Cyclic Scheduling in Flow Lines: Modeling Observations, Effective Heuristics and a Cycle Time Minimization Procedure

Selcuk Karabati*
Bilkent University, Management Department, Faculty of Business Administration, Bilkent, Ankara 06533, Turkey

Panagiotis Kouvelis
Duke University, The Fuqua School of Business, Durham, North Carolina 27708

In this paper we address the cyclic scheduling problem in flow lines. We develop a modeling framework and an integer programming formulation of the problem. We subsequently present exact and approximate solution procedures. The exact solution procedure is a branch-and-bound algorithm which uses Lagrangian and station-based relaxations of the integer programming formulation of the problem as the lower bounding method. Our heuristic procedures show a performance superior to the available ones in the literature. Finally, we address the stability issue in cyclic scheduling, demonstrate its relationship to the work-in-progress inventory control of a flow line, and present a very simple procedure to generate stable schedules in flow lines. © 1996 John Wiley & Sons, Inc.

1. INTRODUCTION

In this paper we address the cyclic scheduling problem of a flow line with the performance objective of maximizing the throughput rate of the line. Some of the advantages of cyclic scheduling policies over conventional scheduling techniques for current flow line environments are mentioned in the research literature (Matsuo [12], Lee and Posner [9]) and include benefits such as better station utilization, smoother finished goods inventory levels, and implementation convenience due to the simplicity of cyclic schedules.

The operating environment of a flow line under a cyclic scheduling policy can be described as follows. The system consists of m stations in series with finite or infinite capacity buffers between the stations. Let r_l be the number of units of item l, $l = 1, 2, \ldots, L$, required to meet a production target of the line over a planning horizon, and $r = (r_1, \ldots, r_L)$ be the production requirement vector. If q is the greatest common divisor of integers $r_1, \ldots, r_L$, then the vector

$$r^* = \left(\frac{r_1}{q}, \ldots, \frac{r_L}{q}\right)$$

is referred to as the Minimal Part Set (MPS) (term introduced by Hitz [8]). It represents the smallest part set having the same proportions as the production requirement vector.

* Current address: College of Administrative Sciences and Economics, Koç University, Cayir Cad. No. 5, Istinye, Istanbul, 80860, Turkey

Naval Research Logistics, Vol. 43, pp. 211–231(1996)

CCC 0894-069X/96/020211-21

Under a cyclic scheduling policy, the flow line will produce an integral multiple of MPSs. A part set that is an integral multiple of MPS is represented through its production requirement vector

$$r' = \alpha r^* = \left(\alpha \frac{r_1}{q}, \ldots, \alpha \frac{r_L}{q}\right)$$

where α is an integral constant of proportionality and a divisor of q. The above part set is referred to as the α-multiple MPS (α-MPS). In our analyzed flow line environment the processing times of the various items are assumed to be known and deterministic. Note that an MPS (or α-MPS) contains more than one unit of item l, when $r_l/q > 1$, $l = 1, 2, \ldots, L$. However, each element of the MPS represents a job to be processed by the system, and our scheduling task is to sequence these jobs. We treat every element of the MPS as an individual job, although some of these jobs may belong to the same item group. Therefore, the total number of jobs in an MPS is equal to $n = \sum_{l=1}^{L} r_l/q$. We denote by p_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, m$, the processing time of job i on station j. Then our cyclic scheduling problem is to find the optimal sequence of jobs in a prespecified part set (i.e., α-MPS for a given $\alpha \geq 1$) in order to optimize the throughput rate of the line, or equivalently its cycle time (i.e., the reciprocal of the throughput rate of the line). We are restricting our analysis to the operation of conventional flow lines (i.e., by-passing of stations by some jobs is not allowed). For references on flexible flow lines, which allow station by-passing by some jobs, see Hitz [8] and Wittrock [20].

The cyclic scheduling problem for flow lines has been recently studied in some detail. McCormick et al. [14] established the computational complexity of the problem. The cyclic scheduling problem for cycle time minimization is unary *NP*-complete for arbitrary processing times. A polynomial time algorithm for a special case of the problem has been presented by Matsuo [12]. He analyzed the two station flow line with a zero capacity buffer between the stations. McCormick et al. [13] have presented a heuristic solution procedure for the problem.

The cyclic scheduling problem has been discussed in various settings. Bartholdi [1] has addressed the problem of cyclic staff scheduling and presented a heuristic solution procedure. Graves et al. [6] address the scheduling problem in a reentrant flow shop, and present a heuristic solution procedure for the problem. In Hall [7], the notion of cyclic production scheduling approach and its benefits have been discussed in detail. Roundy [18] addresses the cyclic scheduling problem in a job shop with identical jobs. Cyclic scheduling of a material handling network has been discussed in Lei and Wang [10]. Bowman and Muckstadt [2] address the cyclic scheduling of operations in a multi-machine environment, and analyze the performance of cyclic schedules under stochastic variability in processing times. Dauscha et al. [3] discuss the problem of cycling scheduling of a finite set of tasks, prove an existence criterion for cyclic schedules, and investigate some complexity issues in cyclic scheduling. Whybark [19] describes an actual production planning and control system which is an implementation of the cyclic scheduling approach. In Rao [16] the identical jobs, single end-item cyclic scheduling problem is addressed, and a set of sequencing heuristics is developed and tested. Rao and Jackson [17] study two subproblems that are encountered in the identical jobs, reentrant flow cyclic scheduling problem. Fuxman [4] studies the team organization of workers in mixed-model assembly lines under cyclic production. Loerch and Muckstadt [11] address the cyclic production planning and scheduling problem.

In this paper we develop an integer programming formulation of the cyclic scheduling problem in flow lines. Our optimal procedure for the above formulation is a branch-and-bound approach, and can handle cycling scheduling of 12 jobs per MPS and up to ten stations for various buffer configurations between the stations. For larger size problems we propose two new approximate solution procedures. One of the approximate solution procedures is a simple constructive heuristic, while the other one is based on an early termination of the branch-and-bound procedure. Both heuristic procedures outperform the best available in the literature (see McCormick et al. [13]).

In the cyclic scheduling literature the concept of stability of schedules plays an important role. For a detailed reference on this concept see Lee and Posner [9]. The authors discuss the concept in a general job shop environment with infinite capacity buffers, and present a graph theoretic framework for stable schedule generation. In flow lines the stability of a schedule is directly related to maintaining low work-in-progress (WIP) inventory levels, as we will further discuss later. Another research contribution of our paper is the development of a simple way to generate stable schedules in a flow line environment.

The structure of the paper is the following. In Section 2 we discuss a graph theoretic modeling framework of job completion times in cyclic scheduling flow line environments. Building on the previous modeling framework, in Section 3 we present a formal statement of the cyclic scheduling problem as an integer program. Section 4 discusses the use of relaxation approaches for lower and upper bound generation on the optimal cycle time values of cyclic scheduling problems. Section 4 constitutes the backbone of the branch-and-bound procedure suggested for the optimal solution of the problem. The details of the procedure, as well as our computational experience with it, are presented in Section 5. For the solution of large size problems we suggest the use of heuristic solution procedures whose details and successful performance are documented in Section 6. A stable schedule generation scheme for flow line environments is discussed in Section 7. Section 8 concludes the paper with a summary of our results.

2. A MODELING FRAMEWORK FOR THE CYCLE TIME CRITERION

For the production environment described in Section 1, let us consider the operation of the line under the cyclic scheduling approach. The jobs of an MPS (or α-MPS) go through the system in a given order followed by a second MPS (or a second α-MPS) in the same order and so on. A total of q (or q/α) runs of an MPS (or α-MPS) is required to meet the production target of the line over the planning horizon. Our discussion below concentrates on sequencing jobs of an MPS, however, the same approach can be readily extended to sequencing jobs of an α-MPS.

An MPS schedule will be represented by permutation $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$ where n is the number of jobs in the MPS and $\sigma(i)$ is the ith job in the processing order. Note that we restrict our analysis to permutation schedules only, i.e., no job passing is allowed once the MPS is released to the system. Job $\sigma(i)$ in the rth MPS is said to be the rth repetition of job $\sigma(i)$ and is denoted by $\sigma_r(i)$.

