J. Soc. Korea Ind. Syst. Eng Vol. 38, No. 2 : 47-55, June 2015
http://dx.doi.org/10.11627/jkise.2015.38.2.47
ISSN : 2005-0461(print)
ISSN : 2287-7975(online)

Rescheduling on Parallel Machines with Compressible Processing Times

Suhwan Kim[†]

Department of Military Operations Research, Korea National Defense University

작업시간이 압축 가능한 경우 병렬기계의 재일정계획

김 수 환[†]

국방대학교 군사운영분석학과

This paper deals with rescheduling on unrelated parallel-machines with compressible processing times, assuming that the arrival of a set of new jobs triggers rescheduling. It formulates this rescheduling problem as an assignment problem with a side constraint and proposes a heuristic to solve it. Computational tests evaluate the efficacy of the heuristic.

Keywords : Rescheduling, Parallel Machines, Compressible Processing Time, Heuristic

1. Introduction

It is very common that a set of jobs being processed according to a schedule must be rescheduled because of a disruption. Events that might cause a disruption include the arrival of a new job, the breakdown of a machine, or the unavailability of materials. Most deterministic machine scheduling problems deal with constant job processing times. In various real-life systems, however, processing times may be compressed by allocating additional resources, such as money, overtime, energy, fuel, catalysts, or manpower.

This paper deals with rescheduling on unrelated parallel machines with compressible processing times and proposes a heuristic. We assume that the original set of jobs had been optimally scheduled on unrelated parallel machines to minimize the sum of completion times and the costs of compressing processing times and that a disruption occurs when a set of new jobs arrives. In rescheduling, some additional measures may be needed, because any change of the original schedule may lead to additional cost (e.g., transportation) and rescheduling effort (e.g., job reassignment, workforce rescheduling, and/or customer delivery-date changes). Rescheduling parallel machines may result in reassigning some job to a machine other than the one on which it was originally scheduled, a result we call *machine reassignment*. We consider the disruptions caused by machine reassignments, either as part of the objective function or by imposing a constraint to limit their number. When the cost of reassignment is difficult to estimate, we limit the number of machine reassignments by imposing a constraint, resulting in an assignment problem with a side constraint (APSC) (i.e., an NP-hard problem [13]).

Several papers have contributed to rescheduling on parallel-machines with machine reassignment as an additional measure. For example, Alagoz and Azizoglu [4] and Azizoglu and Alagoz [6] studied the objective of minimizing the sum of completion times and either included penalties for machine reassignment in the objective function or limited the

Received 3 Fabruary 2015; Finally Revised 20 March 2015; Accepted 20 March 2015
† Corresponding Author : ksuhwan@kndu.ac.kr

number of reassignments permitted with an additional constraint. Curry and Peters [8] limited machine reassignments by invoking a constraint and used simulation to demonstrate the effect of reassignment in the parallel-machine scheduling environment. Ozen and Azizoglu [15] considered unrelated parallel machines to minimize the total flow time, restricting the total reassignment cost.

Compressible processing times can provide flexibility for rescheduling by managing processing times. Scheduling with compressible processing times has been studied extensively. Shabtay and Steiner [19] gave an extensive survey on this topic. Gurel and Akturk [11] studied a single machine with compressible processing times with different non-linear compressible cost functions. Yin and Wang [22] studied a single machine to minimize a cost function containing makespan, considering compressible processing times and learning effect. Yang [21] studied rescheduling on a single machine configuration with compressible processing times, allowing time compressions only for newly arrived jobs. Akturk et. al. [3] introduced parallel machine match-up scheduling problems to cope with machine disruptions when processing times of the jobs are compressible at a certain manufacturing cost. Gurel and Cincioglu [10] studied a parallel machine rescheduling problem with compressible processing times for number of disrupted jobs. They considered a bicriteria rescheduling problem to deal with a trade-off between the number of on-time jobs and manufacturing cost, admitting machine reassignment.

Several solution approaches have been proposed for APSC. Murty [14] proposed a ranking algorithm, which generates all assignments in order of increasing cost and selects the least-cost assignment that satisfies the side constraint. Gupta and Sharma [9] developed an enumeration scheme using a search tree in which each node represents a complete, feasible solution, so that a node can be fathomed if the solution it represents is infeasible with respect to the side constraint. Aggarwal [2] presented a two-stage algorithm that applies Lagrange relaxation to eliminate the side constraint and then searches for an optimal solution using Murty's [14] ranking algorithm. Aboudi and Jornsten [1] took a polyhedral approach, presenting several classes of valid inequalities and associated separation problems. Mazzola and Neebe [13] presented a branch-and-bound (B&B) algorithm that incorporates the subgradient method as part of a bounding procedure and a subgradient-based heuristic. Punnen and Aneja [18] observed that the Mazzola-Neebe heuristic is sensitive to the range of the coefficients in the objective function and in the side constraint. Lieshout and Volgenant [12] proposed a B&B algorithm, obtaining lower bounds by utilizing Lagrange relaxation and a heuristic by modifying the B&B algorithm. They mentioned that the B&B algorithm and their heuristic were also affected by the range of coefficients: if the ranges double, their run times also double.

