On Parallel-Machine Scheduling with Operator-Constrained Setups

by J.W. Herrmann, C-Y. Lee

T.R. 94-85

Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry

On parallel-machine scheduling with operator-constrained setups

Jeffrey W. Herrmann
Institute for Systems Research
University of Maryland, College Park

Chung-Yee Lee
Department of Industrial and Systems Engineering
University of Florida

November 1, 1994

Abstract

The processing of a task on a machine often requires an operator to setup the job. In this paper we consider the problem of scheduling a finite set of jobs on a number of identical parallel machines. Each job has a setup that must be performed by an operator, who can perform only one setup at a time. We examine the problems of minimizing the schedule makespan. Our results include complexity proofs, special cases that can be solved in polynomial time, lower bounds, and approximation algorithms with error bounds.

1 Introduction

In many manufacturing environments, a set of machines is overseen by an operator who must setup a machine for processing a task. Once setup, however, the machine can run until the end of the task without operator intervention. In this paper we consider the problem of scheduling such a system of semiautomatic machines.

We are given a set of identical machines, a set of jobs, and one operator. Each job consists of both a setup that must be performed by the operator and a task. Both the setup and the task have to be scheduled on a machine (the task must immediately follow the setup). The operator can perform only one setup at a time; thus, scheduling the jobs on the machines also requires scheduling the setups for the operator. A machine can perform only one task at a time, and preemption is prohibited.

The problem is one of scheduling the tasks on the machines in order to minimize the makespan, which is defined as the time between the start of the first setup and the completion of the last task.

The problem can be denoted as follows: There exists a set of m machines M_j, $j = 1, ..., m$. There exists a set of n jobs J_i, $i = 1, ..., n$. Job J_i requires a setup of time

s_i and processing for time p_i.

This problem is quite similar to the machine interference problems studied by Miller and Berry (1974, 1977), Aronson (1984), and Stecke and Aronson (1985); these problems have many identical jobs preassigned to each machine. In addition, Koulamas and Smith (1988) and Koulamas (1993) study heuristics for parallel machines, preassigned jobs, and a common server. Hall, Potts, and Sriskandarajah (1994) review this previous work and provide a complete analysis of the complexity of various parallel machine scheduling problems with a common server. In this paper we describe the performance of a number of heuristics that can be used to find approximate solutions and a number of special cases that can be solved optimally in polynomial time. This work was done independently of the work by Hall, Potts, and Sriskandarajah.

Researchers have studied many different problems associated with scheduling parallel automatic machines, including those problems with sequence-independent setups, in which case the setups are included in the processing time, and those with sequence-dependent setups, which lead to traveling salesman problems. For these problems, it is assumed that the machines or resources can setup themselves. The problem of minimizing makespan on parallel identical machines with no sequence-dependent setups is a strongly NP-complete problem, as shown by Garey and Johnson (1978). Tang (1990), Geoffrion and Graves (1976), and Parker, Deane, and Holmes (1977) all study parallel-machine scheduling problems with setups or changeover costs. Some resource-constrained scheduling problems have been considered, but all except the simplest problems are computationally complex. Additional research has studied very general formulations. Surveys of such results can be found in Lawler, Lenstra, Rinnooy Kan, and Shmoys (1989) and Cheng and Sin (1990).

The rest of the paper is organized as follows: Section 2 provides a basic result for the general case. In Section 3 we consider some special cases and prove that even a special case of our problem is strongly NP-complete. We discuss in Section 4 a branch-and-bound for the general makespan problem. Section 5 presents a number of heuristics and their corresponding error bounds, and Section 6 presents the results of the empirical testing of those algorithms. Section 7 concludes the paper with a look at possible future directions.

2 Basic Results

Any schedule can be considered as a sequence of setups, since the operator can perform only one setup at a time. For a given schedule, we say that a machine M_j is *available* at time t if the machine completes the processing of the last task at time t.