McCormick et al. [13] have shown that a unit capacity buffer can be represented as a station at which the processing times of all jobs are equal to zero. Therefore, without any loss of generality, we may assume that all buffers have either zero or infinite capacity, because a finite capacity buffer can be represented as a series of unit capacity buffers. This transformation increases the number of stations in our problem, however, as we will dem-

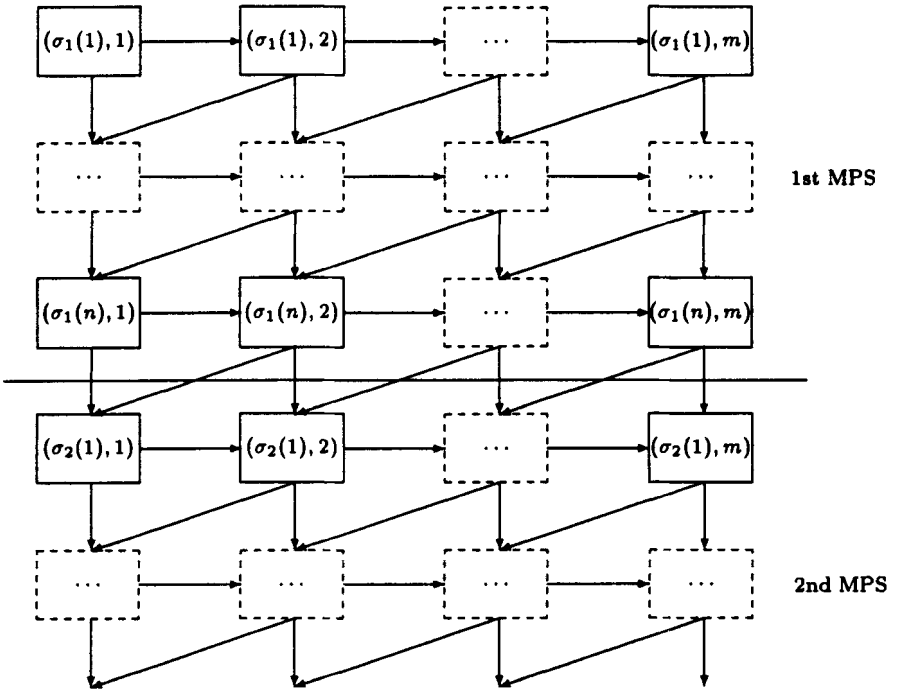

Figure 1.

onstrate later, it does not increase the computational effort required by our optimal solution procedure for the minimum cycle time problem.

Let us consider a flow line with zero capacity buffers. The completion time $C(\sigma_r(i),j)$ of job $\sigma_r(i)$ on station j can be found using the following recursive relationship:

$$C(\sigma_r(i),j) = \max\{C(\sigma_r(i-1),j+1), \max\{C(\sigma_r(i-1),j), C(\sigma_r(i),j-1)\} + p_{\sigma(i),j}\}. \quad (1)$$

This recursive relationship can be represented by the directed graph presented in Figure 1. Vertices $(\sigma_r(i),j)$ are defined for each element $\sigma(i)$ of the permutation, for $r = 1, 2, \ldots,$ and each station j. Directed arcs are defined from each vertex $(\sigma_r(i),j)$ towards vertices $(\sigma_r(i+1),j)$, $(\sigma_r(i),j+1)$ and $(\sigma_r(i+1),j-1)$, where $C(\sigma_1(0),j) = 0, j = 1, 2, \ldots, m$, $C(\sigma_r(i),0) = 0, r \geq 1, i = 1, 2, \ldots, n$, $\sigma_r(0) = \sigma_{r-1}(n), r > 1$, and $\sigma_r(n+1) = \sigma_{r+1}(1), r \geq 1$, due to the cyclic nature of the problem. The directed arc from vertex $(\sigma_r(i),j)$ to vertex $(\sigma_r(i+1),j-1)$ accounts for the blocking of jobs due to the zero capacity buffer between stations $j-1$ and j, and therefore should be removed from the graph if there exists an infinite capacity buffer between stations $j-1$ and j. A weight $p_{\sigma(i),j}$ is associated with vertex $(\sigma_r(i),j)$, if the vertex is entered via either the horizontal or the vertical directed arc that are adjacent to vertex $(\sigma_r(i),j)$. We note that when an operation is blocked, i.e., the corresponding vertex is entered via the diagonal directed arc, its processing time does not affect its completion time, and therefore the weight of the vertex is set equal to zero.

Given the above defined graph, the completion time $C(\sigma_r(i),j)$ of job $\sigma_r(i)$ on station j

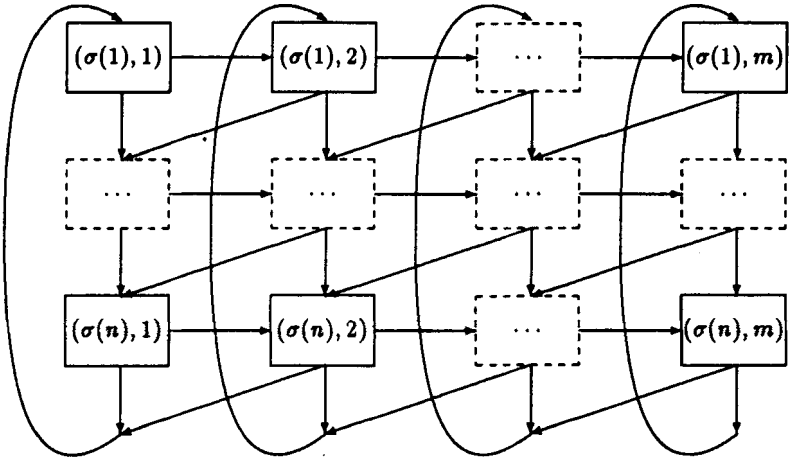

Figure 2.

is equal to the weight of the maximum-weighted directed path from $(\sigma_1(1), 1)$ to $(\sigma_r(i), j)$ in the graph, as follows immediately from the recursive relationship (1).

The performance criterion we want to optimize is the throughput rate of the flow line, or equivalently its cycle time. To define the cycle time of a flow line in our modeling framework, we need to introduce the notion of a cylinder formed by the directed graph of a single MPS, which is depicted in Figure 2. The relationship of the cycle time to the above concept comes from the following theorem:

THEOREM 1: (McCormick et al. [13]) The weight of the maximum-weighted path around a cylinder formed by the directed graph of a single MPS equals the cycle time.

Our representation of the MPS cylinder is slightly different than that of McCormick et al.'s, in our cylinder the processing times are represented as vertex weights instead of arc weights. This approach is necessitated by our formulation of the cyclic scheduling problem.

We now introduce a convenient way of computing the weight of a path around the cylinder. Let τ be a path around the cylinder, i.e., a chain of vertices that starts at a particular vertex and ends at the same vertex after completing one tour around the cylinder. We note that in a given tour around the cylinder no vertex is visited more than once, because the directed graph is planar and contains no sub-tours. Let w_i^τ, $i = 1, \ldots, n$, be the station index of the right-most operation of job $\sigma(i)$ on path τ. We have the following property:

$$w_n^\tau - 1 \leq w_1^\tau, \qquad \text{and } w_{i-1}^\tau - 1 \leq w_i^\tau, \qquad i = 2, 3, \ldots, n,$$

with the equality holding if job $\sigma(i)$ is blocked on station w_i^τ by job $\sigma(i-1)$ on the tour defined by path τ (i.e., vertex $(\sigma(i), w_i^\tau)$ is entered via vertex $(\sigma(i-1), w_{i-1}^\tau)$, where $w_{i-1}^\tau = w_i^\tau + 1$). It follows that $p_{\sigma(i),k}$ is included in the weight of path τ if and only if $w_{i-1}^\tau \leq k \leq w_i^\tau$. Let $f(\sigma, \tau)$ be the weight of path τ, when the cylinder is formed using the permutation schedule σ. We can now compute $f(\sigma, \tau)$ as follows:

