GENERATING ALL EFFICIENT SOLUTIONS OF A RESCHEDULING PROBLEM ON UNRELATED PARALLEL MACHINES

Melih Özlen
Department of Industrial Engineering,
Hacettepe University, Ankara 06800, Turkey
Phone: +90 312 297 6885
Fax: +90 312 297 2078
E-Mail: mozlen@hacettepe.edu.tr

Meral Azizoğlu[1]
Department of Industrial Engineering,
Middle East Technical University, Ankara 06531, Turkey
Phone: +90 312 210 22 81
Fax: +90 312 210 47 86
E-Mail: meral@ie.metu.edu.tr

Abstract: In this paper, we consider a rescheduling problem where a set of jobs has already been assigned to unrelated parallel machines. When a disruption occurs on one of the machines, the affected jobs are rescheduled, considering the efficiency and stability measures. Our efficiency measure is the total flow time and stability measure is the total reassignment cost caused by the differences in the machine allocations in the initial and new schedules. We propose a branch and bound algorithm to generate all efficient solutions with respect to our efficiency and stability measures. We improve the efficiency of the algorithm by incorporating powerful reduction and bounding mechanisms. Our computational tests on large sized problem instances have revealed the satisfactory behavior of our algorithm.

Keywords: Rescheduling; Efficient Solutions; Branch and Bound Algorithm

[1] *Corresponding Author*

1. Introduction

Majority of the scheduling literature assumes an environment that works smoothly without any disruption. However, in practice the manufacturing environments are often prone to disruptions like machine breakdowns, new order arrivals, order cancellations, changes in order specifications, and material shortages. Such disruptions may make the initial scheduling plan hard to implement and may arise a need for rescheduling.

We consider a rescheduling problem where a number of parallel machines are disrupted, hence blocked, for a specified time period. We study the most general parallel machine environment, i.e., unrelated parallel machines, where the processing time of a job is dependent on the machine it is assigned. We assume that the customer promises are given and machine allocations are made according to the initial schedule plan. After the disruption, we aim to minimize the total flow time of the jobs that have not started yet. However, the new minimum total flow time schedule may deviate from the initial schedule, in terms of the machine allocations. The deviations due to the machine allocations should be minimized, in particular when initial preparations like machine setups, tool loadings, labor assignments, are made according to the initial plan. Such rescheduling problems fall within the scope of the disruption management area (Clausen, 2001).

Despite its practical importance, the literature on the rescheduling problems is relatively scarce. We refer the reader to Aytug *et al*. (2005) and Vieira, Herrmann, and Edward (2003) for the extensive review of the literature. Aktürk and Görgülü (1999) and Li and Shaw (1996), Raheja and Subramaniam (2002), Mason, Jin and Wessels (2004) and Abumaizar and Svetska (1997) consider multi-stage environments. Wu, Storer and Chang (1993), Daniels and Kouvelis (1995), Ünal, Uzsoy and Kıran (1997), O'Donovan, Uzsoy and McKay (1999), Hall and Potts (2004) and Qi, Bard and Yu (2006) study rescheduling problems on a single machine. The most note-worthy rescheduling studies in parallel machine environments are due to Church and Uzsoy (1992), Bean *et al.* (1991), Leung and Pinedo (2004), Alagöz and Azizoğlu (2003), Azizoğlu and Alagöz (2005), Curry and Peters (2005) and Özlen and Azizoğlu (2007). Church and Uzsoy (1992) consider single and parallel machines to minimize the maximum lateness and the number of times rescheduling is done. They provide a simulation study to test the efficiencies of some rescheduling methodologies like periodic, event-driven and continuous rescheduling. Bean et al. (1991) consider a rescheduling problem with release dates and parallel machines. Their approach reconstructs the part of the schedule after the disruption so as to match the initial schedule at some future time. Leung and Pinedo (2004) consider parallel machines with deadlines and precedence relations and with three objectives: total completion time, makespan and maximum lateness. Alagöz and Azizoğlu (2003) and Azizoğlu and Alagöz (2005) consider the trade-off between the total flow time and number of reassigned jobs criteria in identical parallel machine environments. Özlen and Azizoğlu (2007) consider the total flow time and total reassignment cost in unrelated parallel machine environments. They provide polynomial-time solution methods to the hierarchical optimization problems of the two measures and propose non-polynomial exact algorithms to generate all efficient solutions and to minimize a specified function of the measures. Curry and Peters (2005) consider the total disruption cost as a stability measure and total tardiness as an efficiency measure. They propose a simulation study to test the performances of some heuristic procedures and rescheduling strategies.

We consider the trade-off between the efficiency of the new schedule, measured by its total flow time and the stability measured by the total reassignment cost caused by the differences between the initial and new machine allocations. Our efficiency measure, total flow time, is the total time that the jobs spent in the shop floor, hence it is a direct indication of the work-in-process inventory levels which is an important concern of many manufacturers.

Our stability measure, total reassignment cost, is an important concern particularly in Flexible Manufacturing Systems and Supply Chains. In Flexible Manufacturing Systems, the setup costs are incurred when the tools are allocated in advance according to the initial job assignments (see Olumolade and Norrie, 1996). Hence retooling of the machines due to the changes in their job assignments may require additional time and cost. We assume the reassignment costs are dependent on the machines that the jobs are assigned in the new schedule. The locations of the initial and new machines may be important in defining machine dependent reassignment costs, in particular when the tools/equipments required by the reassigned jobs should be transported between initial and new machines. Moreover, the jobs may have different setup requirements on different machines, due to the different tools that are initially loaded on their tool magazines. Another practical situation where the reassignment costs may find its application is the supply chains where the jobs are assigned to the facilities at different locations. The reassignment costs may well represent the cost of transporting the job from one location to another.

We generate all non-dominated, i.e., efficient, solutions with respect to the total flow time and total reassignment cost criteria in unrelated parallel machine environments. The generation of all efficient solutions is an important concern for a decision maker who is interested in screening all non-dominated solutions and selecting his/her optimal solutions by considering the trade-offs between the two objectives. For example the decision maker may want to know the amount he/she has to sacrifice from the total flow time value for a certain amount of reduction in the total reassignment cost. To generate the efficient set, we develop a branch and bound algorithm and improve the efficiency of the algorithm by incorporating powerful reduction and bounding mechanisms.

The most closely related study to ours is Özlen and Azizoğlu (2007). Özlen and Azizoğlu (2007) present an algorithm to minimize a general nondecreasing function of the total flow time and total reassignment cost criteria. They compare their algorithm with a classical approach of generating all efficient schedules by evaluating each efficient schedule and selecting the one with minimum objective function value. The classical approach generates each efficient solution by solving an NP-hard singly constrained assignment problem.

Our main contribution is the branch and bound algorithm to generate the efficient set for the total flow time and total reassignment cost criteria. Our algorithm generates all efficient solutions simultaneously, unlike the classical approach used in Özlen and Azizoğlu (2007) that generates the set sequentially. We solve a single problem whereas the classical approach solves *r* singly constrained assignment problems if there are *r* efficient solutions. The computational results have also verified the superiority of our algorithm over the classical approach. Hence our algorithm is the best performing algorithm in the literature.

The problem we study is shown to be NP-hard which suggests that any optimization procedure will run into computational difficulties as the problem size increases. There is, however, the practical question concerning the problem sizes that are solvable in reasonable time. Our computational results suggest that the answer to this question for our branch and bound algorithm is 100 jobs and 12 machines. Hence our branch and bound algorithm is the unique reported approach that can be used to solve such large sized problem instances.

The rest of the paper is organized as follows. In Section 2, we give the basic definitions, introduce our notation and define the problem. We also state previous results that are pertinent to our problem. Section 3 presents our procedure to find the extreme supported efficient solutions. In Section 4, we present our branch and bound algorithm designed for generating all efficient solutions. We present the results of our experiments in Section 5 and conclude in Section 6.

