SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853-3801

TECHNICAL REPORT NO. 941

A

November 1990

STOCHASTIC ANALYSIS OF CYCLIC SCHEDULES

By

R.A. Bowman¹ and J.A. Muckstadt²

¹Union College, Schenectady, NY.

²This author's research was supported in part by grants from the IBM Corporation and by the National Science Foundation grant DDM-8819542.

ABSTRACT

A cyclic schedule is a sequence of operations on a set of machines that is repeated indefinitely. We model cyclic schedules as Markov chains and use ergodic theory to analyze and improve the performance of cyclic schedules in environments with machine breakdowns, yield losses, and other sources of variability. The concept of cyclic task criticality is developed as a natural extension of task criticalities in PERT networks. We show that cyclic task criticalities can and should be used to guide the management of cyclic schedules.

The problem of scheduling manufacturing operations has received considerable attention during the past few decades. Many objectives have been proposed and numerous techniques for meeting those objectives have been developed and analyzed. Most of the objectives involve meeting some sort of throughput or due-date criteria with minimal work-in-process (WIP) inventory. Typically, a graph relating WIP vs. throughput is constructed for the application of interest and a point in the knee of the curve (with high throughput but before the WIP level gets too large) is selected. An objective of scheduling techniques is to enable a favorable shift of the entire curve. For example, a fundamental aspect of just-in-time (JIT) systems is to reduce inventory levels gradually. This reduction enables the source of problems to be identified and corrected. This corrective action, in turn, enables further reductions in inventory, which will then expose new problem areas. This process of continuous improvement never ends.

Cyclic schedules are used in industry for a number of reasons. Once an effective sequence of operations on each machine is established, it can then be followed repeatedly. The advantages of such an approach are the ability to plan the movement of material throughout one or more facilities, to communicate the schedule requirements throughout the facilities and to identify and remove problem spots (due to the repetition involved) more easily. The primary disadvantage of cyclic scheduling (as with any preset scheduling technique) is that variability (machine breakdowns, yield losses, variable processing times, changes in demand required) will certainly keep the actual implementation of a schedule from occurring exactly as planned. Our goal in this paper is to present a method for analyzing and improving the performance of a cyclic schedule by evaluating the effect of stochastic variability on performance. As with JIT systems, the basic approach will be to analyze the performance of a schedule, identify the areas where improvement will have the most effect, make the improvements, and repeat the process.

In Section 1, we briefly review some related literature. We describe and analyze the Markovian PERT networks approach given by Kulkarni and Adlakha (1986). We show that a regular schedule with a specified start and end (a finite fixed sequence schedule) can be modeled as a Markovian PERT

network and the algorithms of Kulkarni and Adlakha (1986) can be used to calculate much useful information about the performance of such schedules. Extension of many of these algorithms to cover events rather than states is also described since this is of key interest for a cyclic scheduling application.

In Section 2, we present theoretical justification for the rather intuitive idea that management should focus its attention on the tasks that are likely to be on the critical path (those with the highest arc criticality in the PERT network). We describe relationships among arc criticalities in a Markovian PERT network and exploit them in an algorithm to calculate the exact criticality of each arc in the network. This algorithm does not require enumeration of all paths in the network nor does it require intermediate assessment of path criticalities as does the one described by Kulkarni and Adlakha (1986).

In Section 3, we extend the Markov chain model to cover the case of cyclic schedules. We show that the ergodic performance of a cyclic schedule can be assessed using the algorithms for regular Markovian PERT networks as building blocks in combination with standard ergodic theory of Markov chains. We define the concept of cyclic task criticalities and develop an algorithm to calculate the exact cyclic criticality for each task in the cyclic schedule.

The models of Sections 2 and 3 are used to improve the performance of a finite or cyclic schedule as described in Section 4. The Markov chain model naturally leads to special simulation approaches to exploit the Markovian structure. An algorithm is presented for allocating a budget effectively. The solutions indicate possible management actions for improving the performance of a cyclic schedule. The algorithm can use either the exact analysis algorithms of Sections 2 and 3 or simulations based on the Markov models.

We present computational results in Section 5. Because the analysis algorithms are highly dependent on problem structure, it is impossible to provide a complete, general assessment of their efficiency. Results are presented for several problems of various sizes; however, to show the relative accuracy and efficiency of a Markovian simulation approach, we compare our approach for improving schedule performance (in which variability is explicitly considered) to a linear programming approach (which does not consider the effect of variability).

A brief discussion of modeling considerations is given in Section 6 along with recommendations for future research.

1. LITERATURE REVIEW

Maxwell, Muckstadt, Thomas, and VanderEecken (1983) stress the need for an integrated hierarchical approach for selecting production lot sizes, buffer stocks, and detailed schedules. In this vein, cyclic schedules are the natural complementary technique for implementing lot sizes or production intervals established for the case where demands are reasonably consistent, that is, where the coefficient of variation of the demand process is less than about .15. In particular, recent work on lot sizing in which demand is constant and continuous (Roundy (1985, 1986), Maxwell and Muckstadt (1985), Jackson, Maxwell, and Muckstadt (1988)) is based on the concepts of nestedness and power-of-two (times a base planning period) reorder intervals. These results establish production intervals that are realistic and provably effective for such demand patterns. The production intervals that result from these algorithms are natural inputs to cyclic scheduling algorithms. The powers-of-two constraint ensures that the cycle time is equal to the longest production interval and the number of lots per cycle for each operation on each product is an integer power of two.

Usually each operation on each product is done on a specified single machine. Thus we assume each task is one operation on one lot on a particular product on a specified machine. A cyclic schedule then indicates the sequence of tasks to conduct on each machine. The precedence structure for tasks in a cyclic schedule includes both material flow and machine sequence constraints. In general, even in the deterministic case, finding the best precedence structure is a hard combinatorial optimization problem. For a given precedence structure, the best timing of tasks within a cycle in the deterministic setting can be determined by a simple linear program (Roundy (1988)).

Hitz (1980), Graves, et al. (1983), Wittrock (1985), and McCormick (1988) all study cyclic scheduling in specific (deterministic) settings. Del Pilar and Jackson (1990) address the general production scheduling problem by using the lot sizes given by the algorithm of Jackson, Maxwell, and

Muckstadt (1988) and heuristically sequencing the resulting lots. Whybark (1984) describes an implementation of a cyclic scheduling approach and gives an excellent discussion of practical issues, costs and benefits associated with the approach.

All the previously mentioned works are based on the assumption that no uncertainty is present; there is no variability in processing times and there are no machine breakdowns. Zhang and Graves (1989) study cyclic scheduling in a stochastic environment that, in particular, allows for machine breakdowns. They take a cyclic precedence structure as input and address the problem of selecting start times for the tasks (within a cycle) to minimize the weighted (by importance) sum of ergodic delays in starting time for the tasks. Our model differs by assuming that each task starts as soon as its precedent tasks are completed. The decision to idle a machine is expressed through precedence relations rather than times.

The precedence structure among the tasks that are to be completed within a cyclic schedule gives rise to a cyclic network. If a finite number of cycles is considered with a specified initial schedule status, the network is a standard network with a single start node and a single end node. If the task times (arc lengths) are random variables, this network would be a PERT network. Thus, the extensive literature on PERT networks is applicable to the problem and, in fact, provides many of the building blocks for the approach taken in this paper. See Elmaghraby (1977) for an excellent review of PERT (as well as deterministic) network theory.

Kulkarni and Adlakha (1986) describe the analysis of Markovian PERT Networks (MPN's), which are PERT networks with independent exponentially distributed task times. As they point out, non-exponential task times can be approximated with mixtures of sums of exponentials while retaining the structure of an MPN. By classifying tasks (arcs) as active (being executed), dormant (finished, but at least one task to which it is precedent cannot start because some other precedence constraint has not been satisfied) or idle (neither active nor dormant), they show that $\{s(t), t \geq 0\}$ is a continuous time Markov chain, where $s(t)$ keeps track of which arcs are active, dormant, or idle at time t . When modeling a schedule as a PERT network, the existence of machines allows the state $(s(t))$ to be more

compactly and intuitively represented. The following example will help clarify how to model a schedule as an MPN, which will allow the theorem of Kulkarni and Adlakha (1986) to be re-stated.

Consider the problem of completing 4 tasks on two machines such that task 4 represents the next operation on the same lot of material as task 1. Tasks 2 and 3 are independent of each other and of tasks 1 and 4 (see Figure 1).

Figure 1. Finite Schedule Example

The associated network for this schedule is shown in Figure 2. T_i is the time associated with task i . Nodes s and t represent the start and finish of the schedule, respectively, while node i ($i = 1, 2, 3, 4$) represents the start of task i . The requirement that task 4 cannot begin before task 1 finishes is a material flow precedence constraint while the requirements that task 1 finish before task 2 begins and task 3 finish before task 4 begins are sequence precedence constraints. The time to complete the schedule is equal to the length of the longest path (or critical path) in the network.

Figure 2. Corresponding Finite Network

If the T_i 's are random variables, then the critical path (hence the time to complete the schedule) is random as well. This network is a special case of a PERT network. Van Slyke (1963)

described how to use Monte Carlo simulation to estimate the critical path length distribution and moments as well as path criticality and arc criticality, where the criticality of a path is defined to be the probability that the path is the longest path in the network. The criticality of an arc is defined to be the probability that the arc is on the longest path in the network.

Assume that the first two moments of each task time are known (or can be estimated) and that these moments include variability associated with the processing times, setups, yield, and machine breakdowns. If the variance is less than or equal to the square of the mean (i.e. the squared coefficient of variation (SCV) is less than or equal to 1) for each task, the first two moments of each task time can be matched by representing each task time as a sum of exponentially distributed random variables. Bowman (1990) presents an algorithm for generating the means of these random variables that minimizes the number of random variables being summed. A method for handling the case of SCV's greater than 1 is also developed in that reference. For ease of reading, we assume hereafter that all task times are exponentially distributed so that the arcs in the network correspond one-to-one with the tasks in the schedule. We give results for arcs and show how these results can easily be extended to tasks that are series of arcs.

