Short-term production scheduling of an automated manufacturing facility

by Stanley B. Gershwin
Ramakrishna Akella
Yong F. Choong

We describe a new implementation of the Kimemia-Gershwin hierarchical policy for the real-time scheduling of flexible manufacturing systems. Major improvements result at all three levels of the policy. The algorithm simplification, resulting in substantial reductions of off-line and on-line computation time, is reported, as is the improvement in performance through the elimination of chattering. Simulation results based on a detailed model of a printed circuit card assembly facility are summarized.

1. Introduction

This paper describes major progress in the work reported by Kimemia [1] and Kimemia and Gershwin [2] on the on-line scheduling of flexible manufacturing systems. Major improvements to all levels of the hierarchical policy are reported and simulation results are summarized. The test bed used to evaluate the policy is a printed circuit card assembly facility. The results indicate that the approach is practical, well-behaved, and robust. A full description of the results appears in [3, 4].

A flexible manufacturing system (FMS) is one in which a family of related parts can be made simultaneously. It consists of a set of computer-controlled machines and transportation elements. The changeover time between different operations at a machine is small compared with operation times.

Processing a mix of parts makes it possible to utilize the machines more fully than otherwise. This is because different parts spend different amounts of time at the machines. Each part type may use some machines heavily and others very little or not at all. If complementary part types are selected for simultaneous production, the machines that are lightly used by some parts can be loaded with others that use them heavily.

In principle, balancing can keep several machines busy at the same time. However, scheduling such a system is difficult because there are several machines, several part types, and many parts. In addition, like all manufacturing systems, an FMS is subject to random disturbances in the form of machine failures and repairs, material unavailability, "hot" items or batches, and other phenomena. These disruptive effects complicate an already difficult optimization problem.

The short-term hierarchical scheduler is outlined in **Figure 1**. Its purpose is to mitigate the effects of major disruptions. It is assumed that production requirements are specified at a higher level of the hierarchy, which also provides initial data

on the mean time between failures (MTBF) and the mean time to repair (MTTR). The scheduler's key features are that it respects the capacity constraints of the system and provides for production losses due to machine failures by building up inventory.

In Kimemia and Gershwin's formulation, the top level of the algorithm required the solution of a difficult partial differential equation (Bellman's equation). Here we show that a simple heuristic procedure works well enough that a numerical solution is not necessary.

The middle level of the algorithm formerly required the solution of a linear programming problem at every time instant. While this was a burden on the computer resources, its more serious consequence was "chattering," the rapid switching of control parameters (here, the production rates of parts) from one value to another. This reduced the effectiveness of the algorithm, in terms of both performance and computation time. Now we need to solve the linear program only at machine failure and repair times, and chattering is entirely eliminated.

In the earlier work, the lower level of the algorithm sometimes caused the accumulation of material in the system. A new lower level has been devised which eliminates this problem and which is far simpler to program.

The paper is organized as follows. In Section 2, we describe Kimemia and Gershwin's continuous time formulation of the short-term scheduling problem. We also comment on the nature of the off-line and on-line computational issues. In Section 3, we describe the simplification that can be achieved in the computation of the off-line parameters. The modifications to achieve reduced computation and chattering in generating the production rates on line are described in Section 4. Section 5 describes the lower level that effectively translates production rates into part dispatch times. The manufacturing system that was used as test bed for our simulations is presented in Section 6. In Section 7, we describe the simulation results, concluding in Section 8.

2. Overview of hierarchical policy

The purpose of the short-term FMS scheduling algorithm is to solve the following problem: When should parts (whose operation times at machines are on the order of seconds or minutes) be dispatched into an FMS whose machines are unreliable (with mean times between failures and mean times to repair on the order of hours) to satisfy production requirements that are specified for a week? Kimemia and Gershwin decomposed the problem into two parts: a high-level continuous dynamic programming problem to determine the instantaneous production rates, and a combinatorial algorithm to determine the dispatch times at the bottom level.