Lemma 1 (First Available Machine) *For any regular objective function, there exists an optimal schedule in which each setup is done on the first available machine. (There exists no machine that is available earlier than the machine on which the job is setup.)*

Proof. Suppose we are given an optimal schedule with the following property: Job J_j is setup on machine M_k at time t_j, M_k is available at some time $t_k \leq t_j$, and machine M_r is available at some time $t_r < t_k$. Let J_h be the job setup on M_r at time $t_h \geq t_j$, if such a job exists. (See Figure 1a.) M_r is idle between t_r and t_h. We can move to M_r all of the jobs on M_k from J_j to the end of the given schedule if we simultaneously move to M_k all of the jobs on M_r from J_h to the end of the given schedule (if any). Because $t_r < t_k \leq t_j$ and $t_h > t_j \geq t_k$, it may be possible to begin tasks on M_k and M_r earlier. (See Figure 1b.) In any case, this switch does not delay any tasks, and J_j has been started on an machine available earlier. By repeating such switches, one can create a schedule in which each job begins on the first available machine and no completion time has been increased; therefore, the value of any regular objective function has not been increased, and the new schedule must also be optimal. Q.E.D.

Lemma 2 (Sum of Setups) *Suppose that the jobs are numbered such that $s_1 \leq s_2 \leq \ldots \leq s_n$. If there exists $k < m$ such that $s_1 + s_2 + \ldots + s_k \geq \max\{p_i\}$, then the following algorithm will find an optimal solution, and the minimal makespan can be achieved using only $k+1$ machines.*

Algorithm 1.

Step 1. Find the $k+1$ jobs with the smallest processing times. Let S be the set of such jobs.

Step 2. Sequence the jobs that are not in S in any order and sequence S in decreasing order of processing times.

Step 3. Starting with the jobs not in S, assign the first job to M_1, the next to M_2, and so on. After scheduling a job on M_{k+1}, begin again with M_1. Continue with the jobs in S.

Proof. Consider a schedule created by Algorithm 1. The time t between setups on the same machine will at least the time necessary to perform the k setups on the other machines in use. Thus, $t \geq s_1 + \ldots + s_k \geq \max\{p_i\}$, so the processing of a task on a machine cannot delay the next setup on that machine. Thus, the operator is able to perform all setups continuously, and the use of additional machines cannot improve the schedule. Since each machine ends with a job from S, the completion time of some job in S determines the makespan of the schedule. Since the operator performs setups continuously, minimizing the makespan is equivalent to scheduling the operator to minimize $\max\{S_j + p_j\}$, where S_j is the completion of the setup. Sequencing the jobs in S longest processing time first yields an optimal schedule. Q.E.D.

3 Special Cases

In this section we consider two special cases: in Case 1, all $p_i = p$; in Case 2, all $s_i = s$. Similar results were proved independently by Hall, Potts, and Sriskandarajah (1994).

Case 1. All $p_i = p$.

Lemma 3 *If $p_i = p$ for all jobs J_i, then there exists an optimal schedule in which the operator performs setups cyclically on $M_1, M_2, \ldots, M_m$.*

Proof. This result follows directly from the first available machine result. Consider the construction of a schedule. At the beginning of the schedule, all m machines are available, so the first m setups can be performed on the machines in any order, including $M_1, M_2, \ldots, M_m$. Since the processing times are all equal, the first available machine will be the one that was setup first. Thus the operator repeats the cycle $M_1, M_2, \ldots, M_m$. Q.E.D.

The following special case of Case 1 can be solved by extending Lemma 2:

Corollary 1 *Suppose that the jobs are numbered such that $s_1 \leq s_2 \leq \ldots \leq s_n$. If $p_i = p$ for all jobs J_i and there exists $k < m$ such that $s_1 + s_2 + \ldots + s_k \geq p$, then sequencing the jobs in any order and assigning them to $k + 1$ machines is an optimal solution.*

Theorem 1 (Case 1) *If there are two machines, all $p_i = p$, and the s_i are arbitrary, then minimizing the makespan is an NP-complete problem.*

Proof: Transform into our problem a PARTITION PROBLEM WITH EQUAL CARDINALITY: Given $a_1, ..., a_{2n}$ with $\sum a_i = 2A$, can we partition $a_1, ..., a_{2n}$ into two sets so that each set has n elements with total sum equal to A? (This problem is NP-complete according to Garey and Johnson, 1979.)