2. Problem Definition

We consider an unrelated parallel machine environment and assume that the initial schedule is known. There is a disruption of D time units on one of the machines, say machine DM, after executing the initial schedule for DT time units. The job that is being processed on DM, and the jobs that start on or after DT on other machines are to be rescheduled at time DT. We assume that there are n such jobs. Once we take the reference starting point from time zero to DT, our rescheduling problem reduces to scheduling n jobs, available at time zero, on m unrelated parallel machines where machine j becomes available at time a_j. Accordingly, $a_{DM} = D$ and a_j is the completion time of the job processed at time DT on non-disrupted machine j. Note that, multiple simultaneous disruptions can also be handled by letting $a_j = D_j$ where D_j is the time at which the disruption on machine j, is recovered. Each job should be assigned to one of the machines. Job i should be processed by p_{ij} time units without interruption if assigned to machine j.

The scheduling cost, that defines our efficiency measure, is the total flow time, F. The flow time of a job is the time it spends in the system and total flow time is the total time spent by all jobs. As we assume all zero ready times, the total flow time and total completion time are equivalent measures. If we let C_i denote the completion time of job i in the new schedule, then the total flow time, F , is $\sum_{i=1}^{n} C_i$.

The schedule deviation cost that defines our stability measure is the total reassignment cost. The reassignment cost for job i on machine j is rc_{ij}. We can interpret rc_{ij} as the additional cost incurred due to the reassignment of job i to machine j. We set rc_{ij} to 0 if job i is assigned to machine j in the initial schedule.

We assume D, DT, p_{ij}, a_j and rc_{ij} are all integers.

The total reassignment cost, RC, is $\sum_i \sum_j rc_{ij} x_{ij}$ where x_{ij} is a binary variable that takes on value 1 if job i is assigned to machine j in the new schedule and 0 otherwise.

A schedule S is said to be efficient with respect to F and RC, if there exists no schedule S' with $F(S') \leq F(S)$ and $RC(S') \leq RC(S)$ with at least one strict inequality.

An efficient solution $s \in S$ is supported if it optimizes any weighted sum of F and RC. In other words, $s \in S$ is a supported efficient solution, if it is one of the optimal solutions to the rescheduling problem with the objective function $w_1\, RC + w_2\, F$ for any non-negative weights w_1, w_2.

A supported efficient solution $s \in S$ is extreme supported efficient if it can be found by varying the values of w_1 and w_2 . A supported efficient solution $s \in S$ is nonextreme supported efficient if it lies on the convex combination of two adjacent extreme supported efficient solutions. An efficient solution $s \in S$ is unsupported if it is not optimal for any weighted sum of F and RC objectives.

The standard classification scheme for scheduling problems uses three-field representation $\alpha | \beta | \gamma$ where α is the machine environment, β specifies the constraints or special characteristics of the problem and γ is the objective function (see Lawler et al., 1989). We consider unrelated parallel machines, hence set $\alpha = R$. When the parallel machines are identical, i.e., $p_{ij} = p_i$ for all i and j, we set $\alpha = P$. We use the following constraints in β field:

β : a_j: initial machine available times

β : $RC \leq k$: total reassignment cost is at most k

We consider F and RC as efficiency and stability measures, hence we have,

γ : F : minimizing the total flow time

γ : RC : minimizing the total reassignment cost

γ : F, RC : generating all efficient solutions with respect to F and RC

Özlen and Azizoğlu (2007) formulate the $R | a_j | F$ problem, as an assignment model with the following decision variable:

$$X_{ikj} = \begin{cases} 1 & \text{if job } i \text{ is scheduled } k^{th} \text{ position from last on machine } j \\ 0 & \text{otherwise} \end{cases}$$

The objective function, F, is expressed as:

$$\sum_{i=1}^{n} \sum_{k=1}^{n} \sum_{j=1}^{m} (kp_{ij} + a_j) X_{ikj} \tag{1}$$

kp_{ij} is the contribution of the processing time of job i to the total flow time if sequenced at k^{th} position from last on machine j and a_j is the start time of the first job on machine j.

The constraint sets are explained below.

Constraint set (2) states that each job is assigned exactly once.

$$\sum_{k=1}^{n} \sum_{j=1}^{m} X_{ikj} = 1 \qquad \forall\, i \tag{2}$$

Constraint set (3) ensures that at most one job is assigned to each position of each machine.

$$\sum_{i=1}^{n} X_{ikj} \leq 1 \qquad \forall\, j, k \tag{3}$$

Constraint set (4) requires that the jobs cannot be preempted or split.

$$X_{ikj} \in \{0,1\} \qquad \forall\, i, j, k \tag{4}$$

The total reassignment cost, *RC*, is expressed as $\sum_{i,j,k} rc_{ij} X_{ikj}$. When the constraint $RC \le k$, i.e., $\sum_{i,j,k} rc_{ij} X_{ikj} \le k$, is added to the above model, it becomes a singly-constrained assignment problem and is represented by $R \mid a_j, RC \le k \mid F$.

The singly-constrained assignment problem is NP-Hard (see Aggarwal, 1985), so is the $R \mid a_j, RC \le k \mid F$ problem. The optimal solution to the $R \mid a_j, RC \le k \mid F$ problem is efficient provided that the ties are broken in favor of *RC*, i.e., the $R \mid a_j, RC \le k \mid F + \varepsilon_{RC} RC$ problem is solved for a sufficiently small value of ε_{RC}. Özlen and Azizoğlu (2007) show that when $\varepsilon_{RC} < \dfrac{1}{\sum_{i=1}^{n} Max_j\{rc_{ij}\}}$, the resulting solution is efficient. This follows the schedule that solves the $R \mid a_j, RC \le k \mid F + \dfrac{1}{\sum_{i=1}^{n} Max_j\{rc_{ij}\} + 1} RC$ problem is efficient. In this study we use $\dfrac{1}{\sum_{i=1}^{n} Max_j\{rc_{ij}\} + 1}$ for ε_{RC}.

The generation of all efficient solutions of a bicriteria assignment problem is NP-hard (see Pryzbylski, Gandibleuz and Ehrgott, 2008), so is the generation of all efficient solutions to our rescheduling problem with total flow time and total reassignment cost criteria, i.e., the $R \mid a_j \mid F, RC$ problem. Özlen and Azizoğlu (2007) propose a classical approach to solve the $R \mid a_j \mid F, RC$ problem. They vary *k* between a lower bound on the total reassignment cost, RC_{LB} and an upper bound on the total reassignment cost, RC_{UB}. RC_{LB} is found by applying the right-shift strategy to the initial schedule. The right-shift strategy shifts all jobs on *DM*, *D* time units to the right in time horizon, while preserving the other job assignments. Note that RC_{LB} is zero as the reassignment cost for a job is zero if it is assigned to the same machine in the initial and new schedules. RC_{UB} is the *RC* value that solves the $R \mid a_j \mid F + \varepsilon_{RC} RC$ problem.

The RC_{LB} and RC_{UB} schedules define the boundary solutions of the efficient set. Procedure 1 below is the stepwise description of the classical algorithm proposed in Özlen and Azizoğlu (2007).

Procedure 1. Generating All Efficient Solutions

Step 0. Find RC_{LB} and RC_{UB}

RC_{LB} = RC value of the right-shift schedule = 0

RC_{UB} = RC value that solves the $R \mid a_j \mid F + \varepsilon_{RC}RC$ problem

Let $k = RC_{UB} - 1$

Step 1. Solve the $R \mid a_j, RC \leq k \mid F + \varepsilon_{RC}RC$ problem.

Let (F^*, RC^*) be the solution.

Step 2. If $RC^* = RC_{LB} = 0$ then STOP.