The continuous part is treated as a dynamic programming problem. As such, it is naturally divided further into two

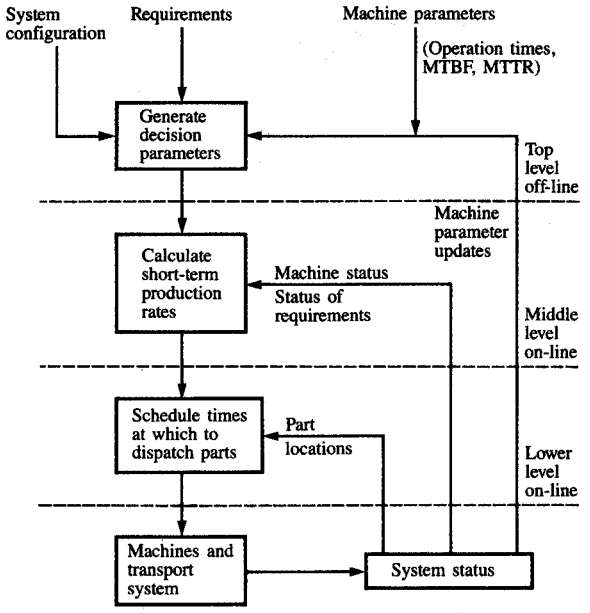

Figure 1

Short-term production hierarchy.

levels. The top level calculates a value or cost-to-go function and is executed off-line. The middle level is the realization of the maximum principle. It uses the cost-to-go function to determine instantaneous flow rates and part mixes.

Assume that the production requirements are stated in the form of a demand rate vector $d(t)$. Let the instantaneous production rate vector be denoted $u(t)$. Define $x(t)$ to be production surplus. It is the cumulative difference between production and demand and satisfies

$$\frac{dx}{dt} = u(t) - d(t). \tag{1}$$

If $x(t)$ is positive, more material has been produced than is currently required. This surplus or safety stock helps to ensure that material is always available over the planning horizon. However, it has a cost. Expensive floor space and material-handling systems must be devoted to storage. In addition, working capital has been expended in the acquisition and processing of stored materials. This capital is not recovered until the processing is complete and the inventory is sold.

If $x(t)$ is negative, there is a backlog, which is even more costly. Backlog represents either starved machines downstream or unsatisfied customers. In the former case, valuable capital is under-utilized; in the latter, sales and goodwill may be lost.

A cost $g(x)$ representing both the inventory and backlog costs can be assigned. The objective of the policy is to

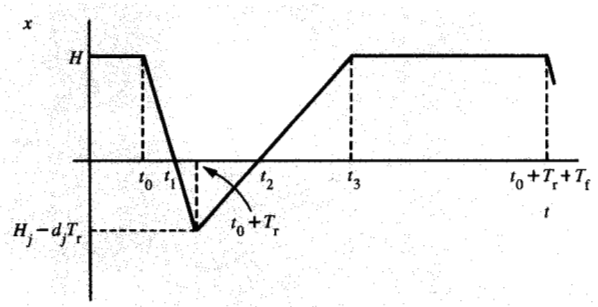

Figure 2

Simplified trajectory.

compute production rates to meet production targets while minimizing the total cost.

However, the choice of the production rate is not unconstrained. The production rate vector u is limited by the capabilities of the machines. Let part type j require time τ_{ij} for all of its operations on machine i. (Note that the order in which parts go to machines is not relevant to this calculation, nor is the number of times a part visits a machine. For simplicity, we assume here that there is only one path for each part.) Then

$$\sum_{j=1}^{n} \tau_{ij} u_j(t) \leq \alpha_i(t), \tag{2a}$$

where $\alpha_i(t)$ is 1 if machine i is operational and 0 if it is down. More generally, if there is a set of identical type-i machines, $\alpha_i(t)$ is the number of these that are operational at time t. Note also that

$$u_j \geq 0. \tag{2b}$$

Inequalities (2a) and (2b) can also be written as

$$u(t) \in \Omega[\alpha(t)]. \tag{2}$$

These requirements and constraints on the production rates can be expressed as a dynamic optimization problem [3]:

$$\begin{cases} J[x(0), \alpha(0)] = \min\limits_{u(\cdot)} E\left\{\int g[x(t)]dt \mid x(0), \alpha(0)\right\} \\ \text{subject to (1), (2), and initial} \\ \text{conditions } x(0) \text{ and } \alpha(0). \end{cases} \tag{3}$$

The scheduling policy that results from solving (3) can be decomposed into the three levels shown in Fig. 1:

1. Top level: Evaluation of $J(\cdot, \cdot)$ off-line.
2. Middle level: The on-line computation of the instantaneous production rates is achieved at this level. It can be shown [3] that the solution is given by solving

$$\begin{cases} \text{minimize } \dfrac{\partial J(x, \alpha)}{\partial x} u \\ \text{subject to } u \in \Omega(\alpha). \end{cases} \tag{4}$$

   Note that this requires the results of the computation at the top level.
3. Lower level: The production rates u are translated into actual part dispatch times.

