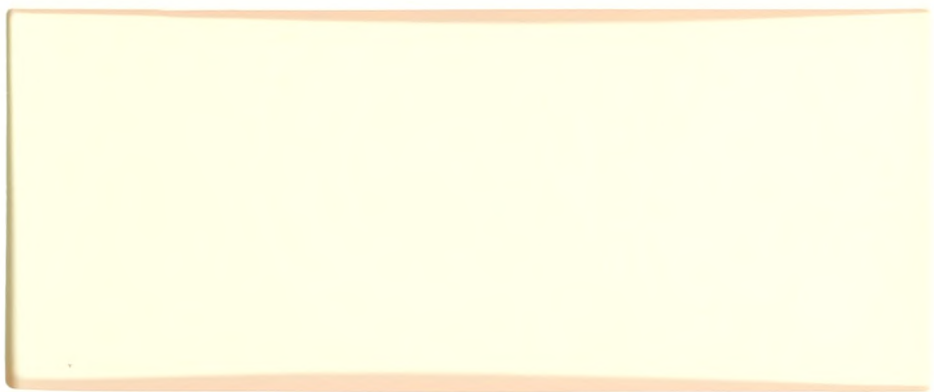
WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

Slack-based Lower Bound and Heuristic for Permutation Flowshop Models

Anantaram Balakrishnan,
Srimathy Gopalakrishnan,
and Atsushi Kurebayashi

Sloan WP# 3573–93 June, 1993

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139



Slack-based Lower Bound and Heuristic for Permutation Flowshop Models

Anantaram Balakrishnan,
Srimathy Gopalakrishnan,
and Atsushi Kurebayashi

Sloan WP# 3573–93 June, 1993

Slack-based Lower Bound and Heuristic for Permutation Flowshop Models[†]

Anantaram Balakrishnan
Sloan School of Management
M. I. T., Cambridge, MA 02139

Srimathy Gopalakrishnan
Operations Research Center
M. I. T., Cambridge, MA 02139

Atsushi Kurebayashi
Nippondenso Co. Ltd.
Kariya-city, Japan

ABSTRACT

This paper addresses optimal job sequencing decisions for various classes of permutation flowshops. We first describe a framework to classify flowshop scheduling problems based on the level of intermediate storage, job transfer mechanism, and objective function. We discuss the interrelationships between various flowshop models, develop a new slack-based lower bound for the total processing time of each machine, and describe an assignment-patching heuristic to generate effective job sequences. Our computational results show that, compared to previous approaches, the slack-based lower bound is effective when job processing times are random or have a trend, and the assignment-patching heuristic performs well for all models. By locally improving the heuristic solution using a two-exchange procedure, we are able to generate solutions that are within 10% from optimal for most scenarios.

[†] Supported in part by a research grant from MIT's *Leaders for Manufacturing* program

1. Introduction

Flowshop scheduling involves sequencing a given set of jobs (or products, items, batches) that follow the same processing route through successive stages (or operations, machines). The flowshop scheduling literature cites many manufacturing and service applications such as sequencing programs on a computer system, managing operations at chemical and other continuous processing plants, and assembling consumer durable products on an assembly line. Flowshops are inherently easier to manage compared to job shops; they offer advantages of lower inventories, quick response, and rapid feedback for process control and improvement Consequently, companies with complex manufacturing operations seek to streamline their product flows by creating "factories within the factory" (Skinner [1974]) using, say, group technology methods; this rationalization exercise often transforms a huge job shop into a network of embedded flowshop operations. The availability of flexible machines has accentuated this trend. For instance, several electronics companies now operate mixed-model assembly lines that can each assemble many different circuit boards in small lot sizes.

With more than two machines, the problem of minimizing the makespan for a given set of jobs in a permutation flowshop is known to be NP-complete (Lawler, Lenstra, and Rinnooy Kan [1982]). Consequently, research has focussed on determining good lower bounds, and developing exact (enumerative) and heuristic algorithms (see, for example, McMahon and Burton [1967], Campbell, Dudek and Smith [1970], Reddi and Ramamoorthy [1972], Baker [1975], Dannenbring [1977], Lageweg, Lenstra, and Rinnooy Kan [1978], King and Spachis [1980], Szwarc [1983], Nawaz, Enscore, and Ham [1983], Park, Pegden and Enscore [1984], Hundal and Rajagopal [1988], and Rajendran and Chaudhari [1990]). The vast majority of this literature deals with the problem of miniming makespan for a permutation flowshop with infinite intermediate storage capacity and unrestricted or free flow of jobs between machines (hence, machines are never blocked). We refer to this model as the "standard" flowshop

model. Other models covered in the literature include the no-wait case in which each job must be processed without waiting at intermediate operations (e.g., Reddi and Ramamoorthy [1972], Szwarc [1983]), and systems with limited or no intermediate storage between stages (e.g., Leisten [1990]). Some recent work has addressed optimization models for cyclic scheduling environments (e.g., McCormick et al. [1989]).

This paper develops and tests lower bounds and heuristics for several different flowshop scheduling models. We first describe (in Section 2) a scheme for classifying flowshop problems based on the level of intermediate storage capacity, type of job transfer mechanism, and optimization objective. We consider four different intermediate storage-job transfer combinations, and two objective functions–makespan and cycle time. To clarify the distinction between these eight models and explore their interrelationships, we examine their integer programming formulations. Section 3 develops a new slack-based lower bounding method that examines the slack or forced idle time of each machine to compute a lower bound on the minimum makespan or cycle time. Section 4 describes an assignment-patching heuristic algorithm, motivated by principles underlying no-wait scheduling, that solves a traveling salesman subproblem using estimates of the slack time between successive jobs. In Section 5, we apply the lower bounding and heuristic method to different flowshop models, and compare their performance with previous approaches (e.g., Lageweg et al.'s [1978] lower bounding method for the standard model). When the job processing times are random or have an increasing or decreasing trend, our slack-based lower bounding method performs well for all models; the bound is not as effective for problem instances with correlated processing times. The assignment-patching heuristic compares favorably with Campbell, Dudek and Smith's [1970] heuristic algorithm for all models and processing time distributions. Augmented with a local improvement procedure, this heuristic appears to be a versatile and effective method to generate near-optimal flowshop schedules for different contexts.

2. Permutation flowshop scheduling: Alternative models and their interrelationships

Given n available jobs and m machines, the flowshop scheduling problem involves sequencing the n jobs to optimize some criterion of interest. All jobs have the same processing sequence but different processing times, machines can process only one job at a time, and jobs cannot undergo simultaneous processing at multiple machines. In permutation flowshops, every machine processes the jobs in the same order. Thus, after we release jobs into the system in a certain sequence, they follow the FCFS discipline at all intermediate queues. We assume that the schedule does not contain any "unforced" idleness, i.e., machines are not idle unless they are "blocked" by downstream machines, or the upstream queue is empty (i.e., machines are "starved"), or timing constraints prevent commencing operations on the next job. Depending on the application context, the scheduling problem might include certain constraints on material storage and movement, and the optimization objective might vary. To distinguish between the common flowshop scheduling models discussed in the literature, we first describe a classification scheme that is analagous to Graham et al.'s (1979) framework to classify general machine scheduling problems based on the machine environment, job characteristics, and optimality criterion. We illustrate the differences between the various flowshop models in terms of their integer programming formulations, and relate their respective upper and lower bounds based on these formulation differences. We also introduce a new objective function, cycle time minimization, that applies to flowshops with cyclic scheduling policies.

2.1 Flowshop model classification

We classify flowshop scheduling models based on three problem characteristics: (1) intermediate storage capacity, (2) job transfer mechanism, and (3) optimization objective.

2.1.1 Intermediate storage capacity

Intermediate storage capacity refers to the maximum number of jobs that an intermediate queue between two successive stages can accommodate. The standard flowshop scheduling model (addressed by Johnson [1954] and others) ignores such queue limits, effectively assuming that intermediate storage capacity is *infinite* (or $\geq (n-1)$). We denote problems with unlimited intermediate storage capacity as **IIS** (infinite intermediate storage) problems. In some applications, upstream operations can be blocked due to inadequate downstream storage capacity, and so we must explicitly account for this capacity constraint in the scheduling model. For instance, the length of the conveyor segment connecting adjacent operations in an electronics assembly line limits the number of boards waiting to be processed at each station. We refer to tightly coupled flowshops with limited ($< (n-1)$) intermediate storage capacity as **FIS** (finite intermediate storage) systems. We distinguish the special case when all intermediate storage capacities are zero as the **NIS** (no intermediate storage) case. For the flowshop models that we consider in this paper, we can transform any FIS problem instance into an equivalent NIS problem instance by introducing dummy intermediate operations with zero processing times, one corresponding to each intermediate storage location. Therefore, in our subsequent discussions, we consider only two options for intermediate storage: IIS and NIS.

