MINIMIZING MAKESPAN IN A CLASS OF REENTRANT SHOPS

M. Y. WANG

University of Toronto, Toronto, Canada

S. P. SETHI

University of Texas, Dallas

S. L. VAN DE VELDE

Erasmus University, Rotterdam, The Netherlands

(Received September 1993; revisions received August 1994; accepted January 1995)

We study the problem of scheduling a chain-reentrant shop, in which each job goes for its processing first to a machine called the primary machine, then to a number of other machines in a fixed sequence, and finally back to the primary machine for its last operation. The problem is to schedule the jobs so as to minimize the makespan. This problem is unary *NP*-hard for a general number of machines. We focus in particular on the two-machine case that is also at least binary *NP*-hard. We prove some properties that identify a specific class of optimal schedules, and then use these properties in designing an approximation algorithm and a branch-and-bound type optimization algorithm. The approximation algorithm, of which we present three versions, has a worst-case performance guarantee of 3/2 along with an excellent empirical performance. The optimization algorithm solves large instances quickly. Finally, we identify a few well solvable special cases and present a pseudo-polynomial algorithm for the case in which the first and the last operations of any job (on the primary machine) are identical.

Ever since Johnson's (1954) seminal work for solving the problem of scheduling the two-machine flowshop to minimize makespan, the flowshop scheduling problem has become an important paradigm in the literature. Although larger, more complicated flowshop scheduling problems have turned out to be intractable, Johnson's elegant scheduling rule continues to be a useful component in many heuristics for solving them.

Recently, a new type of manufacturing shop, the *reentrant shop*, has come into prominence. The basic characteristic of a reentrant shop is that jobs visit a certain machine or a set of machines more than once. Such a shop reflects modern electronic processing and certain manufacturing environments. A typical example is signal processing (Gupta 1993), in which signal pulses have to go to a computer for preprocessing, and then through the sensing and command system for transmission and retrieval, and finally back to the computer for postprocessing. More complicated examples are assembly of printed circuit boards (Noble 1989) and wafer fabrication (Elliot 1989).

A promising way to address the reentrant shop is to view it as a flowshop with an additional complication. While this complication renders even a simple two-machine reentrant shop scheduling problem intractable, there appears to be the possibility of once again invoking Johnson's rule by exploiting the inherent flowshop structure, and solving the problem approximately and efficiently.

The purpose of this paper, therefore, is to concentrate on problems of scheduling jobs in reentrant shops that, other than being reentrant, have all the flowshop features and for which efficient, if not optimal, algorithms can be developed. In particular, we first extend the model of the signal processing to cover a special class of reentrant shops, which we shall label as *chain-reentrant shops*. In a generic chain-reentrant shop, the technological constraints demand that each job is first processed on the primary machine M_1, then on $(m - 1)$ secondary machines in the order $M_2, M_3, \ldots, M_m$, and finally again on M_1 for a finishing operation. We then focus on the two-stage version of the problem, for which we are able to provide a partial characterization of a set of optimal solutions.

As in Johnson (1954), we select the objective of minimizing makespan. This allows us to apply Johnson's algorithm in the development of an efficient algorithm to solve the two-machine chain-reentrant shop problem approximately. Our approach, partitioning jobs into groups and applying Johnson's rule to each group, is an extension of the classical work of Johnson, and this study may serve as a stepping stone to further analysis of larger, more complicated reentrant shop scheduling problems. We also note that in high-tech manufacturing as in the examples given above, production is usually capacity-driven rather than order-driven; in this sense, minimizing makespan is a natural objective.

Before describing the plan of the paper at the end of this section, let us briefly review the related literature. A closely related problem is analyzed by Lev and Adiri (1984), whose objective is to minimize the makespan of a V-shop in which jobs go through m machines following the route $M_1, M_2, \ldots, M_{m-1}, M_m, M_{m-1}, \ldots, M_2, M_1$. They prove the problem to be binary *NP*-hard (Garey and Johnson 1979) and propose polynomial-time algorithms for several special cases. Recently, a combinatorial analysis of the problem of scheduling reentrant shops is made by

Subject classifications: Production/scheduling: deterministic. reentrant shop scheduling.
Area of review: MANUFACTURING, PRODUCTION AND SCHEDULING.

0030-364X/97/4505-0702 $05.00

Kubiak et al. (1996), who consider a class of reentrant shops in which jobs follow the route $M_1, M_2, M_1, M_3, \ldots, M_1, M_m, M_1$. Their objective is to minimize the mean flow time. They show that the *shortest-processing-time* (SPT) rule is optimal provided that certain restrictive conditions hold.

There is another line of research involving reentrant flows which, while not directly related to the scheduling problems under consideration, is worth mentioning. It concerns optimal flow rate control policies and their long-run stabilities in dynamic, and sometimes stochastic, manufacturing systems. Bai and Gershwin (1990), Yan et al. (1992), and Sethi and Zhou (1994) have studied the problem of production planning in stochastic manufacturing systems with reentrant flows by formulating them as optimal flow rate control problems. Kumar and Seidman (1990), on the other hand, demonstrate the possibility of a destabilizing positive feedback effect in such systems. Lu and Kumar (1991) and Bramcon (1993) provide further examples of what are termed *reentrant lines* that are unstable under certain dispatching policies; see Kumar (1993) for a survey of this work.

The paper is organized as follows. We first draw a parallel between chain-reentrant shops and flowshops. The similarity of the two is not merely apparent, but also substantial. Indeed, in the context of minimizing makespan, we prove in Section 2 that the well-known permutation dominance property for flowshops holds also for the chain-reentrant shops. We also show that the problem of minimizing makespan for the chain-reentrant shop is unary *NP*-hard for more than two machines and at least binary *NP*-hard for exactly two machines.

In the remainder of the paper we focus on the two-machine case. Our main results are developed in Section 3, where we derive a strong dominance property for two-machine chain-reentrant shops. This property reduces our problem effectively to a partitioning problem. More specifically, we show in Theorem 3 that there is an optimal schedule σ that can be considered to comprise three parts: $\sigma = \sigma_1\sigma_2\sigma_3$. In σ_1, the jobs are sequenced according to Johnson's rule applied to their first and second operations; in σ_2, the jobs are arbitrarily sequenced; and in σ_3, the jobs are sequenced according to Johnson's rule applied to their second and third operations. Based on this characterization of a class of optimal solutions, we present a branch-and-bound optimization algorithm in Section 4, and we develop approximation algorithms with worst-case performance guarantee of 3/2 in Section 5. In Section 6, we report on our computational experiments that evaluate the performance of the approximation algorithms against the minimal makespan. The results show that the approximation algorithms find optimal solutions quickly for most of the instances that we have generated. In Section 7, we point out a few well solvable special cases, and also present a pseudo-polynomial dynamic programming algorithm for the case in which both operations of a job on the primary machine are identical. Section 8 concludes the paper.

Table I
Notations

M_k, $k = 1, 2, \ldots, m$	The kth machine. M_1 is also called the primary machine.
J_j, $j = 1, 2, \ldots, n$	Job j.
$\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$	The set of jobs to be processed.
O_{jk}	The kth operation of J_j, also referred to as the k-operation of J_j, $k = 1, 2, \ldots, m + 1$. It is nonpreemptive.
p_{jk}	The nonnegative processing time of O_{jk}; we assume without loss of generality that all processing times are integral.
$S_{jk}(\sigma)$, $C_{jk}(\sigma)$	The start and finish time of O_{jk} in a given schedule σ. We often write them simply as S_{jk} and C_{jk} when there is no confusion; note that $C_{jk} = S_{jk} + p_{jk}$.
$C_k(\mathcal{A})$	The minimum completion time of the last k-operation in a schedule for the ordered job set $\mathcal{A}$.