To begin our analysis, let us arbitrarily number the machines from 1 to m and then number the arcs consecutively working from left to right beginning with machine 1, and continuing until arcs related to machine m are numbered. The state of the network (schedule) is given by the vector s :

$$s = [s_i, i = 1, 2, \dots, m].$$

where s_i is the number of the arc most recently completed on machine i ($\equiv 0$ if nothing has yet been completed on machine i).

The main theorem of Kulkarni and Adlakha (1986) can now be re-stated for the scheduling application as follows:

Theorem 1.

Let

$S = \{s: s \text{ is feasible for the precedence constraints}\},$

$\theta_j = \text{mean length of arc } j,$

$PR(a^j) = \{\text{arc } a^i: a^i \text{ is precedent to } a^j\},$

$M(a^j) = \text{machine corresponding to arc } j,$

$L(k) = \text{highest number of an arc corresponding to machine } k,$

$s(t) = \text{state at time } t,$

$\vec{e}_j = \text{the } m\text{-vector with } 1 \text{ as the } j\text{th element and } 0\text{'s elsewhere.}$

Then:

$\{s(t), t \geq 0\}$ is a continuous time Markov chain on S with infinitesimal generator

matrix $Q = [q(s^A, s^B)], s^A \text{ and } s^B \in S, \text{ where, for } j = 1, \dots, m:$

$$q(s, s + \vec{e}_j) = 1/\theta_{s_j+1} \text{ if } s_{M(a^i)} \geq i \forall a^i \in PR(a^{s_j+1}) \text{ and } s_j + 1 \leq L(j), \quad (1)$$

$$q(s, s) = - \sum_{j=1}^m q(s, s + \vec{e}_j), \quad (2)$$

$$q(s^A, s^B) = 0, \text{ otherwise.} \quad (3)$$

Transitions in the Markov chain correspond to tasks being completed in the schedule. Generating the states and infinitesimal generator matrix is simply a matter of examining the precedence constraints to see which tasks can be completed from each state. Note that in the statement of this theorem (and theorems that are presented later in this paper), we suppress the dependency of the state on the time t to simplify notation.

Kulkarni and Adlakha (1986) show that the state where all tasks are idle is the unique absorbing state and that the time to complete the project is the time until absorption occurs. For our application, the absorbing state occurs when all tasks have been completed ($s_i = L(i) \forall i$). The schedule completion time is a phase-type random variable corresponding to the time until absorption of the Markov chain of Theorem 1. Bowman (1990) provides an algorithm for generating the states $s^1, s^2, \dots, s^n$ (we will always use superscripts to number the states and subscripts to denote elements of

the state vector) and the infinitesimal generator matrix Q , such that q_{ij} is the rate of transitions from state s^i to state s^j given that the schedule is in state s^i . This algorithm numbers the states in increasing order of the number of tasks that must be completed to reach the state. This means that s^n is the absorbing state. States that require the same number of tasks to be completed are said to be on the same level. For example, the schedule of Figure 1.1 has $s^1 = \langle 0,0 \rangle$ as the only state in level 0 (0 tasks complete), $s^2 = \langle 1,0 \rangle$ and $s^3 = \langle 0,3 \rangle$ in level 1, $s^4 = \langle 2,0 \rangle$ and $s^5 = \langle 1,3 \rangle$ in level 2, $s^6 = \langle 2,3 \rangle$ and $s^7 = \langle 1,4 \rangle$ in level 3, and $s^8 = \langle 2,4 \rangle$ as the absorbing state in level 4. The positive entries in the Q matrix are $q_{12}, q_{13}, q_{24}, q_{25}, q_{35}, q_{46}, q_{56}, q_{57}, q_{68}$, and q_{78} .

We illustrate the algorithms given by Kulkarni and Adlakha (1986) by calculating the expected time to complete the schedule (reach state s^n) given that we are in state s^j . This time is denoted by $EF(s^j)$. We see that $EF(s^n) = 0$, and

$$EF(s^j) = \frac{1}{-q_{jj}} + \sum_{i>j} \frac{q_{ji}}{-q_{jj}} EF(s^i), \text{ for } j = n-1, n-2, \dots, 2, 1. \quad (4)$$

For example, for the schedule of Figure 1.1, we have:

$$EF(s^5) = \frac{1}{-q_{55}} + \frac{q_{56}}{-q_{55}} EF(s^6) + \frac{q_{57}}{-q_{55}} EF(s^7).$$

Of course, $EF(s^1)$ is the expected time to complete the schedule. Algorithms are also given to calculate (1) the probability that state s^j is reached ($P(s^j)$), (2) the variance of the time from the reaching of a state until the schedule is completed, (3) the expected value ($ET(s^j)$) and variance of the time until state s^j is reached (given that it is reached), and (4) the distribution of the schedule completion time (numerically evaluated to a desired accuracy). Bowman (1990) shows how these algorithms can be revised to give information about events as well as states. As an example, an event could be the completion of operation 1. Let $ETE(E^k)$ be the expected time until event k occurs, and $H(E^k)$ be the set of states for which event k has already happened. Then

$$ETE(E^k) = \sum_{\substack{i: \\ i < j \\ s^i \notin H(E^k)}} \sum_{\substack{j: \\ s^j \in H(E^k)}} P(s^i) \left(\frac{q_{ij}}{-q_{ii}} \right) \left[ET(s^i) + \frac{1}{-q_{ii}} \right]. \quad (5)$$

The event will happen in a transition from a state $s^i \notin H(E^k)$ to a state $s^j \in H(E^k)$, and the formula is just the average of the time until such transitions occur weighted by the probability that they occur. For example, if E^k is the event that operation 4 has started in the schedule of Figure 1.1, then

$$H(E^k) = \{s^5, s^6, s^7, s^8\}, \quad \text{and}$$

$$\begin{aligned} ETE(E^k) &= P(s^2) \left(\frac{q_{25}}{-q_{22}} \right) [ET(s^2) + \frac{1}{-q_{22}}] + P(s^3) \left(\frac{q_{35}}{-q_{33}} \right) [ET(s^3) + \frac{1}{-q_{33}}] \\ &\quad + P(s^4) \left(\frac{q_{46}}{-q_{44}} \right) [ET(s^4) + \frac{1}{-q_{44}}]. \end{aligned}$$

2. TASK CRITICALITIES IN FINITE SCHEDULES

Suppose we find that the expected time to complete the schedule is too large or that the probability that a certain product being completed by its due date is too low. Management usually can take action to improve the situation. For example, effort could be concentrated on speeding up a particular task, or parts could be purchased from an outside vendor. These examples mathematically correspond to reducing some subset of arc length means at some cost. Intuitively, management would like to spend its money or resources on tasks that “hold things up”. A task “holds things up” if its corresponding arc is on the critical path. Arc criticality would seem to be important information needed to make such decisions. The following theorem justifies this intuition.

Theorem 2. *Let T be the time to absorption of the Markov chain described in Theorem 1. Let $C(a^j)$ be the criticality of arc a^j and let the r.v. X_j be its length (which has mean θ_j). Then:*

$$\frac{\partial E(T)}{\partial \theta_j} = C(a^j) E(X_j \mid \text{arc } a^j \text{ is on the critical path}) / \theta_j. \quad (6)$$

A proof of the theorem is given in the appendix. The exact form of the theorem depends only on θ_j being a scale parameter and has easy extensions to other distributions of arc lengths as long as

the stochastic derivative $\frac{\partial E(T)}{\partial \theta_j}$ can be evaluated. If θ_j is a location parameter (e.g. the mean of a normal r.v.), then $\frac{\partial X_j}{\partial \theta_j} = 1$, and the theorem would conclude that $\frac{\partial E(T)}{\partial \theta_j} = C(a^j)$. In fact, $E[X_j \mid \text{arc } a^j \text{ is on the critical path}]$ is greater than θ_j ; but they are approximately equal for large networks, so that $\frac{\partial E(T)}{\partial \theta_j} \approx C(a^j)$. The appendix shows that $\frac{\partial E(T)}{\partial \theta_j}$ corresponds to the infinitesimal perturbation analysis (IPA) estimator and can be evaluated through simulation. $C(a^j)$ can be exactly evaluated using the algorithm to be given shortly.

The theorem does not apply at $\theta_j = 0$. A small change in θ_j from or to 0 represents a change in the precedence structure of the network and can only be evaluated by revising the Markov chain. Also, the theorem refers to arc length means which, in general, are only the building blocks of tasks. The following corollary extends the result to the task means.

Corollary 2.1. *If X_j , the length of task j , is the sum of independent exponential r.v.'s, has total mean θ_j , has criticality $C(t^j)$, and has a constant SCV, then*

$$\frac{\partial E(T)}{\partial \theta_j} = C(t^j)E(X_j \mid \text{task } j \text{ is on the critical path})/\theta_j. \quad (7)$$

A proof is given in the appendix. Corollary 3.1 establishes the existence of a unique consistent task criticality.

As mentioned, the gradient could be estimated through simulation or the criticalities could be evaluated exactly using the approach of Kulkarni and Adlaha (1986). However, their approach requires enumeration of paths in the network and evaluation of the criticality of each path. As we will shortly observe, task criticalities can be evaluated more directly and efficiently by exploiting the relationships between task criticalities. To establish this alternative procedure, we present an additional concept.

Definition. *The transition from state s^i to state s^j enables arc k if the arc that was completed (causing the transition) is in $PR(a^k)$ and all other arcs in $PR(a^k)$ have already been completed in state s^i . $E(a^k) = \{(i,j): \text{the transition from state } s^i \text{ to state } s^j \text{ enables arc } a^k\}$.*

Let $C(a^j | s^i)$ be the criticality of arc a^j given that state s^i is reached (defined $\forall a^j, s^i$ such that arc a^j is the next arc due to be completed on its machine in state s^i).