2.1.2 Job transfer mechanism

The second flowshop characteristic, *job transfer mechanism*, refers to the timing constraints associated with moving jobs to or from each machine. We consider three types of job transfer mechanisms: *free flow* (**FR**), *no wait* (**NW**), and *synchronized* (**SYN**). The *FR case* corresponds to unrestricted transfer of jobs from machines except due to downstream storage capacity constraints. When a machine finishes processing a job, we can immediately unload this job if the downstream queue is not full (or, in the NIS case, if the downstream machine is idle), and start processing the next available job in queue; otherwise, the job waits in the machine until the downstream machine completes its processing. The FR discipline often

applies to flowshops with asynchronous materials handling systems that use pallets to transfer products between machines. The standard flowshop scheduling model assumes an IIS system with FR job transfers.

The *NW case* represents the most restrictive job transfer mechanism. Under this scheme, once a job is started (i.e., released for the first processing step) it must be completed without any intermediate waiting (at intermediate machines or in storage); hence, NW systems do not have intermediate storage capacity. Hot rolling operations provide one example of NW scheduling. In this context, the flowshop is a series of hot rolling mills connected by conveyors, and each incoming job or workpiece is a solid, rectangular metal ingot. The successive rolling mills progressively reduce the thickness and elongate the workpiece; the finished products are plates or sheets with varying thicknesses and lengths. The processing time at each rolling mill varies by job, depending on the starting and finished dimensions of the workpiece. Each ingot is preheated before it is released for processing; since maintaining the elevated temperature is critical for hot rolling, the workpiece must proceed from one rolling mill to the next without any interruptions. Consequently, the release of ingots to the line must be staggered ensure NW processing.

The third job transfer mechanism, *synchronized flow*, is common in assembly systems that use a single conveyor belt with equally pitched fixtures to transport workpieces from one station to the next. In the SYN system, the process start times (and job unloading times) are coordinated across machines, i.e., all machines start processing the next available job in queue at the same instant of time. When a machine completes its operations on a job, it holds this job until all other machines finish processing their current jobs; the conveyor then moves forward to unload the current job, and load the job (if available) in the next position on the conveyor. Since having infinite storage capacity at any stage effectively decouples the line at that stage, we consider only the NIS case (and hence the FIS case using our transformation) with SYN flow.

2.1.3 Optimization objective

The standard flowshop scheduling model seeks the job sequence that minimizes *makespan*, i.e., the completion time of the last job in the sequence on the last machine. Some researchers have considered alternative performance measures such as minimizing the weighted sum of job completion times, or reducing lateness or tardiness (with respect to given job due dates).

For periodic or cyclic scheduling environments that repeat the same schedule at regular intervals, the *interval time* for each machine and the *cycle time* for each batch of n jobs are important performance metrics. We define the *interval time* of a machine as the elapsed time between the start of the first job and the completion of the last job on that machine. Thus, interval time is the sum of job processing times on the machine plus any forced idle time due to machine starvation, blocking, or timing constraints. The *cycle time* for a group of n jobs is the maximum interval time over all machines. To achieve regular (repeating) processing schedules, the length of the scheduling cycle (i.e., the time between the release of successive batches) must be greater than or equal to the cycle time. Another objective function of interest is the *total interval time* of all machines. This objective is relevant when the cost of operating a machine is proportional to the length of its interval time, for example, if operators can be reassigned to other machines after a machine completes all its processing in a cycle. In our subsequent discussions, we focus on the minimum makespan and cycle time objectives which we denote as **M** and **C**, respectively. Our observations concerning cycle time also apply to the total interval time objective.

We can use these three problem characteristics–intermediate storage capacity, job transfer mechanism, and optimization objective–to distinguish between various flowshop scheduling contexts. Thus, NIS/FR/M refers to the problem of minimizing makespan in a flowshop with no intermediate storage but free flow job transfers. As we have indicated, not all combinations

of options are compatible or practical; for instance, IIS/NW/M is not meaningful since having infinite intermediate storage is unnecessary when jobs are not permitted to wait. Similarly, we ingore the IIS/SYN combination. We will consider four combinations of intermediate storage and job transfer characteristics–IIS/FR, NIS/FR, NIS/NW, and NIS/SYN–with each of the two objective functions M and C. For the NIS/SYN/C model, we consider only a single scheduling cycle, i.e., we ignore the synchronization requirements to dovetail adjacent cycles.

This classification scheme covers many of the flowshop scheduling models discussed in the literature, but is not necessarily exhaustive. For instance, we have not considered systems with different batch sizes for processing and intermachine transfers, or batch processing stations (e.g., heat treatment ovens) that can simultaneously process multiple jobs.

2.2 Formulations and bounds for different models

In this section, we first present a mixed integer formulation for the standard model, and discuss enhancements needed to incorporate intermediate storage capacity limitations (i.e., NIS), NW and SYN flow, and cycle time minimization. We are given n jobs, j = 1, ... n, that require processing at machines k = 1, ..., m in sequence. Job j requires a processing time of P_{jk} at machine k, for all j = 1, ..., n, and k = 1, ...,m. The integer programming formulation for the standard flowshop scheduling problem uses the following three sets of decision variables:

X_{ij} = $\begin{cases} 1 & \text{if we assign job j to the } i^{th} \text{ position in the sequence, and} \\ 0 & \text{otherwise;} \end{cases}$

T_{ik} = starting time of the i^{th} job in the sequence on machine k; and,

S_{ik} = processing time of the i^{th} job in the sequence on machine k.

2.2.1 Minimum makespan models

We first formulate the standard model, and then indicate how the other three minimum makespan models differ from the standard formulation.

The Standard model: IIS/FR/M

$$\text{minimize} \quad T_{nm} + S_{nm} \tag{2.1}$$

subject to:

Sequencing constraints:

$$\sum_{i=1}^{n} X_{ij} = 1 \qquad \text{for all } j = 1, \ldots, n, \tag{2.2}$$

$$\sum_{j=1}^{n} X_{ij} = 1 \qquad \text{for all } i = 1, \ldots, n, \tag{2.3}$$

Processing time equations:

$$S_{ik} = \sum_{j=1}^{n} P_{jk} X_{ij} \qquad \text{for all } k = 1, \ldots, m, \; i = 1, \ldots, n, \tag{2.4}$$

Job starting time constraints:

$$T_{i(k+1)} \geq T_{ik} + S_{ik} \qquad \text{for all } k = 1, \ldots, m-1, \; i = 1, \ldots, n. \tag{2.5}$$

Machine starting time constraints:

$$T_{(i+1)k} \geq T_{ik} + S_{ik} \qquad \text{for all } k = 1, \ldots, m, \; i = 1, \ldots, n-1, \tag{2.6}$$

Non-negativity and integrality constraints:

$$X_{ij} \in \{0,1\} \qquad \text{for all } i, j = 1, \ldots, n, \text{ and} \tag{2.7a}$$

$$T_{ik} \geq 0 \qquad \text{for all } i = 1, \ldots, n, \; k = 1, \ldots, m. \tag{2.7b}$$

The objective function (2.1) minimizes the makespan, which is the completion time for the n^{th} job in sequence on the last machine. The sequencing constraints (2.2) and (2.3) ensure that every job is assigned to one position in the sequence, and each position contains only one job. Equations (2.4) determine the appropriate processing time for each job in the chosen sequence. If we assign job j to the i^{th} position in the sequence, then $X_{ij} = 1$ and equation (2.4) sets S_{ik} equal to the processing time of job j on machine k. Constraint (2.5) specifies that the downstream machine (k+1) can start processing the i^{th} job only after machine k completes this job, while constraint (2.6) ensures that machine k starts processing the $(i+1)^{st}$ job only after it finishes the i^{th} job.

The NIS models:

With infinite intermediate storage and free flow, the i^{th} job is unloaded from machine k as soon as its operations at that machine are completed. Therefore, the $(i+1)^{st}$ job can start on machine k immediately after the i^{th} job is completed; constraint (2.6) expresses this condition. However, for the NIS case, since the system does not have storage capacity between machines, the i^{th} must wait in machine k until machine (k+1) completes the $(i-1)^{st}$ job; machines (k+1) and k can then start processing the i^{th} and $(i+1)^{st}$ jobs. The following machine blocking constraint imposes this requirement.

Blocking constraints:

$$T_{(i+1)k} \geq T_{i(k+1)} \qquad \text{for all } k = 1, \ldots, m-1,\ i = 1, \ldots, n-1. \quad (2.8)$$

Adding this set of constraints to the IIS/FR/M formulation gives the formulation for the **NIS/FR/M** model. The NIS/NW/M and NIS/SYN/M models that we discuss next also contain these constraints.