1. PROBLEM DEFINITION AND NOTATION

Refer to Table I for the notations used in this paper. In a chain-reentrant shop, each machine handles no more than one job at a time, and each machine is continuously available from time zero onward. Each job J_j starts on M_1, then goes to $M_2, M_3, \ldots, M_m$ in that order before it returns to M_1 to undergo its final operation. For any job J_j, $O_{j,k+1}$ cannot start before O_{jk} finishes. An unlimited buffer is available for each machine's output. A feasible schedule satisfies these conditions and specifies a start time S_{jk} and a completion time C_{jk} for each operation O_{jk}. The problem is to find a schedule that minimizes the maximum completion time known as the makespan and defined as

$$C^*_{\max} = \min_{\sigma} C_{\max}(\sigma) = \min_{\sigma} \max_{1 \leq j \leq n} C_{j,m+1}(\sigma).$$

For ease of exposition, we shall use $x \prec y$, $x \preccurlyeq y$, $x \succ y$, and $x \succcurlyeq y$ to mean, respectively, *x precedes y*, *x immediately precedes y*, *x follows y*, *and x immediately follows y*.

Before proceeding to the next section, let us note that we shall follow the nomenclature for scheduling problems proposed by Graham et al. (1979). Accordingly, the problem of minimizing makespan in an m-machine chain-reentrant shop is identified by the three-tuple $Fm|$chain-reentrant$|C_{\max}$.

2. THE COMPLEXITY AND A DOMINANCE PROPERTY

In this section we first show that the problem $Fm|$chain-reentrant$|C_{\max}$ is *NP*-hard. We then prove two dominance properties that facilitate the characterization of a set of optimal schedules in the next section.

Theorem 1. *The problem $Fm|$chain-reentrant$|C_{max}$ is at least binary NP-hard for $m = 2$. Moreover, it is unary NP-hard for $m \geq 3$.*

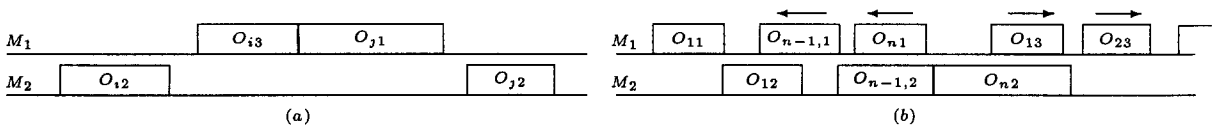

Figure 1. Gantt Chart illustrating the proof of Lemma 1.

Proof. For $m = 2$, see Lev and Adiri (1984). For $m \geq 3$, it is sufficient to prove for $m = 3$. This can be done by a reduction from the decision variant of the three-machine flowshop problem, which is unary *NP*-complete (Garey et al. 1976), as follows.

Given any instance of $F3\|C_{\max}$ with job set $\mathcal{J}$ and $\{a_j, b_j, c_j\}$ as the processing times, we construct an instance of $F3|\text{chain-reentrant}|C_{\max}$ by assigning $p_{j1} = a_j$, $p_{j2} = b_j$, $p_{j3} = c_j$, and $p_{j4} = 0$ for every $J_j \in \mathcal{J}$. It follows immediately that the instance of $F3|\text{chain-reentrant}|C_{\max}$ is solved if and only if the corresponding instance of $F3\|C_{\max}$ is solved. □

Remark 1. For $m = 2$, the chain-reentrant problem is equivalent to the *V*-shop problem introduced by Lev and Adiri (1984). They claim that $V2\|C_{\max}$ is *binary NP*-hard but provide no proof. The claim is yet to be proved.

In chain-reentrant shops, difficulties arise when jobs return to M_1. Not only do the first entries of jobs have to be scheduled, but also all their second entries to M_1. A feasible schedule allows both entries to compete for M_1. Also difficult is the situation in which one job follows another job at one machine and then passes it at some other machine. Such schedules increase the search space for optimal solutions and, at the same time, make it cumbersome to keep track of jobs on the shop floor. Even though they may offer a shorter makespan, they are usually not considered *desirable schedules*. Hence, it is important to know the conditions under which such schedules can be avoided without incurring a longer makespan.

In what follows, we shall say that an entry is *compact* if all the operations of that entry are scheduled contiguously. For example, the first entry is compact if all the operations O_{j1}, $j = 1, \ldots, n$, are scheduled together and there is no idle time between O_{i1} and O_{j1}, whenever $O_{i1} \preccurlyeq O_{j1}$. If all the entries to M_k are compact, we say that the schedule is *compact on* M_k. Moreover, a schedule is *compact* if it is compact on every machine. Also, we say a schedule is *no-passing* if no job passes another in the schedule.

We use the following compactness property to establish a *dominance property* similar to the one that exists for flowshops.

Lemma 1. *To minimize makespan, it is sufficient to consider schedules that are compact on* M_1.

Proof. Consider any feasible schedule σ. We show that σ can be converted into a schedule that is compact on M_1 without increasing the makespan. Refer to the Gantt Chart of Figure 1(a). Suppose there is a pair $O_{i,m+1} \preccurlyeq O_{j1}$. Then interchanging their positions is feasible and does not increase the makespan. When all the 1-operations are scheduled before any $(m + 1)$-operation, then assume without loss of generality that the 1-operations are scheduled in the order $O_{11}, O_{21}, \ldots, O_{n1}$. For $j = 1, \ldots, n - 1$, if there is an idle time between $O_{j1} \preccurlyeq O_{j+1,1}$, we can eliminate it by moving $O_{j+1,1}$ to the left (i.e., scheduling it earlier in time, Figure 1(b)). Clearly such a move does not destroy the feasibility and has no impact on the makespan. Now reindex the jobs in order of the sequence for the $(m + 1)$-operations, that is, assume that the $(m + 1)$-operations are scheduled in the order $O_{1,m+1}, O_{2,m+1}, \ldots, O_{n,m+1}$. For $j = n - 1, \ldots, 1$, if there is an idle time between $O_{j,m+1} \preccurlyeq O_{j+1,m+1}$, we can eliminate it by moving $O_{j,m+1}$ to the right (i.e., scheduling it later in time). Again, such a move is feasible and does not affect the makespan. □

The following dominance property can now be established.

Theorem 2. *To minimize makespan, it is sufficient to consider schedules that are compact on* M_1 *and in which jobs do not pass each other from* M_1 *to* M_2 *and from* M_m *to* M_1.

Proof. In view of Lemma 1, consider any optimal schedule σ that is compact on M_1 but in which jobs pass each other from M_1 to M_2 and from M_{m-1} to M_1. Let J_i and J_j be the first pair in σ such that $O_{i1} \preccurlyeq O_{j1}$ and $O_{i2} \succ O_{j2}$, i.e., J_i passes job J_j on M_2; see Figure 2(a). Clearly interchanging O_{i1} and O_{j1} is feasible, preserves the compactness on M_1, and leaves the makespan unchanged, since no job completion time is affected in doing so.

Now let J_i and J_j be the last pair such that $O_{i,m+1} \preccurlyeq O_{j,m+1}$ and $O_{im} \succ O_{jm}$; refer to Figure 2(b). Interchanging $O_{i,m+1}$ and $O_{j,m+1}$ is feasible and preserves compactness on M_1 as well as the makespan. □

The dominance property stipulated by Theorem 2 is weaker than that for flowshop problems: the property may not hold for regular measures other than makespan. For example, in the case of minimizing mean flow time, Lemma 1 does not hold in general, and so the first and the

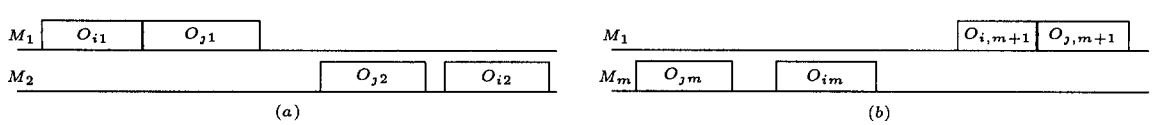

Figure 2. The Gantt Chart illustrating the proof of Theorem 2.

