February, 1984 LIDS-FR-1356

# SHORT TERM PRODUCTION SCHEDULING OF AN AUTOMATED MANUFACTURING FACILITY

by

Stanley B. Gershwin,

Ramakrishna Akella, and

Yong Choong

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, Massachusetts 02139

## ABSTRACT

We describe extensions to the on-line hierarchical scheduling scheme for flexible manufacturing systems of Kimemia and Gershwin. Major improvements to all levels of the algorithm are reported, including algorithm simplification, substantial reductions of off-line and on-line computation time, and improvement of performance by elimination of chattering. Simulation results based on a detailed model of an IBM printed circuit card assembly facility are presented.

## ACKNOWLEDGMENTS

We are grateful for research support which has been provided by the Manufacturing Research Center of the Thomas J. Watson Research Center of the International Business Machines Corporation; and by the U. S. Army Human Engineering Laboratory under Contract DAAK11-82-K-0018. We are also grateful for the guidance of Mr. Mike Kutcher and Dr. Chacko Abraham and for the early participation of Ms. Ethel Sherry-Gordon.

## 1. INTRODUCTION

This paper describes extensions to the work reported by Kimemia (1982) and Kimemia and Gershwin (1983) on the on-line scheduling of flexible manufacturing systems. Major improvements to all levels of the hierarchical algorithm are reported and simulation results are presented. The results indicate that the approach is practical, well-behaved, and robust.

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 do require them.

In principle, therefore, line 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, a FMS is subject to random disturbances in the form of machine failures and repairs, material unavailability, "hot" items or batches, and other phenomena. These effects further complicate an already difficult optimization problem.

Figure 1 represents the hierarchical nature of manufacturing system scheduling. At the longest time scale, decisions concerning total annual production and acquisition or reallocation of capital equipment are made on the basis of estimated demand.

Each month, production plans are made after demand updates are available. It is necessary, at that level, to choose part families. That is, parts that can share a set of production machines may not be able to share them simultaneously because of limitations, such as on tooling capacity. Once parts are grouped into families, the volume of production of each family is determined. That is, the sizes of batches are calculated. In addition, the times at which families are changed are found. At these moments, setup time costs are incurred.

On a weekly basis, short-term demands are scheduled. These are high-priority demands of random amounts that occur at random times and that must be satisfied quickly. They should be treated in a way that does not unnecessarily disrupt the production of the material that was previously planned. The batch sizes and setup times already determined may have to be adjusted. At a still lower level is the dispatch of individual parts into the system. The objective here is to produce the amounts specified

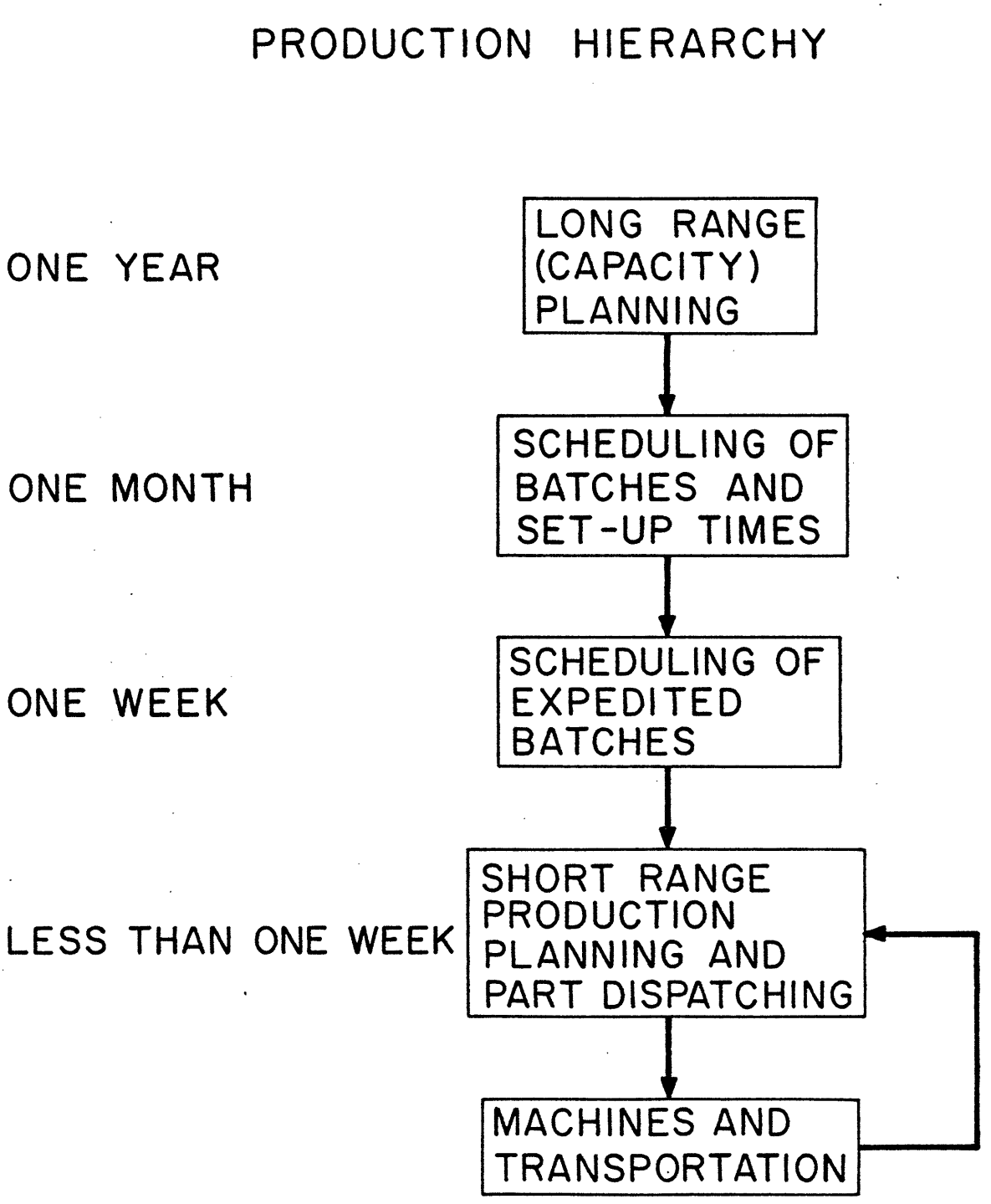


Figure 1. Long Term Production Hierarchy.

earlier at close to the required times. This level must be robust in the presence of such random events as machine failures and repairs, worker absences, and material shortages which can perturb the production process.

Kimemia and Gershwin describe a hierarchical approach to the low-level scheduling of an FMS (Figure 2). They restrict their attention to repairs and failures as sources of randomness. Their method is based on the approximate representation of the movement of discrete material as a continuous flow. A dynamic programming problem is formulated whose solution is the optimal mix of part production rates as a function of machine states (failed or operational) and cumulative production. The approximate numerical solution of this problem is proposed for the top level of this hierarchical algorithm (GENERATE DECISION TABLES). This calculation is expected to be performed off-line. A simplification is described in Section 3.

At the middle level is the implementation of the solution of the dynamic programming problem (CALCULATE SHORT TERM PRODUCTION RATES and CALCULATE ROUTE SPLITS). That is, it is the mix of production rates determined by the current machine states and deviation of cumulative production from cumulative demand. Kimemia and Gershwin showed that the evaluation of these quantities involves the solution of a linear programming problem at each time instant. Although the problem is not large and the simplex algorithm is efficient, the amount of on-line computation they required was larger than necessary. More importantly, this procedure led to "chattering," (a rapid switching between one production rate mix and another) which reduced the performance of the algorithm. Section 4 of this paper describes a method that treats both difficulties of the middle level.

The lowest level of the hierarchical algorithm has the task of converting the continuous instantaneous production rates calculated at the middle level to time instants at which parts are loaded into the system (SCHEDULE TIMES AT WHICH TO DISPATCH PARTS). At this level, the discrete nature of the production process can no longer be avoided. Kimemia proposed a dispatch strategy which was effective for systems that were not heavily loaded, although it appeared to lead to excessive inventory accumulation in the system. An alternative is pesented in Section 5, which is also computationally more efficient.

Simulation results are summarized in Section 6. They are described in full detail in Akella, Choong, and Gershwin (1984). The simulation is described in Akella, Bevans, and Choong (1984).