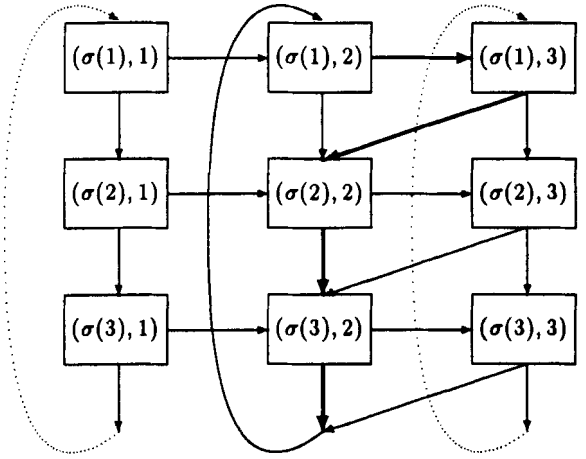

Figure 3. Directed graph of Example 1.

$$f(\sigma, \tau) = \sum_{k=w_n^\tau}^{w_1^\tau} p_{\sigma(1),k} + \sum_{k=w_1^\tau}^{w_2^\tau} p_{\sigma(2),k} + \cdots + \sum_{k=w_{n-1}^\tau}^{w_n^\tau} p_{\sigma(n),k}. \tag{2}$$

For any given path $\tau \in T$, we can determine the unique set of integers $w^\tau = (1 \le w_i^\tau \le m, i = 1, 2, \ldots, n)$ to represent the weight of the path using the above functional form.

We will illustrate the characterization of a path by a set of integers using the following example.

EXAMPLE 1: Consider a three jobs per MPS three station problem where the buffer capacity between stations 1 and 2 is infinitely large and the buffer capacity between stations 2 and 3 is zero. The cylinder of this problem is presented in Figure 3. The path τ is given by the dark lines in Figure 3. Now the weight of path τ is equal to

$$f(\sigma, \tau) = p_{\sigma(1),2} + p_{\sigma(1),3} + p_{\sigma(3),2}.$$

We note that operation $(\sigma(2), 2)$ is blocked by operation $(\sigma(1), 3)$ and therefore $p_{\sigma(2),2}$ is not included in $f(\sigma, \tau)$. The right-most operation of $\sigma(1)$ on path τ is $(\sigma(1), 3)$, therefore $w_1^\tau = 3$. Similarly, $w_2^\tau = 2$, and $w_3^\tau = 2$. Now, let $w^\tau = (3, 2, 2)$, then

$$f(\sigma, \tau) = \sum_{k=2}^{3} p_{\sigma(1),k} + \sum_{k=3}^{2} p_{\sigma(2),k} + \sum_{k=2}^{2} p_{\sigma(3),k} = (p_{\sigma(1),2} + p_{\sigma(1),3}) + (0) + (p_{\sigma(3),2}).$$

The representation of $f(\sigma, \tau)$ in the above functional form will play an important role in the development of an integer programming formulation for the cyclic scheduling problem.

3. AN INTEGER PROGRAMMING FORMULATION OF THE MINIMUM CYCLE TIME PROBLEM

In this section we present a modeling framework for the minimum cycle time problem. In developing our framework we will use Theorem 1 in Section 2. Theorem 1 states that

the weight of the maximum-weighted path around a cylinder formed by a single MPS equals the cycle time. Let τ be a path around the cylinder as depicted in Figure 2, and T be the set of all paths around the cylinder. We note that the elements of set T are independent of the sequence of jobs, i.e., the structure of the cylinder is the same for all permutation sequences, only the vertex weights are affected by the sequence of jobs. We denote by $f(\sigma, \tau)$ the weight of path $\tau \in T$ when the cylinder is formed (i.e., vertex weights are determined) using MPS schedule σ. Let λ_σ denote the cycle time of this schedule. Then λ_σ is given by

$$\lambda_\sigma = \max_{\tau \in T} f(\sigma, \tau).$$

If we let S be the set of all MPS schedules, the minimum cycle time problem can be formulated as

$$\lambda^* = \min_{\sigma \in S} \max_{\tau \in T} f(\sigma, \tau), \tag{3}$$

and λ^* is the optimal cycle time for the problem. In the rest of the paper we will use the abbreviation $n/m/P, D/\lambda$ for the minimum cycle time problem with n jobs per MPS, m stations, and buffer configuration D. The buffer configuration indicates the number of buffers between adjacent stations. For example, for $m = 4$, $D = (2, 1, 2)$ describes a buffer configuration with two unit capacity buffers between the first and second stations, one unit capacity buffer between the second and third stations, and two unit capacity buffers between the third and fourth stations. The symbol P reminds us of our restricted attention to permutation MPS schedules, and λ refers to the cycle time criterion.

Using the above established modeling framework, we can now state the following proposition.

PROPOSITION 1:

$$\max_{\tau \in T} \min_{\sigma \in S} f(\sigma, \tau) \leq \min_{\sigma \in S} \max_{\tau \in T} f(\sigma, \tau). \tag{4}$$

PROOF: Let $\sigma_0 \in S$ be a specific MPS schedule and $\tau_0 \in T$ be a path around the cylinder. Then

$$\min_{\sigma \in S} f(\sigma, \tau_0) \leq f(\sigma_0, \tau_0) \leq \max_{\tau \in T} f(\sigma_0, \tau). \tag{5}$$

However, (5) holds for every $\sigma_0 \in S$ and $\tau_0 \in T$, therefore the validity of (4) follows immediately. □

Proposition 1 states that for any path around the cylinder, $\tau \in T$, the solution to the optimization problem

$$(AP_\tau): \min_{\sigma \in S} f(\sigma, \tau)$$

provides a lower bound for the problem as formulated in (3). Some further insight on the nature of (AP_τ) is provided by the following result.

PROPOSITION 2: (AP_τ) is equivalent to a linear assignment problem.

PROOF: According to (2), for a specific path $\tau \in T$, there exists a unique set of integers $w^\tau = (w_1^\tau, w_2^\tau, \ldots, w_n^\tau)$ such that for any $\sigma \in S$, we have the following:

$$f(\sigma, \tau) = \sum_{r=1}^{n} \sum_{j=1}^{n} \sum_{k=w_{j-1}^\tau}^{w_j^\tau} p_{r,k} 1\{\sigma(j) = r\}, \tag{6}$$

where $1\{\cdot\}$ is an indicator function and $w_0^\tau = w_n^\tau$. Let us denote by

$$a_{r,j}^\tau = \sum_{k=w_{j-1}^\tau}^{w_j^\tau} p_{r,k}.$$

Then (6) can be rewritten as

$$f(\sigma, \tau) = \sum_{r=1}^{n} \sum_{j=1}^{n} a_{r,j}^\tau 1\{\sigma(j) = r\}. \tag{7}$$

Using relationship (7) one can easily observe that (AP_τ) is equivalent to a linear assignment problem with cost matrix $A^\tau = (a_{r,j}^\tau)$. □

We can write an integer programming formulation of the cyclic scheduling problem as follows: We first introduce the set of binary decision variables $x_{i,j} \in \{0, 1\}$, to equivalently describe a schedule as

$$(\sigma \in S) \leftrightarrow X = (x_{i,j}),$$

where

$$x_{i,j} = \begin{cases} 1, & \text{if } \sigma(j) = i, \\ 0, & \text{otherwise.} \end{cases}$$

Using the above set of decision variables and relationship (7) we can state the problem in an equivalent way as follows:

$$(IP): f^* = \min M$$

subject to

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^\tau x_{i,j} \le M, \qquad \tau \in T, \tag{8}$$

$$\sum_{i=1}^{n} x_{i,j} = 1, \qquad j = 1, \ldots, n, \tag{9}$$

$$\sum_{j=1}^{n} x_{i,j} = 1, \qquad i = 1, \ldots, n, \tag{10}$$

$$x_{i,j} \in \{0, 1\}, \qquad i, j = 1, \ldots, n. \tag{11}$$

In the next section we present a lower bounding scheme for the $n/m/P$, D/λ problem which is based on a relaxation of formulation (IP).