In our study, we consider rescheduling on unrelated parallel machines with compressible processing times and propose a heuristic which is less sensitive to the range of coefficients. The body of this paper is organized in four sections. Section 2 formulates the scheduling and rescheduling problems we study. Section 3 presents our LP(Linear Programming)-based heuristic for APSC. Section 4 describes a computational evaluation of the proposed heuristic. Finally, section 5 relates our conclusions and fertile directions for future research.

2. Problem Definitions

This section is comprised of four subsections: the first introduces the underlying scheduling problem and each of the subsequent three subsections formulates a different version of the rescheduling problem. The first version includes a cost for each machine reassignment; the second incorporates a constraint to limit the number of such reassignments; and the third minimizes compression costs and introduces a constraint to limit the sum of completion times. We assume that rescheduling is triggered by the arrival of a set of new jobs.

2.1 $Rm|x_{ij} \le u_{ij}|\Sigma C_j + \Sigma c_{ij}x_{ij}$

We use the standard three-field classification scheme (i.e., $\alpha|\beta|\gamma$) of Pinedo [14] to denote the scheduling problem as $Rm|x_{ij} \le u_{ij}|\Sigma C_j + \Sigma c_{ij}x_{ij}$, where the α field uses Rm to denote m unrelated parallel machines; the β field gives x_{ij}, the actual amount of time compression for job j on machine $i(0 \le x_{ij} \le p_{ij})$; and u_{ij}, the maximum possible time compression for job j on machine $i(0 \le x_{ij} \le \min(u_{ij}, p_{ij}))$; the γ field specifies the objective, which involves the sum of completion times, C_j, and compression costs for which c_{ij} denotes the cost per unit time of compressing job j on machine i. This subsection formulates this scheduling problem. Throughout this paper, we use the following notation :

Indices

i	machines
j	jobs
k	sequence positions

Index sets

I	unrelated parallel machines; $I=\{1, \cdots, m\}$
J	jobs to be processed; $J=\{1, \cdots, n\}$
K	sequence positions; $K=J$

Parameters

C_j	completion time of job j
c_{ij}	cost per unit time of compressing job j on machine i
p_{ij}	normal (*i.e.*, uncompressed) processing time of job j on machine i
u_{ij}	maximum possible time compression of job j on machine i

Decision variables

x_{ij}	amount of processing time compression of job j on machine i (i.e., actual processing time is $(p_{ij}-x_{ij})$)
y_{ijk}	1 if job j is scheduled as the k^{th} to last job on machine i, 0 otherwise.

For this scheduling problem, Alidaee and Ahmadian [5] proved the following Proposition, which is based on a result of Vickson [20] :

Proposition In the $Rm|x_{ij} \le u_{ij}|\Sigma C_j + \Sigma c_{ij}x_{ij}$ scheduling problem, there exists an optimal schedule in which $x_{ij}=0$ if $c_{ij} \ge k$ and $x_{ij}=u_{ij}$ if $c_{ij} <$ (i.e., its processing time is not changed or fully compressed from the normal processing time (p_{ij}), depending on the sequence position of the job.) (*Proof.* see [5]). □

Based on the Proposition, the underlying scheduling problem can be formulated by extending the $Rm|\Sigma C_j$ scheduling model as follows [17] :

$P(0)$:

$$\text{Min} \quad \sum_{i\in I}\sum_{j\in J}\sum_{k\in K}[kp_{ij}+\min\{0, (c_{ij}-k)\mu_{ij}\}]y_{ijk} \qquad (1)$$

$$s.t. \quad \sum_{i\in I}\sum_{k\in K} y_{ijk}=1 \qquad j\in J \qquad (2)$$

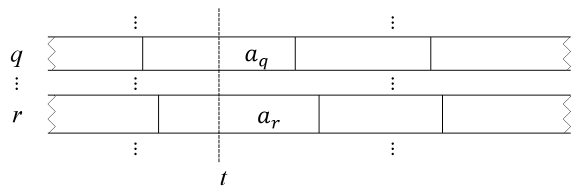
$$\sum_{j\in J} y_{ijk} \le 1 \qquad i\in I,\ k\in K \qquad (3)$$

$$y_{ijk}\in\{0, 1\} \qquad i\in I,\ j\in J,\ k\in K \qquad (4)$$

Objective function (1) minimizes the sum of completion times and compression costs. Constraints (2) assure that each job is scheduled exactly once and (3) assure that each position on each machine is taken by at most one job. Constraints (4) require all decision variables to be nonnegative. Problem $P(0)$ is the well-known assignment problem, which is totally unimodular, so that an optimal integer solution can be prescribed by linear programming in polynomial time.