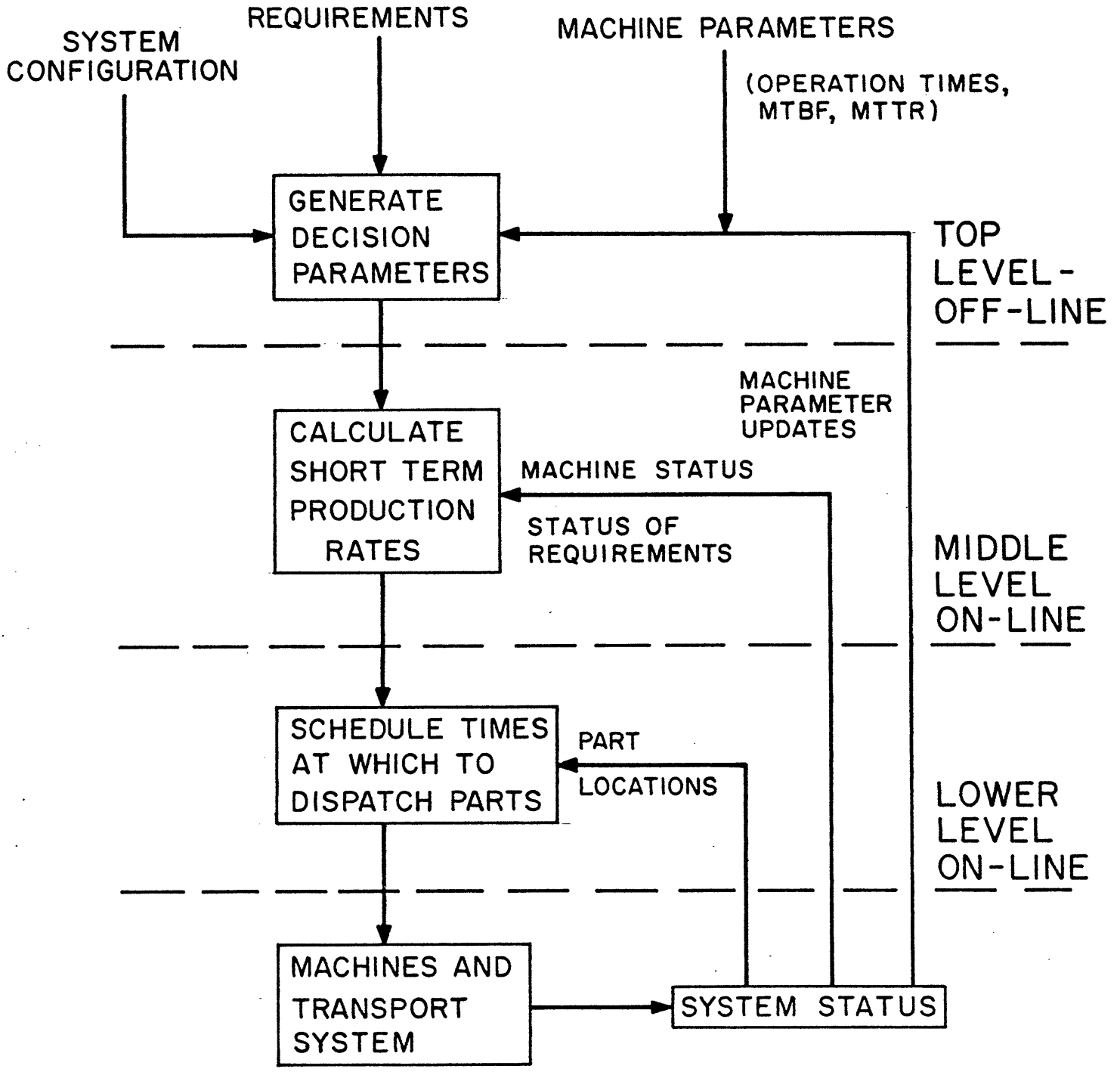


Figure 2. Short Term Production Hierarchy.

## 2.REVIEW OF HIERARCHICAL SCHEDULING

The purpose of the 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's approach decomposed the problem into two parts: a continuous dynamic programming problem to determine the instantaneous production rates and a combinatorial algorithm to determine the dispatch times so that the actual production rates agree with those that were calculated.

The continuous part was further divided into the top and middle levels. The top level calculated a value or cost-to-go function and was executed off-line. The middle level used the cost-to-go function to determine instantaneous flow rates and part mixes.

### Continuous Formulation

Assume that the production requirements are stated in the form of a demand rate vector d(t). Let the instanteous 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

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

If x(t) is positive, more material has been produced than is currently required. This surplus or safety stock is helpful to insure 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 also costly. Backlog represents either starved machines downstream or unsatisfied customers. In the former case, valuable capital is underutilized; in the latter, sales and good will may be lost.

The production rate vector u is limited by the capabilities of the machines. Let part type j require time $\tau_{ij}$ on machine i

for all of its operations. (Note that the order in which parts go to machines is not relevant for 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 while machine i is operational, the flow rates of all part types must satisfy

$$\sum_j \tau_{ij} u_j \leq 1.$$

If machine i is not operational, then no parts that go to it should be allowed into the system. That is, if $\tau_{ij} > 0$ and machine i is down, then $u_j = 0$. This is equivalent to requiring

$$\sum_j \tau_{ij} u_j \leq 0.$$

These inequalities can be combined as

$$\sum_j \tau_{ij} u_j(t) \leq \alpha_i(t) \qquad (2a)$$

where the right-hand side 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

$$u_j \geq 0. \qquad (2b)$$

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

$$u(t) \in \Omega(\alpha(t)). \qquad (2)$$

It is convenient to assume that failure and repair times are exponentially distributed. This makes the mathematical analysis tractable. In a production environment, more information about failure and repair times may be available than is represented by this distribution. Appropriate modifications can be made at the upper and lower levels of the hierarchy.

Top Level

As noted above, costs are incurred when x is far from zero. Kimemia and Gershwin describe the following optimization problem:

$$\text{minimize } E \int g(x(t))\, dt$$

$$\text{subject to (1), (2), and initial conditions } x(0) \text{ and } \alpha(0). \qquad (3)$$

The optimal value of the cost of this problem is called

$J(x(0),\alpha(0))$. The calculation performed at the top level is the approximate numerical evaluation of this function. Its derivative is used in the middle level to determine $u(x,\alpha)$, as described below.

The function $J(x,\alpha)$ satisfies a high order nonlinear set of coupled partial differential equations. Problem (3) is computationally difficult to solve since there is no analytic solution and no exact decomposition. An approximate technique solution is suggested by Kimemia and Gershwin. The simulation experience reported below suggests that the simpler approximation of Section 3 suffices for practical purposes.

Figure 2 displays the structure of the hierarchical algorithm. The numerical solution to problem (3) is implemented off-line at the top level.

Middle Level

Kimemia and Gershwin show that the solution to dynamic programming problem (3), the optimal production rate vector $u(t)$, satisfies the following linear programming problem. The argument $t$ is suppressed in $x(t)$, $\alpha(t)$, and $u(t)$.

$$\text{minimize} \quad \frac{\partial J(x,\alpha)}{\partial x} u \tag{4}$$

subject to (2).

It is important to note that this is a feedback law. The problem is only specified when $x$ and $\alpha$ are determined. The numerical solution of (4) is implemented on-line at the middle level of the hierarchical algorithm.

Because of the qualitative properties of $J(\ )$ (differentiability, convexity) and $\Omega(\ )$ (a linear polyhedron), problem (4) has certain properties. For every $\alpha$, 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 $R_i$, the value of $u$ that satisfies (4) is the corresponding corner $P_i$.

Kimemia and Gershwin implemented (4) in a simulation by solving it every time step (one minute). This worked well while $x$ was in the interior of a region $R_i$. However, when $x$ crossed certain boundaries between regions, this approach worked poorly. After $x(t)$ crossed such a boundary (called attractive below), the value of $u$ corresponding to the new region $R_j$ was such that the derivative (1) pointed toward $R_i$. When $x(t)$ crossed the bounda-

ry back into $R_i$, the derivative pointed again to $R_j$. Thus, u(t) jumped between adjacent corners $P_i$ and $P_j$ of $\Omega(\alpha)$).

This behavior is called "chattering" and is generally undesirable. In this context, where u(t) represents the rate of flow of discrete items, allowing u(t) to chatter means that the flow rate would change more frequently than parts are loaded into the system. The flow rates are such that at least one of the machines is fully utilized or totally unutilized at all times (since u(t) is at an extreme point of $\Omega(\alpha(t))$). When u jumps frequently from one corner of $\Omega(\alpha)$ to another, the algorithm is trying to switch rapidly from keeping one set of machines fully loaded or unloaded to keeping another fully loaded or unloaded.

It cannot do this successfully: neither set of machines is fully loaded or unloaded. As a result, if the demands on the system are near its capacity, it will fail to meet these demands. This behavior was observed by Kimemia (1982). A method is described in Section 4 that circumvents the chattering problem. A value of u is calculated when x reaches a boundary which prevents this behavior.

An additional benefit is the reduction of on-line computation. Instead of solving a linear program at each time step, we now solve a LP only when a machine state change takes place. A small amount of additional calculation is also required.