4. A LOWER BOUND GENERATION SCHEME FOR THE $n/m/P$, D/λ PROBLEM

In order to develop lower bounds for the $n/m/P$, D/λ problem, we are going to use a relaxation of formulation (IP). In the relaxed problem, we restrict our attention only to a subset T^* of the path set T. The relaxed formulation of (IP) is given as follows:

$$(RIP): f_r^* = \min M$$

subject to

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^{\tau} x_{i,j} \leq M, \qquad \tau \in T^*, \tag{12}$$

(9), (10) and (11).

We now present two lower bounding procedures based on formulation (RIP).

4.1. The Lagrangian Relaxation Lower Bound

In our first lower bounding approach, we dualize the constraint set (12). Let $\mu = \{\mu_\tau \geq 0, \tau \in T^*\}$ be a set of Lagrange multipliers. Then, the Lagrangian relaxation of (RIP) is

$$(LR(\mu)): f(\mu) = \min M\left(1 - \sum_{\tau \in T^*} \mu_\tau\right) + \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\sum_{\tau \in T^*} \mu_\tau a_{i,j}^{\tau}\right) x_{i,j}$$

subject to (9), (10) and (11).

Observe that $f(\mu)$ is bounded only if $\sum_{\tau \in T^*} \mu_\tau = 1$. Without loss of generality, we can always assume that this is the case, since we can normalize the Lagrange multipliers μ_τ, $\tau \in T^*$, by their sum $\sum_{\tau \in T^*} \mu_\tau$. Therefore, we can always state ($LR(\mu)$) as an equivalent linear assignment problem as follows:

$$(LRAP(\mu)): f_{AP}(\mu) = \min \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\sum_{\tau \in T^*} \mu_\tau a_{i,j}^{\tau}\right) x_{i,j}$$

subject to (9), (10) and (11).

Let f_r^{LP} be the optimal objective function value of the linear programming relaxation of (RIP). Let $\Omega = \{\mu \in \mathcal{R}^{|T^*|} \mid \sum_{\tau \in T^*} \mu_\tau = 1 \text{ and } \mu_\tau \geq 0, \tau \in T^*\}$ be a set of multiplier vectors. Using standard results from the Lagrangian relaxation theory (see Nemhauser and Wolsey [15]), we can state the following property.

PROPERTY 1:

$$f_\tau^{LP} = \max_{\mu \in \Omega} f_{AP}(\mu).$$

From our above discussion, we can conclude that f_τ^* is a legitimate lower bound for the $n/m/P, D/\lambda$ problem, i.e., $f_\tau^* \leq f^*$. We can always approximate f_τ^* from below by solving the linear programming relaxation of (RIP). From well known results of Lagrangian relaxation theory, and since the constraint set of $(LRAP(\mu))$ has a network structure, we are guaranteed that our Lagrangian bound cannot be better than the linear programming relaxation bound. The solution of the linear programming relaxation of (RIP), however, proved to be computationally prohibitive when we implemented it within a branch-and-bound approach. Thus, we decided to use a slightly different approach. To obtain f_τ^{LP}, or a lower bound on it, we solve the linear assignment problem $(LRAP(\mu))$ for an appropriately chosen vector $\mu \in \Omega$. The use of a subgradient optimization method helps us update the multiplier vector μ, and within a small number of iterations generates a very good approximation of f_τ^{LP}. Our subgradient optimization procedure starts with an initial multiplier vector $\mu^0 \in \Omega$, and then defines iteratively a sequence of multiplier vectors μ^t by use of the following rule

$$\bar{\mu}_\tau^{t+1} = \max\left\{0, \mu_\tau^t + \frac{\beta^t \zeta_\tau (\tilde{f} - f_{AP}(\mu^t))}{\|\mu\|^2}\right\},$$

where β^t is a scalar, $\tilde{f}$ is an upper bound on f_τ^{LP}, $\zeta = (\zeta_\tau, \tau \in T^*)$ is a subgradient for $f_{AP}(\mu^t)$, and $\|\mu\|$ is the Euclidean norm of the vector μ. The notation $\bar{\mu}^t = (\bar{\mu}_\tau^t, \tau \in T^*)$ denotes the multiplier vector before the required normalization procedure for our method. The above subgradient optimization procedure is an adaptation of the one presented in Goffin [5], and based on results therein it is guaranteed to converge to f_τ^{LP}, when $\sum_{t=0}^{\infty} \beta^t = \infty$ and $\lim_{t \to \infty} \beta^t = 0$. For the complete specification of the above procedure we need to describe the subgradient vector ζ. This is described in Property 2 below.

PROPERTY 2: Let $\zeta_\tau = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^\tau x_{ij}^* - f_{AP}(\mu)$, $\tau \in T^*$, where $X^* = (x_{ij}^*)$ is the optimal solution to the $(LRAP(\mu))$ problem. Then, $\zeta = (\zeta_\tau, \tau \in T^*)$ is a subgradient for $f_{AP}(\mu)$.

PROOF: We present the proof of Property 2 in Appendix A.

4.2. The Contiguous Stations Lower Bound

For flow lines with at least two contiguous stations, i.e., two stations with a zero capacity buffer in between them, a station-based, and potentially tighter, lower bounding procedure can be developed based on formulation (RIP). In formulation (RIP) we use a subset T^* of the path set T. Let j and $j+1$ be two contiguous stations, and T^* be the set of paths around the cylinder formed by stations j and $j+1$. Now (RIP) is equivalent to a two station, zero capacity buffer problem which can be solved in $O(n \log n)$ time (Matsuo [12]). Therefore, in flow lines with contiguous stations the above described procedure can be used to develop lower bounds for the minimum cycle time problem, because, as we have

discussed earlier, the optimal value f_τ^* of formulation (RIP) constitutes a lower bound for the original problem.

We have used these lower bound generation schemes within a branch-and-bound algorithm. We will discuss the implementation and quality of the new bounding techniques in the next section.

5. A BRANCH-AND-BOUND ALGORITHM

In this section we present an optimal algorithm for the $n/m/P, D/\lambda$ problem and report on our computational experience with the algorithm.

5.1. Description of the Algorithm

Our branching and bounding schemes are as follows:

Branching Scheme: The branching scheme used in our solution procedure defines the nodes of the search tree as partial schedules. A node at the kth level of the search tree corresponds to a partial schedule of size k, i.e., a schedule where assignments are made to the first k positions. When branching from a node of the search tree at the kth level, we create $(n - k)$ nodes at the $(k + 1)$-st level by appending each unassigned job of the node to the end of the partial schedule of size k.

Note that in our branching scheme we have assumed that MPSs are formed of nonidentical jobs. However, when there are some identical jobs in the MPS, in order to decrease the size of the search tree, the above branching scheme can be modified to account for these similarities.

Bounding Scheme: In writing relaxation (RIP) for development of the Lagrangian lower bound, we choose $|T^*|$ paths around the MPS cylinder. In the current version of the algorithm, $|T^*|$ is created as follows: First we randomly create a permutation schedule, and determine its maximum-weighted path (see Section 2 for the determination of the maximum-weighted path). This path is then included in $|T^*|$. Next, we solve (AP_τ) (see Section 3) on this path to generate another schedule. We then determine the maximum-weighted path of the new schedule, and include it in $|T^*|$. We continue by solving (AP_τ) on this new path to generate a new schedule. This procedure is repeated until $|T^*|$ reaches the required size. If a newly generated path is already in $|T^*|$, the procedure is restarted with another randomly created schedule. In the current version of the algorithm we use 10 paths to obtain the Lagrangian lower bounds.

The selection of paths is performed at the initial node of the search tree. However, as we move down in the search tree the size of the partial schedule (i.e., the number of fixed jobs) that corresponds to a particular node increases, and the set T^* is modified so that the selected paths attain their maximum possible values on these partial schedules. This is accomplished by determining the maximum-weighted paths on the MPS cylinder's fixed part, i.e., the part of the MPS cylinder that corresponds to the partial schedule of the node under consideration. These paths are later appended to the paths chosen at the initial node of the search tree.

The performance of our algorithm is very much affected by the selection of the initial multiplier vector μ in the subgradient optimization method. Due to the above mentioned selection of T^*, the paths of a node and its immediate descendant nodes are very similar, and in most cases the multiplier vector obtained in the last iteration of the subgradient

method is a very good initial multiplier vector for the immediate descendant nodes of this particular node. In the current version of the algorithm, at each node of the search tree multipliers are updated four times, except the initial node where we iterate 20 times to obtain the initial multipliers.