Theorem 3.

$$(1) \quad \forall s \in S \text{ such that } s_j + 1 = L(j), \text{ and } s_k = L(k) \quad \forall k \neq j: \\ C(a^{s_j+1} | s) = 1 \quad \text{and} \quad C(a^{s_j+1}) = P(s). \quad (8)$$

For all other states $s^i \in S$:

Let $s^x = s^i + \vec{e}^{M(a^j)}$ be the state entered from state s^i if arc a^j is completed.
Then, $\forall a^j, s^i$ s.t. $C(a^j | s^i)$ is defined:

$$(2) \quad C(a^j | s^i) = \sum_{k: (i,x) \in E(a^k)} \left(\frac{q(i,x)}{-q(i,i)} \right) C(a^k | s^x) + \sum_{\substack{\ell: q(i,\ell) > 0 \\ \ell \neq x}} \left(\frac{q(i,\ell)}{-q(i,i)} \right) C(a^j | s^\ell), \text{ and} \quad (9)$$

$$(3) \quad C(a^j) = \sum_{\substack{i: C(a^j | s^i) \\ \text{is defined}}} P(s^i) \sum_{k: (i,x) \in E(a^k)} \left(\frac{q(i,x)}{-q(i,i)} \right) C(a^k | s^x). \quad (10)$$

Proof:

(1) This follows immediately from the fact that if an arc has no successors, it will be on the critical path if and only if the state is reached where that arc is the only arc in the network remaining to be completed. (Otherwise, of course, the arc could be infinitesimally lengthened with no effect on the scheduled completion time).

(2) The first summation represents transitions caused by the completion of arc a^j . In this case, arc a^j is on the critical path from state s^i if and only if it enables an arc that is on the critical path from state s^x (or else it could be infinitesimally lengthened with no effect on T). The second term represents transitions from s^i caused by some arc other than arc a^j completing. In this case, arc a^j is critical from state s^i if and only if it is critical from the state s^ℓ that is entered. The events are weighted by their respective probabilities.

(3) If an arc does have successor arcs, it will be on the critical path if and only if it enables one of its successor arcs that is critical (otherwise it could be infinitesimally lengthened with no effect on T). The result follows from the unconditioning of the conditional first summation in (2).

This theorem suggests that one pass backwards through the Q matrix is needed to compute all criticalities, starting with the arcs covered by (1). The upper triangularity of Q ensures that all terms will be available when needed. Storage is made more efficient by the fact that it is only necessary to store two levels of the conditional arc criticalities - the level being computed and the level with one more arc completed.

Example: *For our example problem assume that the task times are not only exponential but have unit mean. Then, part 1 of Theorem 3 states that*

$$\begin{aligned} C(a^2|s^7) = 1 \quad \text{and} \quad C(a^2) = P(s^7) = \frac{3}{8}, \quad (C(a^4|s^7) = 0). \\ C(a^4|s^6) = 1 \quad \text{and} \quad C(a^4) = P(s^6) = \frac{5}{8}, \quad (C(a^2|s^6) = 0). \end{aligned}$$

Starting with states 6 and 7 (level 3) we work back through the lower numbered states and levels using part 2 of Theorem 3, and obtain the following results:

$$\begin{aligned} C(a^2|s^5) = \frac{1}{2} \quad C(a^2|s^7) = \frac{1}{2}, & \quad C(a^2|s^2) = \frac{1}{2} \quad C(a^2|s^5) = \frac{1}{4}, \\ C(a^4|s^5) = \frac{1}{2} \quad C(a^4|s^6) = \frac{1}{2}, & \quad C(a^3|s^2) = \frac{1}{2} \quad C(a^3|s^4) + \frac{1}{2} \quad C(a^4|s^5) = \frac{3}{4}, \\ C(a^3|s^4) = C(a^4|s^6) = 1, & \quad C(a^1|s^1) = \frac{1}{2} \quad C(a^2|s^2) + \frac{1}{2} \quad C(a^1|s^3) = \frac{5}{8}, \\ C(a^1|s^3) = C(a^2|s^5) + C(a^4|s^5) = 1, & \quad C(a^3|s^1) = \frac{1}{2} \quad C(a^3|s^2) = \frac{3}{8}, \\ C(a^4|s^3) = C(a^4|s^5) = \frac{1}{2}. & \end{aligned}$$

Part 3 of Theorem 3 then gives us the remaining unconditional arc criticalities:

$$\begin{aligned} C(a^1) = P(s^1)[\frac{1}{2} \quad C(a^2|s^2)] + P(s^3)[C(a^2|s^5) + C(a^4|s^5)] = \frac{5}{8}. \\ C(a^3) = P(s^2)[\frac{1}{2} \quad C(a^4|s^5)] + P(s^4)[C(a^4|s^6)] = \frac{3}{8}. \end{aligned}$$

In computer implementations, the probabilities $(P(s^j)'s)$ should be computed first in a forward pass through the Q matrix and then the unconditional criticalities from part 3 can be computed concurrently with conditional criticalities with a backward pass through the Q matrix.

Observe that the conditional criticalities for each state sum to 1 if the sum is taken only across arcs that could complete from the state. For example, $C(a^4|s^3)$ is defined, but it is not possible for arc a^4 to complete from state s^3 (since arc a^4 requires both arcs a^1 and a^3 to be completed). Thus $C(a^4|s^3)$ would not be included in the sum described for state s^3 . This observation will be very important in assessing criticalities for the case of a cyclic network in Section 3.

Corollary 3.1. *Suppose task W is represented by a sequence of arcs, $a^{F(W)}, a^{F(W)+1}, \dots, a^{L(W)}$, where $F(W)$ (resp. $L(W)$) is the first (resp. last) arc. Then*

$$C(a^{F(W)}) = C(a^{F(W)+1}) = \dots = C(a^{L(W)}) \equiv C(t^W). \quad (11)$$

Proof: Consider $F(W) \leq i < L(W)$. Arc a^{i+1} is enabled exactly when arc a^i completes (independent of the states involved). Theorem 3 implies $C(a^i) = C(a^{i+1})$. The result follows by induction on i .

To this point we have presented theoretical results for a finite schedule modeled as a Markov chain. We will now extend these results to cyclic schedules, taking into account their true “wrap-around” nature.

3. MARKOVIAN CYCLIC SCHEDULES

Recall that we defined a cyclic schedule to be a sequence of tasks that is repeated over and over again so that each task is done exactly once each cycle. If one task (operation) on each machine is

arbitrarily selected as the “first” operation, then the operation immediately preceding it on the same machine is correspondingly the “last” operation performed on that machine in the previous cycle. “First” and “last” only have meaning in visually depicting the schedule. Material flow precedent constraints can also go across cycles and it is possible to have a constraint that goes across more than two cycles. Thus, for each arc a^i in the precedent set of arc a^j , we associate a non-negative integer C_{ij} that is the number of cycles ahead of a^j that a^i must be done before a^j can start. These numbers C_{ij} are key to understanding cyclic schedules and the difference between cyclic schedules in a deterministic vs. stochastic environment. Notice that any sequence is feasible - the C_{ij} ’s can be adjusted to make this true. A good sequence would lead to low WIP and high throughput. In the deterministic case, this corresponds to zeroes for as many C_{ij} ’s as possible with tasks completed just-in-time. In a stochastic environment, longer expected times between tasks and their material flow successor tasks may be beneficial to reduce the probability of waiting for material and to increase the throughput rate. Selecting a sequence that best balances the need for high throughput with the need for low WIP is not easy to do, and is a topic of future research which is briefly addressed in Sections 4 and 5. The models presented in this section will allow alternatives to be properly compared and should also provide help in generating good alternatives.

Recall that we modeled finite non-cyclic schedules as Markov chains. The following theorem shows that it is possible (and natural) to model a cyclic schedule as a Markov chain as well.

Theorem 4.

Let

$$\begin{aligned}
 A_i(t) &= \text{the number of most recently completed arc on machine } i \text{ at time } t, \\
 R_i(t) &= \text{the number of cycles ahead (+) or behind (-) the arc most recently} \\
 &\quad \text{completed on machine } i \text{ is relative to the arc most recently completed} \\
 &\quad \text{on machine } 1 \text{ at time } t \text{ (with respect to the numbering convention),} \\
 R_1 &\equiv 0,
 \end{aligned}$$

$$A(t) = (A_1(t), A_2(t), \dots, A_m(t)),$$

$$R(t) = (R_1(t), R_2(t), \dots, R_m(t)),$$

$$s(t) = \begin{pmatrix} A(t) \\ R(t) \end{pmatrix},$$

$$S = \{s: s \text{ is feasible for the precedence constraints}\},$$

$$N(i) = \text{number of arcs on machine } i = L(i) - L(i-1) \quad (L(0) \equiv 0).$$

Then:

$\{s(t), t \geq 0\}$ is a continuous time Markov chain on S with infinitesimal generator matrix

$$Q = [q(s^A, s^B)], \quad s^A \text{ and } s^B \in S, \text{ where:}$$

$$\text{If } A_j \neq L(j) \text{ and, } \forall a^i \in PR(a^{A_j+1}),$$

$$\text{either } (A_{M(a^i)} \geq i \text{ and } R_{M(a^i)} + C_{i, A_j+1} = R_j)$$

$$\text{or } R_{M(a^i)} + C_{i, A_j+1} > R_j,$$

then:

$$(1) \quad q\left(s, s + \begin{pmatrix} \vec{e}_j \\ 0 \end{pmatrix}\right) = \frac{1}{\theta_{A_j+1}}. \quad (12)$$