$k = RC^* - 1$

Go to Step 1

Note that, each iteration of Procedure 1 generates an efficient solution by solving a singly-constrained assignment problem for which any polynomial algorithm cannot exist. Hence an efficient solution is generated in exponential time and there can be at most $RC_{UB} - RC_{LB}$ efficient solutions. We hereafter refer to Procedure 1 as Classical Approach (CA).

3. A Procedure to Generate All Extreme Supported Efficient Solutions

We generate the extreme supported efficient set through the successive solutions of a linear assignment problem. We start with two boundary schedules, S_1 and S_2, identify ranges for the w values of the weighted objective function over which each boundary schedule is better. In doing so, we solve the following equality:

$$wF_1 + (1-w)\, RC_1 = wF_2 + (1-w)\, RC_2 \qquad (5)$$

where (F_i, RC_i) is the (F, RC) values of S_i and S_i s are ordered , $F_i < F_{i+1}$, $RC_i > RC_{i+1}$.

Note that $w = \dfrac{RC_2 - RC_1}{F_1 - F_2 + RC_2 - RC_1}$ solves (5).

At w, S_1 and S_2 have the same objective function values. In ranges [w, 1] and [0, w], S_1 and S_2 are favored respectively. When a new extreme supported efficient solution is added, we reorder the solutions such that $F_1 < F_2 < F_3$ and $RC_1 > RC_2 > RC_3$ and solve the following two equations simultaneously.

$$w_1F_1 + (1-w_1)\, RC_1 = w_1F_2 + (1-w_1)\, RC_2$$

$$w_2F_2 + (1-w_2)\, RC_2 = w_2F_3 + (1-w_2)\, RC_3.$$

In ranges [w_1,1], [w_2, w_1] and [0, w_2], S_1, S_2 and S_3 are the best schedules respectively. Note that the ranges change once a new efficient schedule is identified.

In general, once we have *k* efficient solutions, we solve *k*-1 equations: one for each adjacent pair and find *k* ranges. The exact ranges are identified when all extreme supported solutions are found.

Each iteration of our procedure either finds a new extreme supported efficient solution, or returns a known extreme supported efficient solution, by solving a linear assignment problem with weight w_a. If the former case occurs then there exists an efficient solution between S_a and S_{a+1} and the weights are updated once a new schedule is added. If the latter case occurs then there is no supported efficient solution between S_a and S_{a+1}. We then fix w_a and proceed with w_{a+1} with the hope of generating a new extreme supported schedule. The algorithm terminates whenever all weights are fixed. Below is the stepwise description of the extreme supported efficient set generation algorithm.

Procedure 2 Generating All Extreme Supported Efficient Solutions

Step 0. Form the right shift schedule to find S_1.

Solve the $R \mid a_j \mid F + \varepsilon_{RC} RC$ problem to find S_2.

k = # of extreme supported efficient solutions with fixed ranges

$k = 1$

$$w_1 = \frac{RC_2 - RC_1}{F_1 - F_2 + RC_2 - RC_1}$$

$S_L = S_2$

Step 1. Solve the assignment problem with the following objective

Min $w_k F + (1 - w_k) RC$

Let S_L be the solution

If S_L is one of the extreme solutions, S_1 or S_2 then go to Step 3.

Step 2. If S_L is either S_k or S_{k+1} then fix w_k let k=k+1, go to Step 1

If S_L is a new schedule then reorder the schedules, update w_k and w_{k+1} as follows

$$w_k = \frac{RC_{k+1} - RC_k}{F_k - F_{k+1} + RC_{k+1} - RC_k}$$

$$w_{k+1} = \frac{RC_{k+2} - RC_{k+1}}{F_{k+1} - F_{k+2} + RC_{k+2} - RC_{k+1}}$$

If all w_k values are fixed then go to Step 3 else go to Step 1

Step 3. Stop, all extreme supported efficient solutions are generated.

Procedure 2 is similar to the methods by Aneja and Nair (1979) and Visee *et al.* (1998) proposed for the bicriteria transportation and knapsack problems respectively.

Note that we have n by $n{*}m$ rectangular assignment problems as there are n jobs and $n{*}m$ positions. In solving the assignment problems of Step 0 and Step 2, we use the algorithm by Volgenant (1996) particularly designed for the rectangular assignment problems. The algorithm solves n by $n{*}m$ rectangular assignment problems in order of $n^3{*}m$ steps. The classical assignment algorithms solve the same problem in $n^3{*}m^3$ steps by adding ($n{*}m$-n) dummy nodes.

4. A Branch and Bound Algorithm to Generate All Non-Extreme Supported and Unsupported Efficient Solutions

Recall that the generation of all efficient solutions with respect to total flow time and total reassignment cost criteria, is NP-hard. This result justifies the use of enumeration techniques. We propose a Branch and Bound (BAB) algorithm that makes implicit enumeration of all efficient solutions. Below we give the details of the three phases of our algorithm: initialization, branching and bounding.

Phase 1. Initialization

We generate the approximate non-extreme supported and unsupported efficient solutions in the neighborhood of the solutions found in Section 3 by Procedure 1 and use this set as an initial feasible set. In doing so, we start from the first boundary schedule having the minimum total flow time, thereby the maximum total reassignment cost, of all efficient solutions. We then move the jobs to their initial machines. The resulting schedule is added to the list, if it is not dominated by any schedule in the list. We continue with the schedule from the list having the minimum total flow time. We repeat the procedure, until the other boundary schedule is reached, i.e., the one having maximum total flow time and zero total reassignment cost, is reached. Then we start from this boundary schedule and create new schedules by reassigning the jobs from their initial machines to other machines, while keeping the other assignments. The new schedules, if nondominated, are added to the list. We continue with the new schedule from the list having the minimum total reassignment cost. We terminate whenever the other boundary schedule in the list is reached.

Our BAB starts with the list of approximate efficient solutions. We add a schedule to the list if it is not dominated by the schedules in the list. We remove a schedule from the list if it is dominated by the added schedule.

We next discuss the branching phase.

Phase 2. Branching

We generate the partial solutions, i.e., nodes, of the BAB tree as follows: At each level, we decide on the job that should be assigned to the first available position of the earliest available machine. We also represent a solution in which no further assignment is made to the earliest available machine, this case corresponds to removing that machine from further considerations. In selecting the available job we recognize the optimality of the Shortest Processing Time (SPT) order within each machine for the total flow time objective (see Smith, 1956). Hence we never branch to a node representing the assignment of job i to machine j if $p_{ij} < p_{lj}$ and job l has assigned to machine j in the partial solution.

Figure 1 represents a partial BAB tree for n=7 jobs and m=3 machines problem instance whose data are given in Table 1.

[Insert Table 1 about here]

Note that the SPT orders of the jobs are as follows:

Machine 1	4-6-7-5-1-3-2
Machine 2	1-4-3-2-6-7-5
Machine 3	3-6-2-1-7-4-5

We assume Machine 1 is not available for 98 time units and the initial job assignments are 6-7-5 on machine 1, 1-4-3 on machine 2 and 2 on machine 3.

[Insert Figure 1 about here]

Note that initially a_1=98, a_2=a_3=0, hence machines 2 and 3 are the earliest available machines. Assume we arbitrarily select machine 2 for branching. The first node, called 0, represents the case where no further assignments are made on machine 2. The (r+1)st node at level 1 corresponds to the assignment of the r^{th} job in the SPT order of machine 2. Hence the fourth node represents the assignment of job 3. If node 3 is selected for branching then $a_2 = p_{32} = 33$ and machine 3 becomes the earliest available machine, emanates six nodes, each node representing the assignment of a particular job to its first available position. The fifth node at level 2, is the fourth unscheduled job in the SPT order of machine 3, i.e., job 7. If this node is selected for further branching $a_3 = p_{73} = 72$, machine 2 becomes the earliest available machine. At level 3, there are four candidate partial solutions, as job 3 has assigned to the first position of machine 2 and there are 3 unscheduled jobs that have higher processing times than that of job 3 on machine 2. These jobs are 2, 6 and 5.