No-wait and Synchronized flow models with NIS:

In addition to the blocking constraints (2.8), the no-wait model **NIS/NW/M** requires the starting time for the i^{th} job on machine (k+1) to be equal to its completion time on the previous machine, i.e.,

No-wait constraints:

$$T_{i(k+1)} = T_{ik} + S_{ik} \qquad \text{for all } k = 1, \ldots, m-1,\ i = 1, \ldots, n-1 \quad (2.9)$$

For the **NIS/SYN/M** model, the job starting times must satisfy both the blocking constraints (2.8) and the following synchronization constraints:

Synchronization constraints:

$$T_{(i+1)k} = T_{i(k+1)} \qquad \text{for all } k = 1, \ldots, m-1,\ i = 1, \ldots, n-1. \quad (2.10)$$

Constraints (2.10) require adjacent machines to start processing consecutive jobs (since the system has no intermediate storage) at the same time.

To summarize, the standard IIS/FR/M model is the core formulation to which we add various constraints on the job starting times in order to model the NIS/FR/M, NIS/NW/M, and NIS/SYN/M flowshop scheduling problems. These latter three models all require the blocking constraints (2.8). In addition, the NIS/NW/M model requires the no-wait constraints (2.9), while the NIS/SYN/M model contains the synchronization constraints (2.10). Note that the minimum makespan models do not require an explicit constraint to enforce the "no unforced idleness" condition since the problem always has an optimal solution satisfying this condition. However, as we discuss later, certain minimum cycle time models require a separate constraint for this purpose.

Lower bounds:

IIS/FR/M is a relaxation of the remaining three NIS models (all the NIS models contain the additional blocking constraints (2.8)), and NIS/FR/M is a relaxation of NIS/NW/M and NIS/SYN/M. Therefore, for a given set of problem parameters (number of machines and jobs, processing time on each machine),

(a) the optimal value (makespan) of the IIS/FR/M model or any lower bound on this value is a valid lower bound on the optimal values for all three NIS models; and,

(b) the optimal value of the NIS/FR/M model or its lower bound is a valid lower bound on the optimal values of the NIS/NW/M and NIS/SYN/M models.

In Section 3, we describe a slack-based lower bound for the IIS/FR/M model which, by observation (a), also underestimates the optimal makespan for the NIS models. We subsequently improve these latter lower bounds by accounting for the additional timing constraints in the NIS models.

Upper bounds:

Given any job sequence σ, we can choose appropriate job starting times that satisfy the timing constraints for each of the four models. We say that a schedule of starting times is σ-optimal for model P if it processes jobs in the given sequence σ, satisfies the model's timing constraints, and minimizes the objective value (makespan), where P is one of IIS/FR/M, NIS/FR/M, NIS/NW/M, or NIS/SYN/M. What is the relationship between the σ-optimal makespan values for the different models? We note that the σ-optimal schedules for the NW and SYN models can be adjusted to produce a feasible (but possibly not σ-optimal) for the NIS/FR model with equal or lower makespan; likewise, any σ-optimal NIS/FR schedule is feasible for the IIS/FR case. Therefore, the same relationship that we observed for the lower bounds also applies to the upper bounds. In particular, if M_P^σ denotes the σ-optimal makespan for model P, then:

$$M^\sigma_{IIS/FR/M} \leq M^\sigma_{NIS/FR/M} \leq \min \{M^\sigma_{NIS/NW/M}, M^\sigma_{NIS/SYN/M}\} \quad (2.11)$$

For the special case of two-machine flowshops, we can show that the NIS/FR/M, NIS/NW/M, and NIS/SYN/M models all have a common optimal schedule.

2.2.2 Minimum cycle time models

The integer programming formulations for the minimum cycle time models contain the previous sequencing and timing constraints (2.2) to (2.7), and the special constraints (2.8) to (2.10) for the NIS versions. In addition, the cycle time formulations require a new set of decision variables and constraints to model the cycle time objective, and two of the four models require special constraints to prevent unforced idleness.

To express cycle time in terms of the sequencing and timing decision variables, we define *interval time* variables I_k, for all k = 1, ..., m, and a variable C representing the *cycle time* of the schedule. Since the interval time for machine k is the difference between the completion

time for the last (i.e., n^{th}) job and the start time of the first job on that machine, we introduce the following defining equations:

Interval time equations:

$$I_k = T_{nk} + S_{nk} - T_{1k} \quad \text{for all } k = 1, ..., m. \tag{2.12}$$

Since cycle time is the maximum interval time I_k over all the machines k, we add the *cycle time constraints*:

$$C \geq I_k \quad \text{for all } k = 1, ..., m, \tag{2.13}$$

and the optimization objective is

$$\textbf{minimize } C. \tag{2.14}$$

For the NIS/SYN/C model, the synchronization constraints (2.10) ensure that machines do not have unforced idleness. For instance, the first and second jobs start processing on machines 2 and 1, respectively, at the same time. Similarly, in the no-wait case, machines are idle only when the input queue is empty. However, the IIS/FR/C and NIS/FR/C models require additional constraints to prevent machines being ready but idle when jobs are waiting in queue (otherwise, we can reduce the cycle time by postponing the first and subsequent jobs). Note that any schedule that does not have unforced idleness must process the first job in the sequence without intermediate waiting. Therefore, for the IIS/FR/C and NIS/FR/C models we add the following set of constraints to enforce the no-wait requirement for the first job.

No-wait processing for first job:

$$T_{1k} = T_{1,k-1} + S_{1k} \quad \text{for all } k = 2, ..., m. \tag{2.15}$$

These constraints are sufficient to ensure feasibility of the schedule, i.e., given any schedule that processes the first job without intermediate waiting, we can adjust the starting times for the remaining jobs to satisfy the no unforced idleness condition without increasing the cycle time.

Adding constraints (2.12), (2.13) and (2.15) to formulation (2.2) to (2.7) of the standard makespan model IIS/FR/M, and replacing the objective function (2.1) with (2.14) gives the

integer programming formulation for the IIS/FR/C model. The NIS/FR/C model has the additional blocking constraints (2.8); replacing constraints (2.15) with the no-wait constraints (2.9) or the synchronization constraints (2.10) in the NIS/FR/C formulation gives the NIS/NW/C or NIS/SYN/C formulations, respectively.

Lower bounds:

IIS/FR/C is a relaxation of the NIS/FR/C model, which in turn is a relaxation of NIS/NW/C (since constraints (2.15) are a subset of the no-wait constraints (2.9)). Therefore, the optimal value or a lower bound for IIS/FR/C is a valid lower bound for NIS/FR/C and NIS/NW/C. However, unlike the makespan models, NIS/SYN/C is "unrelated" to NIS/FR/C since these two formulations have one set of constraints each (constraints (2.10) and (2.15)) that do not belong to the other. A naive lower bound on the optimal cycle time of NIS/SYN/C is the maximum value, over all machines k, of the total processing time TP_k on machine k (this value also underestimates the minimum cycle times for the IIS/FR/C, NIS/FR/C, and NIS/NW/C models). We can possibly improve this bound by adding a lower bound on the machine idle times due to synchronization effects.

Upper bounds:

To relate the upper bounds for various models, we consider a specific job sequence σ, and compare the cycle time of the σ-optimal schedule (with respect to the cycle time criterion) for each model. For any given job sequence σ, the job starting times for the NIS/NW/C model's σ-optimal schedule is feasible for NIS/FR/C; likewise, the σ-optimal schedule for NIS/FR/C is feasible for IIS/FR/C. Therefore, if C_P^σ denotes the σ-optimal cycle time for model P, then:

$$C^{\sigma}_{\text{IIS/FR/C}} \leq C^{\sigma}_{\text{NIS/FR/C}} \leq C^{\sigma}_{\text{NIS/NW/C}}. \qquad (2.16)$$

The σ-optimal schedule for NIS/SYN/C might entail some intermediate waiting time for the first job (due to synchronization constraints) and, therefore, does not necessarily satisfy the no unforced idleness assumption for the NIS/FR/C model. However, we could develop an upper

bound for NIS/FR/C by adding to the NIS/SYN/C cycle time an estimate of the maximum time that the first job must wait at any machine to ensure synchronous operation.

In summary, the different flowshop scheduling models are closely related, and the bounds and solution principles for one model might apply to the other models as well. The next section develops a new slack-based lower bound for the IIS/FR model, and extends it to the NIS/NW and the NIS/FR cases. Section 4 proposes a common heuristic method.