$\left(\begin{array}{l} \text{A task on machine } j \text{ finishes} \\ \text{and it is not the last task on} \\ \text{machine } j \text{ in the cycle} \end{array} \right)$

$$\text{Else if } j \neq 1 \text{ and } A_j = L(j) \text{ and } \forall a^i \in PR(a^{A_j-N(j)+1})$$

$$\text{either } (A_{M(a^i)} \geq i \text{ and } R_{M(a^i)} + C_{i, A_j-N(j)+1} = R_j)$$

$$\text{or } R_{M(a^i)} + C_{i, A_j-N(j)+1} > R_j + 1,$$

then:

$$(2) \quad q\left(s, s + \begin{pmatrix} (1-N(j))\vec{e}_j \\ \vec{e}_j \end{pmatrix}\right) = \frac{1}{\theta_{A_j-N(j)+1}}. \quad (13)$$

$\left(\begin{array}{l} \text{A task on machine } j \neq 1 \text{ finishes} \\ \text{and it is the last task on} \\ \text{machine } j \text{ in the cycle} \end{array} \right)$

$$\text{Else if } A_1 = L(1) \text{ and, } \forall a^i \in PR(a^1),$$

$$\text{either } (A_{M(a^i)} \geq i \text{ and } R_{M(a^i)} + C_{i1} = 1) \text{ or } R_{M(a^i)} + C_{i1} > 1,$$

then:

$$(3) \quad q\left(s, s + \begin{pmatrix} (1-N(1))\vec{e}_1 \\ \vec{e}_1 - \vec{1} \end{pmatrix}\right) = \frac{1}{\theta_1}.$$

$\left(\begin{array}{l} \text{A task on machine 1 finishes} \\ \text{and it is the last task on} \\ \text{machine 1 in the cycle} \end{array} \right)$

(14)

$$(4) \quad q(s, s) = - \sum_{s^*: q(s, s^*) > 0} q(s, s^*). \quad (15)$$

$$(5) \quad q(s^A, s^B) = 0, \text{ otherwise.} \quad (16)$$

The proof of this theorem is straightforward. The increased complexity of notation is necessary because transitions across cycles on the machines are possible. Theoretically, the number of states could be infinite. For example, one machine could have no material flow from any other machines so that it could move ahead completely unconstrained by the other machines. For any practical situation, however, a machine would be stopped if it got too far ahead of the other machines. Thus, even if a machine doesn't receive material from any other machine, it will be assumed that some extra precedence constraint will be present that will prohibit a machine from getting too many cycles ahead of other machines. This will guarantee that the number of states is finite. The rest of this paper deals only with the case where the number of states is finite.

Theorem 4 and its proof are similar to Theorem 1; but, there are several major differences: there is no initial state, there is no final absorbing state, and the Q matrix is not upper triangular. Consequently we will focus on ergodic results rather than the transient results that were of interest for the Markov chain of Theorem 1. States will be revisited and this will correspond to the cyclic schedule reaching the same state that it did in an earlier cycle. An algorithm for generating all states and the Q matrix for the Markov chain of Theorem 4 is given by Bowman (1990).

Let's revisit the example corresponding to Figure 1.1. The schedule in that case can be transformed into a cyclic example by the introduction of one or more wrap-around constraints. One possibility is to require task 4 to be completed before task 1 is started in the next cycle, as is shown in Figure 3.

Figure 3: Cyclic Schedule Example

One possible state the system could be in is $\begin{pmatrix} \langle 2,4 \rangle \\ \langle 0,0 \rangle \end{pmatrix}$, which indicates that machine 1 has just finished task 2 and machine 2 has just finished task 4 during the same cycle. From this state, a transition can

be made to either $\begin{pmatrix} 1,4 \\ 0,-1 \end{pmatrix}$ or $\begin{pmatrix} 2,3 \\ 0,1 \end{pmatrix}$. Each transition occurs at unit rate since we assumed for the example problem that task times are exponentially distributed with unit mean.

As was the case with non-cyclic schedules, the states will be divided into $|t|$ levels, where $|t|$ is the number of tasks in the schedule. From an arbitrary feasible state, each state in a level can be reached through the completion of the same number of tasks. Some easily provable (see Bowman (1990)) facts about the model are:

1. A feasible state can be reached from itself in $|t|$ transitions.
2. Any feasible state can be reached from any other feasible state in a finite number of transitions.
3. Each state can be reached from at least one state in the previous level (level $|t|$ is the level prior to level 1) and can reach at least one state in the next level in a single transition.
4. Transitions are only possible from a state in one level to a state in the next higher level.

These results suggest the definition of a cycle as the completion of $|t|$ arcs or the return to the same level arbitrarily selected as the start of the cycle. The following theorem justifies this definition and shows how to compute ergodic results based on it.

Theorem 5.

Let

$P_{ij} =$ the probability that the next transition into level 1 occurs in state s^j given that the most recent transition into level 1 was in state s^i ,

$T =$ time between visits to level 1 (a random variable),

$E(T|i,j) =$ expected time between visits to level 1 given that you start in state s^i and reach state s^j (in level 1),

$E(T^2|i,j) =$ second moment of the above time,

$P(j|i) =$ probability state s^j (not in level 1) is visited before the next return to level 1 given that you start in state s^i (in level 1),

$X_n =$ number of the state visited on the n^{th} visit to level 1 (a random variable).

Then:

(1) $\{X_n, n = 0, 1, 2, \dots\}$ is a finite state irreducible ergodic Markov chain with probability transition matrix $P = [P_{ij}]$.

$$(2) \quad E(T) = \sum_{i \in \text{level } 1} \pi_i \sum_{j \in \text{level } 1} P_{ij} E(T|i, j), \quad (17)$$

$$E(T^2) = \sum_{i \in \text{level } 1} \pi_i \sum_{j \in \text{level } 1} P_{ij} E(T^2|i, j), \quad (18)$$

where $\vec{\pi}$ is the vector of steady state probabilities of visiting each state in a visit to level 1 that

$$\text{solves } \vec{\pi} = \vec{\pi} P, \quad \sum_{i: s^i \in \text{level } 1} \pi_i = 1.$$

(3) For states s^j not in level 1:

$$\pi_j^* = \sum_{i: s^i \in \text{level } 1} \pi_i P(j|i), \quad (19)$$

where π_j^* = steady state probability of visiting state s^j in a visit to the level that state s^j is in.

$$(4) \quad \lim_{t \rightarrow \infty} \frac{t}{n_i(t)} = E(T) \quad (20)$$

where $n_i(t)$ = number of times arc i has completed by time t .

Parts (1), (2), and (3) follow immediately from standard ergodic Markov chain theory and the observations made previously. The theorem shows how to evaluate the ergodic expected value and variance of cycle time as well as the probability that each state is visited during a cycle. Part (4) follows readily from the assumption that no machine will be allowed to get too far ahead of any other machine, which ensures that each task will be completed $\frac{1}{|t|}$ times per cycle in the long run. This part shows that the expected cycle time agrees with the intuitive definition of a cycle, that is, that a cycle measures the time between successive completions of a task. The problem with the latter somewhat intuitive definition is that the variance depends on the task selected.

The standard algorithms for a non-cyclic schedule can be used (iteratively starting with each state s^i in a “base” level) to calculate P_{ij} , $E(T|i, j)$, $E(T^2|i, j)$, and $P(j|i)$. A solution for $\bar{\pi} = \bar{\pi} P$ can then be obtained and used to calculate $E(T)$, $E(T^2)$ and π_j^* for all states s^j not in the base

level. The base level should be the level with the smallest number of states for maximum efficiency. If one level (or more) has only one state, then the schedule must pass through that state, and the algorithms of section 2 can be used to evaluate the cyclic schedule directly.

It is natural to ask the same types of questions about the cyclic schedule as were asked about the schedule with unique starting and ending states. There are two major differences. As noted earlier, the questions must first be asked about ergodic rather than transient results. For example, what is the best way to reduce the ergodic expected cycle time below some level. Second, the concept of critical path (hence, arc criticality) cannot be applied to a cyclic schedule directly since there is no true end (or beginning) to the network. It is natural (and useful), however, to define the cyclic criticality of an arc j ($CC(a^j)$) as the fraction of times that arc a^j appears in the critical path to the enabling of any arc a^i in the n th cycle as $n \rightarrow \infty$. The existence and uniqueness of cyclic criticalities is guaranteed by the fact that the concept of enabling an arc is a property of transitions that have steady state probabilities as given in Theorem 5. The following theorem establishes relationships among cyclic criticalities.

Theorem 6.