2.2 $Rm|x_{ij} \le u_{ij}, a_i|\Sigma C_j + \Sigma c_{ij}x_{ij} + \Sigma_{j\in J_R} w_{ij}D_{ij}$

We assume that a disruption occurs when a (index) set J_N of new jobs arrives at time t as a schedule for the (index) set of original jobs, J_o, is ongoing. In particular, jobs in (index) set J_P are in process at time t, each on a scheduled machine, which will become available only after completing the job it is processing.



<Figure 1> Jobs in-process at Time t

For example, in <Figure 1>, a set of new jobs arrives at time t, machines q and r are processing scheduled jobs, which have remaining processing times a_q and a_r, respectively. In this case, machines q and r will be available at times $t+a_q$ and $t+a_r$, respectively. Upon arrival of a set of new jobs, delete the sets of completed (J_C) and in-process (J_P) jobs from J_o, and re-index remaining jobs in set $J'=\{J_o \backslash (J_P \cup J_c)\} \cup J_N$. Without loss of generality, we assume throughout this paper that a set of new jobs arrives at time 0.

In this and the following subsections, we formulate important variations of the rescheduling problem. First, we introduce some additional notation :

J_R	index set of originally scheduled jobs remaining after deleting in-process and completed jobs; $J_o \backslash (J_P \cup J_C)$
J_N	index set of newly arrived jobs
J'	index set of jobs to be rescheduled; $J_R \cup J_N$
i_j	machine to which job $j\in J_R$ is assigned in the initial schedule
a_i	time at which machine i will be available, given that new jobs arrive at time 0
w_{ij}	cost of reassigning job $j\in J_R$ onto machine $i\in I\backslash\{i_j\}$
D_{ij}	1 if job $j\in J_R$ is reassigned onto machine $i\in I\backslash\{i_j\}$, 0 otherwise.

We penalize the amount of inconvenience reassignments cause as a part of the objective function or limit the permissible number of reassignments (i.e., amount of inconvenience) by incorporating a constraint. For example, failure to meet a delivery date may affect customer goodwill to a degree that is hard to estimate and, therefore, the amount of inconvenience can be limited by incorporating a constraint to restrict the number of reassignments permitted. However, if machine reassignment incurs additional costs that can be relatively easily valued, the objective function can reflect the trade-off between the costs associated with rescheduling and inconvenience.

The first version of the rescheduling problem can be designated by $Rm\,|\,x_{ij} \le u_{ij},\ a_i\,|\,\Sigma C_j + \Sigma c_{ij}x_{ij} + \Sigma_{j\in J_R} w_{ij}D_{ij}$, which includes an additional measure, $\Sigma_{j\in J_R} w_{ij}D_{ij}$, the total weighted cost of reassigning jobs in set J_R, each of which had been scheduled originally on a different machine. To clarify, the reschedule can assign job j to the same machine (i_j) on which it had been scheduled or reassign it to a different machine $(i\in I\backslash\{i_j\})$. We assume that a job that is being processed when rescheduling occurs remains on the machine that is processing it and its processing time is not compressed.

Since the symbol a_i in the β field represents the time at which machine i will be available; $a_i = 0$ if machine i is not processing a job when the set of new jobs arrives at time 0. If a job $j\in J_R$ is reassigned, $\Sigma_{i\in I\diagdown\{i_j\}} D_{ij} = \Sigma_{i\in I\diagdown\{i_j\}}\Sigma_{k\in J} y_{ijk} = 1$; otherwise, job j remains assigned to machine i_j according to the original schedule, so that $\Sigma_{i\in I\diagdown\{i_j\}} D_{ij} = \Sigma_{i\in I\diagdown\{i_j\}}\Sigma_{k\in J} y_{ijk} = 0$ Rescheduling problem $Rm\,|\,x_{ij} \le u_{ij},\ a_i|\Sigma C_j + \Sigma c_{ij}x_{ij+}\Sigma_{j\in J_R} w_{ij}D_{ij}$ can be formulated as assignment problem P(1).

P(1):

$$\textit{Min}\quad \sum_{i\in I}\sum_{j\in J}\sum_{k\in K}[kp_{ij}+a_i+\min\{0,\,(c_{ij}-k)u_{ij}\}]y_{ijk} + \sum_{j\in J_R}\sum_{i\in I\diagdown\{i_j\}}\sum_{k\in K} w_{ij}y_{ijk} \tag{5}$$

s.t. (2)-(4).

Constraints are those of the assignment problem, so that P(10) can be solved in polynomial time.

2.3 $Rm\,|\,x_{ij} \le u_{ij},\ a_i, \Sigma_{j\in J_R} D_{ij} \le U\,|\,\Sigma C_j + \Sigma c_{ij}x_{ij}$

When the cost of reassignment, (e.g., w_{ij} in P(1)), is difficult to estimate, it may be more appropriate to limit the number of jobs reassigned by imposing a constraint. This version of the rescheduling problem incorporates a constraint to impose a limitation, U, to restrict machine reassignments.