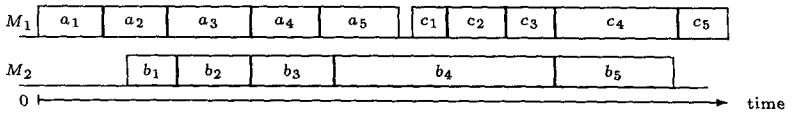

Figure 3. An example of a compact no-passing schedule.

last operations may not be separated as in a compact schedule. Without this separation, Theorem 2 cannot be established.

Since the problem is unary *NP*-hard for $m \geq 3$, we shall from now on concentrate on the two-machine case, for which we are able to derive a stronger dominance property that reduces the $F2|\text{chain-reentrant}|C_{\max}$ problem to essentially a partitioning problem. First, we have the following corollary of Theorem 2.

Corollary 1. *For the $F2|\text{chain-reentrant}|C_{max}$ problem, it suffices to consider only compact no-passing schedules.*

Proof. Suppose σ is an optimal schedule. According to Theorem 2, we may assume σ to be a no-passing schedule that is compact on M_1. It remains to show that operations on M_2 can also be compactified. Let the last 1-operation be O_{i^*1} and the first 3-operation be O_{j^*3}. Then there must be a 2-operation, say, O_{k2}, that is either started in the time interval $[C_{i^*1}, S_{j^*3}]$ or is processed throughout the interval. It is evident that moving all the 2-operations scheduled before O_{k2} to the right in order to eliminate any idle time, and moving all the 2-operations scheduled after O_{k2} to the left in order to eliminate any idle time retains feasibility and does not increase the makespan. □

Figure 3 gives an example of a compact no-passing schedule with five jobs.

3. THE $F2|\text{CHAIN-REENTRANT}|C_{\max}$ PROBLEM

In the rest of the paper, we denote $a_j = p_{j1}$, $b_j = p_{j2}$, $c_j = p_{j3}$, and let $\alpha = \sum_{j=1}^{n} a_j$, $\beta = \sum_{j=1}^{n} b_j$, and $\gamma = \sum_{j=1}^{n} c_j$.

We begin this section with a few observations about a compact no-passing schedule, based on Corollary 1.

Refer to Figure 3. A compact no-passing schedule for a two-machine shop is composed of three blocks of operations: a 1-block consisting of only 1-operations, a 2-block consisting of only 2-operations, and a 3-block consisting of only 3-operations. The operations are scheduled contiguously in each block in the same job order. Thus, a compact no-passing schedule is definitely specified by a permutation of $\mathcal{J}$ except when $C_{\max} = \alpha + \gamma$, in which case there may be some slack in the schedule such that the 2-block can be moved to the left or the right without affecting the makespan. In order to avoid this indefiniteness, we stipulate that in a compact schedule, no block can start processing earlier than required. In other words, we observe the following characteristics of a compact no-passing schedule that are easily verified.

In a compact no-passing schedule,

1. there is a job $J_j \in \mathcal{J}$ such that $S_{j1} + a_j = S_{j2}$; and
2. if $C_{\max} > \alpha + \gamma$, then there is a $J_j \in \mathcal{J}$ such that $S_{j2} + b_j = S_{j3}$.

With this stipulation, our problem becomes a sequencing problem, and we shall therefore use the terms sequence and schedule interchangeably.

Next we establish a critical dominance property that reduces an instance of $F2|\text{chain-reentrant}|C_{\max}$ essentially to a partitioning problem.

For any compact no-passing schedule σ, there exists a job OJ_{k*}, such that

$$k^* = \arg\min_j \{S_{\sigma(j),2} + b_{\sigma(j),2} > \alpha\}.$$

In other words, O_{k^*2} is the first 2-operation that finishes later than α; see Figure 4. Thus, $\mathcal{J}$ is partitioned into three subsets with respect to σ: $\mathcal{J}_1 = \{J_j | J_j < J_{k*}\}$, and $\mathcal{J}_2 = \{J_j | J_j > J_{k*}\}$. Hereafter, job J_{k*} will be called *the partition job*. In Figure 3, e.g., J_4 is the partition job with $\mathcal{J}_1 = \{J_1, J_2, J_3\}$ and $\mathcal{J}_2 = \{J_5\}$. A schedule can therefore be represented by an ordered $\mathcal{J}_1$ followed by J_{k*} and then by an ordered $\mathcal{J}_2$. We define a partition $\{\mathcal{J}_1, J_{k*}, \mathcal{J}_2\}$ to be *admissible* if, and only if, it satisfies

1. $\mathcal{J}_1 \cup \mathcal{J}_2 \cup \{J_{k*}\} = \mathcal{J}$,
2. $C_2(\mathcal{J}_1) \leq \alpha$, and
3. $C_2(\mathcal{J}_1) + b_{k*} > \alpha$.

We observe that the makespan of σ can be thought of as a sum of two parts: the first part is from time zero to the completion time of O_{k*2}, and the second part is from the completion time of O_{k*2} to $C_{\max}$. Ignoring the operations on M_1 between O_{k*1} and O_{k*2}, we can view σ as a two-machine flowshop schedule followed by another two-machine flowshop schedule. Recall that Johnson's rule determines an optimal job sequence for the two-machine flowshop problem. Let the triple $(\mathcal{J}, p, q)$ denote an instance of $F2||C_{\max}$, with processing time vectors p for the 1-operations and q for the 2-operations, and let JOHNSON$(\mathcal{J}, p, q)$ denote the minimum makespan. Johnson's rule can then be stated as follows:

Johnson's rule

1. For any instance $(\mathcal{J}, p, q)$, let $U = \{J_j | p_j \leq q_j\}$ and $W = \mathcal{J} \backslash U$.
2. Sort U in nondecreasing order of p_j and W in nonincreasing order of q_j.

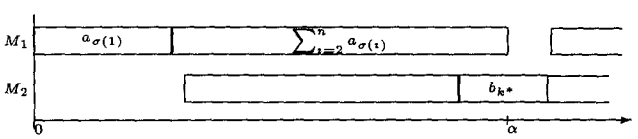

Figure 4. The position of the partition job J_{k*}.

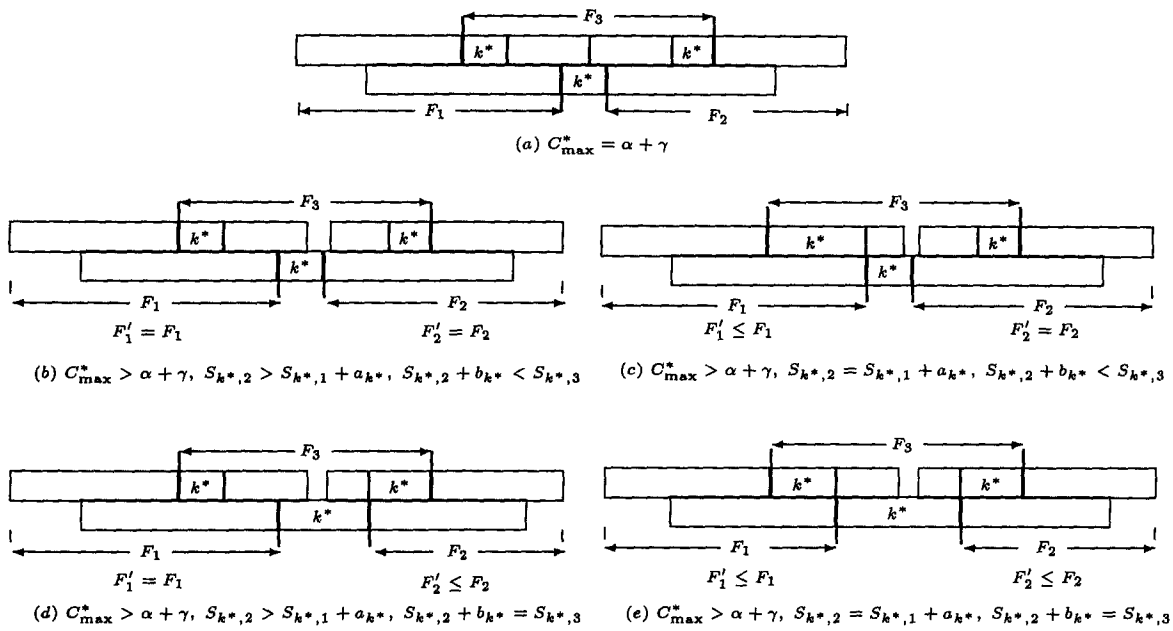

Figure 5. The Gantt Chart illustrating the proof of Theorem 3.

