CHAPTER 4
LOOK-AHEAD SCHEDULING PROBLEMS

In this chapter we discuss the second major area of this research. We study the problem of scheduling a machine that processes jobs headed for two different second-stage machines. We analyze this three-machine problem with three different objective functions: makespan, total flowtime, and number of tardy jobs. We first examine the complexity of the problem and then identify some lower bounds as well as some special cases that can be solved in polynomial time. We develop a number of heuristics that find good solutions to the problem. We also use a branch-and-bound technique to find optimal solutions.

4.1 Introduction

Job shop scheduling includes those scheduling problems in which different jobs may follow different routes through the shop. These problems are generally the hardest to solve optimally, since few properties of optimal schedules are known and the number of possible solutions explodes as the problems increase in size.

Because of the complexity of job shop scheduling, algorithms to find the optimal solution (in a reasonable amount of time) for even the simplest objective functions, e.g. makespan, do not exist. Recent research has shown that bottleneck-based techniques such as the shifting bottleneck algorithm (Adams, Balas, and Zawack, 1988) or bottleneck dynamics (see Morton, 1992, for example) can be successful at finding good schedules. Traditionally, however, researchers have studied (and schedulers have used) dispatching rules to order the jobs waiting for processing at a machine.

Normal dispatching rules consider only the jobs currently in the queue for the machine being scheduled. We define *look-ahead scheduling* as the ability of a sequencing procedure to

include information about the status of machines downstream in the flow, enabling it to make a better solution. Previous techniques that use this look-ahead idea include the work-in-next-queue and number-in-next-queue dispatching rules (Panwalker and Iskander, 1977) and the use of bottleneck starvation avoidance in shop floor control by Glassey and Petrakian (1989). Robinson *et al.* (1993) consider upstream and downstream information in scheduling semiconductor batch operations. Researchers have also studied lot release policies that look ahead to the status of the inventory in front of or arriving at a bottleneck; see for example, Wein (1988), Glassey and Resende (1988), and Leachman, Solorzano, and Glassey (1988). Other researchers have studied procedures that they called look-ahead scheduling (Koulamas and Smith, 1988; Zeestraten, 1990) but the problem setting or interpretation is different.

Consider two examples from the semiconductor test area. In the first, lots of two different products are processed through the same brand workstation. After brand, the lots require electrical testing, but the differences between the products indicate that the lots must be tested on different machines. Or consider the effect of re-entrant flows. The various lots waiting for processing at an electrical test workstation may be at one of two points in their route. At one point, a lots moves to brand after being tested (and it will return to test at some point in the future). At another, it moves to visual/mechanical inspection. In the first case, the brand workstation is sending lots to two different testers; in the second, the tester is sending lots to two dissimilar workstations. If one of the second-stage machines is a bottleneck, it seems clear that the sequencing of lots on the first-stage machine should try to maximize the efficient use of that bottleneck.

We can model this scenario with the following three-machine problem: There are three machines M_0, M_1, and M_2. Each job follows one of two different flows: M_0 - M_1, or M_0 - M_2. Thus, M_0 is feeding the other two machines. If one of these second-stage machines is a bottleneck because the total work required on that machine is the larger than that on the other machine, the sequencing of jobs on M_0 should have as a priority the proper feeding of that

machine. This idea of a bottleneck will not affect our analysis of the problem. We will, however, return to it for the heuristics and the empirical testing.

This problem, which could occur in any number of manufacturing environments, forms an interesting general flow shop problem and a subproblem of the job shop scheduling problem. As a flow shop problem, it is a simple model unlike the multi-machine problems previously discussed in the literature, although work has been done on flexible flow shops with multiple parallel machines at any given stage.

Moreover, the research into our three-machine problem may improve job shop scheduling in two ways: one, the solution procedures can be applied directly to the subproblem of scheduling machines near the bottleneck machine, and two, these techniques may be translated into good look-ahead dispatching rules for scheduling throughout the shop.

In this chapter we investigate three objective functions for this problem: the minimization of makespan, of total flowtime, and of the number of tardy jobs. We are concerned with the analysis of solutions to the problem and the development of heuristics which can be used to find good solutions.

The major contributions of this work include the proof that minimizing makespan is a strongly NP-complete problem, the identification of optimality properties and special cases that can be solved in polynomial time, and the development of an approximation algorithm.

The look-ahead scheduling problems under investigation are as follows (using the numbering given earlier):

4. Three-Machine Look-Ahead Scheduling: Makespan (3MLA-MS)
5. Three-Machine Look-Ahead Scheduling: Flowtime (3MLA-FT)
6. Three-Machine Look-Ahead Scheduling: Number of Tardy Jobs (3MNT)

In this chapter we will look at each of these objective functions. In Chapter 2 we discussed the research relevant to these problems. In the next section we start with the makespan objective. In Section 4.3 we look at minimizing the total flowtime, and Section 4.4 discusses work on minimizing the number of tardy jobs.

4.2 Minimizing the Makespan

As mentioned in the introduction, we are studying a problem that is a subproblem of the general job shop scheduling problem and is also a special case of the general flow shop problem. All analysis of the flow shop starts with Johnson (1954), who studied the minimization of makespan for two-machine flow shop problems and for some special three-machine flow shop problems. His famous algorithm starts jobs with the smallest first-stage tasks as soon as possible and jobs with the smallest second-stage tasks as late as possible.

Special cases of the flow shop makespan problem have been studied by a number of researchers, including Mitten (1958), Conway, Maxwell, and Miller (1967), Burns and Rooker (1975), and Szwarc (1977). Garey, Johnson, and Sethi (1976) proved that the general three-machine problem was NP-complete. Problems with release dates, preemption, precedence constraints, or more than three machines have also been studied.

In the flexible flow shop, more than one machine may be present at a particular stage. Heuristics for this type of problem are discussed by Wittrock (1988), Sriskandarajah and Sethi (1989), Gupta (1988), and Gupta and Tunc (1991). Lee, Cheng, and Lin (1992) study an assembly flow shop problem where each job consists of two subassembly tasks that are assembled in a third operation.

This three-machine problem is therefore closely related to problems previously studied, but the pre-assignment of the jobs to different second-stage machines gives this problem a special structure and leads to interesting results.

4.2.1 Notation

The following list describes the components of the problem and the notation used.

J_j	Job j, $j = 1, \ldots, n$.
M_0	The first-stage machine.
M_1, M_2	The second-stage machines.
H_1	The set of jobs that visit M_0 and then M_1.
H_2	The set of jobs that visit M_0 and then M_2.
p_{0j}	The first-stage processing time of J_j on M_0.
p_{ij}	The second-stage processing time of J_j on M_i, $i = 1$ or 2.

For a given schedule σ, we can calculate the following variables:

C_{0j}	The completion time of J_j on M_0.
C_{ij}	The completion time of J_j on M_i, $i = 1$ or 2.
C_j	$= C_{ij}$, the second-stage completion time of J_j.
C_{max}	$= \max \{C_j\}$, the makespan of the schedule.
$\sum C_j$	the total flowtime of the schedule.

Note that we will call a set of jobs that visit the same second-stage machine a *group*; thus, H_1 and H_2 are each a group of jobs. Each group has a *flow*. The flow for the jobs in H_1 is M_0 - M_1. The flow for the jobs in H_2 is M_0 - M_2. This section is concerned with the problem of minimizing, over all feasible schedules, the makespan of the jobs. We call this problem the Three-Machine Look-Ahead problem - Makespan (3MLA-MS).

4.2.2 Johnson's Algorithm

If we consider just one group and its corresponding flow, the problem of minimizing the makespan of the set of jobs that visit these two machines is the same as Johnson's two-machine flow shop problem. Johnson (1954) provided an optimality rule and an algorithm to solve the problem. If each job to be scheduled has task processing times a_j and b_j on machines one and two respectively, then his rule is as follows:

Johnson's Rule: J_i precedes J_j in an optimal sequence if $\min \{a_i, b_j\} < \min \{a_j, b_i\}$.

This rule is implemented in the following algorithm:

Johnson's Algorithm:

Step 1. Find the smallest unscheduled task processing time. (Break ties arbitrarily.)

Step 2. If this minimum is on the first machine, place the job in the first open position in the schedule. Else, place the job in the last open position in the schedule. Return to Step 1.

4.2.3 Permutation Schedules

In flow shop scheduling, a feasible schedule is called a permutation schedule if the sequence of jobs on each machine is the same. Thus, the sequence of jobs on the first machine uniquely identifies a schedule (assuming all tasks are started as soon as possible). In the three-machine look-ahead problem that we are studying, the two sequences on the second-stage machines consist form two disjoint sets of jobs. Thus, for our problem, we extend the idea of permutation schedules to include schedules where the relative order of two jobs in the same flow is the same at both stages.