3. Slack-based lower bound for Makespan and Cycle Time

Since the minimum makespan problem for flowshops with more than 3 machines is NP-complete, several authors have focussed on developing efficient procedures to estimate the smallest possible makespan for a given problem instance. We can then use these lower bounds in an enumeration procedure such as branch-and-bound (e.g., McMahon and Burton [1967]) to eliminate suboptimal job sequences or stop the search when the incumbent solution is close to optimal. Lageweg, Lenstra, and Rinnooy Kan [1978] describe an effective lower bounding procedure for the standard flowshop model. This method, which we will denote as the LLR method, solves several two-machine flowshop subproblems to generate the bound; it generalizes and outperforms previous lower bounds on the minimum makespan for the IIS/FR/M model. This section proposes a new *slack-based* procedure that provides a lower bound for both the makespan and cycle time minimization models.

3.1 Machine-based lower bounds for the IIS/FR/M model

The slack-based bound belongs to the class of *machine-based* bounds that focus on the total processing time of a particular machine. Machine-based bounds (e.g., Ignall and Schrage [1965], McMahon [1969]) underestimate the makespan by adding the following two terms to the total processing time TP_k on a machine k:

(i) *pre-k processing time*: the minimum time for the first job to complete processing on the first (k–1) machines and reach machine k, and

(ii) *post-k processing time*: the minimum time required to complete processing the last job on the remaining (m–k) machines after machine k.

The maximum value of this sum over all machines k gives a valid lower bound on the minimum makespan, i.e., we compute the machine-based lower bound $\underline{M}$ on the minimum makespan for the IIS/FR/M model as follows:

$$\underline{M} = \max_{k=1,\ldots,m} [\underline{M}(k)], \tag{3.1}$$

where $$\underline{M}(k) = TP_k + \min_{\substack{j,j'=1,\ldots,n \\ j \neq j'}} \{Q_{jk} + R_{j'k}\} \quad \text{for all } k = 1, \ldots, m, \tag{3.2}$$

$$TP_k = \sum_{j=1}^{n} P_{jk} \quad \text{for all } k = 1, \ldots, m, \tag{3.3}$$

$$Q_{jk} = \sum_{l=1}^{k-1} P_{jl}, \text{ and} \tag{3.4}$$

$$R_{jk} = \sum_{l=k+1}^{m} P_{j'l}. \tag{3.5}$$

$\underline{M}(k)$ is the makespan lower bound obtained by using machine k as the reference machine. TP_k is the total processing time on machine k, and Q_{jk} and $R_{j'k}$ correspond respectively to the pre-k processing time for job j and the post-k processing time for job j'. Jobs j and j' are candidate first and last jobs in the sequence. The minimum value of $\{Q_{jk} + R_{j'k}\}$ over all job pairs j and j' gives an underestimate of the total pre-k and post-k processing time in any feasible schedule.

The LLR method generalizes this machine-based bound by considering a block of intermediate machines, say, from machine k1 to machine k2, instead of a single reference machine k. Suppose we can evaluate a lower bound (or the optimal value) on the minimum makespan to complete all jobs on machines k1 to k2, assuming all jobs become simultaneously available (at time 0) at machine k1. Then, adding this makespan or its lower bound to the smallest possible pre-k_1 and post-k_2 processing times over all job pairs j and j' gives a lower

bound on the minimum makespan for the original m-machine problem. If L(k1,k2) denotes the lower bound (or optimal value) of the makespan for machines k1 to k2, the LLR method computes the overall lower bound for as follows:

$$\underline{M} = \max_{k1,k2=1,\ldots m,\ k2\geq k1} \underline{M}(k1,k2), \qquad (3.6)$$

where

$$\underline{M}(k1,k2) = L(k1,k2) + \min_{\substack{j,j'=1,\ldots,n \\ j\neq j'}} \{Q_{jk1}+R_{j'k2}\} \quad \text{for all } k1,k2 = 1, \ldots, m,\ k2 \geq k1. \qquad (3.7)$$

Let us discuss how to compute the k1-to-k2 makespan lower bound L(k1,k2). Note that:

(i) if k1 = k2 = k, then L(k1,k2) equals the total processing time TP_k on machine k, and the LLR bound is the same as the previous machine-based bound (3.2);

(ii) if k2 = (k1+1), i.e., k1 and k2 are adjacent machines, then Johnson's algorithm computes the minimum makespan L(k1,k2) for these two machines, and

(iii) if k2 > (k1+1), then the LLR method computes L(k1,k2) by treating all intermediate machines except machines k1 and k2 as non-bottleneck machines (i.e., these machines effectively introduce a delay, equal to the total intermediate processing time on machines (k1+1) to (k2–1), to transfer each job from machine k1 to machine k2). By appropriately modifying the job processing times, Johnson's algorithm determines the minimum makespan sequence for this two-machine flowshop relaxation with intermediate job transfer delays.

The LLR bound remains valid even if we consider only a subset of machine pairs k1, k2 instead of all m(m–1)/2 possible machine pairs. Since it considers a block of consecutive machines instead of a single reference machine, this bound is superior to the machine-based bound (3.1) but requires additional computational effort ($O(m^2)$ applications of Johnson's algorithm).

3.2 The slack-based bound for IIS/FR models

The LLR method has proven very effective in accelerating branch-and-bound solution procedures for the IIS/FR/M models (Lageweg et al. [1978]). However, because it uses Johnson's algorithm to solve the two-machine flowshop relaxation problems, the bound applies

only to the IIS/FR/M model. The slack-based bound that we describe next overcomes this difficulty, but the single machine version that we discuss might produce inferior bounds for the IIS/FR/M model.

The key observation that motivates the slack-based approach is that the machine-based bound remains valid even if we replace the total processing time TP_k in (3.2) with any valid lower bound on the *interval time* of machine k. The interval time I_k consists of two components: (i) the actual processing time TP_k for the n jobs on machine k, and (ii) the *slack* or "forced" idle time on machine k due to starvation, blocking, or job transfer timing constraints. If we can use the processing time information to determine a lower bound on the slack time, then we can strengthen the machine-based bound (3.2). Next, we discuss how to compute a lower bound for the slack time in an IIS/FR system. Since IIS/FR is the least restrictive intermediate storage and job transfer system , and since we focus on the interval time for a single machine, this lower bound applies to both the makespan and cycle time versions of the other model types as well (except the NIS/SYN/C model as we discussed in Section 2.2.2).

To gain insights about the relationship between machine slack time and job processing times, let us examine the Gantt chart shown in Figure 1 for two adjacent machines, machine (k–1) and machine k. Recall that S_{ik} denotes the processing time of the i^{th} job in sequence on machine k. Figure 1(a) shows the case when the h^{th} job starts processing on machine k immediately after it completes on machine (k–1); in Figure 1(b), the h^{th} job must wait since machine k is busy with prior jobs when machine (k–1) completes this job. Notice that in the IIS/FR system with no unforced idleness, the first job does not wait at any intermediate stage. We let δ_h^k denote the cumulative slack time between the first and h^{th} job on machine k, for all h = 2, ..., n, and all k = 2, ..., m. As Figure 1 shows, this cumulative slack must satisfy the following inequality:

$$\delta_h^k \geq \max\left[0, \left(\sum_{i=2}^{h} S_{i(k-1)} + \delta_h^{k-1}\right) - \sum_{i=1}^{h-1} S_{ik}\right], \tag{3.8}$$

$$= \max\left[0, \left(\sum_{i=1}^{h} S_{i(k-1)} - S_{1(k-1)} + \delta_h^{k-1}\right) - \left(\sum_{i=1}^{h} S_{ik} - S_{hk}\right)\right],$$

$$= \max\left[0, \Delta_h^k + \delta_h^{k-1}\right] \qquad \begin{array}{l}\text{for all } h = 2, \ldots, n, \\ \text{and all } k = 2, \ldots, m\end{array} \tag{3.9}$$

where $\Delta_h^k = \left(\sum_{i=1}^{h} S_{i(k-1)} - \sum_{i=1}^{h} S_{ik}\right) - \left(S_{1(k-1)} - S_{hk}\right).$ (3.10)

The parameter Δ_h^k is the difference in total processing times for the last (h–1) jobs on machine (k–1) and the first (h–1) jobs on machine k. Notice that for h = n, if choose two jobs, say, j and j' as the first and last jobs in the sequence, then (3.10) reduces to

$$\Delta_n^k(j,j') = \left(TP_{k-1} - TP_k\right) - \left(P_{j,k-1} - P_{j'k}\right), \tag{3.11}$$

i.e., we only need to choose the first and last jobs (and not the entire sequence of jobs) in order to calculate the parameter $\Delta_n^k(j,j')$.