Generate an instance for our problem: $p_i = A$ for $i = 1, ..., 2n + 2$. $s_i = a_i$ for $i = 1, ..., 2n$. $s_{2n+1} = s_{2n+2} = 0$. Can we find a feasible schedule with makespan equal to $(n + 2)A$?

Suppose that we have a partition. Let the partition be S_1 and S_2. Rearrange the jobs in S_1 in ascending order of s_i and the jobs in S_2 in descending order of s_i. Assign J_{2n+1} and then S_1 to M_1, and assign S_2 and then J_{2n+2} to M_2. It can be checked that the schedule will have a makespan of $nA + A + A = (n + 2)A$. (There is no idle time in the schedule.) See Figure 2.

Suppose that we have a schedule with makespan equal to $(n + 2)A$. The sum of all setup time and processing time is equal to $(2n + 2 + 2)A = (2n + 4)A$. In order to achieve a makespan of $(n + 2)A$, there will be no idle time on either machine, and two setups will start at time zero. One of these setups will be have to be of length zero. Without loss of generality, let us assign J_{2n+1} as the first job on M_1. Due to the alternating property (Lemma 3), there are exactly $n + 1$ jobs on each machine and the last setup occurs on M_2. Similarly, since there is no idle time, two setups must end at time $(n + 1)A$. And since the operator can do only one setup at a time, one of these setups must be J_{2n+2}, which, as the setup started last, will be on M_2. Thus we have n

non-zero setups on each machine whose total setup time is A. We must have a partition.

Therefore, we have a partition if and only if we have a schedule with makespan equal to $(n+2)A$. Thus, minimizing the makespan is an NP-complete problem. Q.E.D.

Case 2: All $s_i = s$. The following corollary to Lemma 2, which is similar to Corollary 1, provides an optimal solution to a special case of Case 2:

Corollary 2 *If $s_i = s$ for all jobs J_i and there exists $k < m$ such that $ks \geq \max\{p_i\}$, then Algorithm 1 will construct using $k+1$ machines a schedule with optimal makespan.*

If the processing times are large, the following theorem shows that minimizing the makespan is an NP-complete problem in the strong sense even if $m = 2$. This result corresponds to Theorem 2.2 of Hall, Potts, and Sriskandarajah (1994).

Theorem 2 (Case 2) *If there are two machines, all $s_i = s$, and the p_i are arbitrary, then minimizing the makespan is a strongly NP-complete problem.*

Proof. Transform into our problem a 3-PARTITION PROBLEM: Given $a_1, ..., a_{3n}$ with $\sum a_i = nB$, can we partition $a_1, ..., a_{3n}$ into n sets so that each one has 3 elements with sum equal to B? (Without loss of generality, we assume that $B/4 < a_i < B/2$ for all i.) This problem is strongly NP-complete according to Garey and Johnson, 1979.

Generate an instance for our problem. $p_i = 3B$ for $i = 1, ..., n$. $p_i = a_{i-n}$ for $i = n+1, ..., 4n$. $p_i = 0$ for $i = 4n+1$. $s_i = B/2$ for $i = 1, ..., 4n+1$. Can we find a feasible schedule with makespan equal to $3nB + B/2$?

Suppose that we have a partition. Let the n sets of the partition be $S_1, ..., S_n$. S_i also refers to the corresponding three jobs in the set $\{J_i : i = n+1, ..., 4n\}$ whose processing times sum to B. Assign J_1, S_2, J_3, S_4, . . . , J_{n-2}, S_{n-1}, and J_n to M_1 (assuming n is odd). Assign S_1, J_2, S_3, J_4, . . . , S_{n-2}, J_{n-1}, and S_n to M_2. It can be checked that the schedule created will have a makespan of $3nB + B/2$. (There is no idle time.) See Figure 3.