It is known that considering permutation schedules is sufficient for finding optimal solutions for regular two-machine flow shop problems. Thus, for our three-machine look-ahead problems, it seems likely that permutation schedules will also be sufficient, since the problem contains two two-machine flows. We will show that this is indeed true.

Definition: A schedule is a *permutation schedule* if, for all J_j and J_i in H_1 (H_2), J_i precedes J_j on M_0 if and only if J_i precedes J_j on M_1 (M_2).

Theorem 4.1. For any regular performance measure, there exists a permutation schedule that is an optimal schedule.

Proof. First we will show that we can interchange two jobs that are not in the same order at both stages. Given a schedule σ where job J_j directly follows J_i on machine M_1 (or M_2) but J_j precedes J_i on machine M_0, move the M_0 task of J_j after the M_0 task of J_i.

If we look at Figure 4.1, we can observe that moving J_j causes all of the tasks (except J_j) on M_0 to start earlier, which does not delay any (and may expedite some) second-stage tasks. Now, since J_j will complete on M_0 when J_i did ($C'_{0j} = C_{0i}$), the processing of J_j on M_1 will not be delayed. Our interchange, therefore, does not increase the completion time of any job.

Thus, for any given schedule, we can create a corresponding permutation schedule by interchanging the first-stage tasks. None of the job completion times are increased by this construction. Indeed, some of the completion times may be decreased. Therefore, this permutation schedule has a better or equal performance on all regular measures (e.g. makespan, flowtime, number of tardy jobs, maximum lateness). Thus, it is sufficient to consider permutation schedules when trying to minimize these objective functions. QED.

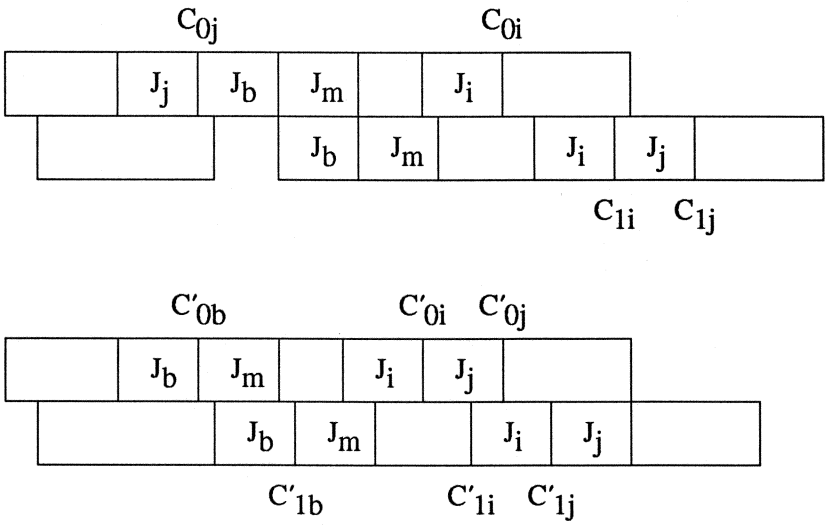

Figure 4.1. The exchange into a permutation schedule for M_0 and M_1. (Machine M_2 not shown.)

We would note here that a permutation schedule preserves the relative order of the sequence for each group. We will use the term *interleaving* for the process of combining two sequences to form a permutation schedule. We will show in Section 4.2.5 that if we are given sequences for each group, then there is a polynomial-time algorithm to find the interleaving that minimizes the makespan of the schedule created. However, the optimal second-stage sequences cannot be determined in polynomial time. In fact, the 3MLA-MS problem is strongly NP-complete, as will see in the next section.

4.2.4 NP-Completeness

In this section, we consider the complexity of the 3MLA-MS problem. Although other researchers (including Gonzalez and Sahni, 1978) have shown the NP-completeness of a number of small shop problems, we cannot determine the complexity of our problem from any of this previous work.

We will therefore prove that 3MLA-MS is strongly NP-complete, which will be done by transforming 3-Partition to 3MLA-MS. (Recall that it is sufficient to consider permutation schedules.) The 3-Partition problem can be stated as follows: Given a set of a_i, $i = 1, \ldots, 3n$, and B, partition these elements into n sets $A_1, \ldots, A_n$, where each set contains three elements and the sum of the three elements equals B. We will make the assumption that for all i, $1/4\ B < a_i < 1/2\ B$. (We can transform any problem without this property into one where it is true.) With this property, there will never be a set of two a_i or a set of four a_i where the sum of the elements equals B.

Theorem 4.2. 3MLA-MS is a strongly NP-complete problem.

Proof. Given an instance of 3-Partition, with a set of a_i, $i = 1, \ldots, 3n$, and B (note all $a_i \geq 1$), create $4n$ jobs, $3n$ of which go from M_0 to M_2 and have processing times $p_{0i} = a_i$ and $p_{2i} = (B+1)a_i$, $i = 1, \ldots, 3n$; let these jobs form a set W. The n remaining jobs that go from M_0 to M_1, $J_{3n+1}, J_{3n+2}, \ldots, J_{4n}$, $p_{0k} = B^2$ for all k and $p_{1k} = (B+1)B$, $k = 3n, \ldots, 4n\text{-}1$, and $p_{1,4n} = B$; let these jobs form a set X. The desired makespan is $M = n(B^2 + B) + B$.

Part 1. If there exists a partition $(A_1, \ldots, A_n)$ such that for all a_i in A_k, $\sum a_i = B$, then the sequence on M_0 of $[A_1\ J_{3n+1}\ A_2\ J_{3n+2} \ldots A_n\ J_{4n}]$ will yield a schedule where the completion time on M_1 will be M and the time on M_2 will be no greater than M. (See Figure 4.2.)

Part 2. Now, suppose there exists no partition. Consider any arbitrary permutation schedule σ. The sequence is composed of consecutive subsets of jobs from W and of jobs from X. Let A_j, $j = 1, \ldots, n+1$ be these subsets of W, where A_j directly precedes the jth job from X, except for A_{n+1}, which is the set of jobs following the last job from X. Note that any of these A_j

may be empty. Let S_j equal the sum of the first-stage task processing times for the jobs in A_j. Let A_k be the first A_j such that S_k is not equal to B. Because there is no partition, there must exist one such A_k among $A_1, \ldots, A_n$.

If $S_k > B$, then the makespan on machine M_1 is delayed by the delay in the kth job in X: $MS(M_1) \geq S_1 + B^2 + \ldots + S_k + B^2 + (n - k)(B^2 + B) + B > kB + kB^2 + (n - k)(B^2 + B) + B = M$. (This assumes that all of the first k-1 jobs have long tasks on M_1, leaving n-1-(k-1) long tasks and one short task on M_1. If one of the first k-1 has the short task, then the makespan is even longer.)

If $S_k < B$, let $S = S_1 + \ldots + S_k < kB$ (and $\leq kB - 1$) and suppose J_m is the first job from W after the kth job from X. Then, the makespan on M_2 is postponed by J_m: $MS(M_2) \geq S + kB^2 + a_m + (B + 1)(nB - S) \geq S + kB^2 + a_m + nB^2 + nB - BS - S$. Including $-BS \geq -kB^2 + B$ implies $MS(M_2) \geq a_m + nB^2 + nB + B > M$. Thus, there exists no schedule with makespan less than or equal to M.

Part 1 and Part 2 of the proof show that there exists a partition if and only if there exists a schedule with the desired makespan. This implies that 3MLA-MS is a strongly NP-complete problem. QED.

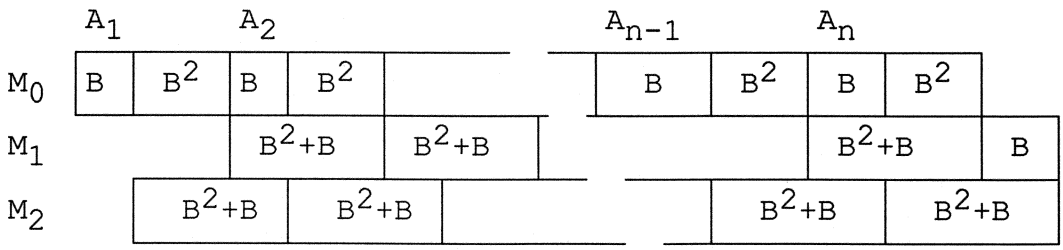

Figure 4.2. A schedule for the 3MLA-MS problem.

4.2.5 Makespan Optimality Conditions and Polynomially-Solvable Cases

This section discusses the optimal combination of two given sequences, lists a few properties of optimal schedules, and presents two special cases that have easily-found optimal solutions. We will see that we can easily form a permutation schedule by optimally interleaving

sequences for each group (Algorithm 4.1). Properties 4.1, 4.2, and 4.3 are dominance properties between jobs in the same group. Theorems 4.3 and 4.4 describe the special cases.