Let

$$\begin{aligned}
 C(a^j, s^n | a^i, s^m) &= \text{the criticality of arc } a^j \text{ from the } n^{\text{th}} \text{ state given that arc} \\
 &\quad a^i \text{ from the } m^{\text{th}} \text{ state of the same level is critical on the next} \\
 &\quad \text{visit to that level (defined } \forall j, n, i, m \text{ such that arc } a^j \text{ is} \\
 &\quad \text{the next arc due to be completed on its machine from state} \\
 &\quad s^n \text{ and arc } a^i \text{ could be the next arc completed from state } s^m), \\
 CC(a^j, s^n) &= \text{cyclic criticality of arc } a^j \text{ from state } s^n \text{ (defined } \forall j, n \text{ such that} \\
 &\quad \text{arc } a^j \text{ is the next arc due to be completed on its machine from} \\
 &\quad \text{state } s^n). \\
 s^{m*} &= \text{the state that would be reached from state } n \text{ if arc } a^j \text{ was completed.}
 \end{aligned}$$

Then

$$(1) \quad CC(a^j, s^n) = \sum_{i: (n, m^*) \in E(a^i)} \left(\frac{q(n, m^*)}{-q(n, n)} \right) CC(a^i, s^{m^*}) + \sum_{\substack{p: q(n, p) > 0 \\ p \neq m^*}} \left(\frac{q(n, p)}{-q(n, n)} \right) CC(a^j, s^p), \quad (21)$$

$\forall j, n \text{ s.t. } CC(a^j, s^n) \text{ is defined.}$

$$(2) \quad CC(a^j) = \sum_{\substack{n: a^j \text{ could be the next arc} \\ \text{completed from state } s^n}} P(s^n) \sum_{i: (n, m^*) \in E(a^i)} \left(\frac{q(n, m^*)}{-q(n, n)} \right) CC(a^i, s^{m^*}). \quad (22)$$

$$(3) \quad CC(a^j, s^n) = \sum_{\substack{i, m: CC(a^i, s^m) \\ \text{is defined}}} C(a^j, s^n | a^i, s^m) CC(a^i, s^m) \quad \forall j, n \text{ s.t. } CC(a^j, s^n) \text{ is defined.} \quad (23)$$

$$(4) \quad \sum_{\substack{j: a^j \text{ could be the next arc} \\ \text{completed from state } s^n}} CC(a^j, s^n) = 1 \quad \forall n. \quad (24)$$

Parts (1) and (2) follow immediately from Theorem 3. Let ℓ be the level that state s^n is in. Part 1 of the theorem says that $CC(a^j, s^n)$ can be expressed as some linear combination of $CC(a^i, s^m)$'s, where s^m is in the next level ($\ell + 1$ unless ℓ is the “last” level, that is, the one that precedes level 1). Repeated application of this fact until we reach level ℓ again tells us that $CC(a^j, s^n)$ can be expressed as a linear combination of $CC(a^i, s^m)$'s, where s^m is in the same level as s^n . Thus, (3) follows. Part (4) provides the necessary scaling of the criticalities. As noted earlier, the conditional criticalities for each state sum to 1 if the sum is taken only across arcs that could directly complete from the state.

By iteratively conditioning on the arcs a^i , and states s^m in a base level, the algorithm given in section 2 for calculating criticalities can be used to solve for the coefficients in (3). Relations (3) and (4) can then be used to solve for the conditional cyclic criticalities for the base level. Relations (1) and (2) can then be used as in Theorem 3. However, we now use the conditional cyclic criticalities for the base level as the end conditions, when solving for the cyclic criticalities.

For the same reasons we were interested in arc criticalities in a finite schedule, we will be interested in cyclic criticalities in a cyclic schedule. In addition, we might also be interested in maintaining buffer stock in a cyclic schedule. Suppose we take the sum in relation (2) over transitions that enable an arc a^j such that the arc that completed, which caused the transition to occur, corresponds to a machine that differs from the one that corresponds to arc a^j . We then obtain the cyclic buffer criticality for task j . This gives the joint probability that task j is critical and that it was waiting for material, which is the same as the infinitesimal effect of adding buffer stock before task j . Note that buffer stock in a cyclic schedule must correspond to tasks as well as machines since it is assumed that the sequence will be maintained. A buffer stock equivalent to an entire lot changes the precedence structure and can only be evaluated by revising the underlying Markov chain. Buffer stocks less than a production lot, if used in a cycle, will be replenished when the material precedent task is completed. Modeling buffer stocks is straightforward and is described by Bowman (1990).

4. AN ALGORITHM FOR SELECTING FROM POSSIBLE MANAGEMENT ACTIONS

An important question that management could ask is how the expected value (or perhaps the expected value plus a multiple of the standard deviation) of schedule completion or cycle time could be reduced below some selected value at minimum cost. Alternatively, one could ask how to best allocate some budget to reduce the expected time or cycle time by the maximum amount. The models and algorithms of the previous sections show how the expected value, standard deviation and task criticalities can be calculated. The task criticalities give a ranking of the infinitesimal effect of reducing task means. Many management actions can be described as a reduction of some subset of task means at some cost. These actions include:

- Crashing the Network - Devote more resources to specific tasks (reducing the mean times) at some cost.
- Reducing Setup Times - Reduce the setup times for specific tasks (hence reducing their mean times) at some cost.

- Purchasing parts - Reduce the lot sizes of all tasks (hence reducing their mean times) required to make the parts which will now be purchased at some cost.
- Improving Machine Efficiency - Reduce the mean times of all tasks on some machine at some cost.
- Shifting Labor - Reduce the mean times of some tasks and increase the mean times of other operations at some cost.

The infinitesimal effect, measured in terms of the change in the expected value of schedule completion time or cycle time per dollar cost, of any of these management actions can be found by taking the dot product of the gradient estimate (vector of criticalities) with the vector of changes in operation times per dollar spent on that action.

Let's consider an example. Suppose management is considering purchasing part number 3 at \$5 per part. Part 3 is an assembly consisting of two part 1's and three part 2's. Part 1 is made by operation 1 at .5 hours per part and $\frac{\partial E(T)}{\partial \theta_1}$ has been estimated at 0.531. Part 2 is made by operation 2 at .25 hours per part and $\frac{\partial E(T)}{\partial \theta_2}$ has been estimated at .0717. The assembly (operation 3) takes .10 hours per part 3 and $\frac{\partial E(T)}{\partial \theta_3}$ has been estimated at 0.821. The infinitesimal effect of purchasing part number 3 is given by:

$$\text{Inf. Effect of Purchasing Part 3} = ((.5)(2)(.531) + (.25)(3)(.717) + (.10)(1)(.821))/5.$$

A greedy approach can then be used to allocate a budget to the various actions based on their infinitesimal effects. Since these infinitesimal effects obviously change continuously as dollars are spent, they must be re-evaluated periodically. If too large a portion of the budget is allocated between re-evaluations of the derivatives, the result could be seriously sub-optimal. If too little is allocated between re-evaluations, the time required to make the re-estimates could be prohibitive. If a simulation is used to estimate the criticalities or exact gradient (IPA estimate for this particular model), the number of replications (or cycles) to use for each estimate must be established. A tradeoff (for a fixed run time) exists between the number of replications and the portion of the budget to

allocate between estimates. Bowman (1990) concluded that 100 replications and a portion of the budget sufficient to reduce the expected cycle time by approximately 1% were reasonable choices.

5. COMPUTATIONAL RESULTS

To better understand the effectiveness of our ideas, we conducted an experiment. In this experiment we found exact task criticalities (or cyclic task criticalities), expected value and standard deviation of schedule completion times (or cycle time), and other performance measures for a series of example problems using the algorithms presented in Sections 2 and 3. The same problems were then analyzed by simulating these Markov chains using Bowman's approach with various numbers of replications. Since exact results were computed, the absolute percent error (APE) of the simulation estimates of the expected value and the standard deviation of schedule completion or cycle time and the average absolute percent error (AAPE) of the simulation estimates of the arc criticalities could be computed. The results are shown in Table I.

All computational results reported below were obtained using programs coded in FORTRAN77 and run on the IBM Supercomputer at Cornell University. We can observe several important facts. First, the run times for the non-cyclic exact analysis are linear in the number of positive Q-matrix entries. This occurs because the algorithms move forward through the Q-matrix and then backward through it. Second, the run times for the cyclic exact analysis are linear in the product of the number of states in the base level (level with the smallest number of states) and the number of positive Q-matrix entries. This happens because algorithms are the same as the non-cyclic algorithm except that they are repeated starting with each state in the "base" level. How this relates to the number of machines and operations is very complicated and very problem specific. Finally, in general, the more freedom each machine has to move unconstrained by the other machines, the larger the run times will be. If we fix all other characteristics, the run times will be exponential in the number of machines.

Two cyclic problems were chosen to demonstrate how a small change in a problem definition can greatly affect the size of the corresponding Markov chain and the consequent run times. The two

Table I. Markovian Simulation vs. Exact Model Calculation

<u>Problem Description</u>	Example Problems						
	Non-Cyclic					Cyclic	
No. Machines	2	3	4	6	9	3	3
No. Operations	12	19	27	72	92	24	24
<u>Exact Model Calculation</u>							
No. States	29	76	378	3696	38,939	165	685
No. Central Level States	--	--	--	--	--	1	31
No. Positive Q-entries	44	143	966	11,844	162,475	360	1668
Run Time (CPUS)	0.38	0.60	1.90	18.98	292.48	0.28	19.72
<u>Markovian Simulation</u>							
100 Reps.:							
Run time (CPUS)	0.12	0.14	0.15	0.24	0.31	0.13	0.13
APE - Exp. Value	0.03	0.45	0.20	1.86	0.04	0.65	0.23
APE - Std. Dev.	0.14	0.90	0.33	0.37	3.57	0.77	0.38
AAPE - Criticalities	4.63	6.05	18.80	13.46	33.55	8.44	7.88
500 Reps.:							
Run time (CPUS)	0.18	0.21	0.25	0.60	0.81	0.24	0.25
APE - Exp. Value	0.25	0.24	0.55	0.79	0.40	0.29	0.43
APE - Std. Dev.	0.47	0.05	0.40	1.19	7.59	0.31	0.57
AAPE - Criticalities	6.32	3.35	11.19	6.00	20.90	8.00	4.27
1000 Reps.:							
Run time (CPUS)	0.22	0.29	0.37	1.06	1.49	0.39	0.37
APE - Exp. Value	0.33	0.32	0.68	0.41	0.07	0.26	0.01
APE - Std. Dev.	0.47	1.36	1.13	1.04	6.13	0.40	0.15
AAPE - Criticalities	3.18	3.86	4.14	4.36	18.16	6.06	3.46
5000 Reps.:							
Run time (CPUS)	0.60	0.97	1.40	4.74	6.69	1.56	1.43
APE - Exp. Value	0.18	0.27	0.06	0.22	0.62	0.05	0.07
APE - Std. Dev.	0.26	0.65	0.30	0.43	4.94	0.26	0.23
AAPE - Criticalities	1.83	0.94	3.02	1.69	10.61	4.72	5.73

problems are the same (a flow shop with all products visiting three machines in numeric order) except for the selection of a constraint to keep machine 1 from getting too far ahead of the other machines. In the first problem, a constraint was included requiring the last task on machine 3 to be completed before the first task on machine 1 could begin in the next cycle. In the second problem, the last task on machine 3 was required to be completed before the first task on machine 1 could begin 3 cycles ahead. This allowed a much larger number of feasible states and increased the run time of the exact analysis, as shown in the table. The difference in the number of states and the run times shows how problem specific the computational efficiency of the exact analysis can be. However, the run times for the Markovian simulations of these problems do not differ much since these run times are linear in the product of the number of replications and the number of arcs in the network.