We have used the alternative contiguous stations-based lower bounding procedure described in Section 4.2, along with the subgradient method, when the flow line under consideration has pairs of contiguous stations. In this case, the lower bounds generated by the station-based procedure are usually better than the bounds obtained using the subgradient method, due to the fact that in this approach (RIP) is directly solved without resorting to a further relaxation.

We also note that both lower bounding procedures generate complete schedules in developing the lower bounds, and these schedules are evaluated for their cycle times to update the upper bound on the optimal cycle time value of the problem. However, the upper bounds generated by the subgradient method have dominated the upper bounds obtained by the contiguous stations-based procedure. This is due to the fact that the contiguous stations-based lower bounding procedure uses local information, i.e., only the processing times of the two contiguous stations, whereas the subgradient method considers the flow line in its entirety. Therefore we have decided to use these two bounding schemes concurrently. Note that, if the flow line does not possess any contiguous stations, the contiguous stations-based procedure cannot be employed and the subgradient method becomes the only possible approach to generate lower bounds.

Fathoming Criterion: If the current upper bound is less than or equal to the lower bound on that subset of schedules.

Node to Examine Next Criterion: In the current version of our algorithm we examine next the subset of schedules that has the smallest lower bound.

5.2. Computational Results

We have tested the performance of the branch-and-bound procedure described in Section 5.1 using two sets of problems. The first set of test problems is the five problems with eight jobs per MPS and nine stations given in McCormick et al. [13]. Our optimal solution procedure solved these problems with an average of 32.6 nodes and 0.19 CPU seconds per problem, on a multi-user IBM 3081-D system. We also note that three of these test problems have some identical jobs and the performance of our approach can be improved by taking the similarity of jobs into account when branching in the search tree.

The second set of test problems is created using integer processing times drawn from a (1–100) uniform distribution. We have tested the performance of the algorithm for problems with 12 nonidentical jobs per MPS. In Table 1, the first column documents the configuration of the line. For each configuration we created 20 problems. In Columns 3 and 4 of Table 1, we present the minimum, average, and maximum difference between the lower bound value of the initial node of the search tree and the optimal cycle time value as a percentage of the optimal cycle time value, for the Lagrangian relaxation and contiguous stations lower bounds, respectively. In Column 5 of Table 1, we present the minimum, average, and maximum difference between the upper bound value of the initial node of the search tree and the optimal cycle time value, as a percentage of the optimal cycle time value. Upper bounds are generated using the LIST heuristic which will be described in Section 6. Columns 6 and 7 of Table 1 tabulate the minimum, average, and maximum number of nodes and CPU times, respectively. Column 8 documents the maximum num-

Table 1. Performance of the branch-and-bound algorithm for 12 jobs per MPS problems.

Flow Line		LB Quality		UB	No of	CPU	No of
		Lagrangian	2-Station	Quality	Nodes	(seconds)	Active Nodes
$m = 3$	Min.	3.30	0.00	0.00	12.00	0.32	11.00
$D = (0, 0)$	Ave.	15.33	2.95	2.29	7438.30	42.53	1845.80
	Max.	23.80	8.49	7.96	31886.00	179.54	7268.00
$m = 4$	Min.	0.32	0.00	0.00	12.00	0.57	11.00
$D = (1, 0, 1)$	Ave.	10.33	3.79	1.86	2593.35	19.18	710.50
	Max.	19.21	25.42	4.41	25475.00	190.98	7786.00
$m = 5$	Min.	3.67	0.00	0.00	12.00	1.20	11.00
$D = (0, 2, 2, 0)$	Ave.	10.20	3.79	1.61	379.90	6.62	179.05
	Max.	17.66	17.89	4.87	1744.00	25.32	746.00
$m = 6$	Min.	6.45	0.00	0.00	12.00	1.40	11.00
$D = (0, 0, 2, 2, 0)$	Ave.	12.65	2.78	2.57	8129.50	153.80	2719.20
	Max.	18.08	8.74	5.49	46348.00	1148.77	19607.00
$m = 7$	Min.	0.00	N/A	0.00	12.00	4.62	11.00
$D = (2, 2, 2, 2, 2, 2)$	Ave.	1.23	N/A	0.00	12.00	5.05	11.00
	Max.	10.89	N/A	0.00	12.00	5.79	11.00
$m = 8$	Min.	4.96	0.00	0.12	12.00	2.76	11.00
$D = (0, 2, 0, 2, 0, 2, 0)$	Ave.	10.77	1.54	2.50	1601.25	53.74	611.45
	Max.	16.22	7.24	5.30	13308.00	469.63	4242.00
$m = 9$	Min.	2.17	0.00	0.00	12.00	5.35	11.00
$D = (2, 0, 3, 0, 0, 3, 0, 2)$	Ave.	13.95	3.22	2.64	8192.10	421.45	2659.25
	Max.	21.41	6.37	5.92	23971.00	1330.33	7144.00
$m = 10$	Min.	3.38	0.00	0.00	12.00	5.42	11.00
$D = (2, 0, 2, 0, 2, 1, 0, 2, 1)$	Ave.	11.66	3.74	2.43	2359.80	120.83	1081.35
	Max.	20.60	14.86	5.39	8640.00	444.00	4439.00

ber of active nodes, i.e., nodes that have been stored for further branching during the implementation of the algorithm. The number of active nodes serves as an indicator of the storage requirements of the algorithm. Based on the results reported in Table 1, we can conclude that the algorithm exhibits a satisfactory performance over all tested problems. We also note that the contiguous stations-based lower bounds dominated the Lagrangian relaxation lower bounds in all of the test problems with contiguous stations.

The size of the problems solvable by the branch-and-bound procedure is limited by the number of jobs which directly affects the size of the solution space, and the buffer configuration of the flow line which largely determines the lower bound quality. The performance of the algorithm improves dramatically as the buffer sizes are increased. Therefore most difficult problems are those with contiguous stations. For $n = 12$, we were able to solve problems with twosomes of contiguous stations, provided that buffer sizes are greater than or equal to 2 between the groups of contiguous stations. Although it is very difficult to determine the limiting number of jobs without considering the buffer configuration of the flow line, for a limited buffers case, 12 seems to be largest number of jobs that can be efficiently handled by our procedure.

6. HEURISTIC SOLUTION PROCEDURES

As polynomial time algorithms are unlikely to exist for the $n/m/P, D/\lambda$ problem, heuristic solution procedures are needed to obtain good schedules for large size problems. McCormick et al. [13] have developed a heuristic algorithm for cycle time minimization based on idle time and blocking time penalties. The McCormick et al. [13] heuristic can be described as follows: For a given partial sequence a penalty is computed for each un-

scheduled job. The penalty of a job is equal to the weighted sum of idle and blocking times caused by this job if it is appended to the end of the partial schedule. Then the job with the smallest penalty is assigned to the end of the partial schedule. The above step is repeated until the schedule is complete. McCormick et al. [13] have also observed that the heuristic that assigns more weight to idle time and blocking time on the bottleneck station performs better. In the rest of this section, we discuss two other effective heuristics for the cyclic scheduling problem.

The first approximate solution procedure we propose is a constructive heuristic. The heuristic starts with a partial MPS schedule of two jobs, along with a list of the remaining jobs. The first job on the list is used to create partial MPS schedules of size three, while keeping the relative positions of the first two jobs the same as in the partial MPS schedule. Among these newly formed partial schedules, we pick the one with the smallest cycle time. We then repeat the same procedure with an initial partial MPS schedule of three jobs and with the second job on the initial list. We repeat the same steps until all jobs are scheduled, i.e., we have a complete MPS schedule. The heuristic creates $\frac{1}{2}(n+1)(n-2)$ partial schedules, and for each partial schedule the cycle time can be computed in $O(nm^2)$ time. Therefore the overall time complexity of the heuristic is $O(n^3m^2)$. The heuristic can be formally described as follows:

Heuristic LIST:

Step 0: Let $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ be an initial sequence of jobs in an MPS.
Initialize $\sigma(1) = \pi(1)$, $\sigma(2) = \pi(2)$ and $k = 3$.
Step 1: For $i = 1$ to k

- Set $\sigma_i(j) = \sigma(j)$ for $j = 1, \ldots, i-1$, and $i > 1$.
- Set $\sigma_i(i) = \pi(k)$, and $\sigma_i(j) = \sigma(j-1)$ for $j = i+1, \ldots, k$.
- Compute the cycle time λ_i^k of partial schedule σ_i.