Kimemia and Gershwin [2] describe a method of computing the required quantities at each of the three levels. At the top level, they suggest a decomposition by which the nth-order Bellman partial differential equation for $J(x, \alpha)$ is replaced by n first-order ordinary differential equations (where n is the number of part types, i.e., the dimensionality of x, u, and d). However, the solution is computation-intensive. In the next three sections, we describe new, computationally effective approximations of the three levels.

3. Top level: Cost-to-go function

We present a quadratic approximation for the value function of the dynamic programming problem (3). Not only does this reduce data requirements, but it also simplifies the middle-level (maximum principle) computation. As a result, the cost-to-go function is

$$J(x, \alpha) = \frac{1}{2} x^T A(\alpha) x + b(\alpha)^T x + c(\alpha), \tag{5}$$

where $A(\alpha)$ is a positive definite diagonal matrix, $b(\alpha)$ is a vector, and $c(\alpha)$ is a scalar. In this section, we describe a method for choosing these coefficients for some values of α.

The function $J[x(t), \alpha]$ is a decreasing function of t when α remains constant. The *hedging point* is the value of x that minimizes $J(x, \alpha)$ for a fixed α. It is the value that x reaches if α stays constant for a long time and if d is feasible, i.e., if $d \in \Omega(\alpha)$. The hedging point is important to the behavior and performance of the algorithm because it is the point that x spends most of its time either moving toward or resting at. Intuitively, it is the level to which one builds up inventory to compensate for future production losses due to machine failures.

Here, the hedging point is given by

$$H_j(\alpha) = -\frac{b_j(\alpha)}{A_{jj}(\alpha)}. \tag{6}$$

In order to estimate the hedging point, consider **Figure 2**, which demonstrates a typical trajectory of $x_j(t)$. Assume that x_j has reached $H_j(\alpha)$, the hedging point corresponding to the machine state before the failure. Then u_j is chosen to be d_j and x_j remains constant.

A failure occurs at time t_0 that forces u_j to be 0. This causes x_j to decrease at rate $-d_j$. In fact, if the failure lasts for a length of time T_r, the minimum value of x_j is

$$H_j - d_j T_r. \tag{7}$$

Recall that T_r, the repair time, is a random variable. Just after the repair (at time $t_0 + T_r$), u_j is assigned the value U_j. Assuming that this value is greater than that of demand d_j, x_j increases at rate $U_j - d_j$ until it reaches the hedging point H_j. At that time, u_j resumes its old value of d_j and x_j stays constant until the next failure, at time $t_0 + T_r + T_f$. T_f is also a random variable.

To simplify the analysis, we assume that

1. The value of u_j is constant between the repair ($t_0 + T_r$) and when x_j reaches H_j.
2. T_r and T_f can be replaced by their expected values, the MTTR and MTBF.
3. The cost function $g(\cdot)$ in (3) penalizes positive areas in Fig. 2 with weight a and negative areas with weight b, where a and b are positive scalars.

The objective of this is to obtain an approximation to $J(\cdot, \cdot)$. We do this by choosing H to minimize the total area under the positive and above the negative parts of the trajectory in Fig. 2. We obtain

$$H_j = \frac{T_r d_j(bU_j - ad_j) - T_f a d_j(U_j - d_j)}{(a + b)U_j}. \tag{8}$$

This approach cannot be applied for machine states in which the demand is not feasible. Either there is no hedging point for such a state, or it is larger than (8).

$A_{jj}(\alpha)$ must be positive in order for J to be convex. Its value reflects the relative priority of part type j. Parts that have great value, or that would cause great difficulty if backlogged, or that pass through relatively unreliable machines should have large values of A_{jj}. For example, in our simulations we let A_{jj} be the number of machines through which part j passed. This was because we treated all parts as equally valuable and because all machines had the same reliability parameters.

4. Middle level: Maximum principle

In this section we describe a computationally effective method of computing the instantaneous production rates.

- *Chattering*

The optimal production rate vector $u(t)$ satisfies the following linear programming problem at every time instant t:

$$\begin{cases} \text{minimize } \dfrac{\partial J(x, \alpha)}{\partial x} u \\ \text{subject to } u \in \Omega(\alpha). \end{cases} \tag{9}$$

This is a feedback law, since the problem is specified only when x and α are determined. The numerical solution of (9) is implemented on-line at the middle level of the hierarchical algorithm.

For every α, x-space is divided into a set of regions (open, connected sets) and the boundaries between them. Each region is associated with a corner of $\Omega(\alpha)$. When x is in the interior of region R_k, the value of u that satisfies (9) is the corresponding corner P_k.

