	**497**	**44**
	14	**566**	**48**	558	46	565	**48**
	15	**482**	**44**	472	42	481	43
	25	**965**	**87**	959	85	**965**	**87**
	26	947	**82**	**952**	**82**	**952**	**82**
	33	**7937**	723	7423	724	7425	**730**
	34	**7810**	**754**	7223	710	7276	720
4	16	**540**	**50**	**540**	**50**	**540**	**50**
	17	**515**	**50**	**515**	**50**	**515**	**50**
	18	**510**	**50**	**510**	**50**	**510**	**50**
	19	**514**	**50**	**514**	**50**	**514**	**50**
	20	**564**	**50**	**564**	**50**	**564**	**50**
	27	**986**	**100**	**986**	**100**	**986**	**100**
	28	**1022**	**100**	**1022**	**100**	**1022**	**100**
	35	**9129**	888	9041	883	8879	**915**
	36	**9461**	**932**	9331	854	9261	919

E. Results of executing NCDRA on identical parallel machines

In this section, the problem $P/r_j, d_j/w_1 * \sum b_j + w_2 * |\mathcal{J}(\mathcal{S})|$ is investigated. NCDRA that was implemented on the problem instances is as same as the one used above, except that machine speed in all of them is equal to 1 ($s_i = \{1\}$) and then the results of which are reported in Table VIII. In this situation, average machine speed has been increased to 1, thereby increasing both benefits and the number of jobs processed. For instance, the average machine speed in problem instances 1 through 5 was equal to 0.96, for which 210 jobs out of the 250 available were completed. In contrast, 277 jobs were processed under the new conditions, indicating that an increase of 0.04 in machine speed led to an increase of 6.8% in the number of jobs processed.

Table VIII. Computational results of NCDRA for $P/r_j, d_j/w_1 * \sum b_j + w_2 * |\mathcal{J}(\mathcal{S})|$

Problem class	*Instance*	$\sum b_i$	$\lvert\mathcal{J}(\mathcal{S})\rvert$
1	1	505	46
	2	542	44
	3	535	44
	4	559	46
	5	567	47
	21	1024	87
	22	1070	90
	29	7703	739
	30	8369	702
2	6	560	50
	7	513	50
	8	413	50
	9	506	50
	10	552	50
	23	1008	99
	24	890	100
	31	8859	867
	32	8881	821
3	11	531	46
	12	588	47
	13	504	46
	14	570	50
	15	500	45
	25	984	92
	26	976	87
	33	8566	725
	34	7825	755
4	16	540	50
	17	515	50
	18	510	50
	19	514	50
	20	564	50

Continue Table VIII. Computational results of NCDRA for $P/r_j, d_j/w_1 * \sum b_j + w_2 * |J(S)|$

Problem class	*Instance*	$\sum b_i$	$\lvert J(S)\rvert$
4	27	986	100
	28	1022	100
	35	8946	918
	36	9562	936

V. CONCLUSION

A mathematical model and a heuristic algorithm, named NCDRA, were developed for solving the problem $Q/r_j, d_j/w_1 * \sum b_j + w_2 * |J(S)|$. It consisted of two phases, the first of which involved applying a method called segmentation to DRs. The BSS procedure was also used to enhance these rules' effectiveness and improve the upper bound. Phase II involved customizing the three exploration operators of swapping, insertion, and inversion as well as their concurrent implementation on the schedule obtained from the BSS to search the neighborhood.

The algorithm developed in this paper was then implemented on problem instances, and the results were recorded. Furthermore, the different parameters of segment length, number of segments, and machine selection policies affecting the performance of the algorithm were investigated. The results obtained from this algorithm and SA were compared with respect to the objective criteria, including the number of jobs processed and benefits. NCDRA was observed to yield better results compared to SA.

Future studies are suggested to consider practical conditions as machine eligibility constraints and machine breakdowns. Moreover, one can investigate the addressed problem considering stochastic or fuzzy processing times and evaluate the robustness of the solutions. As future other lines of research, it would improve the performance of the proposed algorithm by employing predictive methods and techniques in the LS phase. Concurrent employment of strengthened learning and job clustering in the LS phase is expected to lead to greater effects of exploration operators on the solutions found.

REFERENCES

Bard, J. F., & Rojanasoonthon, S. (2006). A branch and price algorithm for parallel machine scheduling with time windows and job priorities. *Naval Research Logistics*, 53(1), 24-44.