Step 2: Let $l = \arg(\min_{1 \le i \le k} \lambda_i^k)$.
Set $\sigma(j) = \sigma_l(j), j = 1, \ldots, k$.
Set $k = k+1$, if $k \le n$ go to Step 1, otherwise stop, σ is the resulting MPS schedule.

Note that in the above heuristic procedure the initial sequence of jobs is an important input to the heuristic. We will use three different initial sequences.

- Jobs are sequenced according to nonincreasing total processing times (we refer to this version of the heuristic as LIST($\sigma_{\max}$)),
- Jobs are sequenced randomly (referred to as LIST(σ_{ran})).
- Jobs are sequenced according to non-decreasing total processing times (referred to as LIST($\sigma_{\min}$)),

Another alternative approximate solution procedure is the early termination of the branch-and-bound (B&B) algorithm. One of the advantages of our lower bounding scheme is that in the process of obtaining a lower bound we also obtain a feasible schedule, as we mentioned in Section 4. We have used the best heuristic solution obtained as an initial upper bound for the B&B procedure and terminated the search after a prespecified number of

Table 2. Performance of heuristics for 20 jobs per MPS problems.

Performance	LIST (σ_{max})	LIST (σ_{ran})	LIST (σ_{min})	McCormick et al.	Early Termination
Best	0.56	0.51	0.20	1.33	0.14
Average	3.27	3.64	3.25	8.23	1.23
Worst	10.04	9.46	8.72	18.18	8.39

nodes were generated in the search tree. The best lower bound obtained so far in the search tree was also used to measure the performance of the proposed heuristic procedures.

The experimental test bed for our heuristics is generated in the following way. We created random problems with 20 and 25 nonidentical jobs per MPS. The processing times of the various jobs on the different stations were drawn from a (1–100) uniform distribution. The 20 jobs per MPS problems were tested on a flow line with eight stations and configuration $D = (0, 0, 1, 0, 1, 0, 0)$. We have experimented with 20 such random problems. The results are reported in Table 2. What we refer to as the performance of the heuristic is $(UB - LB/LB) \times 100$, where UB is the cycle time generated by the heuristic and LB is the best available lower bound for the problem, as obtained from the early termination of the B&B procedure (after 500 nodes were generated in the search tree). The heuristics for which we are reporting results are the LIST(σ_{max}), LIST(σ_{ran}), LIST(σ_{min}), McCormick et al. [13], and the heuristic run of the B&B procedure. For the McCormick et al. [13] heuristic we used ten different combinations of idle and blocking time penalties and reported the best cycle time obtained by these combinations. The first row of Table 2 tabulates the best performance of these heuristics over the 20 random problems, the second row reports their average performances and finally the last row reports the worst exhibited performance of the heuristics. Our results for 25 jobs per MPS problems are reported in a similar manner in Table 3. We report results for 20 random problems nine stations and configuration $D = (0, 2, 0, 0, 1, 0, 1, 0)$. A summary of the observed results is the following. For 20 (25) jobs per MPS problems the average CPU time for the LIST heuristic (over all its variations) was 0.45 (1.47) CPU seconds. The variations of the LIST heuristic exhibited quite similar CPU requirements, and that was the main reason for averaging the CPU time over all of them. Our implementation of the McCormick et al. [13] heuristic had an average CPU time of 0.011 (0.023) seconds. The B&B procedure had an average CPU time of 6.20 (10.00) seconds, excluding the time spent to obtain the first upper bound, i.e., time spent for the LIST and McCormick heuristics. The results of Tables 2 and 3 indicate that the LIST heuristic clearly dominates the McCormick et al. [13] heuristic for any initial sequence selection. We note that the complexities of the McCormick et al. [13] and LIST heuristics are $O(n^2m)$ and $O(n^3m^2)$, respectively, and the improved performance is attained at the expense of larger run times. Among the LIST heuristic variations, the LIST(σ_{min}) and LIST(σ_{ran}) seem to dominate the LIST(σ_{max}). The heuristic performance of the B&B pro-

Table 3. Performance of heuristics for 25 jobs per MPS problems.

Performance	LIST (σ_{max})	LIST (σ_{ran})	LIST (σ_{min})	McCormick et al.	Early Termination
Best	0.17	0.11	0.36	1.41	0.11
Average	2.98	2.63	2.66	7.76	1.35
Worst	9.56	8.82	10.25	17.21	7.97

cedure is very good, and though its computational burden is heavier, still it remains within a very reasonable range. Either the LIST or the heuristic execution of the B&B algorithm can be used effectively for large size real problems.

7. STABILITY OF CYCLIC SCHEDULES

In this section we address the concept of stability of cyclic schedules in flow lines, and present a simple procedure to generate stable schedules. As our discussion will show, the stability concept is closely related to the control of the WIP inventory levels of cyclic schedules, and thus the generation of stable schedules becomes an issue of importance for the scheduler. For example, two schedules having the same cycle time may have different WIP inventory levels in steady state, and our procedure can be used to determine the schedule having the smaller WIP inventory level in steady state without actually implementing the schedules. The control of the WIP level is of particular importance in flow lines with at least one infinite capacity buffer, where an earliest start schedule may result in unbounded WIP inventory levels.

First, we present a formal definition of stable cyclic schedules in flow lines following the work of Lee and Posner [9].

DEFINITION 1: Let σ be an MPS schedule and λ be its cycle time. A schedule is periodic at r_0 if

$$C(\sigma_{r+d}(i), j) - C(\sigma_r(i), j) = d\lambda$$

for some integer d, $i = 1, \ldots, n$, $j = 1, \ldots, m$ and $r \geq r_0$. When $d = 1$, such a schedule is said to be stable at r_0. In particular when $d = 1$ and $r_0 = 1$, such a schedule is simply said to be stable.

The following result of McCormick et al. [13] is important for our further discussion:

THEOREM 2: (McCormick et al. [13]) Let σ be an MPS schedule, and λ be its cycle time. Then in a flow line with finite capacity buffers there always exists an integer k such that

$$C(\sigma_{r+1}(i), j) - C(\sigma_r(i), j) = \lambda, r \geq k$$

for $i = 1, \ldots, n$, and $j = 1, \ldots, m$, where k is finite.

Theorem 2 guarantees that, in a flow line with blocking, i.e., with finite capacity buffers, every schedule becomes stable after a finite number of MPSs. Once the stable state is reached, we can compute the average WIP inventory level using Little's formula:

$$\text{WIP Inventory Level} = \frac{1}{\lambda} \times (\text{MPS Makespan}) \text{ batches/unit time},$$

because in the stable state every MPS will have the same makespan or waiting time in the system. We note that the value of k in Theorem 2 depends largely on the buffer configu-

Table 4. Stability of optimal schedules.

Flow Line	Optimal schedule stable at MPS		
	Minimum	Average	Maximum
$m = 3, D = (0, 0)$	1	1.36	2
$m = 3, D = (5, 0)$	1	4.60	20
$m = 3, D = (3, 3)$	1	3.74	17
$m = 4, D = (0, 0, 0)$	1	1.58	2
$m = 4, D = (0, 7, 0)$	1	9.48	95
$m = 4, D = (3, 3, 3)$	1	5.36	19
$m = 5, D = (0, 0, 0, 0)$	1	1.82	2
$m = 5, D = (2, 0, 3, 0)$	2	5.54	42
$m = 5, D = (3, 3, 3, 3)$	1	5.42	35

ration of the flow line under consideration. In Table 4 we present our computational experience with different flow line configurations for eight jobs per MPS problems. For each flow line configuration we created 50 problems using the procedure described in Section 5.2, and determined an optimal schedule for each problem. In Table 4 we present the minimum, average, and maximum k values over 50 test problems for each flow line configuration. As the results of Table 4 indicate, even with finite capacity buffers, reaching stability may require the completion of a number of MPSs. In such problems, determining the MPS makespan without actually implementing the schedule would be of particular importance for the scheduler.