Lower Level

Kimemia used a simple scheme for loading a mix of material into the system at a prescribed mix of rates. This method was, however, vulnerable to the accumulation of material. A new method is described in Section 5 which does not have this difficulty. Unexpectedly, it is even simpler than the earlier method, and uses less on-line computer resources.

## 3. SIMPLIFICATION OF TOP-LEVEL COMPUTATION

To circumvent the great computational requirements of the top-level dynamic programming problem, a crude approximation has been formulated. Preliminary numerical evidence (Section 6 and Akella, Choong, and Gershwin, 1983) seems to indicate that this approximation is highly satisfactory, at least for a set of important cases. In the following, we assume that d(t) is constant.

Kimemia (1982) and Kimemia and Gershwin (1983) suggest a decomposition by which the n'th order Bellman partial differential equation of the dynamic programming problem is replaced by n first order Bellman ordinary differential equations (where n is the number of part types, ie, the dimensionality of x, u, and d).

Kimemia further suggests approximating the solution to each one-dimensional dynamic programming problem with a quadratic cost function. Not only does this reduce data requirements, but it also simplifies the middle-level computation. As a result, the cost function is then written

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

where $A(\alpha)$ is a diagonal matrix, $b(\alpha)$ is a vector, and $c(\alpha)$ is a scalar whose value is not important. In this section, a simple method for choosing $A(\alpha)$ and $b(\alpha)$ is suggested.

As shown by Kimemia and Gershwin, the function $J(x(t),\alpha)$ is a decreasing function of t when $\alpha$ remains constant. The hedging point, given by

$$H(\alpha) = -A(\alpha)^{-1} b(\alpha) \qquad (6a)$$

is the minimum value of $J(x,\alpha)$ for $\alpha$ fixed. It is the value that x reaches if $\alpha$ stays constant for a long time and if d is feasible, ie if $d \in \Omega(\alpha)$. Since A is diagonal, this can be written

$$H_i(\alpha) = -b_i(\alpha) / A_{ii}(\alpha) \qquad (6b)$$

In the example described in Section 6, there are no backup machines. Therefore, d is feasible when and only when all machines are operational.

In order to calculate the hedging point, consider Figure 3 which demonstrates a typical trajectory of $x_i(t)$. Assume $x_i$ has reached $H_i(\alpha)$, the hedging point corresponding to the machine state before the failure. Then $u_i$ is chosen to be $d_i$ and $x_i$

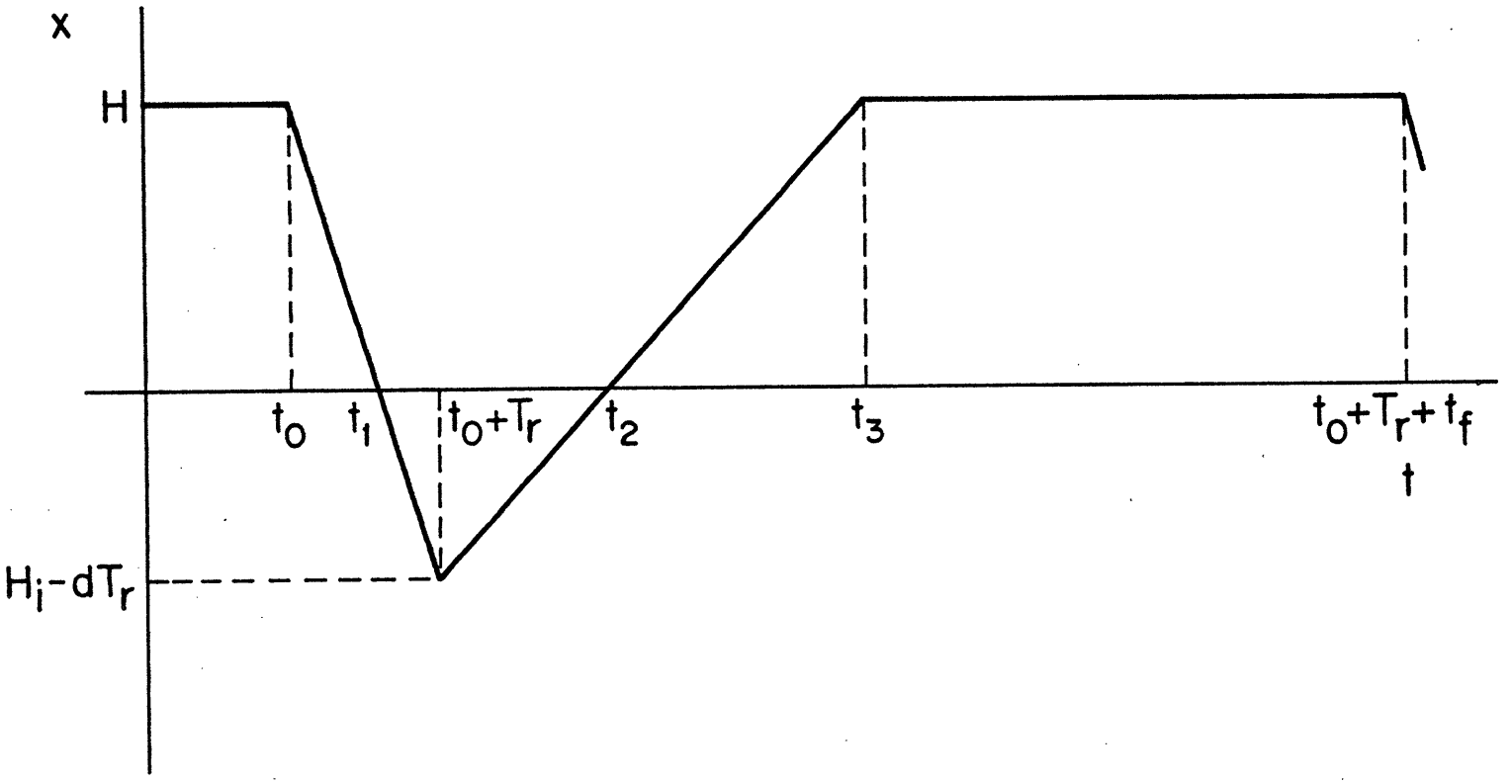


Figure 3. Simplified trajectory of $x_i$.

remains constant.

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

$$H_i - d_i T_r. \tag{7}$$

Just after the repair (at time $t_0 + T_r$), $u_i$ is assigned the value $U_i$. Assuming that this value is greater than demand $d_i$, $x_i$ increases at rate $U_i - d_i$ until it reaches the hedging point $H_i$ (at time $t_3$). At that time, $u_i$ resumes its old value of $d_i$ and $x_i$ stays constant until the next failure, at time $t_0 + T_r + T_f$.

To simplify the analysis, we make several assumptions:

1. $u_i$ is constant between the repair ($t_0 + T_r$) and when $x_i$ reaches $H_i$ ($t_3$). This is false, as indicated by the middle level discussion of Section 4.

2. $T_r$ and $T_f$ can be replaced by their expected values, the MTTR and MTBF. For notational convenience, we continue to use those symbols. A more precise analysis would treat the distributions of repair and failure times and would explicitly treat the effects of early failure and repair. That is, it would deal with the events that $T_r$ is so small that $x_i$ does not become negative, or that $T_f$ is so small that the hedging point is not reached.

3. The cost function g( ) in (3) penalizes positive areas in Figure 3 with weight a and negative areas with weight b, where a and b are positive scalars.

The positive area between $t_0$ and $t_0 + T_r + T_f$ is the area between $t_0$ and $t_1$ plus the area between $t_2$ and $t_0 + T_r + T_f$, where

$$t_1 = t_0 + H_i / d_i,$$

$$t_2 = t_0 + T_r - (H_i - d_i T_r) / (U_i - d_i)$$

and

$$t_3 = t_0 + T_r + d_i T_r / (U_i - d_i).$$

The positive area is

$$\tfrac{1}{2} H_i (t_1 - t_0) + \tfrac{1}{2} H_i (t_3 - t_2) + H_i (t_0 + T_r + T_f - t_3).$$

$$= \frac{1}{2} \frac{H_i^2 U_i}{d_i (U_i - d_i)} + H_i \left[ T_f - \frac{d_i T_r}{U_i - d_i} \right]$$

The first term is clearly positive, and the second term is positive when the demand is feasible because

$$T_f U_i > (T_f + T_r) d_i.$$

The right side of this inequality is the amount of material of type i that is required during an average failure and repair cycle. The left side is the average amount produced during such a cycle. This is because production can only take place during the working portion, whose average length is $T_f$.

The absolute value of the negative area is

$$-1/2 (H_i - d_i T_r) (t_2 - t_1)$$

$$= \frac{1}{2} \frac{(H_i - d_i T_r)^2 U_i}{d_i (U_i - d_i)}$$

which is always positive.