3. Output the ordered U followed by W as the optimal sequence.

If the jobs in a schedule are sequenced according to Johnson's rule, then we call it a *Johnsonian schedule*. We now state the following important property for any instance of the $F2|\text{chain-reentrant}|C_{\max}$ problem.

Theorem 3. *There exists an optimal schedule consisting of an admissible partition $\{\mathcal{J}_1, J_{k*}, \mathcal{J}_2\}$ of $\mathcal{J}$ such that the jobs in $\mathcal{J}_1$ are sequenced according to Johnson's rule for $(\mathcal{J}_1, a, b)$ and the jobs in $\mathcal{J}_2$ are sequenced according to Johnson's rule for $(\mathcal{J}_2, b, c)$. Moreover, the schedule is compact and no-passing.*

Before we prove the theorem, let us describe an algorithm that constructs such a schedule with the property specified in the theorem for any given admissible partition $\{\mathcal{J}_1, J_{k*}, \mathcal{J}_2\}$.

Algorithm 1

STEP 1. Apply Johnson's rule to $(\mathcal{J}_1, a, b)$ and $(\mathcal{J}_2, b, c)$, and let the resulting Johnsonian sequences be σ_1 and σ_2.

STEP 2. Let sequence σ be σ_1 followed by J_{k*} and then by σ_2. Produce a compact schedule with σ.

We are now ready to prove the theorem.

Proof of Theorem 3. It suffices to show that the application of Algorithm 1 to an admissible partition given by an optimal solution will not increase the makespan. Without loss of generality, suppose σ is an optimal compact no-passing schedule. Then we express σ in terms of three partial schedules along with an admissible partition $\{\mathcal{J}_1, J_{k*}, \mathcal{J}_2\}$ as follows:

The first partial schedule, σ_1, consists only of the 1-operations and the 2-operations of the jobs in $\mathcal{J}_1$. The second partial schedule, σ_2, consists only of the 2-operations and the 3-operations of the jobs in $\mathcal{J}_2$. The third partial schedule, σ_3, consists of the 1-operations of the jobs in $\mathcal{J}_2$, the 3-operations of the jobs in $\mathcal{J}_1$, and the operations of J_{k*}. Let F_i be the makespan of σ_i, $i = 1, 2, 3$; see Figure 5.

Now apply Algorithm 1 to the partition $\{\mathcal{J}_1, J_{k*}, \mathcal{J}_2\}$ obtained above. Let the resulting Johnsonian schedules be σ_1', σ_2', and let $\sigma_3' = \sigma_3$. Let F_i' be the makespan of σ_i', $i = 1, 2, 3$. We distinguish five cases (See Figure 5 for illustration):

(a) $C^*_{\max} = \alpha + \gamma$.

(b) $C^*_{\max} > \alpha + \gamma$; $S_{k*,2} > S_{k*,1} + a_{k*}$, and $S_{k*,2} + b_{k*} < S_{k*,3}$. In this case, $F_1' = F_1$, $F_2' = F_2$. Note that σ would not be optimal if $F_1' \le F_1$ or $F_2' \le F_2$ or both. Thus $C_{\max}(\sigma') = C_{\max}(\sigma) = F_1 + b_{k*} + F_2$.

(c) $C^*_{\max} > \alpha + \gamma$; $S_{k*,2} = S_{k*,1} + a_{k*}$, and $S_{k*,2} + b_{k*} < S_{k*,3}$. In this case, $F_1' \le F_1$, $F_2' = F_2$, and $C_{\max}(\sigma') = C_{\max}(\sigma) = \sum_{j\in\mathcal{J}_1} a_j + a_{k*} + b_{k*} + F_2$.

(d) $C^*_{\max} > \alpha + \gamma$; $S_{k*,2} > S_{k*,1} + a_{k*}$, and $S_{k*,2} + b_{k*} = S_{k*,3}$. In this case, $F_1' = F_1$, $F_2' \le F_2$, and $C_{\max}(\sigma') = C_{\max}(\sigma) = F_1 + b_{k*} + c_{k*} + \sum_{j\in\mathcal{J}_2} c_j$.

(e) $C^*_{\max} > \alpha + \gamma$; $S_{k*,2} = S_{k*,1} + a_{k*}$, and $S_{k*,2} + b_{k*} = S_{k*,3}$. In this case, $F_1' \le F_1$, $F_2' \le F_2$, and $C_{\max}(\sigma') = C_{\max}(\sigma) = \sum_{j\in\mathcal{J}_1} a_j + a_{k*} + b_{k*} + c_{k*} + \sum_{j\in\mathcal{J}_2} c_j$.

In each case we have $F_1' \le F_1$ and $F_2' \le F_2$ because of Johnson's rule. Also in each case F_3 is not affected. Therefore, Algorithm 1 preserves the makespan. Moreover, Algorithm 1 produces a compact no-passing schedule.

Remark 2. Note that the partition job is not necessarily unique. By the same token, we can also define the partition job as the last job in σ that starts on M_2 before the

3-block, i.e., $J_{\ell *}$ for ℓ^* such that $S_{\sigma(\ell^*)} = \max_j \{S_{\sigma(j),2} | S_{\sigma(j),2} < S_{\sigma(1),3}\}$. With this partition job, we have an alternative partition of $\mathcal{J}$, $\{\mathcal{J}_3, J_{\ell *}, \mathcal{J}_4\}$. In fact, we can define any job between J_{k*} and $J_{\ell *}$ to be the partition job, and Theorem 3 will still hold. The following corollary is a direct result of this observation.

Corollary 2. *There is an optimal schedule σ that can be specified by a partition of $\mathcal{J}$,*

$$\{\mathcal{J}_1, J_{k*}, \mathcal{J}_5, J_{\ell *}, \mathcal{J}_4\},$$

for some sets $\mathcal{J}_1$, $\mathcal{J}_5$, $\mathcal{J}_4$ and jobs J_{k*} and $J_{\ell *}$, such that the jobs in $\mathcal{J}_1$ are sequenced according to Johnson's rule for 1- and 2-operations, the jobs in $\mathcal{J}_4$ are sequenced according to Johnson's rule for 2- and 3-operations, and the jobs in $\mathcal{J}_5$ are sequenced arbitrarily.

In designing algorithms, this result may further reduce the solution space, especially in cases where $C^*_{\max} \gg \alpha + \gamma$.

We commented earlier on some similarity between flowshops and chain-reentrant shops. The relationship is much deeper between permutation schedules for the three-machine flowshop and compact no-passing schedules for the two-machine chain-reentrant shop. We offer a brief discussion here. For further detail, the reader is referred to Wang (1994).