P(2):

$$\textit{Min}\quad \sum_{i\in I}\sum_{j\in J}\sum_{k\in K}[kp_{ij}+a_i+\min\{0,\,(c_{ij}-k)u_{ij}\}]y_{ijk}$$

s.t. (2)-(4).

$$\sum_{j\in J_R}\sum_{i\in I\diagdown\{i_j\}}\sum_{k\in K} y_{ijk} \le U. \tag{6}$$

Constraint (6) invokes the reassignment limitation. This APSC does not possess the unimodularity property; it is NP-hard [16].

2.4 $Rm\,|\,x_{ij} \le u_{ij},\ a_i, \Sigma C_j \le T\,|\,\Sigma c_{ij}x_{ij}$

In this subsection, we consider that compressible processing times can provide flexibility for rescheduling by managing processing times. Rescheduling upon the arrival of a set of new jobs may increase the sum of completion times, even of originally scheduled jobs. Furthermore, it may be important to allow processing time compression to satisfy customer demands in a more timely manner; thus, total cost may increase due to time compression.

This problem can be designated by $Rm\,|\,x_{ij} \le u_{ij},\ a_i, \Sigma C_j \le T\,|\,\Sigma c_{ij}x_{ij}$, which invokes limit, T, on the sum of completion times. To formulate this problem as an integer program, we introduce the following decision variables, assuming that all processing and compression times are integers.

y_{ijkl}	1 if job j is scheduled as the k^{th} to last job on machine i and its time compression is the integer value l, 0 otherwise

We also formulate this problem as an APSC as follows :

P(3):

$$\textit{Min}\quad \sum_{i\in I}\sum_{j\in J}\sum_{k\in K}\sum_{k\in K} c_{ij}l\,y_{ijkl} \tag{7}$$

$$\textit{s.t.}\quad \sum_{i\in I}\sum_{k\in K}\sum_{l\in U_{ij}} y_{ijkl} = 1 \qquad j\in J' \tag{8}$$

$$\sum_{j\in J}\sum_{I\in U_{ij}} y_{ijkl} \le 1 \qquad i\in l,\ k\in K \tag{9}$$

$$\sum_{i\in I}\sum_{j\in J'}\sum_{k\in K}\sum_{l\in U_{ij}}\{a_i + k(p_{ij} - l)\}y_{ijkl} \le T \tag{10}$$

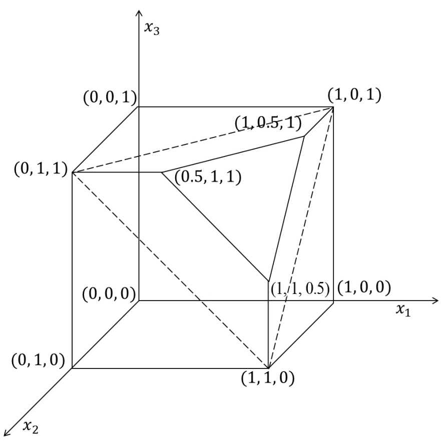
$$y_{ijkl} \in \{0,\,1\} \qquad i\in I,\ j\in J',\ k\in K,\ l\in U_{ij}. \tag{11}$$

where U_{ij} denotes the set of possible compression times for job j on machine i (i.e., $U_{ij} = \{0, 1, \cdots, u_{ij}\}$). Objective function (7) minimizes the total cost of processing time compression. Constraints (8) and (9) assure that each job is scheduled exactly once and each position on each machine is taken by at most one job, respectively. Constraint (10) assures that the sum of completion times does not exceed the limit, T. P(3) has also an additional constraint (10) that makes this problem NP-hard [16].

3. The Linear Relaxation Heuristic

In this section, we propose a heuristic, the Linear Relaxation Heuristic (LRH), which is based on the linear relaxations of P(2) and P(3). Our LRH can be applied to any model that possesses the totally unimodular property, augmented with a single side constraint. It solves the linear relaxation once and combines the logic of sensitivity analysis applied to the right hand side (RHS) of the side constraint with the fact that all extreme points are 0, 1 integer vectors. Note that sensitivity analysis of the RHS specifies the amount by which the RHS value can change before the current basis changes.

To explain the idea of our heuristic, we present an example with feasible region $F = \{(x_1, x_2, x_3) | x_1 + x_2 + x_3 \leq 2.5,\ x_1 \leq 1, x_2 \leq 1,\ x_3 \leq 1, x_1 \geq 0, x_2 \geq 0, x_3 \geq 0\}$ and cost vector $c = (-0.5, -1, -1)$. <Figure 2> shows that the hyperplane represented by the first constraint in the definition of F cuts through the (unit hypercube) polytope described by the other inequalities, forming three fractional extreme points



<Figure 2> Constraint $x_1 + x_2 + x_3 \leq 2.5$ with Different RHS Values

(i.e., (0.5, 1, 1), (1, 0.5, 1), and (1, 1, 0.5)).