The cost function, according to assumption 3, is then

$$a \frac{1}{2}\left[\frac{H_{i2}\, U_i}{d_i\,(U_i - d_i)} + H_i\left[T_f - \frac{d_i\, T_r}{U_i - d_i}\right]\right]$$

$$+ \frac{b\,(H_i - d_i T_r)^2\, U_i}{2\, d_i\,(U_i - d_i)}.$$

This quantity is the cost incurred per average repair and failure cycle of a machine. To find $H_i$, we must minimize it. This is not difficult because the cost is quadratic in H.

The minimizing $H_i$ is

$$\frac{T_r\, d_i\,(b\, U_i - a\, d_i) - T_f\, a\, d_i\,(U_i - d_i)}{(a + b)\, U_i}.$$

This quantity can be shown to be positive.

To further simplify the analysis for the system of Section 6, we assumed that a, b, $T_r$ and $T_f$ and $U_i$ were such that

$$H_i = d_i\, T_r / 2 \qquad (8)$$

where $T_r$ was the average mean time to repair among all the machines that part i visits.

For machine states in which the demand is not feasible, this approach does not apply. In Section 6, we choose the hedging point for such states larger than (8).

$A_{ii}(\alpha)$ must be positive in order for J to be convex. Its value reflects the relative priority of part type i. Parts that have great value, or that would cause great difficulty if backlogged, or that pass through relatively unreliable machines should have larger values of $A_{ii}$. In Section 6, we describe experiments with

$A_{ii}$ = number of machines part type i visits

and other values.

This calculation does not consider the fact that the failure may occur before the state reaches the hedging point or that the repair may occur before the state becomes negative. It assumes a specific form of g. These considerations and others should be the subject of future research, but it is important to observe

that this method produced very satisfactory results, as described in Section 6.

It is also important to observe that the results seem to be insensitive to the values of $A_{ii}$ and $b_i$. This is important because it provides evidence that it is not necessary to obtain an exact solution to the dynamic programming problem (3). Consequently, the off-line part of the algorithm does not appear to require a large computer.

# 4. IMPROVEMENTS TO MIDDLE LEVEL

## 4.1 Characteristics of Solution

### Cone-Shaped Regions

Kimemia shows that the optimal $J(x,\alpha)$ is convex in x for each $\alpha$. He also shows that J decreases when u satisfies (4) and d is feasible. The minimal value of J is achieved when x is at the hedging point. When J is given by (5), its minimum is reached at (6).

Claim: When J is quadratic, the regions of x-space (in which the solution u of (4) is constant at a corner of $\Omega(\alpha)$) are cones. That is, if u is the production rate corresponding to x, then u is also the production rate corresponding to x', where

$$x' - H = s\,(x - H) \tag{9}$$

where s is a positive scalar.

Proof: ($\alpha$ is not written.) The transpose of the coefficient of u in (4), corresponding to x, is $A\,x + b$. Corresponding to x', the transpose of the coefficient of u in (4) is $A\,x' + b$, which can be written

$$A(H + s\,(x - H)) + b$$

or

$$s(Ax + b) + (1-s)(AH + b).$$

Since H satisfies (6), the second term vanishes. The factor s does not change the optimum, and the claim is proven. Note that this is true regardless whether d is feasible.

### Planar Boundaries

Linear program (4) can now be written

$$\begin{aligned} &\text{minimize} && c(x)^T\,u \\ &\text{subject to} && D\,u = e \\ & && u \geqslant 0 \end{aligned} \tag{10}$$

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

$$c(x) = A\,x + b.$$

(Note that arguments $\alpha$ and t are suppressed.) The standard solution of (10) (Luenberger, 1977) breaks u into basic ($u_B$) and non-basic ($u_N$) 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

$$\text{minimize } c_R(x)^T u_N \qquad (11)$$

$$\text{subject to } \quad u_N \geq 0$$

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 called the reduced cost. If all components of $c_R$ are positive, then 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). The simplex method is the widely used technique for changing the basic and non-basic parts of (10).

It is important to note that since c is a function of x, the basic/non-basic breakup of this problem depends on x. That is, the set of components of u (and therefore of c(x) and the set of columns of D) that are treated as basic varies as a function of x.

As stated above, regions correspond to the corners of the constraint set. At every x in region $R_i$, corner $P_i$ is the optimal value of u for (10). In each region, then, there must be a basic/non-basic break-up 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 therefore portions of hyperplanes. This is 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 of Trajectories

Since u is constant throughout a region, dx/dt is also constant. The buffer state x travels along a straight line in the interior of each region. As indicated in Figure 4, such lines may intersect with one or more boundaries of the region. When x reaches a boundary, u and therefore dx/dt changes.

Some boundaries are such that when they are reached, the trajectory continues, after changing direction, into the adjacent region. Others, that we call attractive boundaries, are different. The trajectories on both sides of such boundaries point toward them. Consequently, the trajectories tend to move along the boundaries.

We can now qualitatively describe the trajectory. After a machine state change, x(t) is almost always in the interior of a region. It moves in the characteristic direction of that region (which corresponds to a corner of the $\Omega(\alpha(t))$ polyhedron) until it reaches a boundary. If the boundary is not attractive, x(t) moves in 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.

The trajectory continues until it hits the next attractive boundary. After that, x(t) moves along the intersection of three regions. The production rate vector is on the surface determined by the three corners corresponding to these regions.

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 (that is, if $d \in \Omega(\alpha)$) then 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, for a constant machine state, the future behavior of x(t) would be determined from its current value. We call this the "conditional future trajectory" or the "projected trajectory".

## 4.2 Calculation of the Conditional Future Trajectory

Assume that the conditional future trajectory is to be calculated at time $t_0$. This may be due to a machine state change

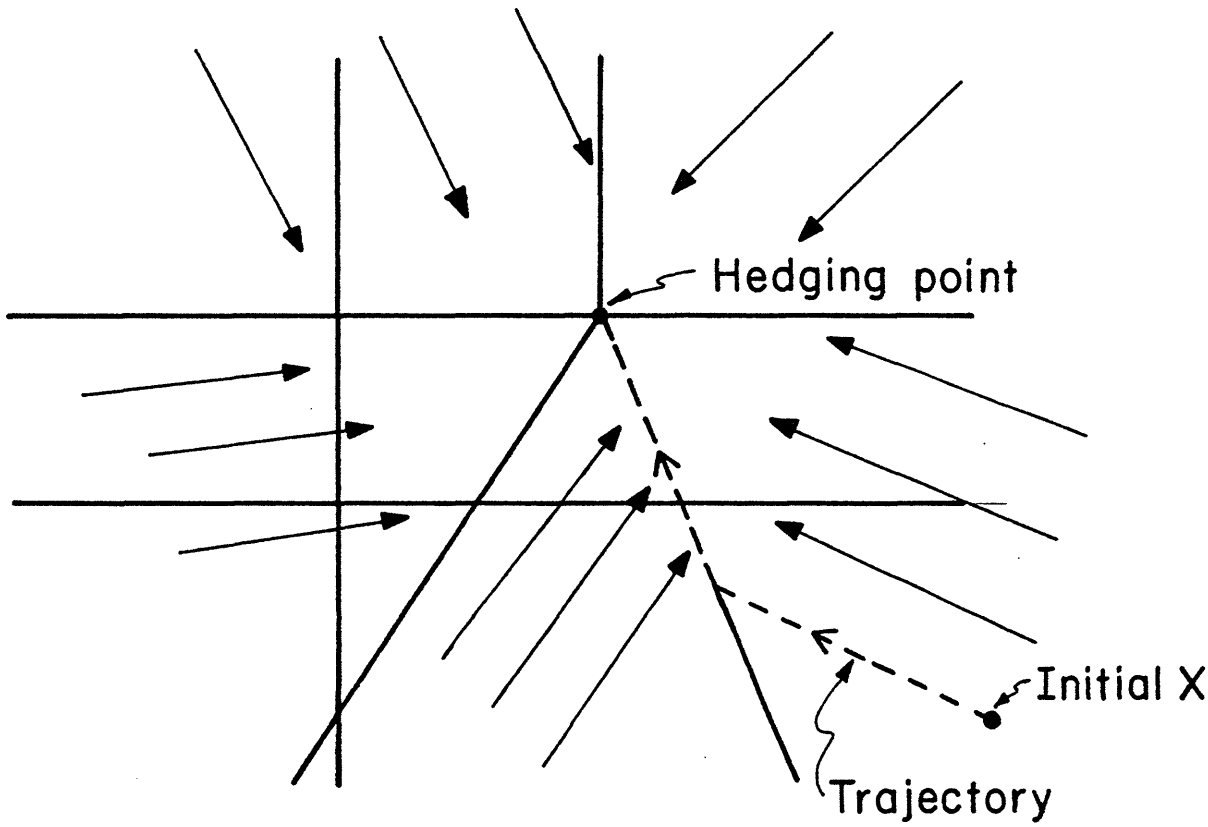


Figure 4. Regions of x-space induced by production policy.

which takes place at that time. It may also be part of a conservative strategy which requires recalculation periodically to ascertain that the system is not drifting from where it is expected to be.