Comparing the two problems, we can see that if we consider only compact schedules, the $F2|\text{chain-reentrant}|C_{\max}$ problem is equivalent to the $F3||C_{\max}$ problem with the additional constraint that no 3-operation can start before all the 1-operations are completed. For a given instance, if $C^*_{\max} > \alpha + \gamma$, then any feasible solution to the $F3||C_{\max}$ problem can be easily converted to a schedule that has the same makespan and satisfies the additional constraint. Let $C^{F}_{\max}{}^*$ denote the makespan of an optimal solution to an instance of $F3||C_{\max}$. Then the above discussion implies that

$$C^*_{\max} = \max\left\{ C^{F}_{\max}{}^*, \sum_{i=1}^{n} (a_i + c_i) \right\}$$

$$= \max\{C^{F}_{\max}{}^*, \alpha + \gamma\}.$$

In other words, the $F2|\text{chain-reentrant}|C_{\max}$ problem is reduced to the $F3||C_{\max}$ problem. If $C^{F}_{\max}{}^* > \alpha + \gamma$, then $C^*_{\max} = C^{F}_{\max}{}^*$ and vice versa. If $C^{F}_{\max}{}^* \leq \alpha + \gamma$, then $C^*_{\max} = \alpha + \gamma$ and vice versa.

4. THE BRANCH-AND-BOUND ALGORITHM

Using Theorem 3, we develop a branch-and-bound algorithm that (implicitly) generates all $n2^{n-1}$ partitions of $\mathcal{J}$ and finds one with the minimum makespan.

The branch-and-bound tree is built up as follows. First, we order and reindex the jobs according to Johnson's rule for the first two operations. At the first level of the tree we designate the partition job; accordingly, there are n nodes at this level, since no job can be ruled out. The tree is binary beyond this level: at each of the lower levels, we decide for the corresponding job whether it will be scheduled before or after the partition job. A node at level ℓ ($\ell = 2, \ldots, n$) corresponds then to a *partial* partition of $\mathcal{J}$, say, $(\mathcal{A}, J_{k*}, \mathcal{B})$. We employ a *depth-first* strategy to explore the search tree.

A node can be discarded if it cannot induce an admissible partition. A sufficient condition for discarding a node is therefore $C_2(\mathcal{A}) > \alpha$; note that $C_2(\mathcal{A})$ is readily computed because of the way the jobs have been reindexed. Also, if $\max\{C_1(\mathcal{A}) + a_{k*}, C_2(\mathcal{A})\} + b_{k*} > \alpha$, then the only admissible partition that the current node can induce is $\{\mathcal{A}, J_{k*}, \mathcal{J}\backslash\mathcal{A}\}$; after evaluating the corresponding schedule, we can discard the node.

In the root node of the tree we compute the following four bounds to verify whether the incumbent upper bound equals the optimal solution value:

1. $\text{lb}^{(1)} = \alpha + \gamma$,
2. $\text{lb}^{(2)} = \text{Johnson}(\mathcal{J}, a, b) + \min_{1 \leq j \leq n} c_j$,
3. $\text{lb}^{(3)} = \text{Johnson}(\mathcal{J}, b, c) + \min_{1 \leq j \leq n} a_j$,
4. $\text{lb}^{(4)} = \beta + \min_{1 \leq j,k \leq n, j \neq k}(a_j + c_k)$.

In the other nodes of the tree, we use two simple but effective lower bounds that are computed in constant time after some preprocessing. The earliest completion time of M_2 in any complete schedule induced by the partial partition $\{\mathcal{A}, J_{k*}, \mathcal{B}\}$ is at least $C_2(\mathcal{A}) + \sum_{J_j \in \mathcal{J}\backslash\mathcal{A}} b_j$. Accordingly, a lower bound on $C^*_{\max}$ is

$$C_2(\mathcal{A}) + \sum_{J_j \in \mathcal{J}\backslash\mathcal{A}} b_j + \min_{J_j \in \mathcal{J}\backslash\mathcal{A}} c_j.$$

The second lower bound that we use is

$$C_3(\mathcal{A}) + \sum_{J_j \in \mathcal{J}\backslash\mathcal{A}} c_j.$$

5. APPROXIMATION ALGORITHMS

The dominance property in Theorem 3 suggests the development of the following three heuristic procedures—H_1, H_2, and H_3—that are shown in our computational experiments to be highly effective. In fact, these procedures find optimal solutions for most instances. They also offer a worst-case performance guarantee of 3/2.

For any instance I of $F2|\text{chain-reentrant}|C_{\max}$, let $C^*_{\max}(I)$ denote the minimum makespan, and $H_j(I)$ be the makespan determined by heuristic H_j, $j = 1, 2, 3$. Furthermore, let the ratio

$$\rho_j = \sup_I \left\{ \frac{H_j(I)}{C^*_{\max}(I)} \right\},$$

denote the *performance guarantee* or the *worst-case ratio* of heuristic H_j, $j = 1, 2, 3$.

Heuristic H_1

1. Schedule the jobs by applying Johnson's rule to $(\mathcal{J}, a, b)$ to obtain sequence σ_1.
2. Schedule the jobs by applying Johnson's rule to $(\mathcal{J}, b, c)$ to obtain sequence σ_2.
3. Let $H_1(I) = \min\{C_{\max}(\sigma_1), C_{\max}(\sigma_2)\}$.

Table II
Experiment I

Distribution: [0, 150], [0, 200], [0, 100]						
n	$\text{opt}_H(n_\ell)$	r_0	$r_1(\text{opt}_1)$	$r_2(\text{opt}_2)$	$r_3(\text{opt}_3)$	n_b, n_t, n_o, n_e
10	192 (156)	0.0001	0.0130 (141)	0.0013 (177)	0.0007 (190)	8, 8, 8, 595
20	196 (187)	0.0000	0.0066 (170)	0.0001 (192)	0.0000 (192)	4, 3, 4, 230
30	198 (191)	0.0000	0.0027 (178)	0.0000 (197)	0.0000 (197)	2, 1, 1, 104
50	199 (200)	0.0000	0.0020 (189)	0.0000 (198)	0.0000 (198)	1, 1, 1, 126
100	200 (200)	0.0000	0.0001 (199)	0.0000 (199)	0.0000 (200)	0, 0, 0, –

It is a naive heuristic, since it does not take advantage of the partition property. Nonetheless, its performance guarantee is 3/2.

Theorem 4. *For any instance* $I = (\mathcal{J}, a, b, c)$ *of* $F2|\text{chain-reentrant}|C_{\max}$, *we have that* $\rho_1 \leq 3/2$. □

The proof is quite straightforward; see Wang (1994). It is based on the lower bound $\max\{\alpha + \gamma, \text{JOHNSON}(\mathcal{J}, a, b), \text{JOHNSON}(\mathcal{J}, b, c)\}$ and the upper bound $\min\{\text{JOHNSON}(\mathcal{J}, a, b) + \gamma, \text{JOHNSON}(\mathcal{J}, b, c) + \alpha\}$ on $H_1(I)$.

The worst-case performance guarantee does not seem to be tight. For the worst instance we managed to find, a randomly generated instance, Heuristic H_1 provides a makespan just 20% greater than the optimal one.

We can take advantage of the partition property and, with a little more effort, improve the empirical performance of the naive heuristic H_1.

Heuristic H_2

1. Apply Johnson's rule to $(\mathcal{J}, a, b)$ to find a sequence σ_1 as in Heuristic H_1. Compute $C_{\max}(\sigma_1)$. If $C_{\max}(\sigma_1) = \alpha + \gamma$, stop.
2. Identify the partition job J_{k*} and hence the partition $\{\mathcal{J}_1, J_{k*}, \mathcal{J}_2\}$.
3. Apply Johnson's rule to $(J_{k*} \cup \mathcal{J}_2, b, c)$, while preserving the sequence of the jobs in $\mathcal{J}_1$. Store the new sequence as σ.
4. Schedule the jobs given σ and compute the makespan $C_{\max}(\sigma)$.

Note from Remark 2 in Section 3 that the partition $\{\mathcal{J}_3, J_{\ell*}, \mathcal{J}_4\}$ can be thought of as a mirror image of the partition $\{\mathcal{J}_1, J_{k*}, \mathcal{J}_2\}$. Hence, we have the following mirror image of the Heuristic H_2.