Because we need to consider only permutation schedules, we can create a single schedule for all three machines from a sequence of the jobs. That is, given a sequence on the first-stage machine we can create a sequence for each second-stage machine. This is done by considering the jobs in the order they appear on the first machine. However, there is no comprehensive rule for determining this sequence. Thus, we will spend some time on how this sequence should be constructed in certain situations.

Interleaving two sequences. Since the problem is NP-complete in the strong sense, heuristic methods for finding good solutions are justified. A very natural heuristic is to schedule the jobs in each group separately. For instance, each group can be scheduled by Johnson's rule. Then we can interleave these two sequences to form a solution to the original problem. The process of combining two sequences to achieve the minimal makespan is called *optimally interleaving*. This is the best combination of these two sequences. Note, however, that the best combination of the two Johnson sequences may not be an optimal solution; the optimal solution may be some combination of sub-optimal solutions to each subproblem. In Section 4.2.9 we present an example where optimally interleaving the Johnson sequences for each group is not optimal.

So, while interleaving the Johnson sequences is a natural heuristic that works well (see Section 4.2.8), we may wish to try other heuristics to the subproblems before interleaving. These are discussed in Section 4.2.7. In this subsection we will describe how two sequences should be interleaved.

Now, suppose we are given two sequences σ_1 and σ_2, one sequence for each group (we have found solutions for each of the subproblems), and we want to find the minimal makespan that can be achieved by combining these two sequences. The following observations will lead us to an algorithm for doing so.

Define C as a makespan that we wish to achieve. To minimize the makespan given the two sequences we need to find the minimal C for which we can find a feasible schedule. We can schedule the tasks on the second-stage machines in the order given by σ_1 and σ_2 and as late as possible, so that the last task on each machine ends at C. In a feasible schedule, the first-stage task for each job has to complete on M_0 before the second-stage task can begin. We need to determine if there is some ordering of the tasks on M_0 so that each task finishes on-time (with respect the second-stage task).

For each job J_j the start time of the second-stage task can be used as the due date d_j for the corresponding first-stage task. We can find a sequence where each first-stage task finishes on-time ($C_{0j} \leq d_j$) if and only if we can find a sequence where the maximum lateness is less than or equal to zero. If we wish to minimize the maximum lateness of the first-stage tasks, we should order them by EDD (Earliest Due Date), according to Jackson (1955). The schedule created is an interleaving of the two given schedules: if J_i precedes J_j in σ_k, then $d_i < d_j$, and J_i will precede J_j on M_0.

Each due date $d_j = C - t_j$, where t_j is the sum of the second-stage processing times of J_j and the jobs J_k that follow J_j on the second-stage machine. See Figure 4.3.

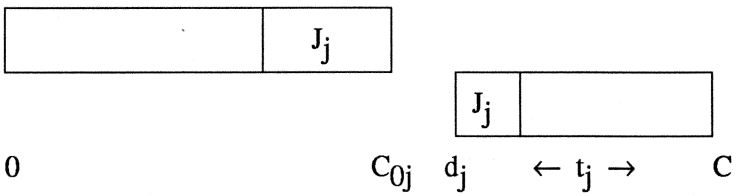

Figure 4.3. The variables associated with J_j (second-stage started late).
Note: Other second-stage machine not shown.

Since the EDD sequence corresponds to sequencing the tasks in decreasing order of t_j, the sequence of jobs is the same for all values of C, including the optimal one. Therefore, we can find the optimal makespan from the feasible schedule with the jobs in this order and all tasks starting as soon as possible. This is the optimal interleaving of two given sequences, described in Algorithm 4.1.

Algorithm 4.1 (Optimal Interleaving): Given a sequence σ_1 for the jobs in H_1 and a sequence σ_2 for the jobs in H_2, perform the following steps to yield a schedule with the minimal makespan:

Step 1. For each J_j in H_1, define A_j as the set of jobs (not including J_j) that follow J_j in σ_1. Then $t_j = p_{1j} + \Sigma_{Aj} p_{1k}$.

Step 2. For each J_j in H_2, define A_j as the set of jobs (not including J_j) that follow J_j in σ_2. Then $t_j = p_{2j} + \Sigma_{Aj} p_{2k}$.

Step 3. Schedule the jobs on M_0 in decreasing order of the t_j, starting at time zero, and start all second-stage tasks as soon as possible.

Note that this algorithm takes O(n) effort, since each group is already in decreasing order of the t_j, and forming the schedule is only combining the two sequences without changing the relative orderings.

In Example 4.1 we perform Algorithm 4.1 on a problem with five jobs and given sequences for each group. We are given sequences [$J_1 J_3 J_2$] and [$J_4 J_5$] for each group. After computing the t_j, we form the interleaved sequence [$J_1 J_3 J_4 J_5 J_2$]. The corresponding schedule has a makespan of 11 (see Figure 4.4).

Example 4.1. Given the following five jobs and the two sequences [$J_1 J_3 J_2$] and [$J_4 J_5$]:

J_j	H_i	p_{0j}	p_{ij}	t_j
J_1	H_1	2	4	9
J_2	H_1	1	1	1
J_3	H_1	2	4	5
J_4	H_2	2	1	4
J_5	H_2	1	3	3

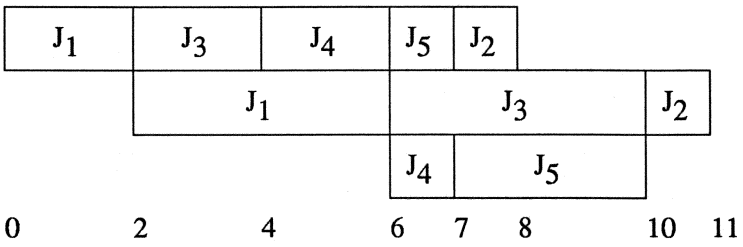

Figure 4.4. Optimal schedule for Example 4.1.

Properties of optimal solutions. In general, we still do not know how to construct the sequences on M_1 and M_2. These subproblems are what is difficult about this problem. Johnson's rule, which is optimal for the two-mchine flow shop, is not always applicable for the 3MLA-MS problem. Of some help, however, are a number of dominance properties for jobs in the same group that limit the number of sequences that need to be considered in searching for the optimal. The first property establishes a precedence relationship between two jobs that is a restrictive form of Johnson's rule. The other two relate two jobs that are processed consecutively on some machine.

Property 4.1. (Absolute Dominance) If both J_i and J_j are in H_k and $p_{0i} \leq p_{0j}$ and $p_{ki} \geq p_{kj}$, then J_i should precede J_j. (If both equalities hold, then the jobs are obviously interchangeable.)

Proof. Suppose we have a schedule σ such that the inequalities above hold and J_j precedes J_i. Without loss of generality, suppose both jobs are in H_1 and that all of the second-stage tasks are started as late as possible. Then it is easy to see that exchanging J_i and J_j on both machines (see Figure 4.5) does not increase the makespan. Thus, it is sufficient to consider schedules where J_i precedes J_j. QED.

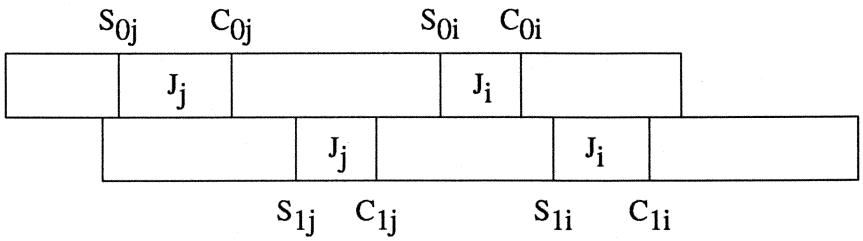

Figure 4.5a. J_i and J_j on M_0 and M_1. (Machine M_2 not shown.)

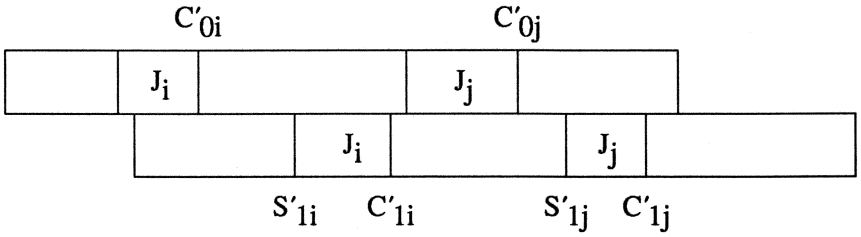

Figure 4.5b. The exchange of J_i and J_j on M_0 and M_1. (Machine M_2 not shown.)