Suppose that we have a feasible schedule with makespan equal to $3nB+B/2$. Since the sum of all setup times and processing times is equal to $3nB + nB + B/2(4n+1) = 6nB + B/2$, there must be no idle time except for the first setup time. Note that $p_i < s = B/2$ for all $i = n+1, ..., 4n$. During the setup for one machine, the other machine must be processing jobs from $J_i : 1 \leq i \leq n$. Otherwise, there will have to be idle time on the other machine during that setup. Furthermore, during the processing of any job from $J_i : n+1 \leq i \leq 4n+1$, the other machine must be processing a job from $J_i : 1 \leq i \leq n$. Otherwise, it can be checked easily that there must be some idle time.

From the above two observations, we conclude that the jobs in $J_i : 1 \leq i \leq n$ must alternate between M_1 and M_2 and one of these jobs must begin processing immediately after the completion of the job on the other machine. Without loss of generality we assign the jobs in numeric order. Note that during the processing of any job from $J_i : 1 \leq i < n$, the other machine must process exactly three jobs from $J_i : n+1 \leq i \leq 4n+1$ since the processing time of any two jobs is less than B, and the processing time of any four jobs is greater than B. This implies that the sum of these three job processing times is equal to B. Once we have $n-1$ sets of jobs whose processing times sum to B, we know that the processing times of the last set of three jobs must also equal B. Thus, we have a partition.

Therefore, we have a partition if and only if we have a feasible schedule with makespan equal to $3nB + B/2$. This shows that minimizing the makespan is a strongly NP-complete problem. Q.E.D.

4 Branch-and-bound

In this section we disuss a branch-and-bound approach to the general problem: multiple machines and no constraints on the processing times and setup times. We will present lower bounds used in a straight-forward branch-and-bound procedure for the problem and a truncated implementation of the procedure.

With the first available machine property in place, it is clear that we can search for the optimal schedule by searching over all sequences of jobs. From any partial schedule, a branch is added for each of the unscheduled jobs. When adding a job to a partial schedule, the new job is setup on the first available machine (or any one of the first available machines). No branching on the machine assignment is necessary.

We now describe two lower bounds for a partial schedule. In order to do so, we will introduce some notation associated with a partial schedule.

m is the number of machines.

A is the set of scheduled jobs.

N is the set of unscheduled jobs.

t is the completion time of the last setup.

t_k is the completion time of the last job scheduled on M_k.

$F(t)$ is the set of machines $M_k : t_k \geq t$.

The maximum t_k is a trivial lower bound. For the first new lower bound, the number of machines is increased to the number of tasks. The operator must perform each remaining setup, and the lower bound on the makespan can be found by renumbering the jobs in LPT order: $p_1 \geq p_2 \geq \ldots \geq p_n$.

$$LB_1 = t + \max_{J_j \in N}\{ \sum_{J_i \in N, i \leq j} s_i + p_j \}.$$

The second lower bound recognizes that all remaining setups and jobs must be done after the last scheduled setup.

$$LB_2 = t + \frac{1}{m}(\sum_{M_k \in F(t)} (t_k - t) + \sum_{J_j \in N} (s_j + p_j)).$$

These lower bounds can be extended to form global lower bounds for a problem instance (N is the set of all jobs):

$$LB_1 = \max_{J_j \in N}\{ \sum_{i \leq j} s_i + p_j \}.$$

$$LB_2 = \frac{1}{m} \sum_{J_j \in N} (s_j + p_j).$$

While a branch-and-bound procedure may take, in the worst case, an exponentially large amount of effort to search the entire solution space, it is often able to find very good solutions at the beginning of its search. The remainder of the search is spent finding a few incremental improvements or verifying that no better solution exists elsewhere. Thus, it may be possible to employ a branch-and-bound search to find very good solutions in a reasonable amount of time. We implemented a truncated (depth-first) branch-and-bound search that halts, if the search is not yet finished, after searching 100,000 nodes. The best solution found so far is an approximate but hopefully optimal or near-optimal schedule. The results of using this procedure, which will still require more effort than a standard scheduling heuristic, are reported in the section on empirical results.