Note that there will be a maximum of $n+m$-1 levels, as n jobs will be assigned and there can be at most m-1 no further machine assignment (node 0) decisions.

We let M_i denote the set of machines that cannot process job i. Job i cannot be processed by machine j, if such an assignment violates the SPT order or cannot yield an efficient solution. An assignment of job i to machine j violates the SPT order if $p_{ij} < p_{L_j,j}$ where L_j is the last job assigned to machine j in the partial schedule.

We calculate a lower bound for each job that can be assigned to the earliest available machine. We next discuss the lower bounds.

Phase 3. Bounding

We let σ denote the current partial schedule and $\bar{\sigma}$ is the set of unscheduled jobs. We let $P_F(\sigma)$ and $P_{RC}(\sigma)$ be the total flow time, F and total reassignment cost, RC of partial schedule σ. $LB_F(\sigma)$ and $LB_{RC}(\sigma)$ are the lower bounds on the F and RC values of the partial schedule σ respectively.

Theorem 1 below states a lower bound on the total reassignment cost of the unscheduled jobs of all efficient schedules.

Theorem 1: $\sum_{i\in\bar{\sigma}} \text{Min}_{j\notin M_i}\{rc_{ij}\}$ is a lower bound on the total reassignment cost of the unscheduled jobs of all efficient schedules.

Proof: In all efficient schedules job i cannot be assigned to any machine in set M_i, without violating the SPT order. Hence the reassignment cost of job i is no smaller than $Min_{j\notin M_i}\{rc_{ij}\}$. This follows, the total reassignment cost of any efficient schedule, over all unscheduled jobs, i.e., the jobs that are not in σ, cannot be greater than $\sum_{i\in\bar{\sigma}} \text{Min}_{j\notin M_i}\{rc_{ij}\}$. #

$UB_F(RC)$ is an upper bound on the F values of the efficient solutions having a total reassignment cost of at least RC. Similarly $UB_{RC}(F)$ is an upper bound on the RC values of the efficient solutions having a total flow time value of at least F units. When job i is assigned to machine j and appended to σ, a lower bound on the total flow time value, is

$P_F(\sigma)+(a_j+p_{ij})$ + $\sum_{l\in\bar{\sigma}}\text{Min}_{r\notin M_l}\{a_r+p_{lr}\}$. If this bound is no smaller than $UB_F(LB_{RC}(\sigma))$ (an upper bound on the F values of the schedules having a total reassignment cost of at least $LB_{RC}(\sigma)$) then σ is dominated by the schedule of our list having a total flow time value of $UB_F(LB_{RC}(\sigma))$.

Similarly, if $P_{RC}(\sigma) + rc_{i\,j}+ \sum_{l\in\bar{\sigma}}\text{Min}_{r\notin M_l}\{rc_{lr}\}\geq UB_{RC}(LB_F(\sigma))$ then σ is dominated by the schedule in our list having a total reassignment cost of $UB_{RC}(LB_F(\sigma))$.

Hence an assignment of job i to machine j is avoided if either

$P_F(\sigma) + (a_j + p_{ij}) + \sum_{l\in\bar{\sigma}}\text{Min}_{r\notin M_l}\{a_r+p_{lr}\} \geq UB_F(LB_{RC}(\sigma))$ or

$P_{RC}(\sigma) + rc_{ij} + \sum_{l\in\bar{\sigma}}\text{Min}_{r\notin M_l}\{rc_{lr}\}\geq UB_{RC}(LB_F(\sigma))$

We hereafter refer to the above conditions as efficiency rules.

We let R_j denote the set of jobs that can be processed on machine j. Among the machines with $R_j \neq 0$, we select the earliest available one. If the first unscheduled job of the SPT order on the selected machine, cannot be assigned to any other machine, we fix that job on that machine and update the M_i sets, earliest available times and proceed. For each job in R_j, we calculate a lower bound on RC and two lower bounds on F values.

Lower bound on RC, $LB_{RC}(\sigma)$

A lower bound on RC is $LB_{RC}(\sigma)= P_{RC}(\sigma) + LB_{RC}(\bar{\sigma})$, where

$P_{RC}(\sigma)$ = total reassignment cost of the jobs in σ

$LB_{RC}(\bar{\sigma})$= a lower bound on the optimal total reassignment cost of the unscheduled jobs, i.e., the jobs that are not in σ.

Using the result of Theorem 1, we let $LB_{RC}(\bar{\sigma}) = \sum_{i\in\bar{\sigma}}\text{Min}_{j\notin M_i}\{rc_{ij}\}$ and hence choose a reassignment cost among the jobs that can be assigned without violating the SPT order and having a potential of generating an efficient solution.

Referring to the BAB tree of Figure 1, M_j values are calculated according to the SPT order. For a partial schedule where jobs 3 and 7 are assigned to machines 2 and 3 respectively, the lower bound can be calculated as follows:

$M_1=\{2, 3\}$, $M_2=\{3\}$, $M_4=\{2\}$, $M_5=\{\ \}$, $M_6=\{3\}$

$P_{RC}(\sigma) = rc_{73}$

$LB_{RC}(\sigma) = rc_{11}$ + Min$\{rc_{41}, rc_{43}\}$+ Min$\{rc_{61}, rc_{62}\}$

as jobs 1, 4 and 6 cannot be assigned to their initial machines, job 1 can only be assigned to machine 1 according to the SPT order.

Lower bound on F

We propose two procedures to find a lower bound on the optimal flow time of the unscheduled jobs

i. ***Lower Bound 1, $LB_{F1}(\sigma)$***

We assume all machines are identical and let $p_i = Min_{j \notin M_i}\{p_{ij}\}$. Note that p_i is the minimum processing time for job *i*, among the machines that it can be assigned without violating the SPT order and efficiency rules.

The new problem is the $P \mid a_j \mid \sum C_i$ problem of the scheduling literature whose optimal solution is due to following rule by Kaspi and Montreuil (1988): Order the jobs by SPT and assign them to the first available machine, in rotation. An optimal *F* value of the new identical machine problem, is a lower bound on the optimal *F* value of the original unrelated machine problem. The theorem below states this result formally.

***Theorem 2* :** The *F* value that solves the $P \mid a_j \mid \sum C_i$ problem with $p_i = Min_{j \notin M_i}\{p_{ij}\}$ is a lower bound on the total flow time of the unscheduled jobs in all efficient schedules.

Proof: In all efficient schedules job *i* cannot be assigned to any machine in set M_i, without violating the SPT order and efficiency rules. Hence the processing incurred due to job *i* cannot be smaller than $Min_{j \notin M_i}\{p_{ij}\}$. This follows, the total flow time of any efficient solution over all unscheduled jobs, i.e., the jobs that are not in σ, cannot be greater than the optimal *F* value of the $P \mid a_j \mid \sum C_i$ problem with $p_i = Min_{j \notin M_i}\{p_{ij}\}$. #

In our example, a lower bound on the *F* value of a partial schedule σ, in which jobs 3 and 7 are assigned to machines 2 and 3 respectively is found as follows:

$p_1 = p_{11} = 67$

$p_2 =$ Min $\{p_{21}, p_{22}\} = 44$

$p_4 =$ Min $\{p_{41}\ p_{43}\} = 14$

$p_5 =$ Min $\{p_{51}, p_{52}, p_{53}\} = 54$

$p_6 =$ Min $\{p_{61}, p_{62}\} = 22$

The SPT order with p_i values is 4-6-2-5-1.