Property 4.2. (Consecutive Dominance) If both J_i and J_j are in H_k and are processed consecutively on M_0, they should be sequenced by Johnson's rule (i.e., if min $\{p_{0i}, p_{kj}\}$ < min $\{p_{ki}, p_{0j}\}$, then J_i should immediately precede J_j).

Proof. The fact that both jobs are processed consecutively on both machines implies that Johnson's Rule applies to them. If the jobs are not sequenced according to Johnson's Rule, they can easily be exchanged, and the makespan of the schedule is not increased by the switch. QED.

This property is stated for two jobs from the same group processed consecutively. It can be extended to a set of three of more consecutive jobs on M_0 from the same group:

Corollary 4.1. (Batch Dominance) The jobs in each batch of a schedule should be ordered by Johnson's Rule, where a batch is defined as a set of consecutive jobs on M_0 from the same group.

Property 4.3. (Small Second-stage Dominance) Suppose J_i and J_j are in H_k and are processed consecutively on M_k. If $p_{0i} \geq p_{ki}, p_{0j} \geq p_{kj}$, and $p_{ki} > p_{kj}$, then J_i should precede J_j.

Proof. Without loss of generality, suppose J_i and J_j are in H_1. Consider a schedule where the two groups have been optimally interleaved and the second-stage tasks are started as late as possible. Then, if there are any jobs between J_i and J_j on M_0, they must be from H_2, and the start times of these jobs on M_2 are between the start times of J_i and J_j on M_1. (See Figures 4.6a and 4.6b.) Suppose J_j precedes J_i. Let us interchange these jobs on both M_0 and M_1 and check that we still have a feasible schedule without increasing the makespan. Let y be the old start time of J_j on M_1. Let z be the old completion time of J_i on M_0. For J_j, the new completion time on M_0

equals z. Since J_i can be on only one machine at a time, $z \leq y + p_{1j}$, the old start time of J_i on M_1. Thus, $z < y + p_{1i}$, the new start time of J_j on M_1, and the schedule is still feasible for J_j.

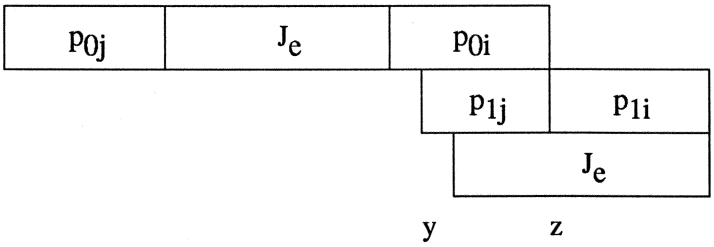

Figure 4.6a. (second-stage tasks started late).

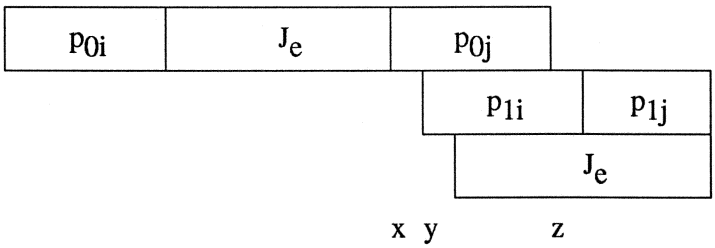

Figure 4.6b. (second-stage tasks started late).

Now, let x be the new start time of J_j on M_0. Thus $y \geq z - p_{1j} \geq z - p_{0j} = x$. Now, for J_i and for all jobs J_e between J_j and J_i, the new completion time is less than x and thus y and thus the start time of their second-stage tasks. We can therefore switch J_j and J_i without increasing the makespan of the schedule. QED.

Corollary 4.2. If for some H_k, $p_{0j} \geq p_{kj}$ for all J_j, then the jobs in H_k should be ordered by LPT on the M_k tasks.

We note that if Corollary 4.2 is true for both H_1 and H_2, we have the first special case mentioned below (Theorem 4.3).

Polynomially-solvable special cases. Like other three-machine flow shop problems, the 3MLA-MS problem has special cases that can be solved in polynomial time. Theorems 4.3 and 4.4 present such special cases.

Theorem 4.3. If $p_{1k} \leq p_{0k}$ for all J_k in H_1 and $p_{2j} \leq p_{0j}$ for all J_j in H_2, then the optimal solution can be found by ordering the jobs in each group by their second-stage processing times, longest processing time first, and optimally interleaving the two sequences.

Proof. This is true because Corollary 4.2 holds for both groups. QED.

Theorem 4.4. If, for some H_i, min $\{p_{0j} : J_j \in H_1 \cup H_2\} \geq$ max $\{p_{ij} : J_j \in H_i\}$, then the optimal schedule can be found by sequencing the jobs in each group by Johnson's Rule and optimally interleaving these sequences.

Proof. By the given, we know that the conditions of Corollary 4.2 hold for the jobs in H_i. Thus, the jobs in this group should be ordered by longest second-stage task processing time first. For these jobs, this sequence is the same as the sequence given by Johnson's rule. Now, if we can determine the optimal ordering of the jobs in H_k, the other group, we can interleave the groups to derive an optimal schedule. We will show that these jobs should be ordered by Johnson's rule.

Consider an optimally-interleaved schedule where J_h is processed immediately before J_j on M_k, and where the jobs are not ordered by Johnson's rule. That is, min $\{p_{0j}, p_{kh}\}$ < min $\{p_{0h}, p_{kj}\}$.

If there are no jobs from H_i between J_h and J_j on M_0, then Property 4.2 implies that the jobs should be interchanged.

Else, let J_m be the job from H_i processed just before J_j. If J_j is not the last job on M_0, then we can move J_j to immediately after J_h. Because the jobs that were between J_h and J_j have short second-stage tasks, delaying these jobs does not delay the processing of any successive jobs. Now J_h and J_j are consecutively processed on M_0, and Property 4.2 implies that the jobs should be interchanged.

Finally, if J_j is the last job on M_0 (and M_k), then the fact that the schedule is optimally interleaved implies that $p_{kj} \leq p_{im}$, since J_m is the last job on M_i. By the given, p_{im} is less than or equal to p_{0h} and p_{0j}. Thus, min $\{p_{0j}, p_{kh}\}$ < min $\{p_{0h}, p_{kj}\}$ implies that p_{kh} < min $\{p_{0h}, p_{kj}\}$ (since $p_{kj} \leq p_{0j}$) and consequently $p_{kh} < p_{0h}$ and $p_{kh} < p_{kj}$. Property 4.3 implies that J_j should precede J_h.

We have thus shown that the jobs in H_k should be ordered by Johnson's rule. This gives us an optimal ordering for H_k which can be interleaved with H_i to solve the problem. QED.

4.2.6 Branch-and-Bound Algorithm

In this section we will identify three lower bounds on the makespan; we will take the maximum of these to form our overall lower bound. We will also discuss the use of a branch-and-bound technique for finding optimal solutions for the 3MLA-MS problem. We will use the overall lower bound in the analysis of the worst-case performance of an approximation algorithm (Section 4.2.9). We will begin by discussing the three component lower bounds.

For the first component bound, consider only the jobs in H_1. Order these jobs using Johnson's rule and determine for each J_j a completion time C_{1j} on M_1. Then, $LB_1 = \max \{C_{1j}\}$. Similarly, $LB_2 = \max \{C_{2j}\}$.

The third component bound takes into account that all of the jobs use M_0. We relax the problem by dropping the second-stage assignments of the jobs and allowing an infinite number of second-stage machines. In fact, however, we only need n second-stage machines, one for each job. If each job has a separate second-stage machine, then the minimal makespan, which will be a lower bound on the optimal makespan for the original problem, is the maximum sum of first-stage completion time plus second-stage task processing time. By an argument similar to that of Section 1.4, we can find the minimal makespan by sequencing the jobs by their second-stage task processing times, longest first. We schedule the jobs in this order on M_0. The second-stage completion time of any job J_j in H_i is $C_{0j} + p_{ij}$. Therefore, $LB_3 = \max \{C_{0j} + p_{ij}\}$. Note that LB_3 will be greater than $\sum p_{0k}$, since there exists some job J_j that has $C_{0j} = \sum p_{0k}$.

Our lower bound for a given problem (hereafter referred to by the variable LB) will be the maximum of these three lower bounds for the makespan: (i) LB_1, the minimum possible makespan of the jobs in set H_1, (ii) LB_2, the makespan of the jobs in set H_2, and (iii) LB_3, the minimal makespan if there exist an infinite number of second-stage machines. That is, $LB = \max \{LB_1, LB_2, LB_3\}$.

In a standard branch-and-bound algorithm, each node will consist of a partial schedule of jobs. From a node, we exclude certain branches using the above dominance properties and create a lower bound using straight-forward extensions of the above lower bounds.