5 Heuristics

Due to the complexity of the scheduling problem, we are unable to find optimal solutions in reasonable time. Therefore, we concentrate on constructing algorithms that find high-quality approximate solutions and finding bounds on their worst-case performance. The average performance of these heuristics on a number of randomly generated problem sets is described in the next section. Although heuristics have been proposed for the scheduling of semiautomatic machines (Miller and Berry, 1974; Koulamas and Smith, 1988), the problems studied in these papers have preassigned jobs. We consider heuristics that assign jobs to machines and schedule the work of the operator.

Combine. We begin by including each setup as part of the job processing time. After assigning the jobs to the machines, we will use a simple rule to order the setups for the operator. The assignment of the jobs is the classic parallel-machine problem of minimizing the makespan on a fixed number of machines. This problem has been studied intensively, and a number of approximation algorithms have been proposed and analyzed. The Combine algorithm described by Lee and Massey (1988) uses the Longest Processing Time (LPT) list schedule as the incumbent and then applies MULTIFIT to reduce the makespan. This algorithm is an improvement to the MULTIFIT algorithm proposed by Coffman, Garey, and Johnson (1978).

In order to find a feasible schedule for the operator, the LPT ordering of the jobs created by the first-fit-decreasing procedure is maintained on each machine. After finishing one setup, the operator does the next setup on the first available machine. If more than one machine is available, the operator does the next setup for the available machine with the most remaining work.

LPT list schedule. An alternative heuristic is to create a list schedule directly by re-ordering the entire set of jobs. The first job is setup on the first machine. The operator performs the setup for each subsequent job on the first available machine. In the LPT list schedule, the jobs are ordered by decreasing setup plus processing time: $s_1 + p_1 \geq s_2 + p_2 \geq \ldots \geq s_n + p_n$.

Shortest Setup Time list. For this list schedule, the jobs are sequenced in order of increasing setup times: $s_1 \leq s_2 \leq \ldots \leq s_n$. The goal is to get each machine working as soon as possible.

Mixed list. This list schedule places shorter setups first and shorter processing times last. Specifically, the *mixed* sequence is formed by renumbering the jobs so that if $i < j$, then $\min\{s_i, p_j\} \leq \min\{s_j, p_i\}$. Such an ordering can always be performed.

If the shortest setup time is greater than or equal to the longest processing time, then Lemma 2 applies, and the mixed list finds the optimal solution, since it will put the shortest task last, creating a schedule with makespan equal to $\sum s_i + p_{\min}$.

Worst-Case Performance.

Let us denote our problem by P and define a new problem P' such that $s'_j = 0$ and $p'_j = s_j + p_j$. Let $C^*(P)$ denote the optimal makespan for P, and let $C^*(P')$ denote the optimal makespan for P'. Since the operator is never busy in problem P', whenever a machine finishes processing a job, the setup for the next job can follow immediately. Thus P' is a m parallel identical machine scheduling problem with no setups and $p'_i = s_i + p_i$. It is clear that the optimal makespan for P' is no larger than the optimal makespan for P:

Property 1 $C^*(P') \leq C^*(P)$.