Inequality (3.9) provides a lower bound on machine k's slack time δ_h^k in terms of the job processing times on machines (k–1) and k, and the slack time δ_h^{k-1} on machine (k-1). In turn, we can express δ_h^{k-1} in terms of the slack time δ_h^{k-2} on machine (k–2) and the processing times on machines (k-2) and (k-1), and so on. Making these substitutions in inequality (3.9) we get

$$\delta_h^k \geq \left[0, \Delta_h^k + \max\left\{0, \Delta_h^{k-1} + \max\left\{ \ldots + \max\left\{0, \Delta_h^2 + \delta_h^1\right\} \ldots\right\}\right\}\right]. \tag{3.12}$$

Thus, the calculation of slack time on the k^{th} machine takes into account the smallest slack times necessary for all previous machines.

Since the first machine does not have any slack time in an IIS/FR system, $\delta_1^h = 0$ for all h = 2, ..., n. And, for a given choice of the first and last jobs j and j' in the sequence, we can

compute $\Delta_n^k(j,j')$ using equation (3.11) for all machines k. Substituting these values in the right-hand side of equation (3.12) gives a lower bound, say, $\alpha_n^k(j,j')$ on $\delta_n^k(j,j')$, the total slack time on machine k if jobs j and j' are the first and last jobs in the sequence. The minimum value of $\alpha_n^k(j,j')$ over all the job pairs j and j' gives a lower bound on total slack time on machine k in any feasible IIS/FR schedule. Adding this lower bound to the total processing time TP_k gives a lower bound, say, $\underline{I}_k$ on machine k's *interval time* in any feasible schedule, i.e.,

$$\underline{I}_k = TP_k + \min_{\substack{j,j'=1,\ldots,n \\ j \neq j'}} \{\alpha_n^k(j,j')\} \qquad (3.13)$$

For the makespan minimizing model IIS/FR/M, we can further add the pre-k and post-k processing times as in (3.2) to get the slack-based lower bound on the optimal makespan. We summarize this slack-based bounding procedure below.

Slack-based bounding method for IIS/FR/M

For every machine k = 1, ..., m,

For every job pair j, j' = 1, ..., n, $j \neq j'$,

compute $\Delta_n^k(j,j')$ using equation (3.11);

compute $\alpha_n^k(j,j')$ by substituting $\Delta_n^k(j,j')$ values and $\delta_n^1 = 0$ in RHS of (3.12);

Compute slack-based lower bound $\underline{M}^{slack}(k)$ on the minimum makespan using machine k as the reference machine:

$$\underline{M}^{slack}(k) = TP_k + \min_{\substack{j,j'=1,\ldots,n \\ j \neq j'}} \{\alpha_n^k(j,j') + Q_{jk} + R_{j'k}\}. \qquad (3.14)$$

The slack-based bound on the optimal value of the IIS/FR/M model is:

$$\underline{M}^{slack} = \max_{k=1,\ldots,m} [\underline{M}^{slack}(k)]. \qquad (3.15)$$

Since the slack-based bounding method adds the non-negative term $\alpha_n^k(j,j')$ to TP_k, it produces a lower bound that is at least as tight as the machine-based lower bound (3.1). Observe from (3.12) and the definition of Δ_h^k (equation (3.10)) that the slack-based lower bound $\underline{M}^{slack}(k)$ depends on the differences in processing times at successive machines prior to

machine k. We, therefore, expect the slack-based bound to outperform the machine-based bound when job processing times have a negative trend (i.e., processing times decrease downstream). Our computational results, reported in Section 5, confirm this behavior.

Finally, as we indicated previously, the slack-based method provides bounds on the minimum cycle time as well. Since $\{\alpha_n^k(j,j') + TP_k\}$ is an underestimate of the interval time on machine k in any IIS/FR schedule that processes job j first and job j' last, and since cycle time is the maximum interval time over all machines, the following expression provides a slack-based *lower bound on the minimum cycle time*:

$$\underline{C}^{slack} = \max_{k=1,\ldots,m} \left\{ \min_{\substack{j,j'=1,\ldots,n \\ j \neq j'}} \left[TP_k + \alpha_n^k(j,j') \right] \right\} \qquad (3.16)$$

3.3 Improved Slack-based lower bound for NIS models

Although the slack-based bounds of Section 3.2 also apply to NIS models, we can exploit the special structure of the NIS system to further strengthen these bounds. Recall that for the IIS/FR case we set the total slack time δ_h^1 for the first machine to 0 for all h = 2, ..., n. With no intermediate storage, however, jobs must wait in the first machine (or we must delay their release) until the second machine becomes available, and so δ_h^1 might be positive. If we can determine a lower bound on this value then we can add it to the right-hand side of inequality (3.12), thus increasing $\alpha_n^k(j,j')$ and the overall makespan or cycle time lower bound. We first discuss a way to compute a lower bound on δ_n^1 for NIS/NW systems, and subsequently show how to adapt this method to the NIS/FR model.

Reddi and Ramamoorthy (1972) have shown that the NIS/NW/M problem is equivalent to an asymmetric traveling salesman problem (TSP) defined over a network whose nodes correspond to jobs. The distance from node i to node j in this network represents the necessary

delay L_{ij} in releasing job j (to ensure that it does not wait at intermediate stages) when it immediately follows job i in the sequence. We compute L_{ij} as follows:

$$L_{ij} = \max_{1 \le g \le m-1} \left[\sum_{k=1}^{g} (P_{i(k+1)} - P_{jk}),\ 0 \right], \qquad \text{for all } i,j = 1, \ldots, n,\ i \neq j. \tag{3.17}$$

To develop a lower bound on the first machine's total slack time for **NIS/NW** without actually sequencing the jobs, we use the following principle.

Consider a given choice of first and last jobs j and j'. For each job $i \neq j, j'$, let λ_i be the smallest delay that any successor job will experience, i.e.,

$$\lambda_i = \underset{l=1, \ldots, n,\ l \neq i,j}{\textbf{minimum}} L_{il} \qquad \text{for all } i = 1, \ldots, n,\ i \neq j, j'. \tag{3.18a}$$

For the first job j, we define

$$\lambda_j = \underset{l=1, \ldots, n,\ l \neq j,j'}{\textbf{minimum}} L_{jl}\ . \tag{3.18b}$$

Then, if jobs j and j' are scheduled first and last in the sequence, the first machine must have a total slack of at least

$$\delta_n^1(j,j') = \sum_{i=1}^{n, i \neq j'} \lambda_i \tag{3.19}$$

to ensure NW job transfers. Therefore, we substitute $\delta_n^1(j,j')$ in place of δ_n^1 in the right-hand side of (3.12) to compute the lower bound $\alpha_n^k(j,j')$ on the slack time in machine k; using this improved value of $\alpha_n^k(j,j')$ in (3.14) and (3.16) gives tighter lower bounds on the optimal values of the no-wait models NIS/NW/M and NIS/NW/C, respectively. Szwarc (1983) has proposed an alternate lower bound for the NIS/NW/M problem based upon the optimal values of a series of two-machine NIS/NW/M subproblems (solved using Gilmore-Gomory's [1964] algorithm), analagous to the LLR bound for the IIS/FR/M model. In Section 5, we compare the performance of the slack-based bound for NIS/NW/M with the Szwarc bound for our test problems.

For the **NIS/FR** case, jobs will still experience a delay at the first machine due to blocking by machine 2. However, because the job transfer mechanism is free flow rather than no-wait, the effect of downstream machines on the release time delay is difficult to incorporate. We, therefore, consider only the first two machines in defining the i-to-j delay parameters L_{ij} in equation (3.17), i.e., we set

$$L_{ij} = \max \{0, P_{i2} - P_{j1}\} \qquad \text{for all } i,j = 1, \ldots, n, i \neq j. \qquad (3.20)$$

Since the NIS system does not have any storage capacity between machines 1 and 2, job j must wait at least L_{ij} time units if it follows job i. As in the NIS/NW case, we use these L_{ij} values to compute λ_i (using equations (3.18)) and $\delta_n^1(j,j')$ (using equation (3.19)), thus improving the interval time lower bound $\underline{I}_k$.

4. Assignment-Patching heuristic for flowshop scheduling

The slack-based lower bounds provide benchmarks to evaluate the quality of heuristic solutions for various flowshop scheduling models. Instead of implementing different specialized heuristics for IIS/FR, NIS/FR, NIS/NW and NIS/SYN, we apply a single optimization-based method called the *assignment-patching heuristic* to generate a promising job sequence σ, and then evaluate the objective value for the σ-optimal schedule corresponding to the model of interest. Subsequently, we improve this solution using a local interchange procedure. The assignment-patching heuristic exploits the TSP structure of the NIS/NW/M and NIS/NW/C models. It uses slightly different arc length parameters for the makespan and cycle time objectives; we describe the makespan version first.