In order to help prune branches from the search tree, we will use the optimal properties that we developed in Section 4.2.5. We will also use a dominance property (Property 4.4) that is more dependent upon the current makespan of each machine (where t_i is the makespan of M_i).

Property 4.4. For a given partial schedule, if J_j is unscheduled and in H_k (where H_m is the other group), $t_k \geq \sum p_{0i}$ (the sum over all jobs), and $t_m < \sum p_{0i}$, then J_j should not precede any job from H_m.

It is easy to show that if, in any schedule constructed from this partial schedule, J_j does precede any of the jobs from H_m, we can move J_j to the last position without increasing the makespan.

We used the branch-and-bound algorithm to solve a number of problems. On the set of 15-job problems, the running time was usually less than one second (on a 386 PC), although it was much greater for two problems, one of which had a lower bound that was not tight. See Table 4.1.

Table 4.1. Statistics on branch-and-bound procedure for LA154.

Problem	Lower Bound	Optimal	Time	Nodes
1	2942	2942	0.11	38
2	2080	2080	0.11	39
3	3138	3138	0.16	47
4	3122	3122	0.11	37
5	1956	1956	0.16	56
6	2034	2034	0.22	51
7	2187	2187	0.17	57
8	1919	1919	0.17	71
9	1911	1911	73.92	61916
10	1945	1956	637.30	546654

Note: Time measured in seconds.

4.2.7 Heuristics

Since 3MLA-MS is a strongly NP-complete problem and the branch-and-bound procedure may occasionally take too long to find the optimal solution, the use of approximation algorithms is a preferable alternative. Our discussion so far has led us one very natural heuristic, the interleaving of the Johnson sequences. We will also introduce another combination that will be of use later.

Johnson Interleaved. Order the jobs in each group using Johnson's rule. Optimally interleave them using Algorithm 4.1.

Merged Johnson. Order the jobs in each group using Johnson's rule. Select the group with the smallest total task processing time on M_0 (if the totals are equal, pick one arbitrarily). Start the schedule with all of the jobs from this group. Follow them with the jobs from the other group.

4.2.8 Empirical Results

In this section we report on the empirical testing of our heuristics. We discuss the problem sets generated, our methodology, and our results.

In order to study the scheduling of a bottleneck machine, consider one of the second-stage machines, say M_1, as a bottleneck operation. As a bottleneck, the workload of M_1 should be greater than that of M_0 or M_2. Therefore, we will construct problems where the total processing time on M_1 is likely to be the largest.

Three problem sets (LA154, LA304, LA504) were created using uniform distributions to generate processing times. The mean fraction of jobs in each group and the mean processing times were set such that M_1 would have more work to do than M_2. These problems had 15, 30, and 50 jobs. There were ten problems in each set.

Table 4.2. Characteristics of 3MLA-MS problem instances.

Set	Jobs	Average number of jobs in H_1	M_0 times (range)	M_1 times	M_2 times
LA304	30	15	20-60	40-120	30-90
LA154	15	5	80-160	240-480	120-200
LA504	50	37	30-60	40-80	60-120

Another problem set (LA201) was created based upon the problem structure used in the NP-completeness proof. Ten sets of 15 random integers ranging from 1 to 10 was created (with an adjustment to insure that the total processing time was divisible by 5), and a 20-job problem was created from each set of these a_j using the construction in Theorem 4.2. For this problem we knew a good lower bound, although we determined later that two of the embedded 3-Partition problems had no solution and thus the lower bound could not be achieved.

For each problem, our lower bound on the makespan was the maximum of the three component bounds (see Section 4.2.6). The best component bound depended upon which machine had the largest workload for any problem. Since M_1 generally had the most processing time, the M_0 - M_1 bound was usually greatest.

For the 15-job problems, we were able to calculate the value of the optimal solution. For the other uniform problems we were able to find heuristic solutions that achieved the lower bound, implying that our lower bounds were tight and that we had found optimal solutions.

On the 20-job hard problems, we could solve the imbedded 3-Partition problem. For 8 of the 10 problems, there does exist a partition and thus we know the optimal makespan. For the other two, we do not know the optimal, although we came very close with the heuristics.

The results of the Johnson Interleaved and Johnson Merged heuristics on minimizing the makespan are shown in Table 4.3. Performance is the relative deviation from the optimal (if known) or the lower bound. We observed that the Johnson Interleaved algorithm found near-optimal or optimal solutions for all of these problems. The performance of the Johnson Merged algorithm varied from good on the 30- and 50-job problems to poor on the 20-job problems. The Johnson Interleaved heuristic outperformed a number of other sequencing rules, and these results were consistent with testing on a number of other problem sets with different problem structures.

Table 4.3. Makespan summary table for 3MLA-MS problem. Peformance is relative deviation from optimal or lower bound.

Set	Johnson Interleaved	Johnson Merged
LA 154	0.50	2.44
LA 304	0.00	0.35
LA 504	0.00	0.40
LA 201	0.42	23.14

4.2.9 Heuristic Error Bounds

It is often possible to evaluate a heuristic by analyzing its worst-case behavior. Heuristics that minimize the makespan of scheduling problems are especially open for this kind of analysis due to the simplicity of the objective function. Since the Johnson Interleaved heuristic performed the best in our empirical study, we are most interested in determining the worst-case performance of this procedure.

We will begin our analysis with the Merged Johnson procedure. For this bound we will make use of the lower bounds derived in Section 4.2.6. Recall that our lower bound $LB = \max \{LB_1, LB_2, LB_3\}$.

Theorem 4.5. For a given instance of 3MLA-MS, the schedule created by the Merged Johnson heuristic has a makespan less than 3/2 *LB*.

Proof. Without loss of generality suppose that $\Sigma_{H1} p_{0j} \leq \Sigma_{H2} p_{0j}$. Now, $LB_3 > \Sigma p_{0j}$, the sum over all of the jobs, implying $\Sigma_{H1} p_{0j} < 1/2\ LB_3 \leq 1/2\ LB$. LB_k be the minimal makespan achievable by the jobs in H_k, found by scheduling the jobs using Johnson's rule. The makespan of the scheduled created using the Merged Johnson heuristic is max $\{LB_1, \Sigma_{H1} p_{0j} + LB_2\}$. This is less than max $\{LB, 1/2\ LB + LB\} = 3/2\ LB$. QED.

Now, let us consider the Johnson Interleaved heuristic. Since this algorithm combines the two Johnson sequences optimally, its performance cannot be worse than that of the Merged Johnson heuristic. Thus we can make the following statement.

Corollary 4.3. For a given instance of 3MLA-MS, the schedule created by the Johnson Interleaved heuristic has a makespan less than 3/2 *LB*.

Theorem 4.6. The maximum error of the Johnson Interleaved algorithm relative to the optimal makespan is one-half.

Proof. If C^* is the value of the optimal makespan and C_{JI} the makespan of the schedule created by the Johnson Interleaved algorithm, $LB \leq C^* \leq C_{JI} < 3/2\ LB \leq 3/2\ C^*$. Thus, $(C_{JI} - C^*) / C^* < 1/2$. QED.

Let us make a few observations about our algorithm. First, the error bound can be extended to the case where there are $m > 2$ groups (each group with a different flow to a different second-stage machine). The maximum relative error of the Johnson Interleaved algorithm is $1 - 1/m$ in this case.

Second, the Johnson Interleaved algorithm interleaves the groups by looking ahead to the future workload of the second-stage machines (the sum of the remaining second-stage task processing times).

Finally, our error bound of one-half is tight. In the following problem instance, the bound is achieved in the limit.

Example 4.2. Given n, let C^* be $2n + 1$. For H_1, construct jobs $J_1, \ldots, J_n$ with the following characteristics:

$i = 1, \ldots, n - 1$:	$p_{0i} = 1$	$p_{1i} = 1$
$i = n$:	$p_{0n} = 1$	$p_{1n} = n$.

Construct in H_2 jobs $J_{n+1}, \ldots, J_{2n}$ with similar characteristics:

$i = n + 1, \ldots, 2n - 1$:	$p_{0i} = 1$	$p_{1i} = 1$
$i = 2n$:	$p_{0,2n} = 1$	$p_{1,2n} = n$.

Now, since ties can be broken arbitrarily, each group is already ordered by Johnson's rule (we could force this ordering by subtracting a small amount from the appropriate first-stage tasks). An optimal schedule can be achieved by interleaving the sequences $[\, J_n J_1 \ldots J_{n-1} \,]$ and

[$J_{2n}\, J_{n+1} \ldots J_{2n-1}$]. The makespan of such a schedule is $C^* = 2n + 1$. The interleaving of the Johnson sequences yields a schedule with makespan $3n$, as does merging the Johnson sequences. As n goes to infinity, the ratio of each of these makespans to C^* goes to 1.5. See Figures 4.7, 4.8, 4.9 ($J_x = J_{10}$) for the optimal, interleaved, and merged schedules with $n = 5$.