As soon as the machine state change occurs, linear program (10) is solved. Thus the basic/non-basic split is determined and the $c_R(x)$ function is known.

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

$$h(x(t)) = f^T ( x(t) - x(t_1) ).$$

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. (It should be remembered that this is an on-line computation that is taking place at time $t_0$. The future trajectory is 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 $h(x(t_1)) = 0$,

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

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:

$$\begin{aligned} &\text{minimize} \quad c(x)^T \; u \\ &\text{subject to } D\,u = e \\ &\qquad\qquad u \geq 0 \\ &\qquad f^T u = f^T d \end{aligned} \tag{13}$$

By adding equation (12) to (13), we are requiring that the solution keeps x(t) on the boundary. We are also replacing the reduced cost which has become zero with a new equation, so that the new problem has a unique solution.

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, the surfaces that u is found on in $\Omega(\alpha)$) increase in dimension. That is, u is first on a corner. When the fist attractive boundary is encountered, u is on the edge formed by the convex combination of the corners corresponding to the regions adjacent to the boundary. When the next attractive boundary is reached, u is a convex combination of three corners, and so forth.

At the same time, x is found in regions of decreasing dimension. After a machine state change, $x(t_0)$ is in the interior of a region of full dimensionality. The first attractive boundary x(t) reaches is a hyperplane separating regions of full dimensionality, so its dimensionality is one less than full. The next boundary is the intersection of two such boundaries and thus has dimensionality one smaller.

Since this is a finite dimensional system, this process must terminate. There are two cases. If the demand is feasible, ie if d is a feasible solution of (10), then d is a feasible solution of (13). This is because d satisfies (12) for all f. As new constaints of the form (12) are added to (13), d remains feasible. Finally, if enough linearly independent constraints are added, there is only a single feasible solution to (13) and that is u=d.

Since the dynamics of x are given by (1), x remains constant when u=d. The value of this constant is the hedging point, discussed above, which is the minimum of $J(x, \alpha)$ for the current value of $\alpha$.

If the demand is not feasible, u cannot be equal to d and thus x cannot become constant. Instead, the process described above terminates with u satisfying linear program (13) including one or more constraints of the form (12). The vector x(t) is eventually of the form

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

Since d is not feasible, some or all of the components of u - d are negative. The corresponding components of x decrease without limit.

## 4.3 COMPUTATIONAL CONSIDERATIONS

The conditional future trajectory is calculated whenever the machine state changes, either due to a failure or a repair. It may also be calculated under other conditions: periodically, to ensure that the actual trajectory is close to the projected trajectory; or after unanticipated events such as parts not being loaded into the system in the prescribed manner.

To begin the computation, a linear programming problem (10) must be solved. The number of variables (production rates and slack variables) is the number of part types plus the number of distinct machines. The computational effort is not large when the number of variables is on the order of 20; it increases as a polynomial function and is substantial when there are 100 variables.

As each boundary is reached, one (if unattractive) or two (if attractive) additional programs are solved. The numerical effort is very small, however, since each starting basic feasible solution is the solution of the previous problem. We expect that no more than a few pivots of the simplex method will be required to find each new solution.

## 5. LOWER LEVEL

### Previous Part Loading Scheme

Kimemia (1982) suggested a simple part loading scheme for the lower level: for each part type, establish a set of time windows. The length of each window is the inverse of the production rate (u). The loading rule is: load one and only one part of each type during each one of its windows.

This rule is uncomplicated, easy to program, and consistent with the hierarchical approach. As implemented by Kimemia, however, it suffered from an accumulation of in-process inventory when machines failed and were repaired.

This is because the windows are reinitialized whenever a machine state changes. Consider a part that goes to two successive machines with no alternative routes. When the second machine fails, one part of that type is likely to be in the system. No additional parts are allowed in. When the repair occurs, a new part is loaded immediately. There are now two parts of that type in the system. There is no mechanism to reduce that number except a failure of the first machine.

If there is another failure/repair cycle of the second machine before the first, the number of parts goes up by one more. If there are several part types that visit both machines in the same sequence, the number of parts in the system goes up by that number for each failure/repair cycle of the second machine that occurs while the first machine is operational.

This problem could have been treated by refraining, under some circumstances, from loading parts. For example, after a repair, the window might not be started until the old part is completed at the machine that was repaired. However, this adds undesirable complication to the logic.

### New Part Loading Scheme

In the current research, an alternative has been used instead which is simple, easy to program, consistent with the hierarchical approach, and effective. It is based on the conditional future trajectory (x(s), s≥t) rather than the current value of the production rate (u(t)).

Define the actual surplus of part type i at time t to be

$$x^A_i(t) = [\text{number of parts of type } i \text{ loaded during } [0,t]] - d_i\, t.$$

Note that $x^A_i(t)$ is an irregular sawtooth function of time. It jumps by 1 each time a part is loaded. At other times, it

decreases at rate $d_i$.

The loading strategy insures that $x^A(t)$ is near $x(t)$. The strategy is: at each time step t, load a part of type i if

$$x^A_i(t) < x_i(t). \qquad (14)$$

Do not load a part of type i otherwise.

A rule is required to resolve conflicts; it probably does not matter what that rule is since conflicts will not arise very often. One reasonable rule is: of all the parts that are candidates for loading, load the one with the smallest i. Other such rules are: load the part with the greatest short-term or long term production rates.

## Characteristics of New Strategy

Strategy (14) is easy to implement. It does not suffer from the problem described above. Just after the failure of the second machine, the projected trajectory is recalculated. Because the production rate ofparts of type i is zero,

$$x_i(s) = x_i(t_0) - d_i\ (s-t_0).$$

If, at time $t_0$, $K_i(t_0)$ parts of type i were produced, it is fair to assume

$$x^A_i(t_0) = K_i(t_0) - d_i t_0 > x_i(t_0)$$

since otherwise a part would have been loaded exactly as the machine failed. This implies that

$$x^A_i(s) > x_i(s)$$

until the repair since if no additional parts are loaded, the actual surplus is given by

$$x^A_i(s) = K_i(t_0) - d_i\ s.$$

Therefore no additional parts are loaded and, more important,

$$x^A_i(s) - x_i(s)$$

is constant during that period. Consequently, when the repair takes place, there is no reason to load any part immediately; the clock that regulates the loading process is suspended during the machine downtime in a very simple way.

## Behavior of the New Strategy

Figures 5 and 6 demonstrate the behaviors of projected and actual trajectories. They are extracted from the simulation described in Section 6 and in Akella, Choong, and Gershwin (1984) in greater detail. Figure 5 shows a portion of a projected trajectory and of an actual trajectory that was determined by this method. The projected and actual surpluses of part type 1 are shown as functions of time.

The actual trajectory remains as close as possible to the projected trajectory. Recall that there are five other such trajectories that are being treated at the same time. No difficulty is experienced at the time (about 9740) when the loading rate changes.

In Figure 6, the projected and actual surpluses of types 1 and 3 are shown evolving together. All machines were operational for a period that ended at t=9499 (seconds into the shift) when Machine 4 failed. The hedging point had been reached with both $H_1$ and $H_3$ equal to 15. (There were a total of six part types.) Since neither part types 1 or 3 required Machine 4, their production was not curtailed by the failure. In fact, the algorithm took the failure as an opportunity to increase its surplus of those types. Both $H_1$ and $H_3$ were increased to 19. This point was reached at t=10511 and the system remained there until t=13733 at which time the repair took place. The old hedging point was reached again at t=18036.

Only part of the actual trajectory is shown to keep the figure uncluttered. Note how it moves around the projected trajectory until it reaches the new hedging point, where it hovers.

## Conclusion

A new version of the lower level of the algorithm has been described. This version is simple, easy to use, and effective, and it does not suffer from problems of an earlier version.