Property 2 *Suppose that a heuristic generates for P' a sequence H of setups. Let $C_H(P')$ be the makespan of the schedule constructed by implementing H for problem P'. Let $C_{H'}(P)$ be the makespan of the schedule constructed by implementing H for problem P, where the jobs are setup in the same order as in H and on the first available machine. If $C_H(P') \leq (1 + \alpha)C^*(P')$, then $C_{H'}(P) \leq (2 + \alpha)C^*(P)$.*

Proof. First, apply H to P by performing the setups in the same order and on the same machines, and let $C_H(P)$ be the makespan of this schedule. This may violate the first-available machine rule. This schedule can be constructed from the solution to P' by removing the conflict on the operator. Some delaying of setups may be necessary. Consider each setup in the order in which the operator performs them. The setup for $J_{[i+1]}$ starts after the setup for $J_{[i]}$. If these setups overlap, delay $J_{[i+1]}$ and all remaining jobs on all machines by the amount of overlap δ, which is less than $s_{[i]}$. This maintains the relative timing of all remaining jobs. After the setup conflicts have been resolved. no job has been delayed by an amount more than $\sum s_j$. Thus,

$$C_H(P) \leq \sum s_j + C_H(P').$$

Let H' be the solution for P formed by scheduling the setups in the same order as H, this time on the first available machine. By Lemma 1, this cannot increase the makespan.

$$C_{H'}(P) \leq C_H(P)$$

Since $\sum s_j \leq C^*(P)$,

$$C_{H'}(P) \leq C^*(P) + (1+\alpha)C^*(P') \leq (2+\alpha)C^*(P).$$

Q.E.D.

Corollary 3 *Since the worst-case relative error of a list schedule is* $2 - \frac{1}{m}$ *for* P' *(Graham, 1966), the worst-case error of a list schedule for* P *is is* $(3 - \frac{1}{m})C^*(P)$.

This bound includes the Mixed and Shortest Setup list schedules.

Corollary 4 *Since the worst-case relative error of an LPT schedule is* $\frac{4}{3} - \frac{1}{3m}$ *for* P' *(Graham, 1969), the worst-case error of a LPT schedule for* P *is* $(\frac{7}{3} - \frac{1}{3m})C^*(P)$.

Example. Consider the following seven-job, three-machine problem:

Job	Setup	Task
1	1	4
2	2	3
3	1	3
4	2	2
5	1	2
6	1	2
7	2	1

The optimal schedule for P is $C^*(P) = 11$, which can be achieved if the jobs are setup in the order 5, 2, 3, 4, 1, 6, 7. The sum of setups $\sum s_j = 10$. The optimal schedule for P' is $C^*(P') = 9$, and the error bound is $(1+\alpha) = 11/9$. The jobs are numbered in LPT order, and the LPT schedule has a makespan of $C_H(P') = 11$ (see Figure

4a). Including the setups in this schedule increases the makespan to $C_H(P) = 16$ (see Figure 4b). Note $16 \leq \sum s_j + C_H(P') = 10 + 11 = 21$. Rescheduling the setups on the first available machine decreases the setup to $C_{H'}(P) = 12$ (see Figure 4c). Finally, note $12 \leq (2+\alpha)C^*(P) = (20/9)11 = 24\frac{4}{9}$.

Corollary 5 *Since the worst-case relative error of a MULTIFIT schedule is* $1 + \frac{2}{11}$ *for* P' *(Yue, 1990), the worst-case error of the Combine schedule for* P *is* $(2 + \frac{2}{11})C^*(P)$.

6 Empirical Results

In this section we present the results of using various heuristics to solve the makespan problem. We used the four heuristics described in Section 5: the three list-based rules and the Combine heuristic. We also implemented the truncated branch-and-bound search described in Section 4. We tested the algorithms on a number of problem sets of various sizes. The makespan of each solution generated is compared to the lower bounds for each problem. These lower bounds were presented in Section 4. The problem set data and the results are shown in Table 1. The performance is the average (over ten problems) of the relative percentage deviation from the lower bound.