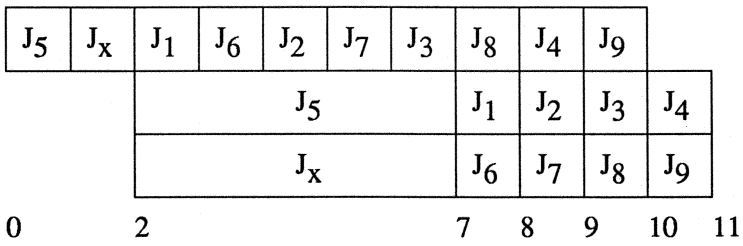

Figure 4.7. Optimal schedule (second-stage tasks started late).

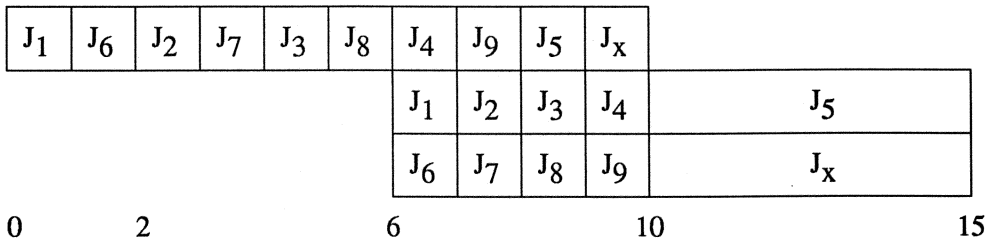

Figure 4.8. Interleaved Johnson schedule (second-stage tasks started late).

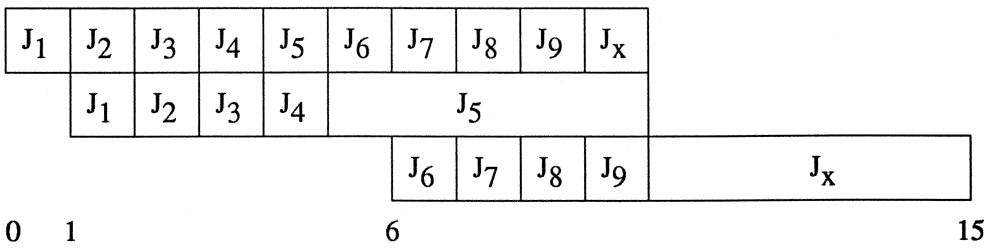

Figure 4.9. Merged Johnson schedule.

4.3 Minimizing the Total Flowtime

In this section we discuss the Three-Machine Look-Ahead problem with the objective of minimizing the total flowtime (3MLA-FT). We know that minimizing total flowtime in a two-machine flow shop is an strongly NP-complete problem (Garey, Johnson, and Sethi, 1976). Since

that problem is a special case of 3MLA-FT, 3MLA-FT must also be a strongly NP-complete problem. Hence, we will investigate optimality conditions, lower bounds, and heuristics. Since total flowtime is a regular objective function, only permutation schedules need be considered for optimality (see Section 4.2.3); thus, a sequence of jobs for M_0 corresponds to a unique schedule on all three machines. The problem notation is the same as that for the 3MLA-MS problem.

A number of researchers have studied the problem of minimizing the total flowtime in a flow shop. This work includes lower bounds, branch-and-bound algorithms, and heuristic approaches. See Ignall and Schrage (1965) for a branch-and-bound and Ahmadi and Bagchi (1990) for improved lower bounds. Szwarc (1983) studies special cases, and Van de Velde (1990) presents a Lagrangian relaxation. Krone and Steiglitz (1974), Kohler and Steiglitz (1975), Miyazaki, Nishiyama, and Hashimoto (1978), and Miyazaki and Nishiyama (1980) all present heuristic approaches. Ahmadi et al. (1989) includes batch processing.

4.3.1 Total Enumeration

Initially, we wanted the compare the difficulty of finding near-optimal solutions for 3MLA-FT to the difficulty of finding near-optimal solutions for 3MLA-MS, where we had very good results. One way in which we could compare the two problems was to compare the range of the objective functions over the domain of all permutation schedules.

For a nine-job problem, a total enumeration of the permutation schedules includes over 300,000 sequences. We picked an arbitrary instance, created each schedule, and measured the makespan and total flowtime of each schedule. This procedure yielded a distribution of makespans and total flowtimes over the set of sequences.

The primary result is that 18.5% of all sequences have makespans within 6.7% of the optimal makespan, but less than one quarter of one percent of all sequences have flowtimes within 6.7% of the optimal flowtime. (See Table 4.4 and Figure 4.10.) Because there are much fewer schedules that are near-optimal, we can infer that the problem of minimizing the total flowtime is a much harder problem than the problem of finding the minimum makespan. Similar

results hold for another nine-job problem and a fifteen-job problem where a random sample of the sequences were examined.

Table 4.4. Distribution of schedule makespans and flowtimes.

Deviation from optimal makespan	Percent of schedules	Deviation from optimal flowtime	Percent of schedules
3.3 %	5.65 %	3.7 %	0.05 %
6.7	18.47	6.7	0.23
10.0	30.13	14.2	2.05
13.3	49.07	21.6	6.85
16.7	62.17	29.1	15.83
20.0	75.41	36.6	29.18
23.3	86.24	44.0	45.37
26.7	92.31	51.5	62.31
30.0	96.43	59.0	77.28
33.3	98.10	66.4	88.65
40.0	100.00	73.9	95.49

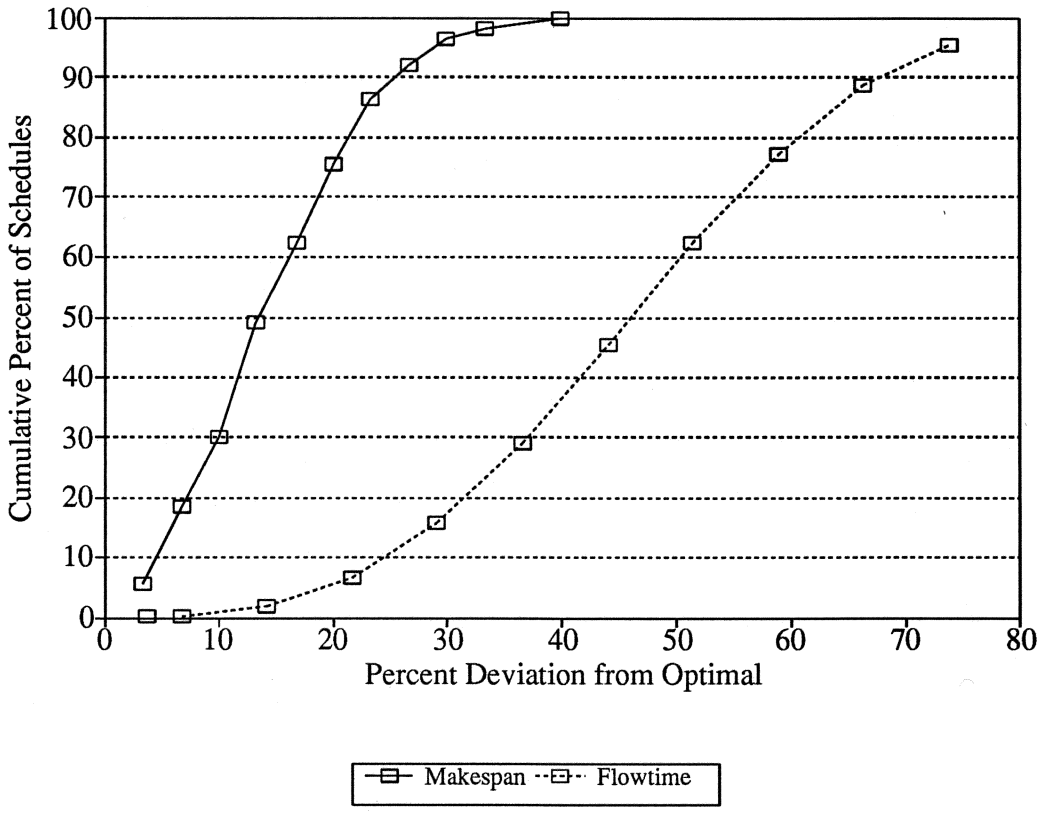

Figure 4.10. Deviation from optimal makespan and flowtime.

4.3.2 Lower Bounds

In this section we discuss lower bounds on the total flowtime. These lower bounds provide us with a way of determining the quality of the solutions produced by our heuristics and can be extended into lower bounds for a branch-and-bound algorithm.