In <Figure 2>, as the RHS of inequality $x_1 + x_2 + x_3 \leq 2.5$ is gradually decreased, the basis does not change but the objective value slowly increases until the RHS reaches the value 2 and the translated hyperplane (represented by the dotted lines) intersects extreme points (0, 1, 1), (1, 0, 1), and (1, 1, 0). Since sensitivity analysis of the RHS provides lower and upper bounds on the RHS coefficient within which the current basis remains optimal, we can use it to determine how much the RHS must change to translate the hyperplane far enough to intersect an adjacent, extreme point, which must be integer, because the underlying polytope is, by assumption, totally unimodular. Like APSC, all extreme points of feasible region F are composed of 0, 1 integer vectors except any fractional extreme points formed by the additional constraint.

Our heuristic solves the linear relaxation of APSC once, then performs a sensitivity analysis of the RHS to determine the value that changes the basis, equivalently, translates the hyperplane until it intersects at least one feasible, integer extreme point of the assignment polytope. If the solution is fractional, the RHS value of the side constraint is replaced with the lower bound of the range and then the heuristic finds an adjacent extreme-point solution by multiplying the inverse of the current basis by the replaced RHS vector. Since the constrained optimal solution to APSC is not always adjacent to the current fractional solution, the heuristic does not always guarantee the optimal solution.

Let z_{LP}^*, δ^*, and $l \leq R \leq u$ denote the optimal solution value, the optimal dual variable value corresponding to the side constraint, and the sensitivity range for the RHS value R of the side constraint in the linear relaxation of APSC, respectively.

Property If $l = R$ or $l \neq R$ and $\delta^* = 0$, the current solution is optimal for the APSC.

Proof. If $l = R$, the side constraint intersects the underlying polytope at an optimal integer extreme point. Now, suppose $l \neq R$ and $\delta^* = 0$. If the RHS of the side constraint in APSC is decreased by $R - l$, the new objective value is $z_{LP}^* + \delta^* (l - R)$. Since $\delta^* = 0$, there is no change in the objective value. This happens when the side constraint is redundant (e.g., if the orientation of the objective function is such that the solution to the linear relaxation is an integer extreme point). □

Letting B and $\begin{bmatrix} 1 \\ R \end{bmatrix}$ be the optimal basis of the linear relaxation of APSC and RHS vector with R as the RHS value of the side constraint, respectively, we now detail LRH in application to solve APSC.

```
LRH
Solve the linear relaxation of APSC.
  If infeasible, STOP : APSC is infeasible.
  Else, Apply sensitivity analysis to determine lower and
  upper bounds for the RHS coefficient of the side con-
  straint, $l \le R \le u$, that allow the current basis to remain
  optimal,
      If the solution is integer, STOP : current solution is
      optimal.
      Else, compute $B^{-1}\begin{bmatrix} 1 \\ l \end{bmatrix}$ to find the new extreme-point
      solution.
```

4. Computational Evaluation

In this section, we evaluate the efficacy of LRH using randomly generated instances of problems P(2) and P(3). We program LRH, using the C/C++ language and the CPLEX 12.1.0 callable library, and perform all computations on a Dell PC running Windows 7 with a 2.67 GHz CPU and 2 GB memory. The first subsection describes test instances and the second relates test results.

4.1 Instance Generation

This section reviews instance generation. For problem P(2), our tests involve three factors, the numbers of machines, old jobs, and new jobs with three levels of each factor : 2, 5, and 10; and 100, 200, and 600; and 10, 50, and 100; respectively. This results in 27 cases as shown in <Table 1> and we generate 20 instances of each randomly.

We generate each independent instance randomly using the discrete uniform distribution (DU); $DU[10, 110]$ for processing times (p_{ij}), costs of processing time compressions (c_{ij}), and machine available times (a_i); $DU[0, 0.5 \times p_{ij}]$, for the maximum possible processing time compressions (u_{ij}); and, finally, $DU[0.2 \times |J_R|, 0.8 \times |J_R|]$ for the machine reassignment limit (U) in which $|J_R|$ denotes the number of old jobs to be rescheduled.

<Table 1> Test Instances for Problem P(2)

Cases	# of machines	# of old jobs	# of new jobs
1, 10, 19	2, 5, 10	100	10
2, 11, 20	2, 5, 10	100	50
3, 12, 21	2, 5, 10	100	100
4, 13, 22	2, 5, 10	200	10
5, 14, 23	2, 5, 10	200	50
6, 15, 24	2, 5, 10	200	100
7, 16, 25	2, 5, 10	600	10
8, 17, 26	2, 5, 10	600	50
9, 18, 27	2, 5, 10	600	100

For problem P(3), we consider the first factor with three levels (2, 5 and 10 machines) but limit the second and third factors to one level (100 old and 50 new jobs) because this problem incorporates variables with four indices and, hence, causes memory issues if more jobs are involved. <Table 2> displays those instances.