From these results we can see that the truncated branch-and-bound procedure found consistently better solutions. On all of the larger problem sets, the branch-and-bound was able to find optimal solutions without searching more than $100,000$ nodes. The Mixed list schedule was the best of the four scheduling heuristics. On the problems in which the setup times were uniformly greater than the processing times, the Mixed list schedule found the optimal solution (since it achieved the lower bound); this was predicted in Section 3. The only significant exception to this dominance are the larger problems in which the setup times were consistently smaller than the processing times (s302az, s306az, s506ay, s506az). For these problems, the LPT and Combine heuristics assign a nearly equal amount of work to each machine, which yields better solutions, since conflict between setups is not a concern.

7 Summary

In this work, we have considered a parallel-machine scheduling problem in which the setup for each job must be performed by an operator. We have proved some fundamental results, including the complexity of the problem, and presented some approximation algorithms for finding near-optimal solutions for the problem of minimizing the makespan. We have investigated the worst-case and average performance of the heuristics. For all heuristics, the worst-case relative performance is less than 3. The heuristics are able to find optimal and near-optimal solutions, but the relative performance of the heuristics depends upon the amount of conflict on the operator. In addition we have identified new special cases that can be solved in polynomial time.

We have begun to look at the problem of minimizing the total flowtime and at problems with precedence constraints between the jobs. Some simple problems with chains of jobs have been considered. If approximation algorithms do not yield good results for some computationally difficult problems, heuristic searches like simulated annealing or genetic algorithms may be useful for finding very good solutions. Additional future work may include the problem of allocating machines to multiple workers in order to optimize system performance.

8 References

Aronson, J.E., "Two heuristics for the deterministic, single operator, multiple machine, multiple run cyclic scheduling problem," *Journal of Operations Management*, Vol. 4, pp. 159-173, 1984.

Cheng, T.C.E., and C.C.S. Sin, "A state-of-the-art review of parallel-machine scheduling research," *European Journal of Operational Research*, Vol. 47, pp. 271-292, 1990.

Coffman, E.G., M. Garey, and D. Johnson, "An application of bin-packing to multiprocessor scheduling," *SIAM Journal of Computing*, Vol. 7, pp. 1-16, 1978.

Garey, M.R., and D.S. Johnson, "Strong NP-completeness results: motivation, examples, and implications," *Journal of the ACM*, Vol. 25, pp. 499-508, 1978.

Garey, M.R., and D.S. Johnson, *Complexity and Intractability*, W.H. Freeman, San Francisco, California, 1979.

Geoffrion, A., and G. Graves, "Scheduling parallel production lines with changeover costs: practical applications of a quadratic assignment/LP approach," *Operations Research*, Vol. 24, pp. 595-610, 1976.

Graham, R.L., "Bounds for certain multiprocessing anomalies," *Bell System Technical Journal*, Vol. 45, pp. 1563-1581, 1966.

Graham, R.L., "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, Vol. 17, pp. 263-269, 1969.

Hall, N.G., C.N. Potts, and C. Sriskandarajah, "Parallel machine scheduling with a common server," Working paper 94-21, College of Business, The Ohio State University, May, 1994.

Koulamas, C.P., "Scheduling on two parallel machines for minimizing machine interference," Working paper, Department of Decision Sciences and Information Systems, Florida International University, 1993.

Koulamas, C.P., and M.L. Smith, "Look-ahead scheduling for minimizing machine interference," *International Journal of Production Research*, Vol. 26, pp. 1523-1533, 1988.

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

Lee, C.-Y., and J.D. Massey, "Multiprocessor scheduling: combining LPT and Multifit," *Discrete Applied Mathematics*, Vol. 20, pp. 233-242, 1988.

Miller, J.G., and W.L. Berry, "Heuristic methods for assigning men to machines: an experimental analysis," *AIIE Transactions*, Vol. 6, pp. 97-104, 1974.

Miller, J.G., and W.L. Berry, "The assignment of men to machines: an application of branch and bound," *Decision Sciences*, Vol. 8, pp. 56-72, 1977.

Parker, R., R. Deane, and R. Holmes, "On the use of a vehicle routing algorithm for the parallel processor problem with sequence dependent changeover costs," *AIIE Transactions*, pp. 155-160, 1977.