4.1 Assignment-Patching heuristic for minimum makespan models

Reddi and Ramamoorthy [1972] transformed the NIS/NW/M model into an equivalent directed traveling salesman problem defined over a network containing (n+1) nodes. Nodes 1, ..., n correspond to the n original jobs, and node (n+1) is a dummy job representing the starting

and completion of the schedule. The length of arc (i,j) for all i, j = 1, ..., n, i ≠ j, is the delay time L_{ij}, computed using (3.17). All the arcs (n+1,j) for j = 1, ..., n, emanating from the dummy node (n+1) have length zero. Arc (i,n+1), for all i = 1, ..., n, incident to node (n+1) has length equal to the post-1 processing time for job i, i.e.,

$$L_{(n+1),j} = 0, \qquad \text{for all } j = 1, ..., n, \text{ and} \qquad (4.1a)$$

$$L_{i,(n+1)} = \sum_{k=2}^{m} P_{ik} \qquad \text{for all } i = 1, ..., n. \qquad (4.1b)$$

Any hamiltonian tour on this network corresponds to a feasible job sequence. If the tour contains arcs (n+1,j) and (j',n+1) then jobs j and j' are the first and last jobs in the sequence, and if it contains arc (p,q), for p,q ≤ n, then job q follows job p. Since the makespan of any job sequence (for NIS/NW systems) is the interval time on machine 1 plus the processing time for the last job on machines 2 through n, the length of the corresponding traveling salesman tour plus the total processing time TP_1 for all jobs on machine 1 equals the makespan. Minimizing the length of the traveling salesman tour, therefore, solves the NIS/NW/M model optimally.

Instead of solving the equivalent TSP optimally, we apply the following *assignment-patching (AP) heuristic*. We can formulate the TSP as an integer program using the binary sequencing variables Y_{ij} for all i, j = 1, ..., n+1, i ≠ j. Y_{ij} is 1 if job i precedes job j, and 0 otherwise. The formulation has assignment constraints (to assign each node to one position in the sequence, and one node to each of the n positions) and subtour breaking constraints (Dantzig, Fulkerson, and Johnson [1954]). If we ignore the subtour breaking constraints, we get the following assignment relaxation of the TSP:

Assignment Relaxation of TSP:

$$\text{minimize} \quad \sum_{j=1}^{n+1} \sum_{j=1}^{n+1} L_{ij} Y_{ij} \qquad (4.2)$$

subject to

$$\sum_{i=1}^{n+1} Y_{ij} = 1 \qquad \text{for all } j = 1, \ldots, n+1, \tag{4.3}$$

$$\sum_{j=1}^{n+1} Y_{ij} = 1 \qquad \text{for all } i = 1, \ldots, n+1, \text{ and} \tag{4.4}$$

$$Y_{ij} \in \{0,1\} \qquad \text{for all } i, j = 1, \ldots, n+1, i \neq j. \tag{4.5}$$

If the assignment solution is a TSP tour, it provides an optimal NIS/NW/M sequence. Otherwise, we must patch together the subtours in the optimal assignment in order to construct a feasible job sequence. We use the following subtour patching method (adapted from Plante and Lowe [1987]).

Subtour patching heuristic:

Step 1: Pick two sub-tours T_1 and T_2 arbitrarily.

Step 2: For every pair of arcs $(i,j) \in T_1$ and $(i',j') \in T_2$, evaluate the incremental cost of replacing these two arcs with the arcs (i,i') and (j,j'). Select the pair of arcs with the minimum incremental cost, and perform the interchange.

Step 3: If the current solution has two or more subtours, repeat Step 1. Otherwise, stop.

The subtour patching heuristic requires at most O(n) iterations, and terminates with a feasible job sequence which we call the *AP makespan solution*. Since the assignment model (4.2) to (4.5) is a relaxation of the TSP representing the NIS/NW/M model, its optimal value is a valid lower bound on the optimal value of NIS/NW/M. In Section 5, we compare this *assignment* lower bound with our slack-based lower bound for our test problems.

4.2 AP heuristic for cycle time minimization models

For cycle time minimization, we examine sequences that reduce the interval time on each of the m machines. To minimize the interval time on any machine k, we must modify the previous TSP arc lengths to represent the delay between jobs i and j on machine k (instead of machine 1).

and set $L_{i,n+1}$ equal to the total processing time TP_k on machine k (instead of the post-1 processing time for job i).

Let us first describe how to compute the delay times for a downstream machine k. We define L_{ij}^k, for all i, j = 1, ..., n, as the necessary delay between jobs i and j on machine k to ensure no-wait processing. Using this notation, the value L_{ij}, computed using equation (3.17), corresponds to L_{ij}^1. Consider a sequence in which job j immediately follows job i, and suppose job i leaves machine 1 at time t. Then, in the no-wait system, machine k must complete processing job i at $\{t + \sum_{h=2}^{k} P_{ih}\}$. Furthermore, job j is released into the line at $\{t + L_{ij}^1\}$, and reaches machine k at $\{t + L_{ij}^1 + \sum_{h=1}^{k-1} P_{jh}\}$. Therefore, the i-to-j slack time L_{ij}^k, which is the time interval between the completion of job i and the arrival of job j at machine k, is:

$$L_{ij}^k = L_{ij}^1 + \sum_{h=1}^{k-1} P_{jh} - \sum_{h=2}^{k} P_{ih}. \qquad (4.6)$$

Similarly, we define

$$L_{i,n+1}^k = TP_k. \qquad (4.7)$$

The TSP with L_{ij}^k and $L_{i,n+1}^k$, for all i, j = 1, ..., n, as arc lengths models the problem of minimizing the interval time on machine k for the NIS/NW system. We solve this TSP approximately using the AP heuristic, for every machine k = 1, ..., m, and choose the best sequence, i.e., the sequence with the smallest cycle time, among the m TSP solutions as the *AP cycle time solution* .

4.3 Local Improvement procedure

The AP makespan or cycle time solution provides an initial candidate sequence σ for all the makespan or cycle time models. For a given model, the initial upper bound is the objective value of the corresponding σ-optimal schedule satisfying the model's intermediate storage and job transfer constraints. We then attempt to improve this upper bound by applying a *local*

exchange heuristic. Consider, for instance, the NIS/SYN/M model. Starting with the AP makespan sequence, for every pair of jobs i and j, we consider a new sequence σ' obtained by interchanging the positions of these two jobs in the current sequence, and compute the objective function value (makespan) for the σ'-optimal NIS/SYN schedule. If σ' has a lower makespan than the current sequence, then we perform this interchange and examine other job pairs. The method stops when no further improvement is possible; we refer to the final job sequence as the *Improved AP solution* .

5. Computational experience

5.1 Test problems

To test the lower bounds and heuristics, we generated random test problems in nine different sizes ranging from 6 jobs and 3 machines to 50 jobs and 3 machines. Table 1 shows the size (number of jobs and machines) for each problem class. We considered four different processing time distributions–random, correlated, trend, and correlated with trend. By *random* processing times we mean independent, identically distributed processing times P_{jk} for all j = 1, ..., n, and k = 1, ..., m. The processing times for a job are said to be *correlated* if they are consistently smaller or greater than average on all machines. On the other hand, job processing times exhibit a *trend* if they increase or decrease for all jobs as we progress down the line.

To generate problem instances corresponding to each of the four processing time distributions, we follow the method used by Lageweg et al. [1978] for their computational tests. Given the number of jobs and machines, our problem generator chooses an integer job processing time P_{jk} for each job j on every machine k from an uniform distribution. The parameters of this uniform distribution reflect the desired correlation and/or trend structure. Problems with *random* processing times have $P_{jk} \overset{iid}{\sim}$ Unif (1,100) for all jobs j and machines k. To introduce correlation in processing times, we generate n additional integers c_j, for all j = 1,

..., n, from the Unif (1, 4) distribution. For problems with *only correlation and no trend* in processing times, we randomly select P_{jk} from the Unif ($20c_j+1$, $20c_j+20$) distribution. Problems with *trend* (but no correlation) in the data are generated by selecting machine k's processing times P_{jk} from the Unif ($12.5(k-1)+1$, $12.5(k-1)+100$) distribution, for all k = 1, ..., m. This scheme increases the average processing time per job at successive stages, i.e., the processing times have an increasing trend. Finally, we construct problem instances with *both correlation and trend* by sampling from the Unif ($2.5(k-1)+20c_j$, $2.5(k-1)+20c_j+20$) distribution, for all j = 1, ..., n, and k = 1, ..., m.