We now restrict our attention to stable schedule generation procedures, which are actually simple MPS release rules for the flow line environment. Lee and Posner [9] have developed a graph theoretic framework for stable schedule generation in job shops with infinite capacity buffers. The complexity of their procedure is $O(N^3)$, where N is the number of operations in the job shop problem. Here we present a stable schedule generation scheme for flow lines with any buffer configuration. The complexity of our procedure is $O(nm^2)$, where n is the number of jobs in the MPS, and m is the number of stations (i.e., there are nm operations). Our stable schedule generation scheme uses the one MPS per cycle time release rule which inserts appropriate idle times before the operations of the 1st job of the 1st MPS so that the desired schedule stability is achieved at the very first MPS.

We present below in detail our algorithm to generate stable schedules in flow lines with or without blocking.

Algorithm STABLE

Step 0: Compute the cycle time λ of schedule σ.
Step 1: Compute $C(\sigma_1(i), j), i = 1, \ldots, n, j = 1, \ldots, m$, and $C(\sigma_2(1), j), j = 1, \ldots, m$ using the one MPS per cycle time release rule.
Step 2: For $k = 2$ to m
Compute $\alpha_k = C(\sigma_2(1), k) - C(\sigma_1(1), k) - \lambda$.
Delay 1st job of the 1st MPS on station k by an amount of time equal to α_k.
Compute $C(\sigma_1(i), j), i = 1, \ldots, n, j = 1, \ldots, m$, and $C(\sigma_2(1), j), j = 1, \ldots, m$, using the one MPS per cycle time release rule and inserted idle times $\alpha_j, j = 1, 2, \ldots, k$.

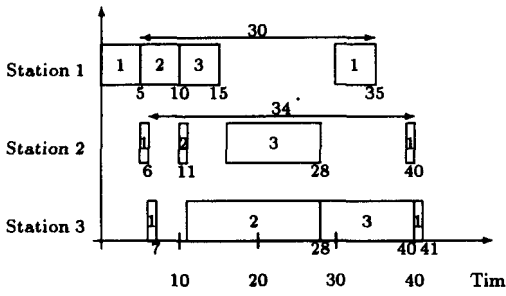

Figure 4(a): Completion times for Step 1.

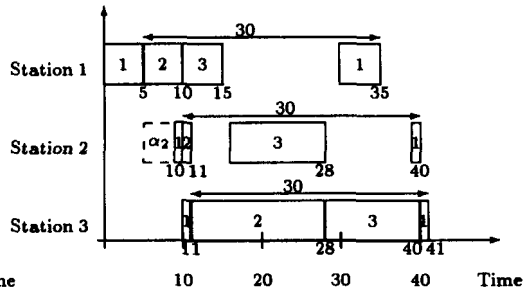

Figure 4(b): Completion times for Step 2.

THEOREM 3: Algorithm STABLE generates a stable schedule with the same throughput rate as the earliest start schedule in $O(nm^2)$ time.

PROOF: We present the proof of Theorem 3 in Appendix B.

EXAMPLE 2: Consider the three jobs per MPS three station problem with zero capacity buffers between the stations, and the following processing times:

	Station		
Job	1	2	3
1	5	1	1
2	5	1	17
3	5	12	12

Suppose $\sigma = (1, 2, 3)$. In Step 0 we obtain the cycle time $\lambda = 30$. We calculate the desired completion times according to Step 1 (i.e., using the one MPS per cycle release rule). The completion times are given in Figure 4(a). In the first iteration of Step 2 we have $\alpha_2 = 40 - 6 - 30 = 4$. After the first job of the 1st MPS is delayed by 4 time units on the second station, we obtain the completion times presented in Figure 4(b). We now let $k = 2 + 1 = 3$. In the 2nd iteration of Step 2 we get $\alpha_3 = 41 - 11 - 30 = 0$, and therefore no idle time is inserted on Station 3. The resulting schedule is stable at the very first MPS. The MPS makespan of this schedule is equal to 40.

8. CONCLUDING REMARKS

Cyclic scheduling policies have proved to be effective approaches in operating medium-to-high volume repetitive manufacturing flow line environments. In the cyclic scheduling environment, the control of the WIP inventory level of the system becomes easier and the complexity of the scheduling problem is drastically reduced. Though the cyclic scheduling problem is *NP*-hard, optimal implicit enumeration procedures, as the one we presented in this paper, can handle medium size problems (e.g., in our computational results we were able to handle rather easily up to 10 station, 12 nonidentical jobs per MPS problems with various buffer configurations). For large size problems the use of heuristic procedures becomes a necessity. Our simple LIST heuristic can handle effectively (within less than 3–4%

deviation from the lower bound on average) most of the real size problems (20 to 25 jobs per MPS). For a computational requirement of less than 15 CPU seconds, a heuristic run (early termination) of our optimal procedure can lead to schedules within 2% deviation from the lower bound on average. We feel that the exhibited computational performance of the suggested solution approaches in this paper, both in terms of effectiveness and efficiency, meet the scheduling needs of the flow line manager.

Finally, our discussion of the stability of schedules concept for flow line environments brings to the attention of operating managers the need for MPS (batch) release rules in the repetitive manufacturing environment of a flow line. Our suggested stable schedule generation rule has as its main advantage its simplicity, with a subsequent intuitive appeal, and an ease of implementation in any flow line environment, regardless of the buffer configuration of the line.

APPENDIX

A. Proof of Property 2

Let $\mu' \in \Omega$ be a multiplier vector. We want to show that

$$f_{AP}(\mu) + \zeta(\mu' - \mu)^t \geq f_{AP}(\mu'). \tag{13}$$

Note that $(\mu' - \mu)^t$ denotes the transpose of the vector $(\mu' - \mu)$. We can rewrite the LHS of (13) as follows

$$f_{AP}(\mu) + \zeta(\mu' - \mu)^t = f_{AP}(\mu) + \sum_{\tau \in T^*} \left(\left(\sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^{\tau} x_{i,j}^{*} - f_{AP}(\mu) \right) (\mu_\tau' - \mu_\tau) \right)$$

$$= f_{AP}(\mu) + \left(\sum_{\tau \in T^*} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^{\tau} x_{i,j}^{*} \mu_\tau' \right) - f_{AP}(\mu) \left(1 + \sum_{\tau \in T^*} (\mu_\tau' - \mu_\tau) \right)$$

Now using $\sum_{\tau \in T^*} \mu_\tau = 1$ and $\sum_{\tau \in T^*} \mu_\tau' = 1$, we obtain

$$f_{AP}(\mu) + \zeta(\mu' - \mu)^t = \sum_{\tau \in T^*} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^{\tau} x_{i,j}^{*} \mu_\tau' \geq f_{AP}(\mu').$$

The last inequality follows from the fact that $x_{i,j}^{*}$, $i, j = 1, \ldots, n$ is a feasible solution for the $(LRAP(\mu'))$ problem and therefore

$$\sum_{\tau \in T^*} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^{\tau} x_{i,j}^{*} \mu_\tau'$$

is an upper bound on $f_{AP}(\mu')$. □

B. Proof of Theorem 3

We now formally prove that Algorithm STABLE always generates a stable schedule. The following additional notation is needed for the proof. We denote as $L_j, j = 1, \ldots, m$, the weight of the maximum-weighted path around the cylinder that visits vertex $(\sigma_1(1), j)$.

We first assume that $C(\sigma_2(1), j) - C(\sigma_1(1), j) = \lambda, j = 1, \ldots, k-1$ and $C(\sigma_2(1), k) - C(\sigma_1(1), k) > \lambda$ at the $(k-1)$-st iteration of Step 2 and then show that by delaying the 1st job of the 1st MPS on station k by an amount equal to $\alpha_k = C(\sigma_2(1), k) - C(\sigma_1(1), k) - \lambda$ we get

$$C(\sigma_2(1), k) - \bar{C}(\sigma_1(1), k) = \lambda,$$

where $\bar{C}(\sigma_1(1), k)$ denotes the completion time of the 1st job of the 1st MPS on station k after the idle time is

inserted. We also show that at the end of the $(k-1)$-st iteration of Step 2 our initial assumptions still hold. The proof is presented in a logical sequence of appropriately numbered steps for the convenience of the reader.