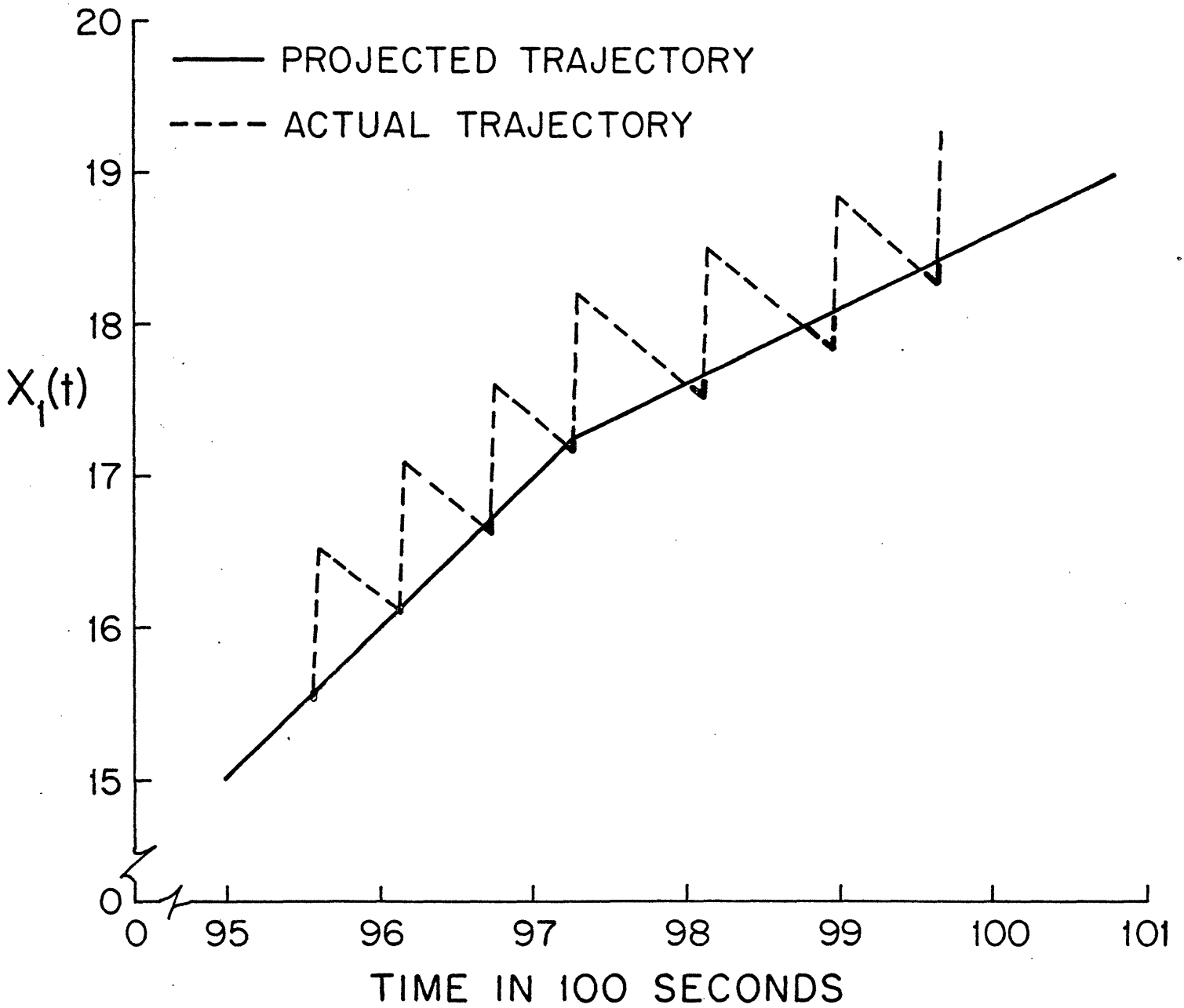


Figure 5. Actual and Projected Trajectories--$x_1$ vs t.

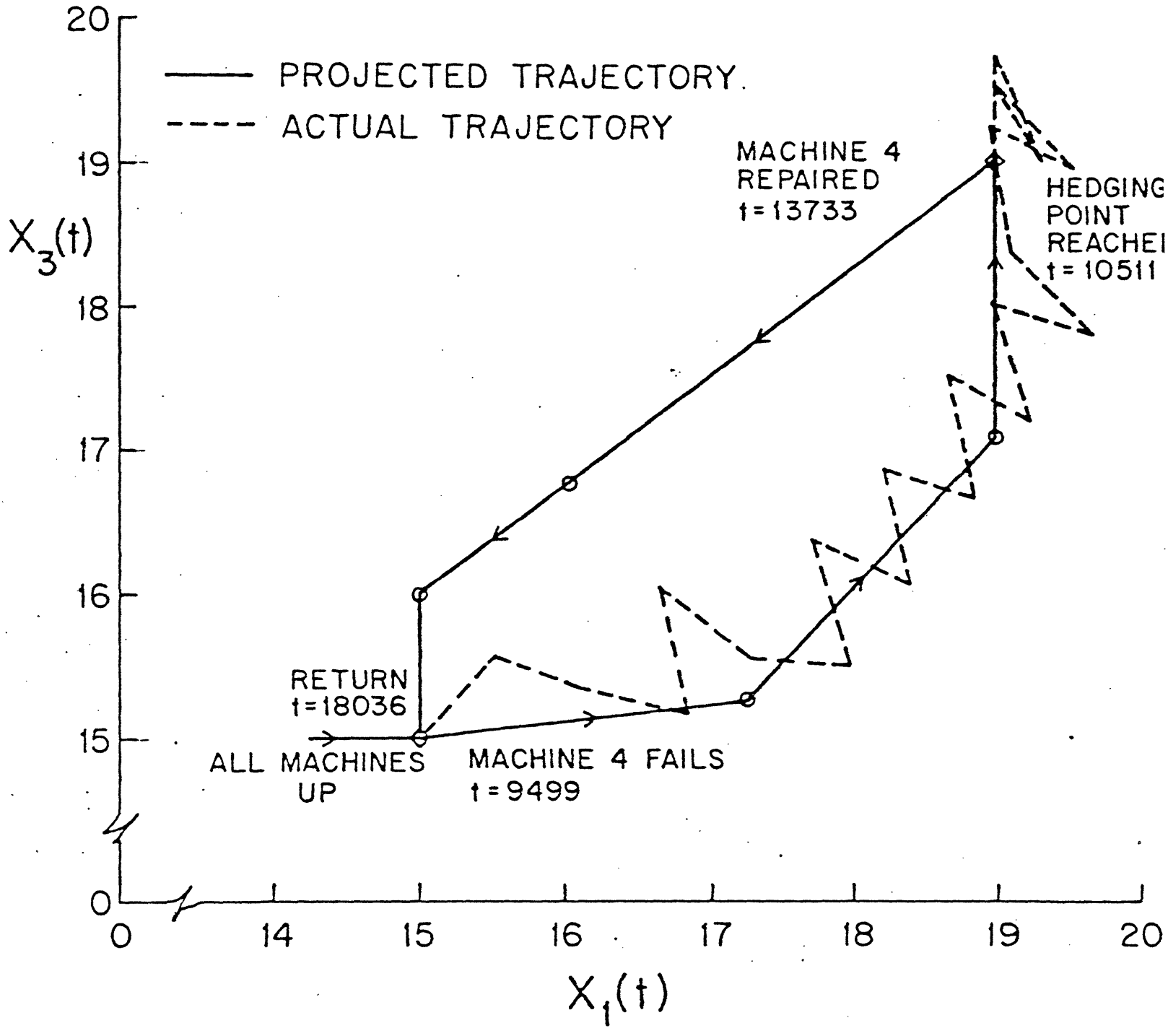


Figure 6. Actual and Projected Trajectories--$x_1$ vs $x_3$.

## 6. SIMULATION RESULTS

A detailed simulation of an IBM flexible manufacturing system was written to test the hierarchical scheduling policy and to compare it with other reasonable ploicies. The simulation is described in Akella, Bevans, and Choong (1984). A full description of the results appears in Akella, Choong, and Gershwin (1984). A summary of the results is presented here.

### System

Figure 7 is a schematic diagram of a simple version of an IBM "Miniline" which inserts electronic components into printed circuit cards. The triangles represent machines; all the machines are different and they each insert one class of components. The ovals are storage buffers. The circles and rectangles are transportation elements. The circles rotate while parts move only in the directions indicated on the rectangles.

Six part types are being made simultaneously. They require operations on one two, or three machines. The machines are able to do different operations on successive parts without time lost for set-up. Failures and repairs on machines take place at random times and the machines all have the same reliability behavior. Each MTBF is 10 hours and each MTTR is 1 hour, so that the efficiencies are all 91%. Demands are chosen so that the machines are utilized 98%, 91%, 96%, and 97% of the expected available time.

### Common Sense Policies

A variety of alternative policies was formulated to compare with the hierarchical policy. These policies shared the same structure but were increasingly sophisticated.

1. Common sense. If more than N parts are in the system, do not load a part. If N or fewer parts are in the system, load the part that is furthest behind or least ahead of demand. Do not allow any part type to get more than K parts ahead of demand.

2. Less sophisticated improvement. This is the same as the previous, except that there are six thresholds, one for each part type. If there are $N_i$ or fewer type i parts in the system, and type i is less than K parts ahead of demand, load a type i part.

3. More sophisticated improvement. This is the same as the previous, except that threshold i is set to zero whenever any machine is down that part i must visit.

These policies have parameters that must be chosen. Little's law (Little, 1961) gives some guidance, but simulation experience indicates that behavior can depend critically on their values.

## Simulation Results

Figure 8 displays the results of four runs of the hierarchical policy and four runs of the common sense policy. All runs were performed with the same seed for the random number generator. That is, each had the same sequence of repairs and failures.