The table shows that very accurate estimates of the expected value and standard deviation of schedule completion times (or cycle times) can be obtained with relatively few replications. The average errors in estimating the criticalities are somewhat larger but are also quite reasonable with a relatively small number of replications. This is particularly true if one is primarily interested in ranking the tasks in order of criticality so that management efforts can be concentrated on the most important ones. In this case, it is not essential to rank the tasks exactly. If one task is ranked ahead of another, however, we would expect that it does not have a criticality that is an order of magnitude lower than the lower ranked task.

A simulation experiment was set up in which three industrial data sets were analyzed using deterministic theory and then the stochastic theory of this paper. The data were finite schedules corresponding to a finite number of cycles of cyclic schedules. Specifically, the objective for each problem was to reduce the expected schedule completion time as much as possible for a given budget by selecting tasks to have their mean times reduced at some cost (i.e. “crashing” the network). If there is no variability, the problem can be solved optimally using a simple linear program and the entire range of optimal solutions can be generated efficiently using a network flow algorithm. The solutions range from each operation having its mean at its lower bound to each operation having its mean at its

current value or upper bound (no money spent). The experiment compared these optimal deterministic solutions with the greedy stochastic gradient approach described in this paper across a wide range of conditions. The stochastic results were obtained using a simulation of the Markov model with the number of replications and the budget portion allocated between re-estimate of the gradient as described previously.

A linear programming solution was obtained to reduce the deterministic longest path (schedule completion time) by 10%, 20%, and 30% for each of the three problems. The minimum cost for each of these solutions was then used as a budget limitation for the greedy stochastic gradient approach. A wide range of variance specifications in terms of SCV's of the task times was examined. A simulation was then run to compare the LP solution with the greedy stochastic gradient (GSG) solution for each specification of the SCV's. Table II shows the percentage reduction in the expected value of schedule completion time obtained from the LP and the GSG and Table III shows the percentage reduction in the standard deviation that was obtained. As expected, if the task times have very small variances, then there is very little difference in the solutions and one would want to use the more efficient and globally-oriented technique of linear programming. As expected, as the amount of variation increases, the GSG method begins to outperform LP significantly. Two interesting results from the tables were not anticipated. First, the GSG method did even better with respect to standard deviation vs. LP than it did with respect to expected value. We believe that this occurred because the GSG method tends to select tasks for reduction that are on many paths that are likely to be critical. Second, randomly assigned SCV's did not favor GSG vs. LP any more than did assigning the same SCV to all tasks. In other words, total variation appears to be more critical than the distribution of the variation. Although one can create examples to counter this observation, the problems we examined did not exhibit such behavior. One would expect that a dominant bottleneck would also favor LP since the smart thing to do (deterministically and stochastically) would be to reduce the bottleneck task times. Although the three problems were significantly different in size and structure, they all had secondary bottlenecks relatively close in utilization to their primary bottleneck. To better characterize the types of problems

Table II. Reduction in Expected Value of Schedule Completion Time Achieved with Greedy Stochastic Gradient vs. LP-Based Arc Length Mean Reduction

SCV	Budget - 10%		Budget = 20%		Budget = 30%	
	LP	GSG	LP	GSG	LP	GSG
0.01	10.0	10.0	19.4	20.0	29.0	30.0
0.10	7.9	8.5	15.9	15.6	24.4	25.5
0.20	7.3	8.0	14.9	16.3	23.0	24.3
0.30	7.0	7.9	14.5	16.0	22.3	23.8
0.40	6.7	7.6	14.0	15.6	21.8	23.3
0.50	6.4	7.5	13.6	15.5	21.4	21.6
0.60	6.5	7.5	13.7	15.5	21.1	22.6
0.70	6.2	7.4	13.2	15.3	20.8	22.2
0.80	6.2	7.5	13.2	15.4	20.6	22.3
0.90	6.2	7.5	13.0	15.1	20.4	22.0
1.00	5.9	7.3	12.9	14.9	20.2	21.8
Random (.05-.50)	7.2	8.0	14.6	16.2	22.6	24.1

Note 1: A budget of x% corresponds to the linear programming cost to reduce the deterministic longest path by x%.

Note 2: The entries for the random squared coefficients of variation are the average of 10 random assignments.

Table III. Reduction in Standard Deviation of Schedule Completion Time Achieved with Greedy Stochastic Gradient vs. LP-Based Arc Length Mean Reduction

SCV	Budget - 10%		Budget = 20%		Budget = 30%	
	LP	GSG	LP	GSG	LP	GSG
0.01	5.7	8.6	17.2	18.8	22.3	22.3
0.10	7.2	11.0	18.2	22.7	27.3	30.7
0.20	4.9	8.5	15.0	18.5	25.6	27.8
0.30	4.7	9.0	14.9	19.2	24.6	27.7
0.40	4.3	9.4	14.1	18.2	23.6	27.8
0.50	4.2	8.4	14.2	19.3	24.2	28.6
0.60	4.2	8.3	13.2	18.7	22.9	27.2
0.70	3.9	8.4	13.1	19.8	22.3	26.8
0.80	3.0	8.3	12.5	19.6	22.5	27.0
0.90	3.8	8.6	12.3	19.1	21.8	26.8
1.00	3.7	8.2	12.6	19.1	22.3	27.4
Random (.05-.50)	5.4	9.8	14.2	20.8	24.7	29.8

Note 1: A budget of x% corresponds to the linear programming cost to reduce the deterministic longest path by x%.

Note 2: The entries for the random squared coefficients of variation are the average of 10 random assignments.

for which one would want to use LP vs. GSG, these same three problems were altered by reducing all task times proportionally, except the bottleneck's, until the secondary bottleneck had 90%, 70%, 50%, 30%, or 10% of the bottleneck utilization. LP and GSG solutions (for a budget that yields a 20% reduction in the deterministic longest path) were then compared with each bottleneck specification in combination with operation time SCV's of .10, .50, and 1.00 (exponential) for each of the three problems. The results (average of the 3 problems) are shown in Table IV (reduction in expected value of schedule completion time achieved) and Table V (reduction in standard deviation of schedule completion time achieved).

As expected, the dominance of the bottleneck was a major factor. For example, if the secondary bottleneck has 90% of the main bottleneck's load, GSG outperformed LP significantly at all 3 SCV settings. With a 10% secondary bottleneck, GSG never outperformed LP. From the table, one could attempt to characterize the type of problems for which one could use a deterministic solution. The results shown, however, are averages for the three problems. This obscures another possible factor which is problem size, in particular, the number of machines. The problem having the largest number of machines and operations required a much more dominant bottleneck before GSG was not favored over LP. In general, however, one can safely say that problems with a large amount of variation and without a single dominant bottleneck are the ones for which the GSG approach should be most preferred over LP.

An investigation was also made into the effect of the distribution of variance into bottlenecks vs. non-bottlenecks. This distribution has a major effect on the behavior of the problem. (For example, increasing the variance of a dominant bottleneck has almost no effect on the expected value of schedule completion time but severely increases the variance, whereas increasing the variance of a bottleneck in a fairly balanced setting significantly increases the expected schedule completion time but only mildly increases the variance). It had little effect, however, on the relative performance of GSG vs. LP and the results, therefore, are not included here.

Table IV. Effect of Degree of Bottleneck Dominance on LP vs. Greedy Stochastic Gradient for Reducing Expected Schedule Completion Time

Radio of Utilization of 2nd Biggest Bottleneck to Biggest Bottleneck	SCV = .10		SCV = .50		SCV = 1.00	
	LP	GSG	LP	GSG	LP	GSG
0.90	15.6	16.9	13.5	15.5	13.1	14.9
0.70	17.2	18.4	15.1	17.2	14.4	16.5
0.50	18.5	19.2	15.6	16.9	14.5	15.6
0.30	19.9	20.0	18.6	19.4	17.3	18.2
0.10	20.1	20.1	20.1	20.0	20.0	19.9

Table V. Effect of Degree of Bottleneck Dominance on LP vs. Greedy Stochastic Gradient for Reducing Standard Deviation of Schedule Completion Time

Radio of Utilization of 2nd Biggest Bottleneck to Biggest Bottleneck	SCV = .10		SCV = .50		SCV = 1.00	
	LP	GSG	LP	GSG	LP	GSG
0.90	17.6	23.8	12.2	21.1	12.8	20.1
0.70	21.6	27.4	17.4	24.1	16.7	22.5
0.50	22.2	24.7	20.0	24.0	18.5	22.2
0.30	16.6	20.0	16.9	20.3	17.5	20.9
0.10	16.9	18.0	16.6	19.0	16.4	18.8

Table VI shows the run times necessary to crash the networks for the 3 problems using GSG. The run times are proportional to the number of arcs in the network. Of course, the number of arcs depends on the number of tasks to be completed and the SCV's of their processing times (see Section 2).

These results were obtained using the underlying Markov chain model of Theorem 1, which applies to finite schedules including a finite number of cycles of a cyclic schedule. Of course, the model of Theorem 4 could be used to evaluate a cyclic schedule more efficiently and directly, since the greedy stochastic gradient approach applies to both models. The linear programming approach, however, does not extend easily. The reported computational results were for problems having finite schedules. This was done only to enable the interesting comparison of deterministic and stochastic approaches. Bowman (1990) presents more computational results for cyclic schedules.