Gershwin, Akella, and Choong [3] show that when J is quadratic, the R_k regions are cones.

Kimemia and Gershwin [2] implemented (9) in a simulation by solving it every time step (one minute). This worked well while x was in the interior of a region R_k. However, when x crossed certain boundaries between regions, this approach worked poorly. After $x(t)$ crossed such a boundary, which we call *attractive*, the value of u corresponding to the new region R_m was such that the derivative (1) pointed toward R_k. When $x(t)$ crossed the boundary back into R_k, the derivative pointed again to R_m. Thus, $u(t)$ jumped between the adjacent corners P_k and P_m of $\Omega(\alpha)$.

This kind of behavior is called *chattering* and is well known in the optimal control literature. Here, it causes the flow rate to change more frequently than parts are loaded into the system. As a result, if the demands on the system are near its capacity, it will fail to meet the demands. This was observed by Kimemia [1].

- *Linear program*

Linear program (9) can be written

$$\begin{cases} \text{minimize } C(x)^T u \\ \text{subject to } Du = e, \\ \qquad u \geq 0, \end{cases} \tag{10}$$

where u has been expanded with slack variables so that the inequality constraint (2a) can be written as an equality, and

$$C(x) = Ax + b = A(x - H).$$

(Note that arguments α and t are suppressed.) The standard solution of (10) [5] breaks u into basic (u_B) and nonbasic (u_N) parts, with $C(x)$ and D broken up correspondingly. The basic part of D is a square, invertible matrix. By using the equality in (10), u_B can be eliminated, and the problem becomes

$$\begin{aligned} &\text{minimize } C_R(x)^T u_N \\ &\text{subject to } u_N \geq 0, \end{aligned} \tag{11}$$

where the constraint on u_B has been suppressed, and where

$$C_R(x)^T = C_N(x)^T - C_B(x)^T D_B^{-1} D_N$$

is the reduced cost. If all components of C_R are positive, there is a solution to (11): $u_N = 0$. This and the corresponding u_B form an optimal solution to (10). Otherwise, (11) does not have a bounded optimal solution and (11) is not equivalent to (10).

It is important to note that since C is a function of x, the basic/nonbasic breakup of this problem depends on x. That

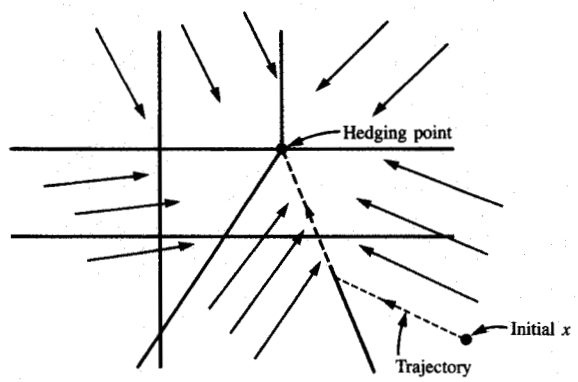

Figure 3

Regions of x-space.

is, the set of components of u that are treated as basic varies as a function of x.

At every x in region R_k, corner P_k is the optimal value of u for (10). In each region, then, there must be a basic/nonbasic breakup of (10) which is constant. Consequently, $C_R(x)$ must be positive everywhere in its own region and it must have some negative components elsewhere. The boundaries of the regions are determined by some components of $C_R(x)$ being equal to zero.

The boundaries of the regions are portions of hyperplanes because $C(x)$ is linear in x. Consequently $C_N(x)^T$ and $C_B(x)^T$ and therefore $C_R(x)$ are also linear in x.

• *Qualitative behavior*

After a machine state change, $x(t)$ is almost always in the interior of a region. Since u is constant throughout a region, dx/dt is also constant. Thus, x travels along a straight line in the interior of each region. As indicated in **Figure 3**, such lines may intersect with one or more boundaries of the region. When x reaches a boundary, u (and therefore dx/dt) changes.

If the boundary is not *attractive*, $x(t)$ moves into the interior of the next region until it reaches the next boundary. The production rate vector u jumps to an adjacent corner. This behavior continues until $x(t)$ encounters an *attractive* boundary. At this time, the trajectory begins to move along the boundary and $u(t)$ jumps to a point on the edge of $\Omega(\alpha)$ between the corners corresponding to the regions on either side of the boundary.

This behavior continues: $x(t)$ moves to lower-dimensional boundaries and $u(t)$ jumps to higher-dimensional faces. It stops when either the machine state changes (that is, a repair or failure takes place) or $u(t)$ becomes constant. If the demand is feasible, the constant value for u is d. When that happens, x also becomes constant and its value is the hedging point. If the demand is not feasible, x does not become constant. Instead, some or all of its components decrease without limit.