Bitar, A., Dauzère-Pérès, S., Yugma, C., & Roussel, R. (2016). A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing. *Journal of Scheduling*, 19(4), 367-376.

Cao, D., Chen, M., & Wan, G. (2005). Parallel machine selection and job scheduling to minimize machine cost and job tardiness. *Computers & Operations Research*, 32(8), 1995-2012.

Croce, D. F., T'kindt V. & Ploton, O. (2021). Parallel machine scheduling with minimum number of tardy jobs: Approximation and exponential algorithms. *Applied Mathematics and Computation*, 397, 125888.

Chen, B., & Vestjens, A.P.A. (1997). Scheduling on identical machines: how good is LPT in an on-line setting?. *Operational Research Letters*, 21, 165-169.

Chung, S.H., Pearn, W.L. & Tai, Y.T. (2009). Fast and effective algorithms for the liquid crystal display module (LCM) scheduling problem with sequence-dependent setup time. Journal of the Operational Research Society, 60, 921–933.

Fanjul-Peyro, L., & Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1), 55-69.

Gholami, O., Sotskov, Y. N., Werner, F., & Zatsiupo, A. S. (2019). Heuristic algorithms to maximize revenue and the number of jobs processed on parallel machines. *Automation and Remote Control*, 80(2), 297-316.

Hoseini, S.F., Omran, M.M., Márquez, A.C. & Makui, A. (2018). Simultaneous optimisation of seaside operations in container terminals: a case study of the Iranian Rajaee port. *International Journal of Shipping and Transport Logistics*, 10(5-6), 587-617.

Huang, J., Li, S., & Chen, Y. (2020). Revenue-optimal task scheduling and resource management for IoT batch jobs in mobile edge computing. *Networking and Applications*, 13, 1776–1787.

Kaban, A. K., Othman, Z., & Rohmah, D. S. (2012). Comparison of dispatching rules in job-shop scheduling problem using simulation: a case study. *International Journal of Simulation Modelling*, 11(3), 129-140.

Kowalczyk, D., & Leus, R. (2017). An exact algorithm for parallel machine scheduling with conflicts. *Journal of Scheduling*, 20(4), 355-372.

Islam, M., Khanna, G., & Sadayappan, P. (2008). Revenue Maximization in Market-Based Parallel Job Schedulers. Ohio State University Library.

Joo, C. M., & Kim, B. S. (2015). Hybrid genetic algorithms with dispatching rules for unrelated parallel machine scheduling with setup time and production availability. *Computers & Industrial Engineering*, 85, 102-109.

Juraszek, J., Sterna, M., & Pesch, E., (2009). Revenue maximization on parallel machines. Institute of Computing Science, Poznan University of Technology, 960-965.

Laha, D., & Gupta, J. N. (2018). An improved cuckoo search algorithm for scheduling jobs on identical parallel machines". *Computers & Industrial Engineering*, 126, 348-360.

Lee, C. H., (2018). A dispatching rule and a random iterated greedy metaheuristic for identical parallel machine scheduling to minimize total tardiness. *International Journal of Production Research*, 56(6), 2292-2308.

Mensendiek, A., Gupta, J. N., & Herrmann, J. (2015). Scheduling identical parallel machines with fixed delivery dates to minimize total tardiness. *European Journal of Operational Research*, 243(2), 514-522.

Nattaf, M., Dauzère-Pérès, S., Yugma, C., & Wu, C. H. (2019). Parallel machine scheduling with time constraints on machine qualifications. *Computers & Operations Research*, 107, 61-76.

Pinedo, M. (2012). Scheduling (Vol. 29). New York: Springer.

Phanden, R. K., Jain, A., & Verma, R. (2013). An approach for integration of process planning and scheduling. *International Journal of Computer Integrated Manufacturing*, 26(4), 284-302. doi:10.1080/0951192X.2012.684721.

Rojanasoonthon, S., Bard, J. F., & Reddy, S. D. (2003). Algorithms for parallel machine scheduling: a case study of the tracking and data relay satellite system. *Journal of the Operational Research Society*, 54(8), 806-821.

Raghu, T. S., & Rajendran, C. (1993). An efficient dynamic dispatching rule for scheduling in a job shop. *International Journal of Production Economics*, 32(3), 301-313.

Yang-Kuei, L., & Chi-Wei, L. (2013). Dispatching rules for unrelated parallel machine scheduling with release dates. *The International Journal of Advanced Manufacturing Technology*, 67(1-4), 269-279. doi:10.1007/s00170-013-4773-8.