The lower bound schedule has the following assignments:

Machine 1	1			a_1=98
Machine 2	4	6	2	a_2=33
Machine 3	5			a_3=72

$LB_{F1}(\bar{\sigma})$ = (98+67) + (33+14) + (33+14+22) + (33+14+22+44) + (72+54) = 520

$P_F(\sigma)$ = 33 + 72 = 105

$LB_{F1}(\sigma)$ = $LB_{F1}(\bar{\sigma}) + P_F(\sigma)$ = 625

We fathom the node, if there is a schedule s' in the list such that $F(s') \le LB_F(\sigma)$ and $RC(s') \le LB_{RC}(\sigma)$. If a node cannot be fathomed by $LB_{F1}(\sigma)$, we calculate a more powerful lower bound, $LB_{F2}(\sigma)$.

ii. ***Lower Bound 2, $LB_{F2}(\sigma)$***

Consider the following assignment model

$$\text{Min} \sum_{i=1}^{n}\sum_{k=1}^{n}\sum_{j=1}^{m}(kp_{ij}+a_j)X_{ikj} + \frac{1}{\sum_{i=1}^{n} Max_j\{rc_{ij}\}+1}\sum_{i,j,k} rc_{ij}X_{ikj}$$

s.t

$$\sum_{k=1}^{n}\sum_{j=1}^{m} X_{ikj} = 1 \qquad \forall\, i$$

$$\sum_{i=1}^{n} X_{ikj} \le 1 \qquad \forall\, j, k$$

$$X_{ikj} \in \{0,1\} \qquad \forall\, i, j, k$$

where $x_{ikj} = \begin{cases} 1 & \text{if job } i \text{ is assigned to } k^{th} \text{ position from last on machine } j \\ 0 & \text{otherwise} \end{cases}$

For a partial schedule σ, where σ_j is the set of jobs assigned to machine j, and n_j is the cardinality of set σ_j we modify a_j values as, $a_j = a_j + \sum_{i \in \sigma_j} p_{ij}$, and solve the assignment model with the following objective function

$$\text{Min} \sum_{i \notin \sigma_j}\sum_{k=1}^{n_j'}\sum_{j=1}^{m}(a_j + kp_{ij})X_{ikj} + \frac{1}{\sum_{i=1}^{n} Max_j\{rc_{ij}\}+1}\sum_{i \notin \sigma_j}\sum_{k=1}^{n_j'}\sum_{j=1}^{m} rc_{ij}X_{ikj}$$

where n_j' is an upper bound on the number of unscheduled jobs that can be assigned to machine j. If the last job assigned to machine j is the l^{th} job of its SPT order then at most n-l more jobs can be assigned to machine j. Moreover the jobs between $l+1$ and n, in the SPT order, may be assigned to the other machines. So, we modify the upper bound, n_j', as the number of unscheduled jobs with no smaller processing time than p_{lj} on machine j and that do not violate the efficiency rules.

Moreover we let $c_{ikj}=M$ if job i is the r^{th} unscheduled job of the Longest Processing Time (LPT) order on machine j such that $r < k$, to avoid the assignment of any job to a position that is higher than its index, thereby avoiding a non-SPT ordering. We then solve the $|\bar{\sigma}|$ x $\sum_{\substack{j=1 \\ j\neq N}}^{m} n'_j$ assignment problem using the rectangular assignment algorithm of Volgenant (1996).

The cost coefficients of the assignment model for a partial schedule, where jobs 3 and 7 are assigned to the first positions of machines 2 and 3 respectively, are calculated as follows: Note that $n'_2=3$ as there are 3 unscheduled jobs having higher processing times than p_{32}, these jobs are 2, 6 and 5. As there are two unscheduled jobs having higher processing times than p_{73}, $n'_3=2$. As there are two scheduled jobs, there can be at most n-2=5 jobs on machine 1. Hence we solve 5 x 10(5+3+2) assignment problem. We set $c_{1k2} = c_{1k3} = M$ for all k as Job 1 cannot be assigned to machines 2 and 3 without violating the SPT order. Jobs 2 and 6 cannot be assigned to machine 3, i.e., $c_{2k2} = c_{6k2} = M$ for k=1, 2. Job 2 cannot be assigned to machine 1, except to its first position, i.e., $c_{2k1} = M$ for $k > 1$, as it is the last job of SPT on machine 1. If we assign job 2 to a later position, the SPT order is violated, as there is no unscheduled job with higher processing time. Moreover, we set c_{651}=M as job 6 cannot be scheduled at the fifth position of machine 1. Job 4 cannot be assigned to machine 2, i.e., $c_{4k2} = M$ for all k. Job 5 is the third longest unscheduled job on machine 1 hence $c_{541} = c_{551} = M$. Similarly job 1 can only be assigned to the first or second positions of machine 1 as it is the second longest unscheduled job. All cost figures are tabulated in Table 2.

[Insert Table 2 about here]

We add term $\varepsilon_{RC} rc_{ij}$ to (i, k, j) if machine j is not the initial machine of job i. For example job 5 is on machine 1 in the initial schedule, hence ε_{RC} appears in all entries for job 5 except the ones on machine 1. The optimal assignment solution gives the following schedule.

Machine 1	4	-	1
Machine 2	2	-	6

Machine 3	5

Note that a_1=98, a_2=33, a_3=72

$LB_{F2}(\bar{\sigma})$ = (98+14) + (98+14+67) + (33+44) + (33+33+64) + (72+86) = 667

$P_F(\sigma)$ = 105

$LB_{F2}(\sigma)$ = $LB_{F2}(\bar{\sigma}) + P_F(\sigma)$ = 772

$F(\bar{s})$ = 772

$RC(\bar{s})$ = $rc_{41} + rc_{11} + rc_{22} + rc_{62} + rc_{52}$

The actual total flow time of the schedule is $F(\bar{s})$ and the actual total reassignment cost is $RC(\bar{s})$. We add schedule $\bar{s}$ to the list of approximate efficient solutions if there does not exist a schedule s' such that $F(s') \le F(\bar{s})$ and $RC(s') \le RC(\bar{s})$. If there exists a schedule $\hat{s}$ in the list such that $F(\hat{s}) \ge F(\bar{s})$ and $RC(\hat{s}) \ge RC(\bar{s})$, then $\hat{s}$ is dominated by $\bar{s}$, and therefore is deleted from the list.

Note that $Max\{LB_{F1}(\sigma), LB_{F2}(\sigma)\}$is a lower bound on the optimal F values of the nodes emanating from σ. Hence when we proceed to the next level we first check whether there exists a schedule s' such that $F(s') \le Max\{LB_{F1}(\sigma), LB_{F2}(\sigma)\}$ and $RC(s') \le RC(\sigma)$. If such a schedule s'exists then we fathom the node, else we calculate the associated lower bound.

Finally we present a pseudo code of our branch and bound algorithm and illustrate it via an example problem.

Pseudo Code of the Branch and Bound Algorithm

```
GenExtSup();  / generate extreme supported solutions
IntApE();        / initialize approximate efficient set
/ use branch and bound to generate all efficient solutions
t=1;                  / initialize with root node
Ins(0,0);          / insert root node into stack
while (t<0) {   / while there is any node waiting in the stack
        Rem(t);            t=t-1;   / remove from top of stack
        minm=EarMac();         / find earliest available machine
        for i {               / all jobs according to SPT on minm
                CreChi(i);        / create child with job i
                LB_RC(σ)=LbRc(i);       / find lower bound rc
                LB_F1(σ)=LbF1(i);       / find lower bound F1
        if (($LB_{RC}(\sigma)<UB_{RC}(LB_{F1}(\sigma))$
            && ($LB_{F1}(\sigma)<UB_F(LB_{RC}(\sigma))$)) {      / check dominance
            $LB_{F2}(\sigma)$=LbF2(i);      / find lower bound F2
            if (($LB_{RC}(\sigma)<UB_{RC}(LB_{F2}(\sigma))$
                && ($LB_{F2}(\sigma)<UB_F(LB_{RC}(\sigma))$))/ check dominance
                t=t+1;  / add i into stack
        } / end if
    }  / end for i
    CreChi(0);      / create child for not assigning any jobs to m
    $LB_{RC}(\sigma)$=LbRc(0);      / find lower bound rc
    $LB_{F1}(\sigma)$=LbF1(0);      / find lower bound F1
    if (($LB_{RC}(\sigma)<UB_{RC}(LB_{F1}(\sigma))$
        && ($LB_{F1}(\sigma)<UB_F(LB_{RC}(\sigma))$)) {      / check dominance
        $LB_{F2}(\sigma)$=LbF2(0);      / find lower bound F2
        if (($LB_{RC}(\sigma)<UB_{RC}(LB_{F2}(\sigma))$
            && ($LB_{F2}(\sigma)<UB_F(LB_{RC}(\sigma))$))      / check dominance
            t=t+1;  / add i into stack
    } / end if
}  / end while
```

Child creation is done by CreChi(i) procedure which is explained as follows:

CreChi(i) first duplicates the information from the parent node, updates it according to the recent assignment, and then checks the first assignable job, shortest processing time job that is assignable according to efficiency rules, of each machine for fixing it to that position. If that job cannot be assigned to any other machine due to SPT and efficiency rule, than it's fixed to that position. Fixing is done iteratively until no further fixing can be done.