Stecke, K.E., and J.E. Aronson, "Review of operator/machine interference models," *International Journal of Production Research*, Vol. 23, pp. 129-151, 1985.

Tang, C.S., "Scheduling batches on parallel machines with major and minor setups," *European Journal of Operational Research*, Vol. 46, pp. 28-37, 1990.

Yue, M., "On the exact upper bound for the MULTIFIT processor scheduling algorithm," in M. Yue, ed., "Operations research in China," *Annals of Operations Research*, Vol. 24, pp. 233-259, 1990.

Table 1. Problems Sets and Empirical Results.
Performance is the average (over ten problems) of the relative percentage deviation from the lower bound.

Problem Set	Jobs	Machines	Setups	Tasks	B&B	Mix	LPT	Setup	Combine
s152ax	15	2	1-10	1-10	4.05	11.32	12.77	16.58	19.94
s152ay	15	2	1-10	1-30	0.50	5.52	7.97	7.54	7.50
s152az	15	2	1-10	11-20	0.68	6.25	6.71	6.25	10.43
s152bx	15	2	1-30	1-10	0.70	1.79	2.76	3.98	8.58
s152by	15	2	1-30	1-30	4.10	11.66	13.28	17.79	25.53
s152bz	15	2	1-30	11-20	9.32	10.24	10.97	12.18	22.85
s152cx	15	2	11-20	1-10	0	0	0.53	1.65	1.86
s152cy	15	2	11-20	1-30	7.19	12.74	12.14	16.14	19.88
s152cz	15	2	11-20	11-20	2.81	5.62	5.72	7.74	13.12
s302ax	30	2	1-10	1-10	8.12	13.72	11.57	16.72	14.69
s302ay	30	2	1-10	1-30	0.38	5.10	5.65	6.25	5.37
s302az	30	2	1-10	11-20	0.27	4.83	2.83	4.83	3.15
s302bx	30	2	1-30	1-10	1.66	1.85	2.12	2.57	6.30
s302by	30	2	1-30	1-30	8.63	14.42	12.58	16.57	19.97
s302bz	30	2	1-30	11-20	13.89	11.70	11.51	12.58	18.99
s302cx	30	2	11-20	1-10	0	0	0.09	1.13	0.15
s302cy	30	2	11-20	1-30	10.24	13.13	11.92	15.46	14.46
s302cz	30	2	11-20	11-20	4.31	6.21	4.77	7.93	5.91
s306ax	30	6	1-10	1-10	0	0	0.32	3.13	0.43
s306ay	30	6	1-10	1-30	0	3.72	4.37	14.47	6.45
s306az	30	6	1-10	11-20	0	6.86	2.29	6.86	2.94
s306bx	30	6	1-30	1-10	0	0	0.42	0.88	0.66
s306by	30	6	1-30	1-30	0	0.18	0.73	2.98	1.14
s306bz	30	6	1-30	11-20	0	0.08	0.46	0.93	0.48
s306cx	30	6	11-20	1-10	0	0	0.09	1.13	0.09
s306cy	30	6	11-20	1-30	0	0	0.35	3.51	0.37
s306cz	30	6	11-20	11-20	0	0	0.09	1.10	0.09
s506ax	50	6	1-10	1-10	0	0.23	0.29	2.02	0.33
s506ay	50	6	1-10	1-30	0	5.72	4.16	11.53	3.55
s506az	50	6	1-10	11-20	0	6.79	2.02	6.79	0.98
s506bx	50	6	1-30	1-10	0	0	0.16	0.67	0.58
s506by	50	6	1-30	1-30	0	0.41	0.43	2.49	0.70
s506bz	50	6	1-30	11-20	0	0.27	0.26	0.93	0.32
s506cx	50	6	11-20	1-10	0	0	0.09	0.66	0.09
s506cy	50	6	11-20	1-30	0	0	0.20	2.05	0.20
s506cz	50	6	11-20	11-20	0	0	0.09	0.65	0.09