For each problem size shown in Table 1, we generated four groups of three problem instances each; the groups correspond to the four processing time distributions–random, with correlation, with trend, and with correlation + trend. We also "inverted" each problem instance by renumbering machine k as machine m+1–k for k = 1, ..., m; thus, original problems with positive trend now have negative trend. To summarize, we generated 24 problem instances–12 original problems consisting of 3 random instances each for four processing time distributions, and 12 inverted problems–for each of the 9 problem sizes, and considered all 8 flowshop models corresponding to each of these 216 problem instances.

5.2 Performance comparisons

Our computational tests seek to address three questions: (i) how good are the slack-based bounds relative to previous lower bounding methods for makespan models; (ii) how does the assignment heuristic perform compared to previous heuristic methods; and, (iii) what is the quality of the heuristic solution obtained by locally improving the assignment-based job sequence? We elaborate on each of these issues next.

5.2.1 Comparison of lower bounds:

Recall that we have three versions of the slack-based bound for the minimum makespan problem: the basic version that provides a lower bound for the IIS/FR/M model, and enhanced versions for the NIS/FR/M and NIS/NW/M models. For the IIS/FR/M model, the LLR bound (described in Section 3.1) based on two-machine subproblem solutions has proven to be the most effective. This bound is also valid for the NIS/FR/M model. We, therefore, compare the slack-based (basic and enhanced) for the IIS/FR/M and NIS/FR/M model with the LLR bound. Note that, using both the slack-based and LLR methods, the lower bound for the NIS/SYN/M model is the same as the NIS/FR/M lower bound, and so we do not perform a separate comparison of bounds for the NIS/SYN/M case. For the NIS/NW/M model, we compare both the Szwarc [1983] lower bound and the optimal value of the assignment problem ((4.2) to (4.5)) with the enhanced slack-based bound. The LLR and Szwarc bounds do not apply to cycle time minimization problems, and so we evaluate the quality of the slack-based bounds based solely on its closeness to the heuristic value (see Section 5.2.3).

5.2.2 Comparison of heuristics:

To evaluate the effectiveness of the AP heuristic, we compare the objective values of the AP solution (before local improvement) and the solution produced by the CDS heuristic (Campbell, Dudek and Smith [1970]) for all model types. The CDS heuristic, developed for the IIS/FR/M model, solves $p \leq (m-1)$ auxiliary two-machine flowshop subproblems using Johnson's algorithm, and selects the best among the p job sequences. The total processing time on the first k machines and last k machines in the original problem serve as the job processing times on each of the two machines in the k^{th} auxiliary two-machine problem. For cycle time minimization models, we augment this method by applying an improvement procedure after sequencing the jobs using Johnson's algorithm.

5.2.3 Gap between upper and lower bounds:

To assess the quality of the improved AP solution (after local improvement), we compute the % gap between the objective value of this solution with the best lower bound (i.e., the better of the slack-based lower bound and the LLR or Szwarc lower bounds) for all 8 models. Small values of this % gap imply that both the lower bounding method and the heuristic procedure are effective.

5.3 Computational results

We implemented the problem generator and the lower bounding methods in FORTRAN on an IBM 4381 computer, and the heuristic procedures in C on a Macintosh computer. Tables 2, 3 and 4 summarize our computational results for the minimum makespan models; Tables 5 and 6 pertain to cycle time minimization models.

Table 2 compares the slack-based bound with the LLR bound for the IIS/FR/M and NIS/FR/M models, and the Szwarc and assignment lower bounds for the NIS/NW/M case. For the LLR bound, we followed Lageweg et al.'s [1978] strategy of solving (m–1) two-machine flowshop subproblems, for k2 = m and k1 = 1, ..., m–1 (see Section 3.1), and selecting the best lower bound. Table 2 shows the average values of the ratio of lower bounds (averaged over 54 problem instances) for each of the four processing time distributions. As expected, the slack-based lower bound performs well for problems with trend in processing times; it also appears to be as tight as the LLR and the Szwarc bounds for problems with random processing times. However, when job processing times are correlated, the difference in total processing times on adjacent machines is small, and hence the slack-based bound is not as effective. We observed that, even for problems with correlated data, the slack-based bound approaches the LLR bound when the number of jobs is very large. For the NIS/NW/M problem, the assignment lower bound (i.e., the optimal value of the assignment subproblem) outperforms the slack-based lower bound.

Tables 3 and 5 compare the performance of the AP heuristic (without local improvement) and the CDS heuristic for the minimum makespan and minimum cycle time models, respectively. Although the AP heuristic is based upon the NIS/NW model, it produces good solutions for all other models as well. For the IIS/FR/M model, the AP heuristic performs as well as the CDS heuristic (which was developed for this model); the AP solution is superior for other models especially when the job processing times are random or have a trend. With random processing times,the AP solution has, on average, 12%, 8%, and 6% lower makespan than the CDS solution for the NIS/NW/M, NIS/FR/M, and NIS/SYN/M models, respectively. Similar performance differentials hold for the cycle time minimization models.

Tables 4 and 6 compare the values of the improved AP solution with the best lower bounds for the makespan and cycle time models, respectively. The average % gaps are 3% or less for the IIS/FR/M, NIS/FR/M (except with random processing times), NIS/NW/M, and IIS/FR/C models, indicating that the heuristic generates near-optimal solutions for these cases, and the lower bounds are tight. The gaps are large for the NIS/FR/C and NIS/NW/C models with correlated processing times; we suspect that the heuristic's performance is quite good even for these cases, but the lower bound is weak.

6. Conclusions

This paper has presented a new slack-based lower bound and an assignment-patching heuristic that apply to a variety of flowshop scheduling contexts. The slack-based bound improves upon the machine-based bound, and applies to makespan, total interval time, and cycle time optimization problems. The computational results show that the performance of the lower bound is as good as the LLR bound for problems with random processing times or with trend in processing times. The AP algorithm, together with heuristic local improvement,

performs well for all scenarios, producing schedules that are within 10% of optimal in most cases.

Scheduling flowshops with synchronized job transfers, and with cycle time and interval time objectives are promising topics to investigate further. Although the slack-based method provides lower bounds for these models as well, we might be able to improve performance by exploiting the models' special structure. Similarly, instead of using a general sequencing algorithm based on no-wait principles, we might consider alternative heuristic methods that are specially adapted to cyclic and synchronous scheduling.