We illustrate the pseudo code via the following 6 jobs, 2 machine problem.

p_{ij}	J1	J2	J3	J4	J5	J6
M1	22	6	44	33	21	97
M2	64	94	72	62	55	79

rc_{ij}	J1	J2	J3	J4	J5	J6
M1	51	60	13	16	10	58
M2	30	37	24	58	22	20

Initial minimum total flow time schedule makes the following assignments.

M1 J2-J5-J1-J4-J3

M2 J6

Total flow time= 369

We assume that a disruption of length 126 time units occurs on M1 at time 0.

DM=1, DT=0, D=126, a_1=126, a_2=0

RC_{LB}=0, F_{UB}=999, right shift schedule

RC_{UB}=80, F_{LB}=852, solution of $R \mid a_j \mid F + \varepsilon_{RC} RC$ problem

The four extreme supported solutions generated by procedure 1 are listed below.

RC	F
0	999
22	893
46	861
80	852

The initial heuristic finds no other approximate efficient solutions and $UB_F(RC)$ is initialized as:

$$UB_F(RC) = \begin{cases} 999 & RC = 0 \\ 998 & 1 \le RC \le 21 \\ 893 & RC = 22 \\ 892 & 23 \le RC \le 45 \\ 861 & RC = 46 \\ 860 & 47 \le RC \le 79 \\ 852 & RC = 852 \end{cases}$$

We initialize B&B by the root node, from root node M2 is identified as the earliest available machine.

At root node with no job assignments, a_1=126, a_2=0, $minm$=2.

Node 1, J5 ➔ M2

$LB_{RC}(\sigma) = 22$, $LB_{F1}(\sigma) = 680$, $UB_F(LB_{RC}(\sigma)) = 893$, $UB_{RC}(LB_{F1}(\sigma)) = 80$

$LB_{F1}(\sigma) < UB_F(LB_{RC}(\sigma))$, $LB_{RC}(\sigma) < UB_{RC}(LB_{F1}(\sigma))$.

$LB_{F2}(\sigma) = 861$, $UB_{RC}(LB_{F2}(\sigma)) = 80$, $(RC(\bar{s}), F(\bar{s})) = (46, 861)$, no need to add.

$LB_{F1}(\sigma) < UB_F(LB_{RC}(\sigma))$, $LB_{RC}(\sigma) < UB_{RC}(LB_{F2}(\sigma))$, add to BAB stack.

Node 2, J4 ➔ M2

Fix J2 ➔ M1, otherwise if J2 ➔ M1 then $LB_{RC} \geq 95$.

Fix J5 ➔ M1, otherwise if J5 ➔ M2 then SPT will be violated.

Fix J1 ➔ M1, otherwise if J1 ➔ M2 then $LB_{RC} \geq 88$.

Fix J3 ➔ M1, otherwise if J3 ➔ M2 then $LB_{RC} \geq 82$.

Fix J6 ➔ M2, otherwise if J6 ➔ M1 then $LB_{RC} \geq 116$.

All jobs are fixed, $(RC(\bar{s}), F(\bar{s})) = (58, 882)$ no need to add since dominated by (46, 861).

Node 3, J1 ➔ M2

$LB_{RC}(\sigma) = 30$, $LB_{F1}(\sigma) = 722$, $UB_F(LB_{RC}(\sigma)) = 892$, $UB_{RC}(LB_{F1}(\sigma)) = 80$.

$LB_{F1}(\sigma) < UB_F(LB_{RC}(\sigma))$, $LB_{RC}(\sigma) < UB_{RC}(LB_{F1}(\sigma))$.

$LB_{F2}(\sigma) = 891$, $(RC(\bar{s}), F(\bar{s})) = (24, 891)$, no need to add since dominated by (46, 861).

$LB_{F1}(\sigma) < UB_F(LB_{RC}(\sigma))$, $LB_{RC}(\sigma) < UB_{RC}(LB_{F2}(\sigma))$, add to BAB stack.

Node 4, J3 ➔ M2

Fix J6 ➔ M2, otherwise if J6 ➔ M1 then $LB_{RC} \geq 82$.

$LB_{RC}(\sigma) = 24$, $LB_{F1}(\sigma) = 867$, $UB_F(LB_{RC}(\sigma)) = 892$, $UB_{RC}(LB_{F1}(\sigma)) = 80$.

$LB_{F1}(\sigma) < UB_F(LB_{RC}(\sigma))$, $LB_{RC}(\sigma) < UB_{RC}(LB_{F1}(\sigma))$.

$LB_{F2}(\sigma) = 891$, $(RC(\bar{s}), F(\bar{s})) = (24, 891)$, add to list since not dominated.

Update $UB_F(RC)$.

$$UB_F(RC) = \begin{cases} 999 & RC = 0 \\ 998 & 1 \leq RC \leq 21 \\ 893 & RC = 22 \\ 892 & RC = 23 \\ 891 & RC = 24 \\ 890 & 25 \leq RC \leq 45 \\ 861 & RC = 46 \\ 860 & 47 \leq RC \leq 79 \\ 852 & RC = 852 \end{cases}$$

$UB_F(LB_{RC}(\sigma)) = 891$, $LB_{F1}(\sigma) \geq UB_F(LB_{RC}(\sigma))$, fathom.

Node 5, J6 ➔ M2

$LB_{RC}(\sigma) = 0$, $LB_{F1}(\sigma) = 729$, $UB_F(LB_{RC}(\sigma)) = 999$, $UB_{RC}(LB_{F1}(\sigma)) = 80$

$LB_{F1}(\sigma) < UB_F(LB_{RC}(\sigma))$, $LB_{RC}(\sigma) < UB_{RC}(LB_{F1}(\sigma))$.

$LB_{F2}(\sigma) = 999$, $(RC(\bar{s}), F(\bar{s})) = (0, 999)$, no need to add.

$LB_{F1}(\sigma) \geq UB_F(LB_{RC}(\sigma))$, fathom.

Node 6, J2 $\Rightarrow$ M2

J6 cannot be assigned to any of the machines due to efficiency rules, fathom.

if J6 $\Rightarrow$ M1 then $LB_{RC} \geq 95$.

If J6 $\Rightarrow$ M2 then SPT will be dominated.

Node 7, Close M2

All jobs are assigned to M1 in SPT order, J2-J5-J1-J4-J3-J6.

$(RC(\bar{s}), F(\bar{s})) = (58, 1269)$ no need to add since dominated by (46, 861).

From Node 3, where J1$\Rightarrow$M2,

a_1=126, a_2=64, P_F=64, P_{RC}=30, $LB_{F2}(\sigma) \geq 886$, *minm*=2.