The lower bounds are similar to those proposed in Ignall and Schrage (1965). The initial lower bound on the total flowtime is calculated by ordering the jobs by their processing times on M_0, shortest processing time first (SPT), and assuming that the second-stage machines have infinite capacity. That is, $LB_1 = \Sigma C_{0j} + \Sigma p_{1j} + \Sigma p_{2j}$.

We compute the second bound by considering each of the second-stage machines. Let a = min $\{p_{0j} : J_j \in H_1\}$ and b = min $\{p_{0j} : J_j \in H_2\}$. Now, a (b) is the earliest time that the second-stage tasks on M_1 (M_2) could start. Then, form two sequences σ_1 and σ_2 of the jobs in H_1 and H_2 respectively by ordering the jobs in each group by the second-stage task processing times, shortest processing time first. Schedule on M_1 the second-stage tasks of the jobs in H_1 in the order given by σ_1. The first task should begin at a, and each successive task should immediately follow (we do not schedule the first-stage tasks). This forms completion times C_{1j} for the jobs. Repeat for the second-stage tasks of the jobs in H_2 in order to calculate C_{2j}.

However, only one machine can start at its earliest time. The other must start at a time no less than $a + b$. Let r be the cardinality of H_1 and s the cardinality of H_2. Either the r tasks on M_1 will be delayed by b, or the s tasks on M_2 will be delayed by a. After calculating rb and sa, we can increase our lower bound by adding the smaller quantity. Thus, the second lower bound is $LB_2 = \Sigma C_{1j} + \Sigma C_{2j} + \min \{rb, sa\}$.

In Example 4.3, we calculate the lower bounds for the problem instance that we introduced in Example 4.1.

Example 4.3. Given the following five jobs in two groups:

J_j	H_i	p_{0j}	p_{ij}	t_j
J_1	H_1	2	4	9
J_2	H_1	1	1	1
J_3	H_1	2	4	5
J_4	H_2	2	1	4
J_5	H_2	1	3	3

Lower Bound 1: Order first-stage by SPT and add second-stage:

p_{0j}	1	1	2	2	2
C_{0j}	1	2	4	6	8

$\Sigma C_{0j} = 21.$ $\Sigma p_{1j} = 9.$ $\Sigma p_{2j} = 4.$ $LB_1 = 34.$

Lower Bound 2: Order each second-stage machine by SPT:

$a = 1.$ $r = 3.$ $b = 1.$ $s = 2.$

p_{1j}	1	4	4	p_{2j}	1	3
C_{1j}	2	6	10	C_{2j}	2	5

$\Sigma C_{1j} = 18.$ $\Sigma C_{2j} = 7.$ $\min\{rb, sa\} = 2.$ $LB_2 = 27.$

4.3.3 Special Case

This section discusses a special case that leads to easily-found optimal solutions.

Theorem 4.7. If $p_{1k} \leq p_{0k}$ for all J_k in H_1 and $p_{2j} \leq p_{0j}$ for all J_j in H_2, then any sequence σ in which the first-stage tasks are in SPT order forms an optimal schedule.

Proof. Suppose we have such a σ. Consider H_1. If J_i is the first job on M_1, then $C_{1i} = C_{0i} + p_{1i}$. If J_k is the next job from H_1 on M_1, then $p_{1i} \leq p_{0i} \leq p_{0k}$. Thus, the second-stage task of J_i completes before (or at the same time as) the first-stage task of J_k and does not delay the second-stage task of J_k. Thus, $C_{1k} = C_{0k} + p_{1k}$. Similarly, this equality is true for all J_k in H_1, and $C_{2j} = C_{0j} + p_{2j}$ for all J_j in H_2. The total flowtime is therefore $\Sigma C_j = \Sigma C_{0j} + \Sigma p_{1j} + \Sigma p_{2j}$. Since the first-stage tasks are in SPT order, this achieves the first lower bound, and this sequence is an optimal schedule. QED.

4.3.4 Empirical Testing

Empirical testing was performed using a number of different heuristics. The three heuristics that performed the best (and very similarly) are described below. The 3MLA-MS problem sets were used as a testbed. For each problem, the four lower bounds on the flowtime were calculated. The best of these was taken as the lower bound. For the fifteen-job problems, we were able to find the optimal solutions from the branch-and-bound algorithm. The performance of a heuristic on a problem was taken as the relative deviation from the optimal solution (if known) or the best lower bound.

SPT-look-ahead. Each group H_1 and H_2 is ordered by the first-stage task processing times (shortest first), forming two sequences. These sequences are interleaved by choosing at each step the first unscheduled job from one sequence or the other. Define at each step the following variables: t_0 is the completion time of the partial schedule on M_0 and t_1 is the completion time of the partial schedule on M_1. Let p_1 (p_2) be the processing time on M_0 of the next job from H_1 (H_2). Note that these are the shortest such task processing times from each set of jobs. If $t_1 - t_0 \leq p_1$, the work-in-process inventory (WIP) waiting at M_1 is low; schedule the job from H_1. If $t_1 - t_0 \geq p_1 + p_2$, then the WIP at M_1 is high; schedule the job from H_2. Else, $p_1 < t_1 - t_0 < p_1 + p_2$, and the WIP is intermediate; schedule the job with shortest first-stage processing time (p_1 or p_2).

WIP-look-ahead. Again, order the jobs in each group by the first-stage task processing times (shortest first). Combine the groups as in SPT-Look-ahead, except for one case: If $p_1 < t_1 - t_0 < p_1 + p_2$, schedule the job from H_1. Thus the work on M_1 is always used to determine which job to schedule.

Johnson-look-ahead. Sequence each group by Johnson's rule. Combine the groups as in SPT-Look-ahead, except for one case: If $p_1 < t_1 - t_0 < p_1 + p_2$, schedule the job from H_1. Thus the work on M_1 is always used to determine which job to schedule.

We tested the heuristics on the problem sets described in Section 1.7. For the fifteen-job problems, we were able to find the optimal solutions from a branch-and-bound algorithm. We

could not find optimal solutions for the larger problems. Therefore, in order to measure the heuristics, we computed a lower bound on the flowtime for each problem. We calculated the lower bounds described in Section 2.1 and took the largest as the lower bound. The performance of a heuristic on a problem was taken as the relative deviation from the optimal solution (if known) or the best lower bound. Due to the special structure of the problem, we could find improve the second lower bound for the instances in Set 4 by determining the optimal total flowtime for the five jobs in H_1.

Minimizing the total flowtime is harder than minimizing the makespan, and in Table 4.5 we report the results of the three heuristics described above. These heuristics were selected because they performed much better than a number of other procedures that combined the groups differently. The look-ahead heuristics found solutions with average total flowtime within eight percent of the optimal value. Because of the special structure of the 3MLA instances in Problem Set LA201, very good solutions were easier to find. The WIP-Look-ahead heuristic was slightly better on all of the other problem sets. However, the heuristics were very close to each other.

Table 4.5. Heuristic performance for the 3MLA-FT Problem.
Performance relative to optimal or lower bound.

Set	SPT Look-ahead	WIP Look-ahead	Johnson Look-ahead
LA 154	7.23	7.13	7.32
LA 304	5.41	5.16	6.15
LA 504	4.95	4.85	6.58
LA 201	1.37	1.37	1.37

4.4 Minimizing the Number of Tardy Jobs

4.4.1 Problem Introduction

The other objective function under consideration in look-ahead scheduling is the number of tardy jobs. This problem models a subproblem of the shop scheduling problem related to the

feeding of a bottleneck machine. The objective mirrors the management concern of customer satisfaction.

Let us call our problem the Three-Machine Number Tardy (3MNT) problem. Recall that only permutation schedules need be considered. Thus a sequence for all of the jobs defines a unique schedule. The problem notation is the same as that for 3MLA-MS, except that we have for each job J_j a due date d_j. Since 3MLA-MS is strongly NP-complete, 3MNT is also strongly NP-complete. Consider an instance where all of the jobs have the same due date: finding a schedule with no tardy jobs is equivalent to finding a schedule with makespan less than or equal to the common due date.

Since the problem is computationally difficult, we will examine a simple lower bound, a special case, and some heuristic approaches to finding solutions.

4.4.2 Lower Bound and Special Case

Good lower bounds are hard to find for the problem of minimizing the number of tardy jobs in a flow shop (Hariri and Potts, 1989). We will make use of a fairly simple one.

The lower bound makes use of the fact that the Moore-Hodgson algorithm will find the optimal number of tardy jobs for a one-machine problem. Given an instance of 3MNT, the due date of each job is adjusted by subtracting the processing time of the second-stage task. These adjusted due dates and the first-stage processing times form a one-machine problem for machine zero. The lower bound is calculated by using the Moore-Hodgson algorithm to optimally sequence these tasks. The number of tardy first-stage tasks is a lower bound on the minimum number of tardy jobs for the 3MNT instance.