1. First we need to show that $C(\sigma_2(1), k) - C(\sigma_1(1), k) \geq \lambda$. We have $C(\sigma_2(1), k) \geq C(\sigma_2(1), k-1) + p_{\sigma(1),k}$ and $C(\sigma_1(1), k) = C(\sigma_1(1), k-1) + p_{\sigma(1),k}$. Therefore $C(\sigma_2(1), k) - C(\sigma_1(1), k) \geq C(\sigma_2(1), k-1) - C(\sigma_1(1), k-1) = \lambda$.

2. Next we note that vertex $(\sigma_1(1), k)$ can be delayed by Δ_k, where Δ_k is given by

$$\Delta_k = C(\sigma_2(1), k) - C(\sigma_1(1), k) - L_k,$$

without increasing $C(\sigma_2(1), k)$, thus Δ_k gives us the minimum slack on all paths that go to vertex $(\sigma_2(1), k)$ through vertex $(\sigma_1(1), k)$. It is also clear that $0 \leq \alpha_k \leq \Delta_k$, because by definition $L_k \leq \lambda$.

3. Now using Part 2 we see that when we delay vertex $(\sigma_1(1), k)$ by an amount equal to α_k, $C(\sigma_2(1), k)$ is not increased and

$$C(\sigma_2(1), k) - \bar{C}(\sigma_1(1), k) = \lambda,$$

where $\bar{C}(\sigma_1(1), k)$ denotes the completion time of the 1st job of the 1st MPS on station k after the idle time is inserted.

4. Finally, we need to show that when we delay vertex $(\sigma_1(1), k)$ by an amount equal to α_k, $C(\sigma_2(1), j), j = 1, \ldots, k-1$, are not increased. (In this part we consider the zero capacity buffers case, however the same arguments hold, with minor modifications, for flow lines with combination of finite and infinite capacity buffers.) Recursive relationship (1) can be rewritten as

$$C(\sigma_2(1), k) = \max\{C(\sigma_1(n), k+1), C(\sigma_2(1), k-1) + p_{\sigma(1),k}\}.$$

We should only consider the case where $C(\sigma_2(1), k) = C(\sigma_1(n), k+1)$, because otherwise $\alpha_k = 0$. From Part 3, we know that $C(\sigma_2(1), k)$ is not going to be increased when we delay vertex $(\sigma_1(1), k)$ by α_k time units. Therefore $C(\sigma_1(n), k+1)$ is not increased as well, otherwise $C(\sigma_2(1), k)$ would have been increased. Now we have

$$C(\sigma_1(n), k+1) = \max\{C(\sigma_1(n-1), k+2), C(\sigma_1(n), k) + p_{\sigma(n),k+1}\}$$

Note that $C(\sigma_1(n-1), k+2)$ and $C(\sigma_1(n), k)$ cannot be increased simultaneously. We now consider the vertex whose completion time is not increased and repeat the same procedure for that particular vertex. By doing so we form a chain of vertices whose completion times have remained the same after vertex $(\sigma_1(1), k)$ is delayed by α_k time units. Any chain of vertices that is formed using the above procedure will eventually end up at vertex $(\sigma_1(1), 1)$, because according to our stable schedule generation scheme $C(\sigma_1(1), 1)$ will remain the same throughout the Algorithm STABLE. The vertices that are included in the chain will form an envelope of non-increased completion times and therefore no completion time in the lower left part of this chain can be increased by delaying vertex $(\sigma_1(1), k)$ by α_k time unit. Note that vertices $(\sigma_2(1), j), j < k$ lie in the lower left part of the envelope and therefore they cannot be increased by delaying vertex $(\sigma_1(1), k)$ by α_k time units.

This completes the proof, because our initial assumption are met for $k = 2$, when we use the one MPS per cycle time release rule, and we construct the stable schedule by repeating Steps 1 and 2 of Algorithm STABLE m-1 times. The complexities of Steps 0, 1, and 2 are $O(nm^2)$, $O(nm)$, and $O(nm^2)$, respectively, and therefore the overall complexity of Algorithm STABLE is $O(nm^2)$. □

REFERENCES

[1] Bartholdi, J.J., "A Guaranteed-Accuracy Round-Off Algorithm for Cyclic Scheduling and Set Covering," *Operations Research,* **29,** 501–510 (1981).

[2] Bowman, R., and Muckstadt, J., "Stochastic Analysis of Cyclic Schedules," *Operations Research,* **41,** 947–958 (1993).

[3] Dauscha, W., Modrow, H.D., and Neumann, A., "On Cyclic Sequence Types for Constructing Cyclic Schedules," *Zeitschrift für OR,* **29,** 1–30 (1985).

[4] Fuxman, L., Mixed-Model Assembly Lines Under Cyclic Production: Models for Productivity Enhancement and Teamwork. Working paper, Wharton School, University of Pennsylvania, 1994.

[5] Goffin, J.L., "On Convergence Rates of Subgradient Optimization Methods," *Mathematical Programming,* **13,** 329–347 (1977).
[6] Graves, S.C., Meal, H.C., Stefak, D., and Zeghmi, A.H., "Scheduling of Re-entrant Flowshops," *Journal of Operations Management,* **3,** 197–207 (1983).
[7] Hall, R.W., "Cyclic Scheduling for Improvement," *International Journal of Production Research,* **26,** 457–472 (1988).
[8] Hitz, K.L., "Scheduling of Flexible Flowshops," Technical report, MIT, Cambridge, MA, 1979.
[9] Lee, T.-E., and Posner, M.E., "Performance Measures and Schedules in Periodic Job Shops," Working paper 1990-012, Dept. of ISE, The Ohio State University, 1991.
[10] Lei, L., and Wang, T.J., "The Minimum Common-Cycle Algorithm for Cyclic Scheduling of Two Hoists with Time Window Constraints," *Management Science,* **37,** 1629–1639 (1991).
[11] Loerch, A.G., and Muckstadt, J.A., "An Approach to Production Planning and Scheduling in Cyclically Scheduled Manufacturing Systems," *International Journal of Production Research,* **32,** 851–872 (1994).
[12] Matsuo, H., "Cyclic Sequencing Problems in the Two-Machine Permutation Flowshop: Complexity, Worst-Case and Average-Case Analysis," *Naval Research Logistics Quarterly,* **37,** 679–694 (1990).
[13] McCormick, S.T., Pinedo, M.L., Shenker, S., and Wolf, B., "Sequencing in an Assembly Line with Blocking to Minimize Cycle Time," *Operations Research,* **37,** 925–935 (1989).
[14] McCormick, S.T., Pinedo, M.L., and Wolf, B., "The Complexity of Scheduling a Flexible Assembly System," Working paper, University of British Columbia, Vancouver BC, Canada, 1987.
[15] Nemhauser, G.L., and Wolsey, L.A., *Integer and Combinatorial Optimization,* Wiley, New York, 1988.
[16] Rao, U.S., "A Study of Sequencing Heuristics for Periodic Production Environment," Working paper, SORIE, Cornell University, Ithaca, NY, 1993.
[17] Rao, U.S., and Jackson, P.L., "Subproblems in Identical Job Cyclic Scheduling: Properties, Complexity, and Solution Approaches," Technical report 956, SORIE, Cornell University, Ithaca, NY, 1991.
[18] Roundy, R.O., "Cyclic Schedules for Job Shops with Identical Jobs," *Mathematics of OR,* **17,** 842–865 (1992).
[19] Whybark, D.C., "Production Planning and Control at Kumera Oy," *Production and Inventory Management,* First Quarter (1984).
[20] Wittrock, R.J., "Scheduling Algorithms for Flexible Flow Lines," *IBM Journal Research Review,* **29,** 401–412 (1985).

Manuscript received July 1992
Revised manuscript received December 1994
Accepted May 17, 1995