Node 8, J3 $\Rightarrow$ M2

$LB_{RC}(\sigma) = 54$, $(LB_{RC}(\sigma), LB_{F2}(\sigma))$ is dominated by (46, 861), fathom.

Node 9, J6 $\Rightarrow$ M2

Fix J2 $\Rightarrow$ M1, otherwise if J2 $\Rightarrow$ M1 then

$LB_{RC} \geq 67$, $(LB_{RC}(\sigma), LB_{F2}(\sigma))$ will be dominated by (46, 861).

Fix J5 $\Rightarrow$ M1, otherwise if J5 $\Rightarrow$ M2 then SPT will be violated.

Fix J4 $\Rightarrow$ M1, otherwise if J4 $\Rightarrow$ M2 then SPT will be violated.

Fix J3 $\Rightarrow$ M1, otherwise if J3 $\Rightarrow$ M2 then SPT will be violated.

All jobs are fixed, $(RC(\bar{s}), F(\bar{s})) = (30, 908)$ no need to add since dominated by (22, 893).

Node 10, J2 $\Rightarrow$ M2

$LB_{RC}(\sigma) = 67$, $(LB_{RC}(\sigma), LB_{F2}(\sigma))$ is dominated by (46, 861), fathom.

Node 11, Close M2

J6 cannot be assigned to M1 due to efficiency rules, fathom.

if J6 $\Rightarrow$ M1 then $LB_{RC} \geq 88$.

From Node 1, where J5$\Rightarrow$M2,

a_1=126, a_2=55, P_F=55, P_{RC}=22, $LB_{F2}(\sigma)\geq 861$, $minm$=2.

Node 12, J4 $\Rightarrow$ M2

$LB_{RC}(\sigma)$ = 80, ($LB_{RC}(\sigma)$, $LB_{F2}(\sigma)$) is dominated by (46, 861), fathom.

Node 13, J1 $\Rightarrow$ M2

$LB_{RC}(\sigma)$ = 52, ($LB_{RC}(\sigma)$, $LB_{F2}(\sigma)$) is dominated by (46, 861), fathom.

Node 14, J3 $\Rightarrow$ M2

$LB_{RC}(\sigma)$ = 46, ($LB_{RC}(\sigma)$, $LB_{F2}(\sigma)$) is dominated by (46, 861), fathom.

Node 15, J6 $\Rightarrow$ M2

Fix J2 $\Rightarrow$ M1, otherwise if J2 $\Rightarrow$ M1 then

$LB_{RC} \geq 57$, ($LB_{RC}(\sigma)$, $LB_{F2}(\sigma)$) will be dominated by (46, 861).

Fix J1 $\Rightarrow$ M1, otherwise if J1 $\Rightarrow$ M2 then SPT will be violated.

Fix J4 $\Rightarrow$ M1, otherwise if J4 $\Rightarrow$ M2 then SPT will be violated.

Fix J3 $\Rightarrow$ M1, otherwise if J3 $\Rightarrow$ M2 then SPT will be violated.

All jobs are fixed, ($RC(\bar{s})$, $F(\bar{s})$) = (22, 893) no need to add since dominated by (22, 893).

Node 16, J2 $\Rightarrow$ M2

$LB_{RC}(\sigma)$ = 59, ($LB_{RC}(\sigma)$, $LB_{F2}(\sigma)$) is dominated by (46, 861), fathom.

Node 17, Close M2

J6 cannot be assigned to M1 due to efficiency rules, fathom.

if J6 $\Rightarrow$ M1 then $LB_{RC} \geq 78$, ($LB_{RC}(\sigma)$, $LB_{F2}(\sigma)$) is dominated by (46, 861), fathom.

No nodes are left in stack, so BAB terminates with the following set of efficient solutions.

RC	*F*
0	999
22	893
24	891
46	861
80	852

The efficient solution (24, 891) is unsupported and all other efficient solutions are extreme supported.

5. Computational Experience

We conduct a computational experiment to assess the efficiency of BAB compared to Classical Approach (CA). We generate random problem instances having 40, 60, 80, 100 jobs and 4, 8, 12 machines. We select two levels for processing times and two levels for reassignment costs to see the effects of the variability of these parameters on the performance of our BAB algorithm. The p_{ij} values are drawn from discrete uniform distributions between [1,100] and [50,100] to represent high and low variability cases respectively. The rc_{ij} values are drawn from discrete uniform distributions between [1,60] and [30,60] to represent high and low variability cases respectively. We set rc_{ij} to zero if machine j is the initial machine of job i.

The disruption duration, D, is set to two levels: Long (L) and Short (S). For level L, D is set to the half of the completion time of the last job on the disrupted machine in the initial schedule. Level S has half of the duration of level L.

We conduct all experiment on a PC with Intel Pentium 4 2.8 Ghz processor and 1 GB of RAM running under Linux, specifically Fedora 5, operating system. We implement our BAB in C, compiled with GCC 4 and utilize Borland C++BuilderX as the development environment. We solve our integer and linear programming models using CPLEX 8.1.1. We set a termination limit of 2 hours for CA and BAB. We find that the problem instances with n=100 jobs could not be solved in 2 hours, when the disruption duration is long, hence did not report the associated results.

We consider 72(3x3x2x2x2)+12=84 problem combinations and generate 10 instances for each combination.. Hence as a total of 840 problem instances are generated and solved.

Tables 3 and 4 report the performance of CA and BAB for p_{ij}~U[1,100] and p_{ij}~U[50,100] respectively. The tables give the average and maximum computation times, number of efficient solutions, and the number of times BAB runs quicker than CA. The average and maximum number of nodes generated by BAB are also included.

[Insert Tables 3 and 4 about here]

From Tables 3 and 4, we can observe the increase in the average number of efficient solutions with increasing n. On average there are 13, 25 and 59 efficient solutions, when there are 40, 60 and 80 jobs, respectively. Moreover the difficulty of attaining an efficient solution increases considerably when n increases. In Table 4, we observe two cases having the same average number of efficient solutions; n=60, m=8, rc_{ij}~U[1,60] and n=80, m=8, rc_{ij}~U[30,60]. CA generates the efficient set of case 1 five times quicker than that of case 2. BAB generates the efficient set of case 1 three times quicker than that of case 2. This is due to the fact, the number of integer variables increases with an increase in n for CA. Similarly, the number of branches increases as a function of n in our BAB algorithm

As m increases, the F and RC ranges decrease and that leads to a decrease in the number of efficient solutions. Note from Table 3 that, when n=80, p_{ij}~U[1,100] and D=L, the average number of efficient solutions decrease with an increase in the number of machines. When there are 4, 8, and 12 machines, the respective average numbers of efficient solutions are 34, 14, and 10. As m increases, the efficient solutions are generated in higher computational times, due to the increase in the number of integer decision variables. Note that the same number of efficient solutions is generated in less effort when m is small. In Table 4, we can observe this effect significantly, for the problems with 80 jobs, D=S and reassignment cost in range between 30 and 60, 10 efficient solutions exist on average for the cases with 8 and 12 machines. CA generates the efficient set in 48 CPU seconds on average when m=8, and in 95 CPU seconds on average when m=12.

In general, the performance of CA is dependent on the number of efficient solutions and number of integer variables (that increases with n and m). Disruption duration, processing time variability, and reassignment cost variability are also effective, as these parameters affect the number of efficient solutions.

We also observe that the disruption duration, processing time and reassignment cost distributions significantly affect the performance of BAB. When the disruption duration is longer, the sequencing alternatives are more and this causes weak differentiation of the partial solutions which in turn increases the difficulty of attaining optimal solutions. This significant behavior can be easily observed from Table 3 for D=S and D=L. Note that the average CPU time of BAB to generate efficient set is 1.9 CPU seconds where the disruption duration is shorter. The CPU time increases to 43.8 seconds where the disruption duration is longer. Whenever the processing times are higher, the disruption durations are longer and thus the problems are harder to solve.