This lower bound is achievable in the following special case:

Theorem 4.8. If $\min \{p_{0j} : J_j \in H_1\} \geq \max \{p_{1j} : J_j \in H_1\}$ and $\min \{p_{0j} : J_j \in H_2\} \geq \max \{p_{2j} : J_j \in H_2\}$, then an optimal sequence can be found be minimizing the number of first-stage tasks that are tardy to the adjusted due dates.

Proof. Because of the conditions above, the completion time of a job J_j in H_i is $C_j = C_{0j} + p_{ij}$. Thus, J_j is tardy if and only if $C_{0j} > d_j - p_{ij}$. QED.

We also tried lower bounds on each of the second-stage machines, but these were not as good. This seems reasonable if the interaction of the two flows is significantly contributing to tardiness.

4.4.3 Heuristics

Simple rules that can be extended for this problem include the Moore-Hodgson algorithm and Earliest Due Date (EDD). These can be expanded by including look-ahead ideas. We also developed a simple problem space genetic algorithm to find good solutions.

Look-ahead rules. The look-ahead extension of the Moore-Hodgson algorithm includes both machines. The jobs are ordered by their due dates and added to the schedule until a tardy job is found. The procedure then determines the critical path of tasks that determines the completion time of the tardy job. Then, each job in the path is evaluated to determine how much the completion time would decrease if that job were removed from the partial schedule. This calculation depends upon whether the job precedes, is, or follows the crossover job (the job whose first and second tasks are in the critical path).

We developed two look-ahead versions of the EDD rule. The first is similar to the Moore-Hodgson rule. The jobs are sequenced by their due dates. They are scheduled one at a time. When a job is tardy, however, we simply remove it to the end of the schedule. (In the Moore-Hodgson rule, we look for the job whose removal helps the most.) This form of the EDD is called EDD-No Tardy.

The other look-ahead version (EDD-Look-ahead) tries to schedule machine one (the bottleneck) carefully. The jobs in each group are ordered by their due dates. The primary idea is to get the jobs from group one to be on-time; group two jobs can be inserted if they don't interfere. At any point in constructing the schedule, we consider the next job from each group. If either would be tardy if scheduled next, we place it at the end of the schedule. Eventually, we get

a job from each group that would be on-time if scheduled next. We then determine if scheduling the group two job next would cause the group one job to be tardy. If not, we schedule this group two job and consider the next job from that group. Else we schedule the group one job next.

Genetic algorithm. After initial testing, we determined that the look-ahead Moore-Hodgson rule found consistently good solutions. Thus, we used a problem space genetic algorithm to adjust the problem data used by this rule so that we could find better solutions. The procedure is similar to those described in Chapter 3. We adjust the due dates using a steady-state genetic algorithm. The population size was 50, and the algorithm generated 1000 new individuals.

4.4.4 Results

In order to compare the heuristics we created a number of problem sets. Each set had 10 similar instances. Problems were created with 15, 30, and 50 jobs. The processing times were chosen randomly, and the due dates were chosen from a range that depended upon the sum of the processing times.

The look-ahead Moore-Hodgson heuristic performs better than any of the other rules (See Table 4.6). A number of rules that used the first-stage due date of a lot were also tested but did not perform as well. The genetic algorithm was able to find slightly better solutions than those found by the Moore-Hodgson rule.

Table 4.6. Summary table of results for 3MNT.

Set	Jobs	Lower Bound	EDD No Tardy	EDD	Moore Look-ahead	Genetic Algorithm
NT 151	15	5.8	7.8	7.3	7.0	6.5
NT 152	15	5.3	7.6	7.1	7.1	6.3
NT 301	30	9.0	12.7	12.5	12.0	10.6
NT 501	50	17.0	21.6	21.4	20.7	20.4

4.5 Application to Job Shop Scheduling

One of the primary motivations for studying these problems was to see if we could develop useful dispatching rules. After considering our results, it seemed clear that look-ahead and look-behind policies similar to those we used on the one-machine and three-machine problems would be intuitively good ways to dispatch jobs. However, we wanted to determine the tradeoffs of using such rules on a number of objectives. To this end, we created a job shop scheduling problem that modeled the semiconductor test area we were studying. This problem had 82 jobs and 23 machines and included various test operations. Processing time data were gathered from some historical lots. We scheduled the jobs under a number of dispatching rule combinations, using standard dispatching rules, look-ahead rules, and look-behind rules. We measured the schedules on four scales: total flowtime, makespan, number of tardy jobs, and total tardiness.

The bottleneck in this problem was a set of burn-in boards. Thus, we developed a look-ahead rule that orders the jobs waiting by their task processing times and uses information about the downstream bottleneck resource (the burn-in board availability) to determine which lot should be scheduled next. We also developed a look-behind rule that sequences lots by EDD and reserves burn-in boards for the next late lot that will be arriving soon. The effort of using these rules is slight for our problem; in a manufacturing environment, the dispatching effort might be more significant.

We used the following scheme in order to see how look-ahead and look-behind rules would influence schedule performance. We allocated the rules by dividing the machines into three areas: electrical test, burn-in, and other. In any given policy, all of the machines in the same area used the same dispatching rule. Our model included one burn-in workstation and 15 testers. We used seven standard rules: SPT, EDD, Slack per Remaining Operation, LPT, Modified Due Date, Earliest Finish Time, and First-In-First-Out in all areas.

For each of the standard rules, we created four policies. In the first, all of the areas used that rule. In the second, the testers used the look-ahead rule (since these were the machines

feeding the burn-in area). In the third, the burn-in area used the look-behind rule. In the fourth, the testers used the look-ahead rule while the burn-in area used the look-behind rule. This yielded 28 policies.

The average results over the seven standard rules are summarized in Table 4.7. We observe that the use of a look-behind rule, which is concerned with expediting late jobs, has a drastic effect on due date-related measures. It is able to reduce total tardiness while increasing the number of tardy jobs. This is a common tradeoff in scheduling problems. The look-ahead rule, which is concerned with avoiding unnecessary delays, reduces the total flowtime and makespan objectives. Our results are the consequence of the specific definitions of these look-ahead and look-behind rules. For other problems, alternative definitions may yield different results. While our results are not proof that look-ahead and look-behind rules are the answer to solving the job shop scheduling problem, the decreases in total flowtime (when the look-ahead rule was used) and total tardiness (with the look-behind rule) did encourage us to use them in our procedures to find good solutions (discussed in Chapter 5).

Table 4.7. Performance of 28 dispatching rule combinations.

	Flowtime	Makespan	Tardy	Tardiness
All policies[a]	2,179,344	88,677	17.1	192,394
Single rules[b]	2,168,014	89,324	16.6	249,968
With Look-ahead[b]	2,135,800	88,850	16.6	243,253
With Look-behind[b]	2,222,639	88,534	17.6	138,130
With both[b]	2,190,921	88,001	17.6	138,226

Notes: a: Average over all 28 policies.
b: Average over seven policies, one for each standard rule.

4.6 Chapter Summary

In this chapter we have examined a special case of the general three-machine flow shop. In this problem, the jobs to be scheduled form two classes with different groups, and we wish to minimize the makespan, the total flowtime, or the number of tardy jobs.

We proved that minimizing the makespan is a strongly NP-complete problem, and we also identified some properties of optimal solutions and some special cases that can be solved in polynomial time. We developed an approximation algorithm, Johnson Interleaved, that can find near-optimal solutions by looking ahead to the future workload of the second-stage machines. We showed that the worst-case error bound for this procedure is one-half and that this bound is tight in the limit. We also developed a branch-and-bound algorithm that can find exact solutions to the problem.

Then we described the problem of minimizing the total flowtime. We presented lower bounds, optimality conditions, and the results of testing on selected problem instances a number of heuristics that look ahead to current workload at the second-stage machines.

Finally we examined the problem of minimizing the number of tardy jobs. We discussed a simple lower bound, a special case that achieves this bound, and a number of simple and look-ahead heuristics. We also showed that a problem space genetic algorithm can find better solutions.

These results have two contributions. First is the analysis of these three-machine problems, problems previously unstudied in the literature. We conclude from the results of our empirical testing that look-ahead heuristics can find good solutions for the problems of minimizing total flowtime and minimizing the number of tardy jobs, and the interleaving procedure minimizes makespan.

Second, while these problems are important questions in their own right, they are also significant as subproblems in a job shop. It is possible to apply our results to the problem of job shop scheduling, either as part of a general scheduling procedure or as an attempt to schedule the bottleneck of a job shop more efficiently.