Consequently, the future behavior of $x(t)$ can be determined from its current value if the machine state remains constant. We call this future behavior the *projected trajectory*.

• *Calculation of the projected trajectory*

Assume that the conditional future trajectory is to be calculated at time t_0. This may be due to a machine state change. As soon as the machine state change occurs (at t_0), linear program (10) is solved. Thus the basic/nonbasic split is determined and the $C_R(x)$ function is known. In general, x appears in the interior of a region, and therefore all components of $C_R(x)$ are strictly positive. One or more components are zero on a boundary of a region.

The production rate vector at $t = t_0$ is denoted u_0. The production rate remains constant at this value until $t = t_1$, which is to be determined. In (t_0, t_1), x is given by

$$x(t) = x(t_0) + (u_0 - d)(t - t_0),$$

where $x(t_1)$ is on a boundary. Then t_1 is the smallest value of t for which some component of $C_R[x(t)]$ is zero. It is easy to calculate this quantity since C_R is linear in x and x is linear in t. Once t_1 is found, $x(t_1)$ is known. Define $h[x(t)]$ to be the component of $C_R[x(t)]$ that reaches zero at $t = t_1$. Because h is a linear scalar function of x, we can write

$$h[x(t)] = f^T[x(t) - x(t_1)],$$

where f is a vector of coefficients.

For $t > t_1$, there are two possibilities. The trajectory may enter the neighboring region and travel in the interior until it reaches the next boundary. Alternatively, it may move along the boundary it has just reached. To determine whether or not the boundary is attractive, we must consider the behavior of $h[x(t)]$ in its neighborhood.

We know that $h(x)$ is negative in the region across the boundary since this is how the regions are defined. We must determine whether h is increasing or decreasing on trajectories inside that region. If h is decreasing, x moves away from the boundary (where h is zero) into the interior. If h is increasing, trajectories move toward the boundary, which must therefore be attractive.

One value of x which is just across the boundary is

$$x'' = x(t_0) + (u_0 - d)(t_1 + \epsilon - t_0)$$

$$= x(t_1) + (u_0 - d)\epsilon.$$

This is the value x would have if u were allowed to be u_0 until $t_1 + \epsilon$.

Let u'' be the solution to (10) in the adjacent region. That is, (10) is solved with x given by x''. (This can be performed efficiently.) Let x^+ be the value of x at $t_1 + \epsilon$ if u'' were used after t_1. That is,

$x^+ = x(t_1) + (u'' - d)\epsilon.$

Then

$h(x'') = f^T(u'' - d)\epsilon.$

Therefore h is increasing and the boundary is attractive if and only if

$f^T(u'' - d) > 0.$

If the boundary is not attractive, define $u_1 = u''$. Then the process is repeated to find t_2, $x(t_2)$, t_3, $x(t_3)$, and so forth until an attractive boundary is encountered. Recall that this is an on-line computation that is taking place at time t_0. The future trajectory is now being planned.

If the boundary is attractive, a value of u must be determined which will keep the trajectory on it. Otherwise chattering will occur. For the trajectory to stay on the boundary,

$h[x(t)] = 0,$

or, since $h[x(t_1)] = 0$,

$$\frac{d}{dt}h[x(t)] = f^T(u - d) = 0. \tag{12}$$

Although u is an optimal solution to (10), it is no longer determined by this linear program. In fact u_0, u'', and any convex combination of them are optimal. This is because one or more of the reduced costs is zero while x is on a boundary. Consequently, the new scalar condition (12) is required to determine the solution. The linear program is modified as follows:

$$\begin{cases} \text{minimize } C(x)^T u \\ \text{subject to } Du = \epsilon, \\ \qquad u \geq 0, \\ \qquad f^T u = f^T d. \end{cases} \tag{13}$$

The solution to (13) is the value of u that keeps the trajectory on the boundary. As before, this value is maintained until a new boundary is encountered.

New boundaries may still be attractive or unattractive. The same tests are performed: x is allowed to move slightly into the next region to determine the value of u. The time derivative of the component of the reduced cost that first reaches zero (h) is examined. If it is negative, the boundary is unattractive and the trajectory enters the new region. If it is positive, a new constraint is added to linear program (13).

Constraints, when added to (13), are not deleted. As the number of constraints increases, u is found on surfaces in $\Omega(\alpha)$ that increase in dimension and x is found in regions of decreasing dimension.