The horizontal axis displays the average number of parts in the system. The vertical axis shows the percentage of the total requirements that was actually produced.

The four hierarchical runs had different values of A and b. The common sense runs had different values of the threshold N. Figure 8 indicates that the hierarchical strategy produced superior results. The in-process inventory was lower and the production percentage was greater: in fact, over 98%. In addition, although the values of the A and b parameters differed considerably, the four hierarchical points are clustered quite close together. This indicates that the policy is not sensitive to these parameters.

Figure 9 demonstrates how the policies satisfy balance requirements. The horizontal axis is the same as the vertical axis of Figure 8. To define balance, let

$$Z_i = \frac{\text{number of type i parts produced during the run}}{\text{number of type i parts required during the run}} .$$

Then balance is defined as

$$\frac{\min Z_i}{\max Z_i} .$$

It is important that balance be near 100% to ensure that only what is required is produced. In particular, if the cards will eventually be assembled into the same pieces of equipment, any deviation from 100% can result in incomplete production. Figure 9 again indicates the superiority of the hierarchical policy.

Akella, Choong, and Gershwin (1984) present a fuller comparison of these and other policies. Different seeds were used in the random number generator, and the same conclusions hold. The more sophisticated policies performed better than the common sense policy, but not as well as the hierarchical policy.

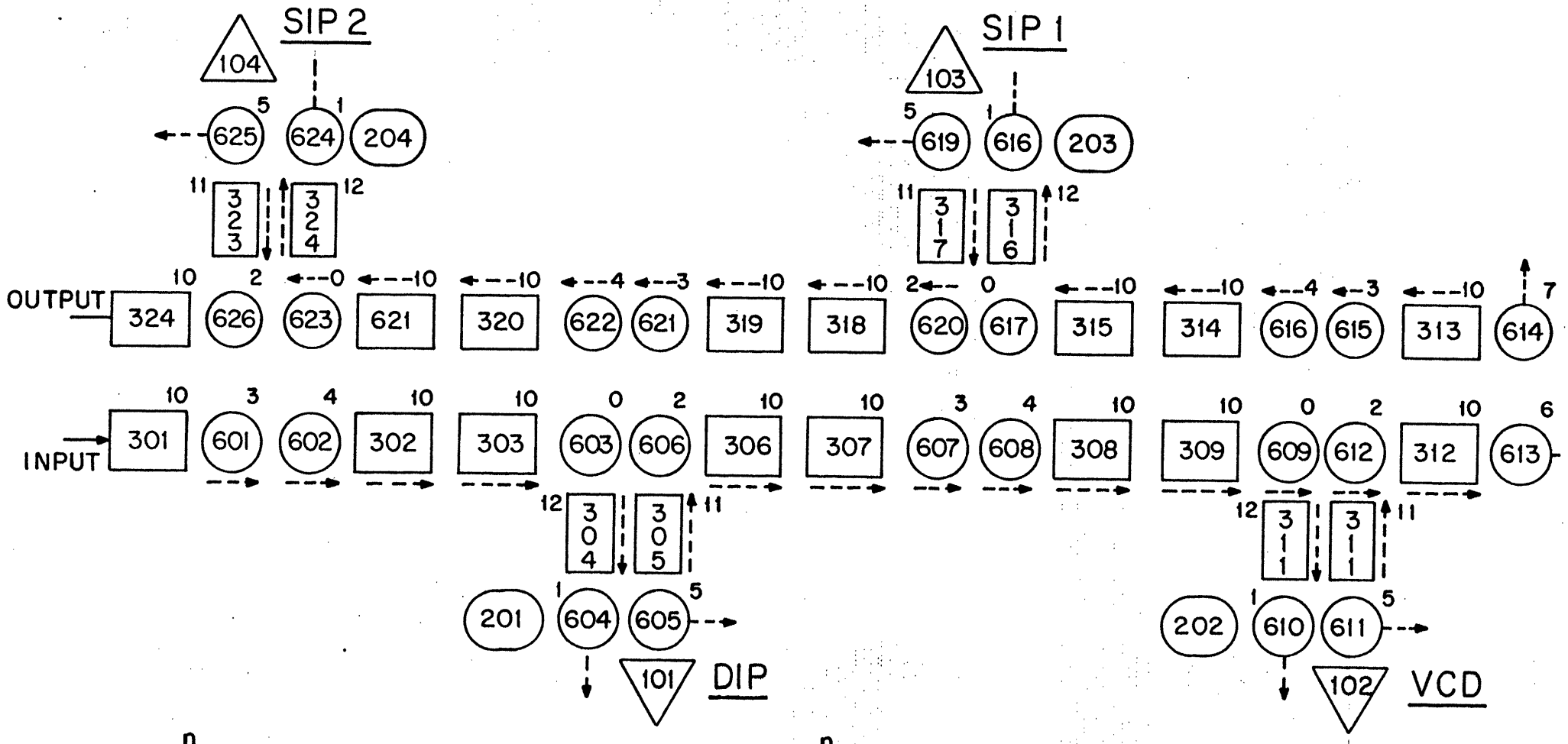


Figure 7. Schematic Layout of IBM Miniline.

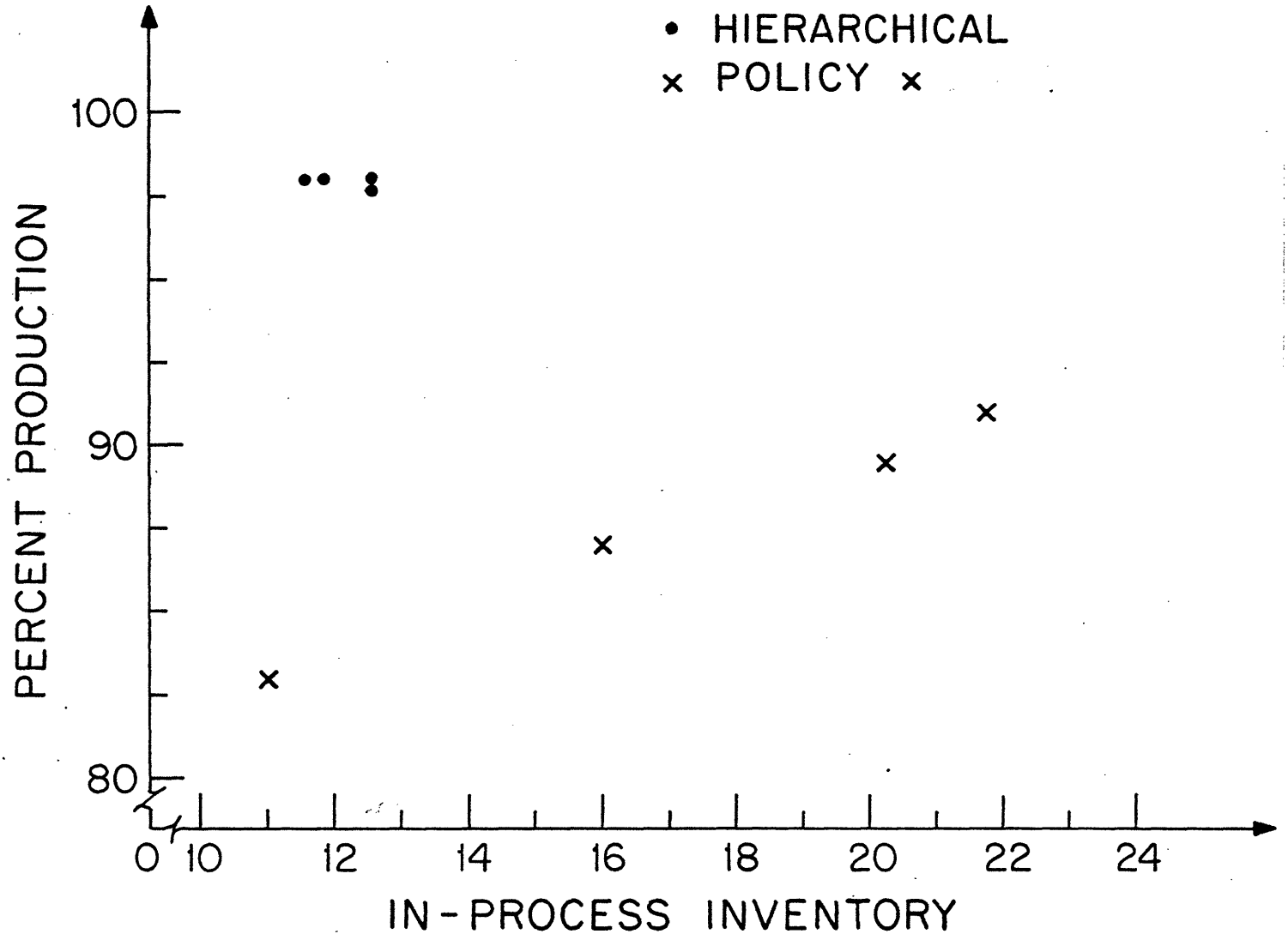


Figure 8. Simulation Results--Total Production Percentage vs. In-Process Inventory.