Heuristic H_3

1. Apply Johnson's rule to $(\mathcal{J}, b, c)$ to find a sequence σ_2. Compute $C_{\max}(\sigma_2)$. If $C_{\max}(\sigma_2) = \alpha + \gamma$, stop.
2. Identify partition $\{\mathcal{J}_3, J_{\ell*}, \mathcal{J}_4\}$ as in H_2.
3. Apply Johnson's rule to $(J_{\ell*} \cup \mathcal{J}_3, a, b)$ to get the final sequence σ as in H_2.
4. Schedule the jobs given σ and compute the makespan $C_{\max}(\sigma)$.

6. COMPUTATIONAL EXPERIMENTS

The purpose of this section is to evaluate the empirical performance of the heuristics discussed in the previous section, especially H_2 and H_3, against optimal solutions. The processing times of the instances follow a uniform distribution, and are randomly generated with six input parameters that define the three intervals: $[\ell_a, u_a]$, $[\ell_b, u_b]$, $[\ell_c, u_c]$. For each set of parameters, 200 instances are generated for a given number of jobs, n. For each instance I, we compute the lower bound $\text{LB}(I) = \max_{1 \leq j \leq 4} lb^{(j)}$ as discussed in Section 4, and heuristics H_1, H_2, and H_3 are applied to compute the first upper bound. Let $\text{UB}(I)$ be the best upper bound obtained by any of the algorithms. If $\text{UB}(I) > \text{LB}(I)$, then the branch-and-bound algorithm is invoked. We impose an upper limit on the number of nodes that the branch-and-bound algorithm has to search. If the limit is reached, the algorithm is stopped, and a "hard" instance is identified. We compare solution values of the heuristics against $C^*_{\max}(I)$, or against $\text{LB}(I)$ if an optimal solution is not obtained.

The results are summarized in Tables II–V. The tables are divided into seven columns. The headers of the columns are:

n = number of jobs;
opt_H = number of instances for which $\text{UB}(I) = C^*_{\max}(I)$;
opt_j = number of instances for which $H_j(I) = C^*_{\max}(I)$;
n_ℓ = number of instances for which $C^*_{\max}(I) = \alpha + \gamma$;
r_0 = upper bound on the average relative gap between $\text{UB}(I)$ and $C^*_{\max}(I)$;
r_j = upper bound on the average relative gap between $H_j(I)$ and $C^*_{\max}(I)$.

Table III
Experiment II

Distribution: [0, 100], [0, 200], [0, 100]						
n	$\text{opt}_H(n_\ell)$	r_0	$r_1(\text{opt}_1)$	$r_2(\text{opt}_2)$	$r_3(\text{opt}_3)$	n_b, n_t, n_o, n_e
20	188 (100)	0.0000	0.0331 (61)	0.0010 (158)	0.0006 (163)	12, 9, 12, 18919
50	183 (99)	0.0000	0.0321 (49)	0.0002 (160)	0.0001 (162)	17, 14, 14, 468
200	194 (98)	0.0000	0.0366 (16)	0.0000 (179)	0.0000 (180)	6, 6, 6, 207
500	200 (84)	0.0000	0.0410 (4)	0.0000 (192)	0.0000 (198)	0, 0, 0, –

Table IV
Experiment III

Distribution: [0, 100], [0, 300], [0, 150]						
n	$\text{opt}_H(n_e)$	r_0	$r_1(\text{opt}_1)$	$r_2(\text{opt}_2)$	$r_3(\text{opt}_3)$	n_b, n_i, n_o, n_e
20	173 (26)	0.0000	0.0343 (10)	0.0010 (141)	0.0011 (128)	27, 21, 27, 48860
50	181 (5)	0.0000	0.0355 (2)	0.0002 (145)	0.0002 (131)	19, 15, 15, 122
200	191 (0)	0.0000	0.0282 (0)	0.0000 (151)	0.0000 (166)	9, 8, 8, 419
500	199 (0)	0.0000	0.0269 (0)	0.0000 (187)	0.0000 (187)	1, 1, 1, 156

The branch-and-bound algorithm is evaluated with a four-tuple (n_b, n_i, n_o, n_e), where

n_b = number of instances for which the algorithm is invoked;
n_i = number of instances for which the algorithm improves UB(I);
n_o = number of instances for which the algorithm finds $C^*_{\max}(I)$;
n_e = average number of nodes for those instances for which $C^*_{\max}(I)$ is found.

The difference $n_b - n_o$ records the number of instances for which the algorithm is terminated when the node limit is reached.

On a DEC 5000 workstation, the times spent by the heuristics are negligible; the results are obtained almost instantaneously. Since the real time spent by any algorithm depends on the model of the computer and its implementation, we believe that n_e is a better measure of the efficiency of the branch-and-bound algorithm. If the number of nodes searched is below 10,000, the result is almost instantaneous. Therefore, unless a "hard" instance is encountered, the time spent by the branch-and-bound algorithm is also negligible.

Table II gives an average picture of the cases where the expected values $E(\beta) = 100n < (75 + 50)n = E(\alpha + \gamma)$. It is seen that these instances are "easy" in general. For example, when there are 10 jobs to schedule, the heuristics found optimal solutions for 192 out of 200 instances. Even the naive heuristic H_1 found optimal solutions in 141 instances. The relative errors are also very low, ranging from 0.007 for H_3 to 0.0130 for H_1. The branch-and-bound algorithm was invoked in only 8 out of the 200 instances, and it found optimal solutions for all of them after searching an average of 595 nodes. The lower bound LB(I) is essentially equal to the optimal value with a relative difference $r_0 = 0.0001$. As the number of jobs n increases, r_0 decreases, and the instances become even easier.

Comparing the results in Table II with those in Table V, where $E(\beta) > E(\alpha + \gamma)$, we see that the instances in Table V are harder. Nonetheless, the heuristics still provided optimal solutions for most of them, and the differences between the heuristic solutions and the lower bounds are once again negligible. An interesting observation, however, is that the corresponding three-machine flowshop problems become easier in the sense that our algorithms solved all except two of them to optimality.

A pattern seems to emerge. As the expected value of β becomes greater than that of the sum of α and γ, the chance that the branch-and-bound algorithm quickly locates the optimal solution decreases. However, the chance that our algorithms solve the corresponding three-machine flowshop problems increases. As the number of jobs increases, the performance of the heuristics H_2 and H_3 becomes even better, while that of H_1 worsens. This pattern further confirms the discussion at the end of Section 3 about the relationship between permutation schedules for the three-machine flowshop and compact no-passing schedules for the two-machine chain-reentrant shop, and strengthens the importance and applicability of the dominance property of Theorem 3.

7. SPECIAL CASES OF $F2|\text{CHAIN-REENTRANT}|C_{\max}$

In this section we identify some special cases that are solvable in polynomial time; we call them *well solvable*. We also develop a pseudo-polynomial algorithm for the case where for any job the processing times of the two operations on the primary machine are equal.