Since this is a finite dimensional system, this process must terminate. The vector $x(t)$ is eventually of the form

$x(t) = x(t_j) + (u - d)t.$

If d is feasible, when enough linearly independent constraints (12) are added to (13), the only feasible solution is $u = d$ and x is constant. If d is not feasible, some or all of the components of $u - d$ are negative. The corresponding components of x decrease without limit.

5. Lower level: Discrete part dispatch

The new part-loading scheme is based on the conditional future trajectory $[x(s), s \geq t]$. Define the actual surplus of part type j at time t to be

$x_j^A(t) = \{\text{number of parts of type } j \text{ loaded during } (0, t)\} - d_j t.$

Note that $x_j^A(t)$ is an irregular sawtooth function of time. It jumps by 1 each time a part is loaded. At other times, it decreases at rate d_j.

The loading strategy ensures that $x^A(t)$ is near $x(t)$. The strategy is this: At each time step t, load a part of type j if

$$x_j^A(t) < x_j(t). \tag{14}$$

Do not load a part of type j otherwise. A rule is required to resolve conflicts; it may not matter what that rule is since conflicts do not arise very often.

6. An automated card assemby line

In this section, we describe a system to which the hierarchical scheduler is applicable. Our purpose in using this system is to assess the scheduler in a realistic setting.

- *Purpose of system*

An automated card assembly line is being built up in stages, through a series of "minilines." The portion of the system of interest to us is the stage consisting of insertion machines. Printed circuit cards from a storage area upstream arrive at the loading area of the insertion stage. Each card is placed in a workholder, which is introduced into the system. It goes to the machines where the electronic components it requires are inserted. It then leaves the system and goes to the downstream stages, which consist of testing and soldering machines.

There are several types of insertion machines, each of which inserts one mechanically distinct type of component. The common ones are SIPs (single in-line package inserters), DIPs (dual in-line package inserters), MODIs (multiform modular inserters) and VCDs (variable center distance inserters). By loading different components, the line can be used to assemble a variety of cards.

In order to concentrate on the operational issues of the FMS, we assume that component loading has already been determined. The changeover time is small among the family of parts producible with a given component loading. We also restrict our attention to the Miniline 1300, whose schematic is shown in **Figure 4**. This consists of a DIP, a VCD, and two SIPs. Each of the machines also has an associated buffer, which can hold 30 parts.