<Table 2> Test Instances for Problem P(3)

Case	# of machines	# of old jobs	# of new jobs
28	2	100	50
29	5	100	50
30	10	100	50

Again, we generate 20 independent instances for each case as described above, except that we generate the limit on the sum of completion times as follows

$$T = DU[0.6 \times t_{\min}, 0.9 \times t_{\min}]$$

where
$t_{\min} = Min\{\Sigma_{i \in I} \Sigma_{j \in J} \Sigma_{k \in J} \Sigma_{l \in U_{ij}} kp_{ij}y_{ijkl}$: s.t. (8), (9) and (11)$\}$ (i.e., the minimum value of the sum of completion times when processing time compression is disallowed). Depending on the value of $t_{\min}$, problem (3) can be infeasible. Thus, we repeat instance generations until we get 20 feasible instances.

4.2 Computational Results

All computational results are averages taken over the 20 instances associated with each case. <Table 3> displays test results for cases 1 through 27. Column N_{opt} gives the number of instances out of 20 that our heuristic solves to optimality. Columns RD_{avg}, $RD_{\max}$, and $RD_{\min}$ provide average, maximum, and minimum relative deviations from the optimal value, respectively.

<Table 3> Test Results for Cases 1 through 27

Case	N_{opt} (#(%))	RD_{avg} (%)	$RD_{\max}$ (%)	$RD_{\min}$ (%)	TR_{avg}	$TR_{\max}$	$TR_{\min}$	T_{avg} (secs)
1	17(85)	0.030	0.354	0	4.566	10.440	2.324	0.098
2	15(75)	0.063	0.519	0	4.801	11.968	3.201	0.206
3	18(90)	0.006	0.089	0	3.292	4.304	2.807	0.490
4	17(85)	0.043	0.339	0	3.270	4.487	2.469	0.612
5	17(85)	0.006	0.099	0	3.586	5.655	2.633	0.897
6	17(85)	0.019	0.180	0	3.539	5.314	2.210	1.595
7	18(90)	0.004	0.051	0	7.868	10.581	5.967	8.752
8	17(85)	0.007	0.084	0	8.290	9.949	6.570	9.984
9	18(90)	0.005	0.095	0	9.775	17.374	7.629	11.171
Average	17.1(85.5)	0.020			5.443			3.756
10	17(85)	0.074	0.967	0	3.335	5.936	2.455	0.267
11	16(80)	0.106	0.969	0	2.502	4.555	2.020	0.694
12	16(80)	0.051	0.650	0	1.934	2.941	1.392	1.864
13	10(50)	0.147	0.627	0	2.186	4.195	1.458	2.356
14	14(70)	0.089	0.605	0	2.122	3.785	1.509	3.505
15	15(75)	0.068	0.522	0	2.293	4.174	1.658	5.106
16	15(75)	0.026	0.238	0	4.503	5.665	3.628	20.306
17	13(65)	0.026	0.166	0	4.869	6.508	3.922	22.710
18	15(75)	0.018	0.175	0	4.934	6.270	3.828	26.981
Average	14.6(72.8)	0.067			3.186			9.310
19	13(65)	0.263	1.442	0	2.575	4.146	1.959	0.755
20	18(90)	0.070	0.946	0	2.098	3.952	1.563	1.790
21	17(85)	0.063	0.623	0	1.779	3.012	1.515	3.800
22	11(55)	0.187	0.785	0	2.140	3.372	1.671	4.256
23	16(80)	0.093	0.600	0	2.013	3.777	1.549	6.778
24	16(80)	0.044	0.327	0	1.887	2.907	1.451	10.695
25	15(75)	0.038	0.262	0	3.611	5.077	2.581	43.042
26	19(95)	0.014	0.285	0	3.411	4.477	2.500	49.057
27	16(80)	0.053	0.375	0	3.655	4.468	2.711	58.771
Average	15.7(78.3)	0.092			2.574			19.883

The relative deviation RD is computed as follows :

$$RD = \frac{z_H - z_{IP}}{z_{IP}} \times 100$$

in which z_H and z_{IP} denote the objective value of LRH and the optimal solution, respectively. Columns TR_{avg}, $TR_{\max}$, and $TR_{\min}$, give average, maximum, and minimum values of CPLEX run time divided by LRH run time, respectively. The last column gives T_{avg}, the average run time of LRH in each case.

Rows 1~9, 10~18, and 19~27 give results for 2, 5, and 10 machines, respectively. The underlined rows following rows 9, 18 and 27 report the average number of optimal solutions out of 20, RD_{avg} and TR_{avg} for all instances involving 2, 5, and 10 machines, respectively.

LRH optimally solves about 78.9% of our test instances. For remaining instances, it provides near-optimal solutions, resulting in an average relative deviation over all test instances of 0.06%. Column TR_{avg} shows that LRH solves instances about 3.7 times faster than CPLEX, on average. In particular, LRH solves instances that have many jobs relative to the number of machines much faster than CPLEX. For example, LRH solves case 9 about 9.8 times faster than CPLEX, on average.