As described in Section 3, the concept of cyclic criticality is also useful for placing buffer stock in a cyclic schedule. As with GSG, one must select a step size (in this case, the amount of buffer stock to position before the cyclic buffer gradient is re-estimated) and, if simulation of the Markov chain is used to estimate the gradient, the number of replications to use for each simulation estimate. For the results that follow, 100 replications were made for each simulation, and the operation with the highest cyclic buffer gradient had an additional buffer stock inserted of size equal to of one-tenth of the operation lot size. We call this the greedy stochastic gradient (GSG) buffering technique.

Clearly, buffer stock is only one way to increase throughput of a cyclic schedule. Other alternatives include cyclic release rules, introduction of safety time into schedules, lot sizing decisions, and improved heuristics for sequencing. Research is needed on all these approaches and a good solution is likely to combine many of them. The GSG buffering technique does, however, provide a new alternative to the obvious approach of moving to a different point on the WIP/throughput trade-off curve.

Table VI. Computation Time: Reduction in Schedule Completion Time Through Reduction of Operation Means

<u>Problem Description</u>	Problem		
	1	2	3
No. of Machines	4	6	15
No. of Operations	27	72	109
<u>Run time (CPUS) to Reduce Expected Completion Time By 10%</u>			
SCV = .10	4.97	7.88	20.38
SCV = .50	1.29	2.55	5.27
SCV = 1.00	0.74	1.72	3.17
SCV = Random (U[.1,.5])	3.13	4.59	13.24

With the goal of obtaining a first characterization of when one might consider GSG buffering, ten scheduling problems were randomly generated and cyclic schedules were built using the heuristic of Graves et al. (1983). The desired cycle time is an input to this heuristic and cycle times were input from 1.0 to 2.0 times the length of the processing time on the bottleneck machine in .05 increments. Thus, 21 schedules were generated for each of the 10 problems. The results for one of these problems is given in Figure 4. Notice that as a larger (deterministic) cycle time was input, in general the expected cycle time went up and the work-in-process inventory (and mean flow time) went down. This is what one would expect, although the progression wasn't always smooth. Some schedules had both higher WIP and higher expected cycle times than others. These "dominated" schedules were thrown out and the remaining schedules formed an "efficient frontier" of cyclic schedules.

Figure 4. Generating a Tradeoff Curve

The next step was to use greedy stochastic gradient buffering to buffer each efficient schedule until it had approximately as much WIP as the next efficient schedule (the one with the next highest WIP). The expected cycle times were then compared to see if buffering gave better or worse results than simply rescheduling. For each of the ten problems, one efficient schedule was also buffered until its WIP equaled the WIP of an efficient schedule that had significantly higher WIP. Thus, the schedule had a large amount of buffer stock. The results for one case are in Figure 5.

Figure 5. GSG Buffering

The results for all 10 problems are summarized in Table VII. This table shows the average decrease in expected cycle time per unit of additional WIP for GSG buffering compared to moving to a different point on the curve (rescheduling with a smaller input cycle time). The comparison is made as a function of the ratio of the input cycle time to the minimum possible cycle time. The table shows that when this ratio is low (i.e. the bottleneck machine is heavily loaded), it was generally better to add buffer stock, and when the ratio was high, it was better to reschedule. This is primarily because the flow time (hence WIP) goes up very rapidly as you generate cyclic schedules with the bottleneck heavily loaded. In this case, it is hard to avoid conflicts on the machine, so that a product ends up flowing across many cycles.

Once again, these results are based on the cyclic schedule being allowed to advance whenever possible. How these results change when release rules (state-based or time-based) are implemented needs to be studied further. Rescheduling is simply a way to affect the amount of safety time in a schedule. Coming up with better methods for establishing the proper amount of safety time in a cyclic schedule is also an area for future research.

Computationally, reducing the expected cycle time of a cyclic schedule by gradient-based reduction of operation means or addition of buffer stock (or any of the other management actions previously discussed that can be evaluated with the gradient) is very similar to crashing a schedule with a fixed starting and ending state. The same factors (number of arcs, number of replications, step size, and amount of reduction desired) affect the run times. Run times for the 10 randomly generated problems are shown in Table VIII.

Table VII. GSG Buffering (GB) vs. Rescheduling (RE) of Cyclic Schedules:
Average Expected Cycle Time/WIP Gradient

Problem No.	Input Cycle Time $\div$ Min. Possible Deterministic Cycle Time					
	1.00 - 1.25		1.25 - 1.50		1.50 and up	
	GB	RE	GB	RE	GB	RE
1	.0654	.0529	.0906	.2030	.1105	.3465
2	.0306	.0267	.1139	.1008	.1376	.3382
3	.0631	.0499	.0781	.0938	.1614	.7795
4	.0377	.0266	.0724	.1277	.1119	.4400
5	.0444	.0543	.0662	.0442	.1658	.5466
6	.0379	.0237	.0730	.1749	.0976	.1276
7	.0502	.0851	.0888	.2489	.1329	.3753
8	.0409	.0471	.0714	.1133	.1177	.2450
9	.0308	.0137	.0964	.4547	.0820	.1673
10	.0249	.0236	.0467	.0354	.0721	.1142
Avg Ratio of GB Gradient to RE Gradient	1.22		0.74		0.42	

Table VIII. Computation Time: Cyclic Schedule Management

Problem	Number Machines	Number Operations	Run Time to Reduce Expected Cycle Time by 10%	
			Using Gradient Buffering	Using GSG Operation Mean Reduction
1	9	92	21.34	17.91
2	8	42	2.77	6.00
3	9	88	18.51	14.78
4	10	130	20.97	35.13
5	10	68	14.42	16.42
6	10	70	10.19	14.89
7	10	57	10.58	10.06
8	9	75	16.32	15.76
9	10	62	14.31	12.02
10	9	102	25.04	19.93

6. CONCLUSIONS

Cyclic scheduling is a practical and effective way to control the flow of material in many environments. Certainly, more research related to this type of scheduling concept is needed. We believe that the primary focus of future research should be to extend the practical applicability of cyclic scheduling while developing a clearer characterization of the types of problems for which cyclic scheduling is a viable solution technique.

Much work also remains to be done on the development of deterministic cyclic scheduling heuristics. Existing heuristics were primarily developed for very specific environments. There are so few general heuristics that it is difficult to evaluate how well they work and how much room for improvement remains. These schedules form the starting point for the analysis presented in this paper. Integration of the cyclic sequence development with the stochastic modeling and analysis presented here would be a useful goal of future research.

The state space size for the Markov chain models can grow so large for many practical problems that exact analysis is impractical. In this case, the Markovian simulation approach is appropriate and effective. However, the exact algorithms have the additional advantage of generating powerful state-based information that simulation can't easily provide. Research that could show how to limit the size of the state space but not significantly affect the accuracy of the model could greatly expand the applicability of exact analysis. For example, prohibiting transitions out of states caused by tasks being completed that are extremely unlikely to be critical from that state may effectively reduce the state space size while retaining model accuracy. Identifying such constraints might also be a good idea for operational considerations. One can theoretically extend the Markov Chain model presented in this paper to cover any feature that does not violate the Markovian property. That is, given the current state, the past and future must be independent. Most extensions expand the state space and many enlarge the representation of the state as well. The loss in efficiency of evaluating the model (through exact analysis or simulation) must then be weighed against the additional insight provided by the more realistic and complex model. Bowman (1990) shows how one could model separate setups,

transfer lots that differ from process lots, separate (major) machine breakdowns, buffer stock, variable lot sizes, and cyclic release rules.

We have specifically identified only a few problems that can be addressed with a greedy gradient-based approach and examined only a portion of these in any detail from a computational viewpoint. More problems need to be constructed to test the effectiveness and robustness of the algorithms.

The idea of characterizing problems where deterministic approaches are preferable to stochastic ones or vice versa is an interesting one. A first attempt for the problem of “crashing” a schedule was made here. Further work on such characterizations of this problem and others is also needed.

The tradeoff between WIP and throughput measures the effectiveness of a cyclic schedule. Much work remains to be done to develop and evaluate release rules, safety time management heuristics, and other methods for improving the efficient frontier of this tradeoff.

The scope of the problem can be broadened to include variability of customer demand. In this case, the value of a schedule would be assessed in terms of the cost and responsiveness of meeting uncertain customer demands. Release rules, safety time and safety stock, and other approaches have to be considered along with the management of finished goods inventory in developing a more integrated approach to broader, more realistic problems. Such research could be extremely valuable in enhancing the usefulness of cyclic scheduling and in characterizing the types of problems that can be successfully addressed with cyclic schedules.

APPENDIX: PROOFS OF GRADIENT RESULTS

Infinitesimal Perturbation Analysis is a technique for estimating the gradient of some performance measure of a simulation with respect to some parameter (vector) by analyzing sample path information. The approach was first outlined by Ho (1979).

The first basic idea of IPA is that each event epoch of a simulation run (or sample path) can be expressed as the sum of some subset of the random variates generated that drive the simulation. This is called the triggering sequence for the event.

The second key idea is that, as long as the distribution functions of the random variables are continuous and 0 at $x = 0$, then the (stochastic) derivative of the event epoch with respect to a specific parameter is the sum of the derivatives of the random variables in the triggering sequence. Applied to the problem considered in this paper, the first idea says:

$$T = \sum_{j=1}^n X_j(\theta) 1(a^j \text{ is on the critical path}). \quad (\text{A1})$$

In this case, T is the time to complete the schedule, $X_j(\theta)$ is the random variable for the length of arc a^j with θ being the parameter vector (of arc length means), and $1(a^j \text{ is on the critical path})$ is one if arc a^j is on the critical path and zero otherwise. The second idea gives us:

$$\frac{\partial T}{\partial \theta_j} = \sum_{j=1}^n \frac{\partial X_j(\theta)}{\partial \theta_j} 1(a^j \text{ is on the critical path}). \quad (\text{A2})$$

This will be referred to as IPA Result 1. These ideas are proven in Glasserman (1988).