Figure 1: Slack time on adjacent machines

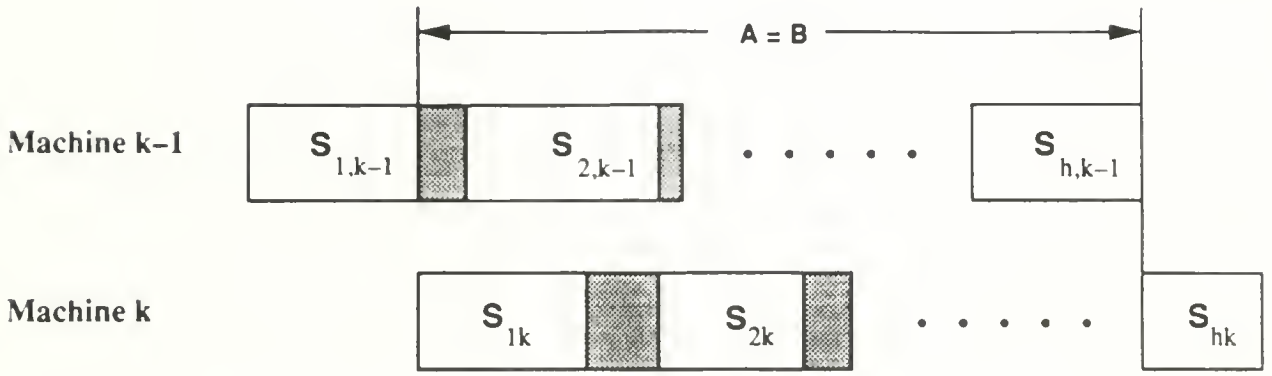


Figure 1(a): No idle time for hth job between machine (k–1) and machine k

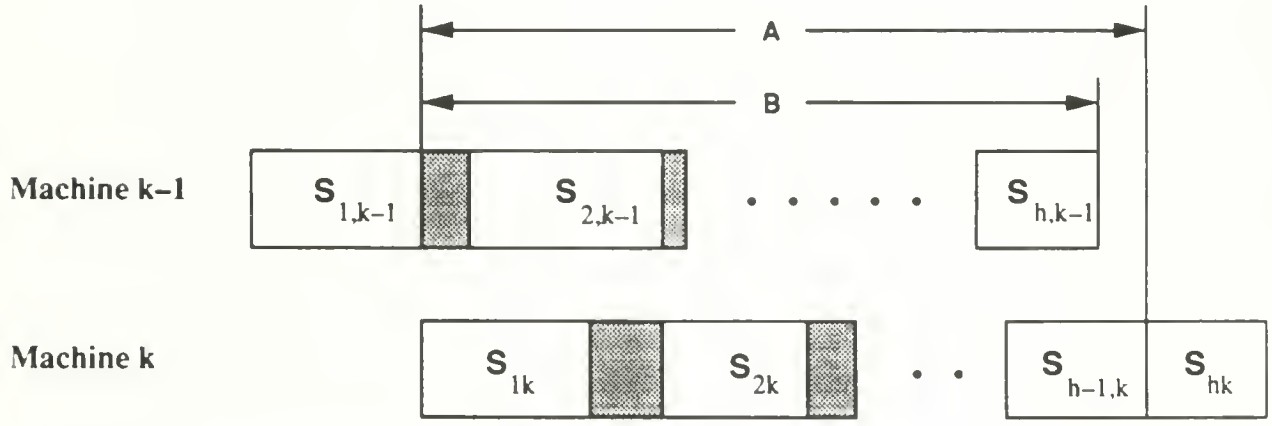


Figure 1(b): Positive idle time for hth job between machine (k–1) and machine k

$$A = \sum_{i=1}^{h-1} S_{ik} + \delta_h^k$$

$$B = \sum_{i=2}^{h} S_{i,k-1} + \delta_h^{k-1}$$

Table 1: Sizes of Test Problems

Problem Group #	Number of jobs (n)	Number of machines (m)
1	6	3
2	6	5
3	6	8
4	10	3
5	10	5
6	15	3
7	20	3
8	20	5
9	50	3

Table 2: Comparison of Lower Bounds for Minimum Makespan Models†

Model (Ratio of bounds)	Proc. Time distribution	*Uncorrelated*	*Correlated*
IIS/FR/M (Slack/LLR)	*No Trend*	0.99	0.81
	Trend	1.00	0.84
NIS/FR/M (Slack/LLR)	*No Trend*	0.99	0.81
	Trend	1.00	0.85
NIS/NW/M (Slack/Szwarc)	*No Trend*	1.01	0.82
	Trend	1.00	0.84
NIS/NW/M (Slack/Assignment)	*No Trend*	0.91	0.95
	Trend	0.97	0.95

† Entries in table represent average values, over 54 problem instances, of Slack-based/LLR, Slack-based/Szwarc or Slack-based/Assignment lower bound ratios

Table 3: Performance of Assignment-Patching Heuristic for Minimum Makespan Models†

Model	Proc. Time distribution	*Uncorrelated*	*Correlated*
IIS/FR/M	*No Trend*	1.00	0.99
	Trend	0.99	0.98
NIS/FR/M	*No Trend*	0.92	0.99
	Trend	0.95	0.98
NIS/NW/M	*No Trend*	0.88	0.98
	Trend	0.94	0.98
NIS/SYN/M	*No Trend*	0.94	0.97
	Trend	0.95	0.97

† Entries in table represent average values, over 54 problem instances, of the ratio of makespans for the Assignment-Patching and CDS heuristic solutions

Table 4: Gap between Upper and Lower Bounds for Minimum Makespan Models†

Model	Proc. Time distribution	*Uncorrelated*	*Correlated*
IIS/FR/M	*No Trend*	0.03	0.01
	Trend	0.01	0.00
NIS/FR/M	*No Trend*	0.09	0.02
	Trend	0.03	0.02
NIS/NW/M	*No Trend*	0.01	0.01
	Trend	0.01	0.01

† Entries in table represent average values, over 54 problem instances, of
Gap = (Makespan of improved heuristic solution – Best Lower Bound)/Best Lower Bound

Table 5: Performance of Assignment-Patching Heuristic for Minimum Cycle Time Models†

Model	Proc. Time distribution	*Uncorrelated*	*Correlated*
IIS/FR/C	*No Trend*	0.99	0.93
	Trend	0.99	0.95
NIS/FR/C	*No Trend*	0.92	0.96
	Trend	0.94	0.96
NIS/NW/C	*No Trend*	0.88	0.95
	Trend	0.92	0.94
NIS/SYN/C	*No Trend*	0.92	0.97
	Trend	0.94	0.97

† Entries in table represent average values, over 54 problem instances, of the ratio of cycle times for the Assignment-Patching and CDS heuristic solutions

Table 6: Gap between Upper and Lower Bounds for Minimum Cycle Time Models†

Model	Proc. Time distribution	*Uncorrelated*	*Correlated*
IIS/FR/C	*No Trend*	0.00	0.00
	Trend	0.00	0.00
NIS/FR/C	*No Trend*	0.12	0.31
	Trend	0.04	0.21
NIS/NW/C	*No Trend*	0.09	0.26
	Trend	0.04	0.22

† Entries in table represent average values, over 54 problem instances, of
Gap = (Cycle Time of improved heur. solution – Slack Lower Bound)/Slack Lower Bound

REFERENCES

Baker, K. R., 1975, A Comparative Study of Flow-Shop Algorithms. *Operations Research*, **23**, 63-73.

Campbell, H. G., Dudek, R. A., and Smith, M. L., 1970, A Heuristic Algorithm for the n Job, m Machine Sequencing Problem. *Management Science*, **16**, B630-B637.

Dannenbring, D. G., 1977, An Evaluation of Flow Shop Sequencing Heuristics. *Management Science*, **23**, 1174-1182.

Dantzig, G. B., Fulkerson, D. R., and Johnson, S. M., 1954, Solution of a Large-scale Traveling Salesman Problem. *Operations Research*, **2**, 393-410.

Gilmore, P. C., and Gomory, R. E., 1964, Sequencing a One State-Variable Machine: A Solvable Case of the Traveling Salesman Problem. *Operations Research*, **12**, 655-679.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G., 1979, Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics*, **5**, 287-326.

Hundal, T. S. and Rajagopal, J., 1988, An Extension of Palmer's Heuristic for the Flow Shop Scheduling Problem. *International Journal of Production Research*, **26**, 1119-1124.

Johnson, S. M., 1954, Optimal Two- and Three-Stage Production Schedules with Setup Times Included. *Naval Research Logistics Quarterly*, **1**, 61-68.

King, J. R., and Spachis, A. S., 1980, Heuristics for Flow-shop Scheduling. *International Journal of Production Research*, **18**, 345-357.

Lageweg, B. J., Lenstra, J. K., and Rinooy Kan, A. H. G., 1978, A General Bounding Scheme for the Permutation Flow-Shop Problem. *Operations Research*, **26**, 53-67.

Lawler, E. L., Lenstra, J. K., Rinooy Kan, A. H. G., 1982, Recent Developments in Deterministic Sequencing and Scheduling: A Survey. *Deterministic Sequencing and Scheduling* , M. A. H. Dempster et al.(eds.), D. Reidel Co., Dordrecht, Holland, 35-74.

Leisten, R., 1990, Flowshop Sequencing Problems with Limited Buffer Storage. *International Journal of Production Research*, **8**, 2085-2100.

McCormick, S. T., Pinedo, M. L., Shenker, S., and Wolf, B., 1989, Sequencing in an Assembly Line with Blocking to Minimize Cycle Time. *Operations Research*, **37**, 925-935.

McMahon, G. B. and Burton, P. G., 1967, Flow-Shop Scheduling with the Branch-and-Bound Method. *Operations Research*, **15**, 473-481.

Nawaz, M. Enscore, E. E., and Ham, I., 1983, A Heuristic Algorithm for the m-Machine, n-Job Flow-Shop Sequencing Problem. *The International Journal of Management Science*, **11**, 91-95.

Park, Y. B., Pegden, D., and Enscore E. E., 1984, A Survey and Evaluation of Static Flowshop Scheduling Heuristics. *International Journal of Production Research*, **22**, 127-141.

Plante, R. D. and Lowe, T. J., 1987, The Product Matrix Traveling Salesman Problem: An Application and Solution Heuristic. *Operations Research*, **35**, 772-783.

Rajendran, C. and Chaudhuri, D., 1990, Heuristic Algorithms for Continuous Flowshop Problems. *Naval Research Logistics* , **37**, 695-705.

Reddi, S. S., and Ramamoorthy, C. V., 1972, On the Flow-shop Sequencing Problem with No Wait in Process. *Operational Research Quarterly*, **21**, 544-549.

Skinner, W., 1974, The Focused Factory. *Harvard Business Review*, 113-122.

Szwarc, W., 1983, Solvable Cases of the Flow-Shop Problem Without Interruptions in Job Processing. *Naval Research Logistics Quarterly*, **30**, 179-183.