We also compare cases in which the ratio of the number of old jobs to the number of new jobs is relatively high with cases for which the ratio is small, in particular, equal to 1. For this test, we adapt medium-sized cases 11, 14, and 17, which all have 50 new jobs but 100, 200, and 600 old jobs, giving ratios of 2, 4 and 12, respectively. To specify new cases, 11^d, 14^d, and 17^d, we change the numbers of old and new jobs, keeping the same total number of jobs in each case (150, 250 and 650, respectively), to make each ratio equal to 1, resulting in 75, 125, and 325 old as well as new jobs in these cases, respectively.

<Table 4> Comparison of Cases i and i^d for $i=$ 11, 14, 17

Case	Ratio #old/#new	N_{opt} (#(%))	RD_{avg} (%)	$RD_{\max}$ (%)	$RD_{\min}$ (%)	T_{avg} (secs)	$T_{\max}$ (secs)	$T_{\min}$ (secs)
11	2	16(80)	0.106	0.969	0	0.694	0.811	0.609
14	4	14(70)	0.089	0.605	0	3.505	4.15	2.964
17	12	13(65)	0.026	0.166	0	22.710	26.317	19.593
11^d	1	14(70)	0.082	0.515	0	0.638	0.78	0.53
14^d	1	17(85)	0.019	0.268	0	3.010	3.276	2.684
17^d	1	15(75)	0.028	0.321	0	23.575	27.768	19.297

<Table 5> Test Results for Cases 28, 29 and 30

Case	N_{opt} (#(%))	RD_{avg} (%)	$RD_{\max}$ (%)	$RD_{\min}$ (%)	TR_{avg} (secs)	$TR_{\max}$ (secs)	$TR_{\min}$ (secs)	T_{avg} (secs)
28	2(10)	0.990	3.303	0	36.912	152.856	4.009	5.911
29	0(0)	0.967	3.183	0.006	24.389	151.122	4.000	15.031
30	0(0)	1.698	6.724	0.083	13.868	79.416	4.828	31.012
Average	0.7(3.3)	1.218			25.056			17.318

<Table 6> Comparison of Cases i and i^d for $i=$ 28, 29, 30

Case	N_{opt} (#(%))	RD_{avg} (%)	$RD_{\max}$ (%)	$RD_{\min}$ (%)	T_{avg} (secs)	$T_{\max}$ (secs)	$T_{\min}$ (secs)
28	2(10)	0.990	3.303	0	5.911	6.801	4.742
29	0(0)	0.967	3.183	0.006	15.031	17.893	10.483
30	0(0)	1.698	6.724	0.083	31.012	36.301	26.864
28^d	0(0)	1.762	6.022	0.074	5.919	7.27	4.617
29^d	0(0)	1.875	3.919	0.164	15.285	17.769	12.293
30^d	0(0)	1.637	4.466	0.054	31.928	35.537	28.174

<Table 4> assesses the difference that the ratio of the number of old jobs to the number of new jobs makes on run time and solution quality by comparing cases 11, 14, and 17 with cases, 11^d, 14^d, and 17^d, respectively. Columns, T_{avg}, $T_{\max}$, and $T_{\min}$, give average, maximum, and minimum values of LRH run time, respectively. <Table 4> shows that neither the run time nor the solution quality of our heuristic is affected by the ratio of the number of old jobs to the number of new jobs, even though the ratios differ widely.

<Table 5> gives test results obtained for cases 28, 29, and 30. These results show that very few of instances are solved optimally but that average relative deviation barely exceeds 1%. However, LRH solves test instances about 25 times faster than CPLEX, on average.

We use cases of problem P(3) to test the sensitivity of our heuristic relative to the values of the objective function and the RHS coefficients. Run time of the heuristic proposed by Lieshout and Volgenant [12] double if the ranges of coefficients double. Thus, we double the values of the coefficients of the objective function and the RHS to define cases cases, 28^d, 29^d, and 30^d, respectively, then generate 20 independent instances of each case to compare test results with cases 28, 29, and 30, respectively.

<Table 6> compares results associated with cases i and i^d for i = 28, 29, 30. The first six columns and the last three columns correspond to those of <Table 2> and <Table 4>, respectively. <Table 6> shows that solution qualities (i.e., RD_{avg}, $RD_{\max}$, and $RD_{\min}$) and T_{avg} for classes i and i^d, i = 28, 29, 30 are almost the same, even though coefficients of the objective function and the RHS are doubled in the latter three cases. We have similar results for problem P(2).

5. Conclusions and Recommendations for Future Research

This paper addresses the problem of rescheduling on unrelated parallel machines with compressible processing times, where the objective is to minimize the sum of completion times, compression costs, and, perhaps, job reassignment costs. We formulate this problem as an APSC, and we introduce a heuristic for problems of this type.