7.1. Well-solvable Cases

A well-solvable case of $F3||C_{\max}$ induces a well-solvable case of $F2|\text{chain-reentrant}|C_{\max}$; see the discussion at the end of Section 3. The Cases 1–3 listed below are therefore well solvable. These special cases are solved by applying

Table V
Experiment IV

Distribution: [0, 100], [0, 400], [0, 150]						
n	$\text{opt}_H(n_e)$	r_0	$r_1(\text{opt}_1)$	$r_2(\text{opt}_2)$	$r_3(\text{opt}_3)$	n_b, n_i, n_o, n_e
20	174 (2)	0.0001	0.0292 (2)	0.0006 (152)	0.0005 (133)	26, 16, 26, 10624
50	186 (0)	0.0000	0.0206 (0)	0.0001 (149)	0.0001 (151)	14, 13, 14, 804
200	195 (0)	0.0000	0.0160 (0)	0.0000 (173)	0.0000 (171)	5, 4, 5, 15521
500	200 (0)	0.0000	0.0152 (0)	0.0000 (189)	0.0000 (198)	0, 0, 0, –

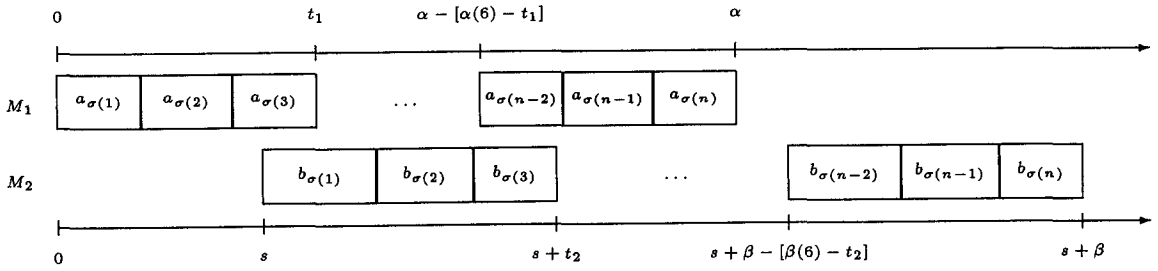

Figure 6. An illustration of the case $a_j = c_j$ with six jobs already scheduled.

Johnson's rule to the pseudo $F2\|C_{\max}$ problem formed by letting $a'_j = a_j + b_j$ and $b'_j = b_j + c_j$. The reader is referred to Baker (1995) for a summary of these cases, and to Burns and Rooker (1976, 1978) and Szwarc (1977) for further details.

Case 1. One machine dominates another in terms of processing times, i.e., one of the following conditions holds:

1. $\min\{a_j\} \geq \max\{b_j\}$.
2. $\min\{c_j\} \geq \max\{b_j\}$.
3. $\min\{b_j\} \geq \max\{a_j\}$.
4. $\min\{b_j\} \geq \max\{c_j\}$.

Case 2. Regressive second stage: $b_j \leq \min\{a_j, c_j\}$ for all j.

Case 3. Constant second stage: $b_j = b = constant$.

7.2. A Special Case: $a = c$

Another natural special case is the one in which operation O_{j1} is identical to operation O_{j3} for all J_j, and hence $a_j = c_j$ for all J_j. This is a good approximation of some real-world situations. For instance, chemical processing on the photolithography station in wafer production takes about the same time for every entry. This special case is still binary *NP*-hard. In this section we prove that the case $a_j = c_j$ for all J_j is solvable in pseudo-polynomial time. Specifically, we present a dynamic programming algorithm that runs in $O(n^2\alpha^2\beta)$ time and $O(n\alpha^2)$ space. Throughout this section we assume that the jobs have been reindexed according to Johnson's rule for (a, b), i.e., for the first two operations. Also, we let $\alpha(j) = \sum_{i=1}^{j} a_i$ and $\beta(j) = \sum_{i=1}^{j} b_i$.

If $a = c$, then according to Theorem 3, there is an optimal compact no-passing schedule in which the ordering of the *left*-jobs, i.e., the jobs before the partition job, are in Johnson's order for (a, b), and the *right*-jobs, i.e. the jobs after the partition job, are in Johnson's order for (b, a).

A compact no-passing schedule σ specifies a unique time point s, at which the 2-block starts. Therefore, given a partition job J_k $(k = 1, \ldots, n)$, we can make a guess at the value of s, $0 \leq s \leq \alpha$, and enumerate all feasible schedules (there might be none) for the jobs $J_1, \ldots, J_j$ $(j = 1, \ldots, n, j \neq k)$ with their 1-operations scheduled in the interval $[0, \alpha]$ and their 2-operations in the interval $[s, s + \beta]$. Since we cannot predetermine the time at which the 3-block starts, we temporarily put all the 3-operations in the interval $[\alpha + \beta, 2\alpha + \beta]$. When a feasible schedule is found, the 3-block will be moved to the left to compute the makespan. To preserve the feasibility, J_j can be moved to the left at most $\delta_j = \min(\beta, S_{j3} - C_{j2})$ units of time, and the 3-block can only be moved to the left $\Delta = \min_{1 \leq j \leq n} \delta_j$ units of time. For any feasible schedule σ for the given s and J_k, we thus define Δ as the *slack* of σ; the makespan of σ is then actually $2\alpha + \beta - \Delta$. The optimal schedule given s and J_k is then the one that maximizes Δ.

For any partition job J_k $(k = 1, \ldots, n)$ and a given integer s, $0 \leq s \leq \alpha$, and for $j = 1, \ldots, k-1, k+1, \ldots, n$, let $F_j^{k,s}(t_1, t_2)$ denote the *maximum* slack for scheduling jobs $J_1, \ldots, J_j$ such that the 1-operations and the 2-operations of the left-jobs among them fill up the intervals $[0, t_1]$ and $[s, s + t_2]$, respectively. The state variables t_1, t_2, and s contain all the information we need: it follows that the 1-operations of the right-jobs are scheduled in the interval $[\alpha - \alpha(j) + t_1, \alpha]$ on M_1, the 2-operations of the right-jobs in the interval $[s + \beta - \beta(j) + t_2, s + \beta]$ on M_2, the 3-operations of the left-jobs in the interval $\alpha + \beta, \alpha + \beta + t_1]$ on M_1, and the 3-operations of the right-jobs in the interval $[2\alpha + \beta - \alpha(j) + t_1, 2\alpha + \beta]$ on M_1. See Figure 6 for an example of a partial schedule in which six jobs, $J_1, \ldots, J_6$, are already scheduled. Note that we only show the 1- and 2-blocks since the 3-block follows.

If we now add $J_j \neq J_k$ to a partial schedule associated with $F_{j-1}^{k,s}(t_1, t_2)$, $t_2 \leq \alpha - s$, we must consider the following cases:

- Scheduling J_j as a left-job: this is feasible only if
  1. $s + t_2 \geq t_1 + a_j$; then the operations of J_j do not overlap in their execution.
  2. $s + t_2 + b_j \leq \alpha$; otherwise, J_j would be the partition job.

  The slack for J_j is then equal to $\delta_j = \min\{\beta, S_{j3} - C_{j2}\} = \min\{\beta, \alpha + \beta + t_1 - s - t_2 - b_j\}$.
- Scheduling J_j as a right-job: this is feasible only if $S_{j2} \equiv s + \beta - \beta(j) + t_2 > \alpha$. The slack for J_j is then equal to $\delta_j = \min\{\beta, 2\alpha - \alpha(j) + \beta(j-1) - s + t_1 - t_2\}$.

We are now ready to set up the recursion. For notational convenience we define

$$\delta_j^L(s, t_1, t_2) = \begin{cases} \min\{\beta, \alpha + \beta + t_1 - s - t_2 - b_j\}, \\ \quad \text{if } t_1 + a_j \leq s + t_2 \leq \alpha - b_j, \\ -\infty, \quad \text{otherwise,} \end{cases}$$

and

$$\delta_j^R(s, t_1, t_2) = \begin{cases} -\infty, & \text{if } s + \beta - \beta(j) + t_2 \leq \alpha, \\ \min\{\beta, 2\alpha - \alpha(j) + \beta(j-1) - s + t_1 - t_2\}, & \text{otherwise.} \end{cases}$$