If the random variables in the sum are exponentially distributed, which they are for the Markov chain model, then, for each random variables X_j , we have:

$$U = 1 - \exp\left\{-\frac{1}{\theta_j} X_j\right\}, \quad (\text{A3})$$

where U is a Uniform (0,1) random variable. Thus:

$$X_j = -\theta_j \ln(1 - U), \text{ and} \quad (A4)$$

$$\frac{\partial X_j}{\partial \theta_j} = -\ln(1 - U) = \frac{X_j}{\theta_j}. \quad (A5)$$

If we integrated this result over the joint distribution of arc lengths (or ran replicates of a simulation and computed the average gradient of the sample paths) we would be evaluating $E\left(\frac{\partial T}{\partial \theta_j}\right)$. Of course, the measure of interest is actually $\frac{\partial E(T)}{\partial \theta_j}$. Glasserman (1988) proves the two are equal if the following two conditions hold:

Let

$\epsilon(s)$ = the set of events that can possibly occur given the system is in state s , and

$p(s'; s, j)$ = the probability of going from s to s' if event j occurs, defined $\forall j \in \epsilon(s)$.

Condition 1: If $\{i, j\} \subseteq \epsilon(s)$ and $p(s'; s, i) > 0$, then $j \in \epsilon(s')$.

Condition 2: For any s if $\{i, j\} \subseteq \epsilon(s)$ and $p(s_1; s, i)p(s_2; s_1, j) > 0$, there is an s_3 such that $p(s_3; s, j) = p(s_2; s_1, j)$ and $p(s_2; s_3, i) = p(s_1; s, i)$.

Condition 1 requires that no event is able to deactivate another. Condition 2 basically requires that if two events are possible from the same state, the event that is reached if they both occur is independent of the order in which they occur. To keep notation as simple as possible, verification of the conditions will be done for the case where i and j do not refer to the completion of the first or last arc corresponding to a machine. These end condition events are conceptually the same, but notationally more difficult.

Verification of Condition 1:

If event i is the completion of arc a^i , it must be that:

$$s' = s + \vec{e}_{M(a^i)} \quad (\text{A6})$$

We also know:

$$s'_{M(a^j)} = s_{M(a^j)} = j - 1, \text{ and,} \quad (\text{A7})$$

since the precedence requirements for arc a^j are met in state s , they must also be met in state s' .

Together, these facts imply $j \in \epsilon(s')$.

Verification of Condition 2:

The events of interest are the completion of arcs. Furthermore, if an arc completes, the state entered is deterministic. If the event i refers to the completion of arc i , it must be that:

$$s_1 = s + \vec{e}_{M(a^i)}, \text{ and} \quad (\text{A8})$$

$$s_2 = s_1 + \vec{e}_{M(a^j)}. \quad (\text{A9})$$

Let $s_3 = s + \vec{e}_{M(a^j)}$, and all conditions are met, since the probabilities in the condition are all equal to one.

The result $\frac{\partial E(T)}{\partial \theta_j} = E\left(\frac{\partial T}{\partial \theta_j}\right)$ will be referred to as IPA Result 2. Theorem 2 can now be proven.

Proof of Theorem 2:

$$\frac{\partial E(T)}{\partial \theta_j} = E\left(\frac{\partial T}{\partial \theta_j}\right) \quad (\text{IPA Result 2})$$

$$= E\left[\sum_{i=1}^n \frac{\partial X_i}{\partial \theta_j} 1(a^j \text{ is on critical path})\right] \quad (\text{IPA Result 1})$$

$$\begin{aligned}
&= E \left[\frac{\partial X_i}{\partial \theta_j} \mathbf{1}(a^j \text{ is on critical path}) \right] \\
&= E \left[\frac{X_j}{\theta_j} \mathbf{1}(a^j \text{ is on critical path}) \right] \\
&= \frac{1}{\theta_j} C(a^j) E \left[X_j \mid a^j \text{ is on critical path} \right]. \tag{A10}
\end{aligned}$$

Proof of Corollary 2.1:

Let $X_j = A_1 + A_2 + \dots + A_n$,

where A_i is exponentially distributed with mean $\lambda(i)$, $i = 1, 2, \dots, n$.

Then $\theta_j = \lambda(1) + \lambda(2) + \dots + \lambda(n)$.

If the SCV of X_j is held constant, then:

$$\begin{aligned}
\theta_j + \Delta\theta_j &= \left(\lambda(1) + \Delta\theta_j \frac{\lambda(1)}{\theta_j} + \dots + (\lambda(n) + \Delta\theta_j \frac{\lambda(n)}{\theta_j}) \right). \\
\Rightarrow \frac{\partial E(T)}{\partial \theta_j} &= \frac{\lambda(1)}{\theta_j} \frac{\partial E(T)}{\partial \lambda(1)} + \dots + \frac{\lambda(n)}{\theta_j} \frac{\partial E(T)}{\partial \lambda(n)} \\
&= \frac{1}{\theta_j} C(\text{arc } 1) E(A_1 \mid \text{arc } 1 \text{ is on critical path}) + \dots \\
&\quad + \frac{1}{\theta_j} C(\text{arc } n) E(A_n \mid \text{arc } n \text{ is on critical path}) \\
&= \frac{1}{\theta_j} C(t^j) E(A_1 \mid \text{task } j \text{ is on critical path}) + \dots \\
&\quad + \frac{1}{\theta_j} C(t^j) E(A_n \mid \text{task } j \text{ is on critical path}) \\
&= \frac{1}{\theta_j} C(t^j) E(X_j \mid \text{task } j \text{ is on critical path}). \tag{A11}
\end{aligned}$$

REFERENCES

- [1] BOWMAN, R.A. (1990). Stochastic Analysis of Cyclic Schedules. Ph.D. Dissertation, School of OR&IE, Cornell University.
- [2] DEL PILAR, R. and P.L. Jackson (1989). The Cyclic Scheduling Problem Formulation. Tech. Rep. 868, School of OR&IE, Cornell University.
- [3] ELMAGHRABY, S.E. (1977). *Activity Networks: Project Planning and Control by Network Models*. John Wiley and Sons, New York.
- [4] FULKERSON, D.R. (1961). A Network Flow Computation for Project Cost Curve. *Mgmt. Sci.*, 7, 2, 167-178.
- [5] GLASSERMAN, P. (1988). Performance Continuity and Differentiability in Monte Carlo Optimization. *Proceedings of the 1988 Winter Simulation Conference*, M. Abrams, P. Haigh, and J. Comfort (eds.).
- [6] GRAVES, S.C., H.C. Meal, D. Stefek, and A.H. Zeghmi (1983). Scheduling of Re-entrant Flow Shops. *J. of Ops. Mgmt.*, 3, 4, 197-203.
- [7] HITZ, K.L. (1980). Scheduling of Flexible Flow Shops - II. Tech Rep. LIDS-R-1049, Laboratory for Information and Decision Systems, Mass. Inst. of Tech.
- [8] HO, Y.C., M.A. Eyler and T.T. Chien (1979). A Gradient Technique for General Buffer Storage Design in a Serial Production Line. *Int. J. Prod. Res.*, 7, 6, 557-580.
- [9] JACKSON, P.L., W.L. Maxwell, and J.A. Muckstadt (1988). Determining Optimal Reorder Intervals in Capacitated Production-Distribution Systems. *Mgmt. Sci.*, 34, 8, 938-958.
- [10] KULKARNI, V.G. and V.G. Adlakha (1986). Markov and Markov-regenerative PERT Networks. *Ops. Res.*, 34, 5, 769-781.
- [11] MAXWELL, W.L. and J.A. Muckstadt (1985). Establishing Consistent and Realistic Reorder Intervals in Production-Distribution Systems. *Ops. Res.*, 33, 6, 1316-1341.
- [12] MAXWELL, W.L., J.A. Muckstadt, L.J. Thomas, and J. VanderEecken (1983). A Modelling Framework for Planning and Control of Production in Discrete Parts Manufacturing and Assembly Systems. *Interfaces*, 13, 6, 92-104.
- [13] McCORMICK, S.T., M.L. Pinedo, S. Shenker, and B. Wolf (1988). Sequencing in an Assembly Line with Blocking to Minimize Cycle Time. Tech. Rep., School of IE&OR, Columbia University.
- [14] ROUNDY, R.O. (1985). 98% Effective Integer-Ratio Lot-Sizing for One Warehouse Multi-Retailer Systems. *Mgmt. Sci.*, 31, 11, 1416-1430.
- [15] ROUNDY, R.O. (1986). A 98% Effective Lot-Sizing Rule for Multi-Product, Multi-Stage Production Inventory Systems. *Math. of Ops. Res.*, 11, 699-727.
- [16] ROUNDY, R.O. (1988). Cyclic Schedules for Shops With Identical Jobs. Tech. Rep. 766, School of OR&IE, Cornell University.

- [17] VAN SLYKE, R.M. (1963). Monte Carlo Methods and the PERT Problem. *Ops. Res.*, 11, 5, 839-860.
- [18] WHYBARK, D.C. (1984). Production Planning and Control at Kumera Oy. *Prod. and Inv. Mgmt.*, 1, 71-82.
- [19] WITTROCK, R.J. (1985). Scheduling Algorithm for Flexible Flow Lines. *IBM J. of Res. and Dev.*, 29, 4, 401-412.
- [20] ZHANG, H. and S.C. Graves (1989). Cyclic Scheduling in a Stochastic Environment. Working Paper. Operations Research Center, Massachusetts Institute of Technology, Cambridge, Massachusetts.