Tests show that our heuristic solves some 78.9% of our randomly generated test instances for cases 1 through 27 to optimality, providing near-optimal solutions for all test instances, and requires much shorter run time than CPLEX, especially on instances of problem P(3) : $Rm|x_{ij} \le u_{ij}, a_i, \Sigma C_j \le T|\Sigma c_{ij}x_{ij}$. Furthermore, our tests show that our heuristic is not sensitive to the values of objective function coefficients or of the RHS coefficient of the side constraint. Tests show that our heuristic resolves rescheduling problems effectively and that it can be applied to find a good solution quickly for cases in which finding an optimal solution may require prohibitive run time.

Our heuristic can be also applied to any problem with a set of constraints that possesses the unimodular property, augmented with a side constraint. We plan to test such numerical instances. In addition to this, we will try to modify our heuristic to solve APSC with multiple side constraints, perhaps by aggregating them.

References

[1] Aboudi, R. and Jornsten, K., Resource constrained assignment problems. *Discrete Appl Math*, 1990, Vol. 26, pp. 175-191.

[2] Aggarwal, V., A lagrangean-relaxation method for the constrained assignment problem. *Compu Oper Res*, 1985, Vol. 12, No. 1, pp. 97-106.

[3] Akturk, M.S., Atamturk, A., and Gurel, S., Parallel machine match-up scheduling with manufacturing cost considerations. *J Sched*, 2010, Vol. 13, pp. 95-110.

[4] Alagoz, O. and Azizoglu, M., Rescheduling of identical parallel machines under machine eligibility constraints. *Eur J Oper Res*, 2003, Vol. 149, pp. 523-532.

[5] Alidaee, B. and Ahmadian, A., Two parallel machine sequencing problems involving controllable job processing times. *Eur J Oper Res*, 1993, Vol. 70, pp. 335-341.

[6] Azizoglu, M. and Alagoz, O., Parallel-machine rescheduling with machine disruptions. *IIE Trans*, 2005, Vol. 37, pp. 1113-1118.

[7] Cheng, T.C.E., Chen, Z.L., and Li, C.L., Parallel-machine scheduling with controllable processing times. *IIE Trans*, 1996, Vol. 28, pp. 177-180.

[8] Curry, J. and Peters, B., Rescheduling parallel machines with stepwise increasing tardiness and machine assignment stability objectives. *Inter J Prod Res*, 2005, Vol. 43, No. 15, pp. 3231-3246.

[9] Gupta, A. and Sharma, J., Tree search method for optimal core management of pressurized water reactors. *Compu Oper Res*, 1981, Vol. 8, No. 4, pp. 263-266.

[10] Gurel, S. and Cincioglu, D., Rescheduling with controllable processing times for number of disrupted jobs and manufacturing cost objectives. *Int J Prod Res*, 2015, Vol. 53, No. 9, pp. 2751-2770.

[11] Gurel, S. and Akturk, M.S., Considering manufacturing cost and scheduling performance on a CNC turning machine. *Eur J Oper Res*, 2007, Vol. 177, pp. 325-343.

[12] Lieshout, P.M.D. and Volgenant, A., A branch-and-bound algorithm for the singly constrained assignment problem. *Eur J Oper Res*, 2007, Vol. 176, pp. 151-161.

[13] Mazzola, J. and Neebe, A.W., Resource-constrained assignment scheduling. *Oper Res*, 1986, Vol. 34, pp. 560-572.

[14] Murty, K., An algorithm for ranking all the assignments in order of increasing cost. *Oper Res*, 1968, Vol. 16, pp. 682-687.

[15] Ozlen, M. and Azizouglu, M., Generating all efficient solutions of a rescheduling problem on unrelated parallel machines. *Int J Prod Res*, 2009, Vol. 47, No. 19, pp. 5245-5270.

[16] Papadimitriou, C.H. and Steiglitz, K., Combination Optimization : Algorithms and Complexity. *Englewood Cliffs*, NJ : Prentice Hall, 1982.

[17] Pinedo, M.L., Scheduling : Theory, Algorithm, and Systems (3rd ed.) : Prentice Hall, 2008.

[18] Punnen, A. and Aneja, Y.P., A tabu search algorithm for the resource-constrained assignment problem. *J Oper Res Soc*, 1995, Vol. 46, pp. 214-220.

[19] Shabtay, D. and Steniner, G., A survey of scheduling with controllable processing times. *Discrete Appl Math*, 2007, Vol. 155, pp. 1643-1666.

[20] Vickson, R.G., Two single-machine sequencing problems involving controllable job processing times. *AIIE Trans*, 1980, Vol. 12, pp. 258-262.

[21] Yang, B., Single machine rescheduling with new jobs arrivals and processing time compression. *Inter J Adv Manuf Tech*, 2007, Vol. 34, pp. 378-384.

[22] Yin, N. and Wang, X.Y., Single-machine scheduling with controllable processing times and learning effect. *Int J Adv Manuf Technol*, 2011, Vol. 54, pp. 743-748.

ORCID

Suhwan Kim | http://orcid.org/0000-0003-4916-1713