When the variability of the processing times or reassignment costs decreases, the differentiation power of the lower bounds decreases as the partial solutions become closer. As the quality of the lower bounds directly affects the performance of BAB, we observe smaller computational times when the ranges are wider. This relation is quite obvious from Table 3 for D=L, the performance of the algorithm depends on the reassignment cost variation. Note that when there are 80 jobs and 4 machines, the efficient set is generated in 22 seconds for low variation case, and in 7 seconds when the variation is high. Moreover, we observe more significant affect of the processing time variability, as the processing time defines the range of efficient solutions more often. One can point out some exceptions which can be attributed to the randomness effect like dominant contributions of few instances to average performance.

Tables 3 and 4 reveal that, BAB outperforms CA in the vast majority of the problem instances. We find that, in 793 out of 840 instances, BAB runs quicker than CA.

6. Conclusions

In this paper, we address a rescheduling problem on unrelated parallel machines. We consider the total flow time as an efficiency measure and total reassignment cost as a stability measure. We generate all efficient solutions with respect to these two measures. Our aim is to help a decision maker who cannot explicitly express his/her preference function, but want to make a choice by screening the non-dominated solutions.

To find an initial set of approximate efficient solutions, we form the extreme supported efficient set by the weighted approach and extend the set in a neighborhood. To generate exact efficient solutions, we propose a branch and bound algorithm. We improve the efficiency of the algorithm by incorporating powerful reduction and bounding mechanisms.

The results of our computational tests have revealed that our branch and bound algorithm can solve problem instances with up to 100 jobs and 12 machines in reasonable solution times. We compare our algorithm with the classical approach used in the previous studies and find that our algorithm performs superior for the majority of the problem instances.

We hope that our study stimulates future work on rescheduling area. The extension of our results to multi-stage environments like flow-shops and job-shops might be an interesting extension. Other two noteworthy extensions are the total weighted flow time measure and a tri-criteria problem that might include a customer related measure. We discuss each extension below:

When the jobs do have different priorities or values, the total weighted flow time would be a more suitable objective than the total flow time. The incorporation of the weights destroys the assignment nature of the model, so the special procedures inherent for assignment problem will not be valid any more. However the total weighted flow time has also a nice property that the optimal solution of the sequencing problem (which is the weighted shortest processing time rule) is known. Thus we can adapt our branching scheme for the total flow time problem to solve its weighted version.

In addition to our producer related efficiency measure of the total flow time, we can consider a customer related efficiency measure, like maximum lateness or total tardiness. In such a case, the rescheduling problem will be treated as a tri-criteria problem together with our stability measure. A customer related measure may also act as a stability measure, once the due-dates are accepted as the promises given according to the completion times in the initial schedule.

REFERENCES

Abumaizar, R.J., and Svestka J.A., Rescheduling job shops under random disruptions, *International Journal of Production Research*, 1997, **35**, 2065--2082.

Aggarwal V., A lagrangean-relaxation method for the constrained assignment problem, *Computers and Operations Research*, 1985, **12**, 97--106.

Aktürk M.S., and Görgülü E., Match-up scheduling under a machine breakdown, *European Journal of Operational Research*, 1999, **112**, 81--97.

Alagöz O., and Azizoğlu M., Rescheduling of identical parallel machines under machine eligibility constraints, *European Journal of Operational Research*, 2003, **149**, 523--532.

Aneja, Y.P., and Nair K.P.K., Bicriteria transportation problem, *Management Science*, 1979, **25**, 73--78.

Aytug H., Lawley M.A., McKay K., Mohan S., and Uzsoy R., Executing production schedules in the face of uncertainties: a review and some future directions, *European Journal of Operational Research*, 2005, **161**, 86--110.

Azizoğlu M., and Alagöz O., Parallel machine rescheduling with machine disruptions, *IIE Transactions*, 2005, **37**, 1113--1118.

Bean J.C., Birge J.R., Mittenthal J., and Noon C.E., Matchup scheduling with multiple resources, release dates and disruptions, *Operations Research*, 1991, **39**, 470--483.

Church L.K., and Uzsoy R., Analysis of periodic and event-driven rescheduling policies in dynamic shops, *International Journal of Computer Integrated Manufacturing*, 1992, **5**, 153--163.

Clausen J., Hansen J., Larsen J., and Larsen A., Disruption management, *ORMS Today*, 2001, **28**, 40--43.

Curry J.,and Peters B., Rescheduling parallel machines with stepwise increasing tardiness and machine assignment stability objectives, *International Journal of Production Research*, 2005, **43**, 3231--3246.

Daniels R.L.,and Kouvelis P., Robust scheduling to hedge against processing time uncertainty in single stage production, *Management Science*, 1995, **41**, 363--376.

Hall N.G., and Potts C.N., Rescheduling for new orders, *Operations Research*, 2004, **52**, 440--453.

Kaspi M., and Montreuil B., On the scheduling of identical parallel processes with arbitrary initial processor available time, Research Report 88-12, School of Industrial Engineering, Purdue University; 1988.

Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G. and Shmoys D.B., Sequencing and scheduling: algorithms and complexity, Reports BS-R8909, Centre for Mathematics and Computers Science, Amsterdam, 1989.

Leung J.Y.-T., and Pinedo M., A note on scheduling parallel machines subject to breakdown and repair, *Naval Research Logistics*, 2004, **51**, 60--71.

Li E., and Shaw W., Flow-time performance of modified-scheduling heuristics in a dynamic rescheduling environment, *Computers & Industrial Engineering*, 1996, **31**, 213--216.

Mason S.J., Jin S., and Wessels C.M., Rescheduling strategies for minimizing total weighted tardiness in complex job shops, *International Journal of Production Research*, 2004, **42**, 613--628.

O'Donovan R., Uzsoy R., and McKay K.N., Predictable scheduling of a single machine with breakdowns and sensitive jobs, *International Journal of Production Research*, 1999, **18**, 4217--4233.

Olumolade M.O., and Norrie D.H., Reactive scheduling system for cellular manufacturing with failure-prone machines, *International Journal of Computer Integrated Manufacturing*. 1996, **9**, 131--144.

Özlen M., and Azizoğlu M., Rescheduling unrelated parallel machines with total flow time and total disruption cost criteria, 2007, Technical Report, Department of IE, METU.

Przybylski A., Gandibleux X., Ehrgott, M., Two phase algorithms for the bi-objective assignment problem, *European Journal of Operational Research*, 2008, **185**, 509--533.

Qi X.T., Bard J.R., and Yu G., Disruption management for machine scheduling: the case of SPT schedules, *International Journal of Production Economics*, 2006, **103**, 166--184.

Raheja A.S., and Subramaniam V., Reactive recovery of job shop schedules – A review, *International Journal of Advanced Manufacturing Technology*, 2002, **19**, 756--763.

Smith, W. E., Various optimizers for single stage production. *Naval Research Logistics Quarterly,* 1956, **70**, 93--113.

Ünal A.T., Uzsoy, R., and Kıran A.S., Rescheduling on a single machine with part-type dependent setup times and deadlines, *Annals of Operations Research*, 1997, **70**, 93--113.

Vieira G.E., Herrmann J.W., and Edward L., Rescheduling manufacturing systems: a framework of strategies, Policies and Methods, *Journal of Scheduling,* 2003, **6**, 39--62.

Visee M., Teghem J., Pirlot M., and Ulungu E.L., Two-phases Method and Branch and Bound Procedures to Solve the Bi–objective Knapsack Problem, *Journal of Global Optimization*, 1998, **12**, 139--155.

Volgenant, A., Linear and semi-assignment problems: a core oriented approach, *Computers and Operations Research*, 1996, **23**, 917--932.

Wu S.D., Storer R.H., and Chang P.C., One-machine rescheduling heuristics with efficiency and stability as criteria, *Computers and Operations Research*, 1993, **20**, 1-14.