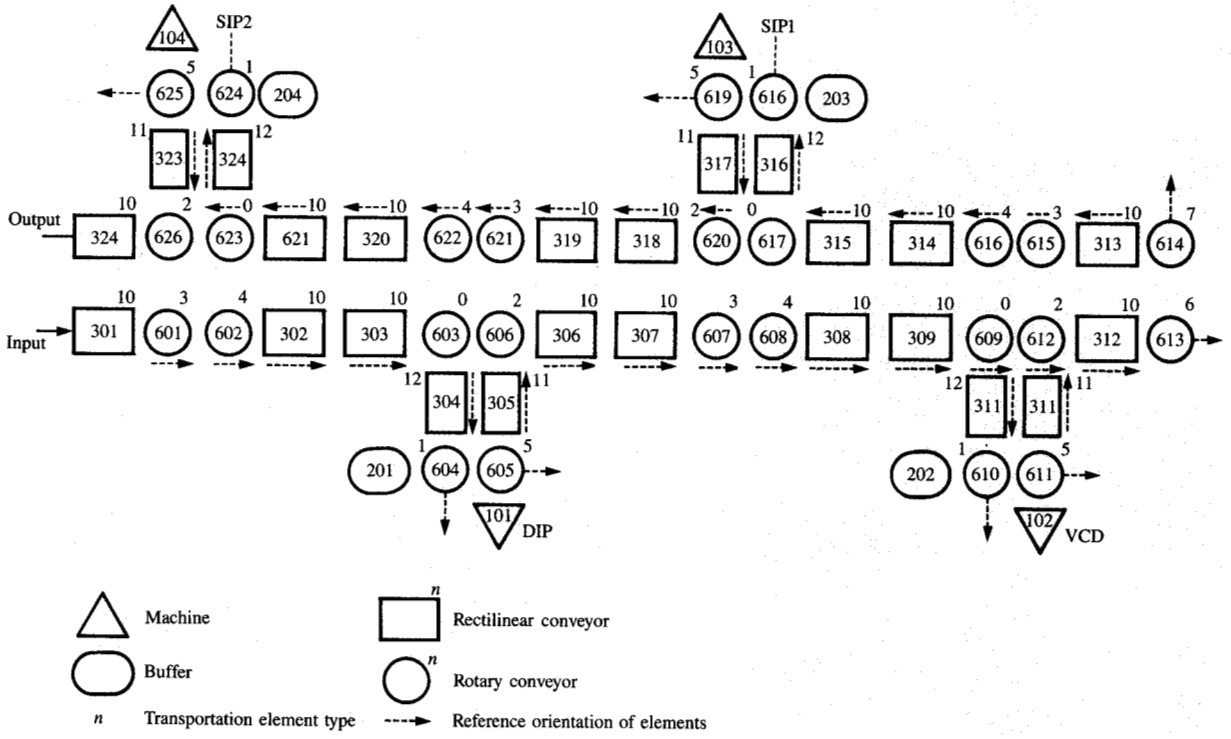

Figure 4

Schematic of Miniline 1300.

Table 1 Transportation mechanism.

Transfer time of card from element to element (straight or rotation)	one second
Rotation time	six seconds
Number of movements to transfer card via rotary mode	one rotation and one transfer

• *Transportation system*

The workholders are loaded at input station 301 and then move to each of the required machines. Movement is along straight or rotating elements. The straight elements are used to move parts in a single, fixed direction and are represented by rectangles. The rotating elements are for 90° turns and are represented by circles. Representative movement times are listed in **Table 1**.

Movement of cards in the vicinity of a work station (insertion machine, associated buffer, and transport elements) follows a common pattern. Cards arrive at a rotating element such as 603 and either turn toward the insertion machines or move straight on. The cards going to machines (e.g., 101) either wait at input elements such as 605 or go into buffers such as 201. After all the required components have been inserted, a similar movement takes the card out of the insertion machine and onto output element 305. After element 606 is rotated toward the work station, the card is placed on it. Element 606 is rotated back to its original position and the card is then loaded onto the next transportation element (306). Finally, after going through the entire system, the cards exit from output element 324.

• *Machine parameters and part data*

The mean time between failures (MTBF) and mean time to repair (MTTR) of the machines are listed in **Table 2**. The average fraction of time a machine is available is the time it is available for production divided by the total time. This quantity, called the efficiency or availability of a machine, is also listed in Table 2.

There are other random perturbations affecting the system. These include machine tool jams, which occur when a machine jams in trying to insert a component. Rather than being regarded as a failure, this small but regular (approximately once every 100 insertions) disturbance can be modeled as part of the processing time.

Normally, there are several part (card) types being processed in the system. We limit our experiment to only six types to better examine the hierarchical policy. Typical demand rates are listed in **Table 3**, with the operation times required by each card type at each of the machines. These include the processing time and the time to move in and out of each machine.

- *Loading*

Loading describes how heavily the machines in a system must be utilized to satisfy demand. The expected utilization of a machine is equal to the ratio of the total machine time required to the expected machine time available. The total machine time required is the product of total demand and processing time. The expected time a machine is available is its time available multiplied by the total time period. Table 2 displays the average utilizations for the machines in the configuration reported in the runs in Section 6. These data were created to impose a heavy loading on the simulated production system. The actual utilization in any sample simulation run depends on the time history of machine failures and repairs during that run. This time sequence determines the actual amount of time a machine is available.

7. Simulation results

A detailed simulation of a flexible manufacturing system was written to test the hierarchical scheduling policy and to compare it with other reasonable policies. The simulation is described in [6]. A full description of the results appears in [4].

The simulation indicates that the method behaves very well. The performance measures that were used were the total production during a day, average work in process, and production balance. The latter measured how well production adhered to the specified ratio of production requirements for different parts.

The hierarchical method was compared with a set of simpler strategies. On all measures, the hierarchical method was superior: production was greater; work in process was less; and balance was closer to 100%, indicating that material was produced in nearly the specified ratio.

It was more robust, in that it was less affected by disruptive events. The difference between its performance on good days and on bad days was less than the difference for other strategies.

It was not vulnerable to parameter variations. A variety of different A and b coefficients chosen for J in (5) made almost no difference in the performance of the algorithm.

8. Summary and conclusions

The hierarchical scheduling policy devised by Kimemia and Gershwin for flexible manufacturing systems has been further developed and tested. This policy is designed to respond to random disruptions of the production process. In its current formulation, it treats unpredictable changes in the operational states of the machines: repairs and failures. All levels of the policy have been improved, and the policy shows great promise for practical application.

Table 2 Machine parameters.