The initialization of the recursion is

$$F_0^{k,s}(t_1, t_2) = \begin{cases} \beta, & \text{if } t_1 = 0,\ t_2 = 0, \text{ and } 0 \leq s \leq \alpha, \\ -\infty, & \text{otherwise,} \end{cases}$$

and the recursion for $k = 1, \ldots, n$, $s = 0, \ldots, \alpha$, $j = 1, \ldots, n, j \neq k$, $0 \leq t_1 \leq \alpha(j)$, $0 \leq t_2 \leq \min\{\alpha - s, \beta(j)\}$, is then given by

$$F_j^{k,s}(t_1, t_2) = \left\{ \begin{array}{l} F_{j-1}^{k,s}(t_1, t_2), \quad \text{if } J_j = J_k, \\ \max\left\{ \begin{array}{l} \min\{F_{j-1}^{k,s}(t_1 - a_j, t_2 - b_j), \\ \delta_j^L(s, t_1 - a_j, t_2 - b_j) \\ \min\{F_{j-1}^{k,s}(t_1, t_2), \delta_j^R(s, t_1, t_2)\} \end{array} \right\}, \end{array} \right.$$

$$\text{if } J_j \neq J_k.$$

Finally, we schedule the partition job J_k. Let $G^{k,s}(t_1, t_2)$ denote the maximum slack if we add J_k to a schedule associated with $F_n^{k,s}(t_1, t_2)$. The only way to schedule J_k in any such schedule is to put its 1-operation in the time slot $[t_1, t_1 + a_k]$ on M_1, its 2-operation in $[s + t_2, s + t_2 + b_k]$ on M_2, and its 3-operation in $[\alpha + \beta + t_1, \alpha + \beta + t_1 + a_k]$ on M_1. Accordingly, we have

$$G^{k,s}(t_1, t_2) = \begin{cases} \min\{F_n^{k,s}(t_1, t_2), \alpha + \beta + t_1 - s - t_2 - b_k\}, \\ \qquad \text{if } s + t_2 \geq t_1 + a_k, \\ -\infty \qquad \text{otherwise.} \end{cases}$$

The overall maximum slack Δ^* can be found as

$$\Delta^* = \max_{1 \leq k \leq n, 0 \leq s \leq \alpha, 0 \leq t_1 \leq \alpha, \alpha - s - a_k \leq t_2 \leq \alpha - s} G^{k,s}(t_1, t_2),$$

and the optimal sequence σ^* is determined by backtracing. The minimal makespan of σ^* is equal to $2\alpha + \beta - \Delta^*$. This dynamic programming algorithm requires $O(n^2\alpha^2\beta)$ time and can be implemented to run in $O(n\alpha^2)$ space.

8. CONCLUDING REMARKS

In this paper we have studied a class of reentrant shops called chain-reentrant shops. The problem of minimizing makespan is shown to be binary *NP*-hard even for the two-machine case. We have proved that when $m = 2$, there is an optimal schedule that can be partitioned into three segments with the first and the third being scheduled according to Johnson's rule and the second scheduled arbitrarily. Based on this result, a branch-and-bound optimization algorithm and three approximation algorithms with worst-case performance guarantee 3/2 are developed. We have tested the approximation algorithms and shown their performance to be excellent. We have also identified a special case that can be solved in pseudo-polynomial time and some other cases that are well solvable.

Two immediate questions remain to be settled for the two-machine problem. First, does there exist a pseudo-polynomial algorithm for its solution? This question concerns the complexity of the problem; in other words, is the problem unary *NP*-hard? Second, is the worst-case performance guarantee of our approximation algorithms tight?

To conclude, the approach we have adopted isolates the effect of the reentrant flow successfully and reduces the problem to a partitioning problem, with each partition representing a simple two-machine flowshop scheduling problem. Our algorithms can be used also to solve effectively three-machine flowshop problems, especially those with makespan greater than the sum of processing times for the 1- and 3-operations. The insight we have gained here is that some jobs are more critical than others; once the positions of these jobs are determined, scheduling of other jobs becomes easier. To extend the idea, we speculate that jobs in a more complicated manufacturing environment can be grouped such that in each group the jobs are scheduled around one or more critical jobs, or on one or two bottleneck machines. Thus our approach suggests an interesting avenue for future research, that is, the possibility of developing new decomposition techniques to be applied to reentrant shops as well as to flowshops with more than two machines.

REFERENCES

BAKER, K. R. 1995. *Elements of Sequencing and Scheduling*, Revised Edition. Hanover, NH.

BAI, S. AND S. B. GERSHWIN. 1990. Scheduling Manufacturing Systems with Work-in-process Inventory. *Proceedings of the 29th IEEE Conference in Decision and Control.* **2,** 557–564.

BRAMCON, M. 1993. Instability of FIFO Queuing Networks. Technical Report, Mathematics Department, University of Wisconsin, Madison, WI.

BURNS, F. AND J. ROOKER. 1976. Johnson's Three-machine Flow Shop Conjecture. *Opns. Res.* **24,** 578–580.

BURNS, F. AND J. ROOKER. 1978. Three Stage Flow Shops with Regressive Second Stage. *Opns. Res.* **26,** 207–208.

ELLIOT, D. J. 1989. *Integrated Circuit Fabrication Technology.* McGraw-Hill Book Company, New York.

GAREY, M. R. AND D. S. JOHNSON. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco.

GAREY, M. R., D. S. JOHNSON, AND R. SETHI. 1976. The Complexity of Flowshop and Jobshop Scheduling. *Math. O.R.* **1,** 117–129.

GRAHAM, R. L., E. L. LAWLER, J. K. LENSTRA, AND A. H. G. RINNOOY KAN. 1979. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Ann. Discrete Math.* **5,** 287–326.

GUPTA, J. N. D. 1993. Private correspondence.

JOHNSON, S. M. 1954. Optimal Two- and Three-stage Production Schedules with Setup Times Included. *Naval Res. Logist.* **1,** 61–68.

KUBIAK, W., S. X. LOU, AND Y. WANG. 1996. Mean Flow Time Minimization in Reentrant Jobshops with Hub. *Opns. Res.* **44,** 764–776.

KUMAR, P. R. 1993. Reentrant Lines. To appear in a special issue of *Queuing Systems: Theory and Applications*.

KUMAR, P. R. AND T. I. SEIDMAN. 1990. Dynamic Instabilities and Stabilization Methods in Distributed Real-time Scheduling of Manufacturing Systems. *IEEE Trans. Automatic Control AC-35*, 289–298.

LAWLER, E. L., J. K. LENSTRA, AND A. H. G. RINNOOY KAN. 1982. Recent Developments in Deterministic Sequencing and Scheduling: A Survey. In *Deterministic and Stochastic Scheduling*, M. A. H. Dempster, J. K. Lenstra, and A. H. G. Rinnooy Kan (eds.). Reidel, Dordrecht, The Netherlands.

LAWLER, E. L., J. K. LENSTRA, A. H. G. RINNOOY KAN, AND D. B. SHMOYS. 1993. Sequencing and Scheduling: Algorithms and Complexity. In the *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*, S. C. Graves, A. H. G. Rinnooy Kan, and P. Zipkin (eds.). North-Holland, Amsterdam, The Netherlands.

LEV, V. AND I. ADIRI. 1984. *V*-shop Scheduling. *European J. Opnl. Res.* **18,** 51–56.

LOU, S. X. C., H. YAN, S. SETHI, A. GARDEL, AND P. DEOSTHALI. 1991. Using Simulation to Test the Robustness of Various Existing Production Control Policies. *Proceedings of Winter Simulation Conference*, 261–269.

LU, S. H. AND P. R. KUMAR. 1991. Distributed Scheduling Based on Due Dates and Buffer Priorities. *IEEE Trans. Automatic Control*, **36,** 1406–1416.

NOBLE, P. J. W. 1989. *Printed Circuit Board Assembly*. John Wiley & Sons, New York.

SETHI, S. P. AND X. ZHOU. 1994. Stochastic Dynamic Jobshop and Hierarchical Production Planning. *IEEE Trans. Automatic Control* **39,** 2061–2075.

SZWARC, W. 1977. Optimal Two Machine Orderings in the $3 \times n$ Flow Shop Problem. *Opns. Res.* **25,** 70–77.

WANG, Y. M. 1994. Scheduling Flowshops with Reentrant Flows and Pallet Requirements. *Ph.D. Thesis, University of Toronto.*