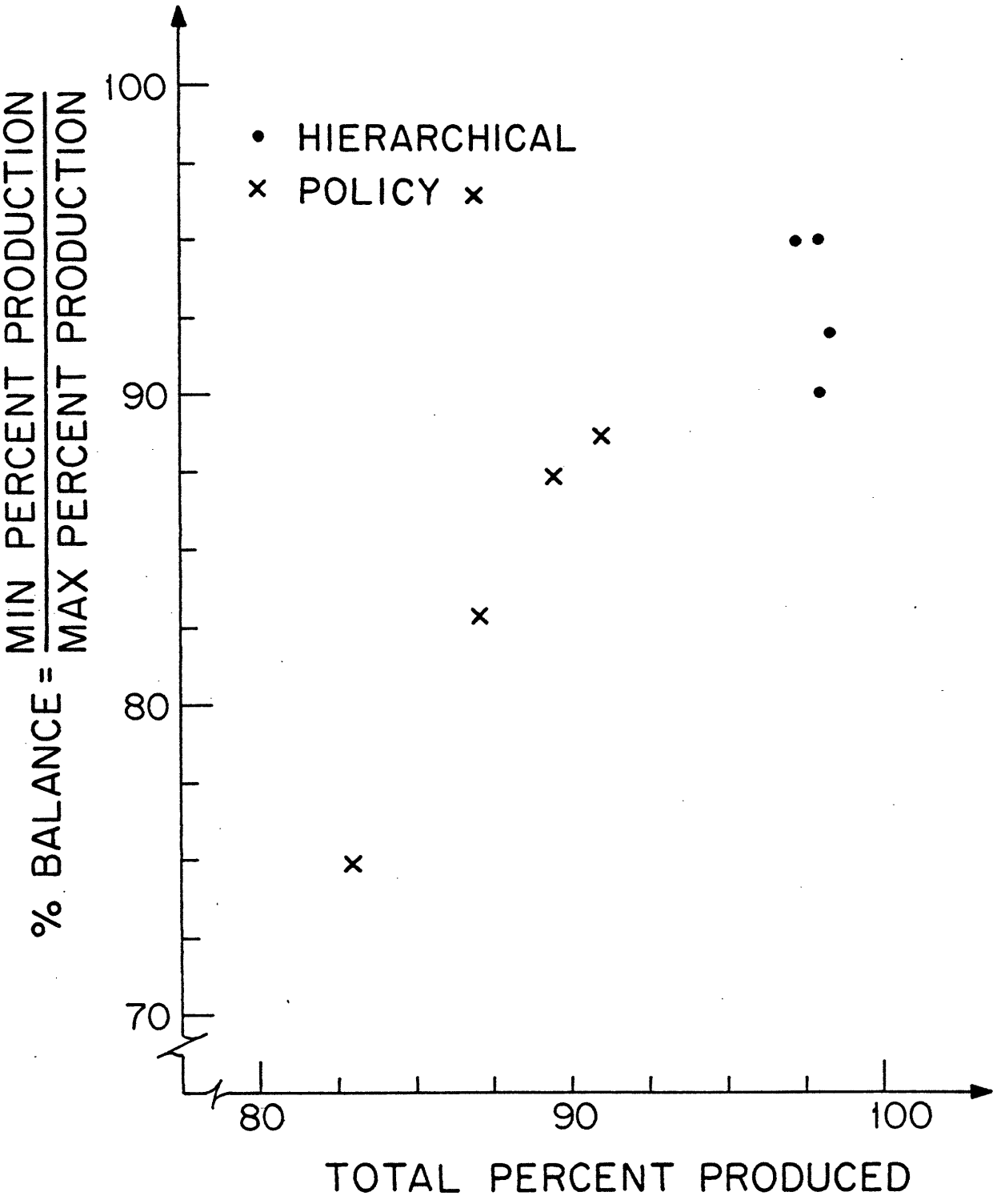


Figure 9. Simulation Results--Balance vs. Total Production Percentage

## 7. SUMMARY AND CONCLUSIONS

### Summary

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:

1. An alternative to the computationally intense off-line calculation of the cost-to-go function has been suggested. It is simple enough as to be essentially instantaneous even on a microprocessor. Preliminary simulation tests indicate that it is accurate enough for the top level of the scheduling hierarchy.

2. The middle level has been made more computationally efficient and more accurate. It is no longer necessary to solve a linear programming problem at each time step. Instead, a handful of closely related linear programming problems are solved whenever there is a change of machine state. Systems with a small number of part types can thus be managed with a small computer. In addition, because of the way the new method plans a trajectory in advance, chattering is avoided. It is thus possible to more accurately calculate desired part flows and to achieve those flows.

3. A new part loading policy has been developed for the lower level. It is based on the trajectory planning method of the middle level. It is simple. It is accurate in that it keeps the actual trajectory close to the projected trajectory. That is, the middle level devises a plan for production until the next machine state change. The lower level loads parts in close agreement with that plan.

### Conclusions

While further simulation tests and more research and development is needed, certain conclusions can be drawn.

1. The method works better than the alternatives considered in Section 6. It gets nearly 100% of the theoretically possible production out of a system and maintains good balance among the parts required.

2. The reason that the method works well computationally is the hierarchical structure. The large, difficult problem is decomposed into a set of small, easy problems.

3. The hierarchical structure is also the reason for the scheduling accuracy of the method. The most important consideration is the capacity constraint (2). This guarantees that material is not loaded into the system at a rate greater than it can handle.

Consequently, large queues do not develop in the buffers or in the transport system. This avoids excessive in-process inventory. It also reduces congestion which can diminish the effective productive capacity of the system.

## Further Research and Development

Research in several areas is suggested by this work:

1. Additional simulation tests are required. A greater variety of cases should be tested to confirm the conclusions stated here.

2. The method proposed for the approximate calculation of the cost-to-go function should be formalized and generalized so that it can be applied to a wider class of systems. The apparent insensitivity of the production process to this function should be better understood.

3. Other kinds of disruptive phenomena should be studied and incorporated in this method. The most important is random demand: the arrival of orders that are not specified in the long range requirements d(t).

4. We have assumed that production requirements have been specified at a higher level of the hierarchy. (See Figure 1.) The calculation of these requirements should be done in a way that takes changeover times between part families into account since it is important to limit time lost due to set-ups. The scheduling of set-ups and of batch sizes is an important research area.

5. The coordination of adjacent cells in a multi-stage factory requires study. For example, how should the scheduling of a cell be affected by the failure of a machine in a neighboring cell?

While further research will enhance this work, its current form can be useful for manufacturers. Development is required in the following areas.

1. Events which are potentially disruptive, but which can be planned in advance must be treated. They include maintenance, training sessions, and so forth. Hedging points should be adjusted in advance of the anticipated events.

2. Software must be written for the implementation of this method. This software must be of commercial quality and it must treat many issues not covered here or in the earlier research papers, including the gathering of data.

REFERENCES

R. Akella, J. P. Bevans, and Y. Choong (1984), "Simulation of a Flexible Electronic Assembly System," Massachusetts Institute of Technology Laboratory for Information and Decision Systems Report to appear.

R. Akella, Y. Choong, and S. B. Gershwin (1984), "Performance of Hierarchical Production Scheduling Policy,"Massachusetts Institute of Technology Laboratory for Information and Decision Systems Report LIDS-FR-1357.

J. G. Kimemia (1982), "Hierarchical Control of Production in Flexible Manufacturing Systems, Massachusetts Institute of Technology Laboratory for Information and Decision Systems Report LIDS-TH-1215.

J. G. Kimemia and S. B. Gershwin (1983), "An Algorithm for the Computer Control of Production in Flexible Manufacturing Systems, IEE Transactions, Volume 15, No. 4, December, 1983, pp. 353-362.

D. Luenberger (1977) Introduction to Linear and Nonlinear Programming, Addison-Wesley.

J. D. C. Little (1961) "A Proof for the Queuing Formula: $L = \lambda W$," Operations Research, Volume 9, Number 3, pp. 383-387.

Captions

Figure 1. Long Term Production Hierarchy.

Figure 2. Short Term Production Hierarchy.

Figure 3. Simplified trajectory of $x_1$.

Figure 4. Regions of x-space induced by production policy.

Figure 5. Actual and Projected Trajectories--$x_1$ vs t.

Figure 6. Actual and Projected Trajectories--$x_1$ vs $x_3$.

Figure 7. Schematic Layout of IBM Miniline.

Figure 8. Simulation Results--Total Production Percentage vs. In-Process Inventory.

Figure 9. Simulation Results--Balance vs. Total Production Percentage