Machine	*MTBF* (min)	*MTTR* (min)	*Efficiency* (%)	*Expected utilization* (%)
1	600	60	90.91	97.68
2	600	60	90.91	91.10
3	600	60	90.91	96.03
4	600	60	90.91	96.58

Table 3 Operation times and demand rates.

Machine	*Operation times* (s) *Part type*					
	1	*2*	*3*	*4*	*5*	*6*
1	40	40	0	0	20	60
2	0	0	60	30	40	40
3	0	100	0	0	70	0
4	0	0	0	80	0	80
	Demand rates (parts/s) *Part type*					
	1	*2*	*3*	*4*	*5*	*6*
	0.0080	0.0070	0.0060	0.0070	0.0025	0.0040

Acknowledgments

Research support was provided by the Manufacturing Research Center of the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, and by the U.S. Army Human Engineering Laboratory, under Contract No. DAAK11-82-K-0018.

References

1. J. G. Kimemia, "Hierarchical Control of Production in Flexible Manufacturing Systems," *Report No. LIDS-TH-1215*, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 1982.
2. J. G. Kimemia and S. B. Gershwin, "An Algorithm for the Computer Control of Production in Flexible Manufacturing Systems, *IIE Trans.* **15,** No. 4, 353–362 (December 1983).
3. S. B. Gershwin, R. Akella, and Y. C. Choong, "Short Term Production Scheduling of an Automated Manufacturing Facility," *Report No. LIDS-FR-1356*, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 1984.
4. R. Akella, Y. Choong, and S. B. Gershwin, "Performance of Hierarchical Production Scheduling Policy," *IEEE Trans. Components, Hybrids, Manuf. Technol.* **CHMT-7,** No. 3, 225–240 (September 1984).

5. D. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley Publishing Co., Reading, MA, 1977.
6. R. Akella, J. P. Bevans, and Y. Choong, "Simulation of a Flexible Electronic Assembly System," *Report No. LIDS-R-1485*, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 1985.

Received December 19, 1984; revised March 14, 1985

Ramakrishna Akella *Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.* Dr. Akella received the B.Tech. degree in electronics from the Indian Institute of Technology, Madras, in 1976, and the Ph.D. degree in systems engineering from the School of Automation, Indian Institute of Science, Bangalore, in 1981. During 1982, he was a CSIR Research Associate at the Indian Institute of Science, and spent the summer as a Postdoctoral Visitor with the Decision and Control Group, Division of Applied Sciences, Harvard University, Cambridge, Massachusetts. Since 1983, he has been a Postdoctoral Associate at the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology. His research interests include computer-aided manufacturing and the hierarchical control of large-scale systems. In particular, through interaction with IBM, he has been active in the modeling, simulation, and real-time scheduling of flexible manufacturing systems. Recent interests include scheduling and other issues for quick-turnaround VLSI systems. Dr. Akella is listed in the *International Who's Who in Engineering*, International Biographical Centre, Cambridge, England. He received the 1984 IBM Postdoctoral Fellow Award from the Manufacturing Research Centre at IBM, Yorktown Heights, New York. Other awards and honors have included a CSIR Research Associateship, an Indian Institute of Science Research Fellowship, the National Merit Scholarship, and a National Science Talent Scholarship.

Yong F. Choong *Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.* Mr. Choong is a graduate student in the Department of Mechanical Engineering. He received the B.Sc. and M.Sc. degrees in mechanical engineering from the Massachusetts Institute of Technology in 1981.

Stanley B. Gershwin *Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.* Dr. Gershwin received the B.S. degree in engineering mathematics from Columbia University, New York, in 1966 and the M.S. and Ph.D. degrees in applied mathematics from Harvard University, Cambridge, Massachusetts, in 1967 and 1971, respectively. He is a Principal Research Scientist at the MIT Laboratory for Information and Decision Systems and a Lecturer in the MIT Department of Electrical Engineering and Computer Science. He is Assistant Director of the MIT Laboratory for Information and Decision Systems, and President of Technical Support Software, Inc. (TSSI). In 1970–1971, he was employed by the Bell Telephone Laboratories in Holmdel, New Jersey, where he studied telephone hardware capacity estimation. At the Charles Stark Draper Laboratory in Cambridge, Massachusetts, from 1971–1975, he investigated problems in manufacturing and transportation. His interest in these areas, as well as control, optimization, and estimation, continues at MIT and at TSSI. Dr. Gershwin is a member of the American Association for the Advancement of Science, the IEEE Control Systems Society, the Institute of Industrial Engineers, the Operations Research Society of America, the Society of Manufacturing Engineers Manufacturing Management Council, and Tau Beta Pi.