Scheduling algorithms for flexible flow lines

by Robert J. Wittrock

This paper discusses scheduling algorithms for a certain kind of manufacturing environment, called the "flexible flow line." Two scheduling problems are considered. "Loading" decides when each part should be loaded into the system. "Mix allocation" selects the daily part mix. The goals are to maximize throughput and reduce WIP. New heuristic algorithms specially suited to solve these problems in the context of a flexible flow line are described. The paper also discusses experience with the use of an experimental implementation of these algorithms to solve such problems arising in a real production line.

1. Introduction

Automated scheduling algorithms can be used to improve productivity in various manufacturing environments. This paper considers one such environment, called the "flexible flow line." Two scheduling problems are considered. "Loading" decides when each part should be loaded into the system. "Mix allocation" selects the daily part mix. The goals are to maximize throughput and reduce work-in-process inventory (WIP). New heuristic algorithms specially suited to solve these problems in the context of a flexible flow line are described. The paper also discusses experience with the use of an experimental implementation of these algorithms to solve such problems arising in a real production line.

A flexible flow line can be defined as follows. Several part types must be produced each day. There are several banks of identical machines. Each part must be processed by at most one machine in each bank. (It may skip a bank.) Each part visits the machine banks in the same order. The time it takes a machine to process a part depends on the part type and the machine bank, but not on the sequence in which the parts are processed. There is no setup time for a machine switching from one part type to another. In front of each individual machine, there is a buffer which has a large capacity and operates on a first-in, first-out basis. There are machines to load and unload parts into and out of the system. Finally, there is an automated transport system to move parts from any machine to any other.

As an example, **Figure 1** schematically portrays the machine configuration of an actual production line. The function of this line is to insert components into printed circuit cards. The line has both numerically controlled tools and robots. There are three banks of machines: two so-called "DIP" inserters, three "SIP" inserters, and three robots, which do "MODULE" insertions. (The terms "DIP" and "SIP" refer to types of components called "Dual Inline Packages" and "Single Inline Packages," respectively.) A card is transported through the system on a "carrier," which holds four cards of the same type. After being loaded into the system, a carrier visits one of the two DIP machines, or skips them both. It then visits one of the three SIP machines (or skips them). Finally, it visits one of the three robots (or skips them) and is then unloaded from the system. Before visiting a machine, the carrier may wait in the machine's buffer if it is busy. Each buffer can hold up to 25 carriers.

To date, this scheduling environment does not appear to have been given much attention in the literature, although certain special cases of it have. For example, if there is only one machine in each bank, and each part visits every bank, the flexible flow line becomes the "flow shop" environment.

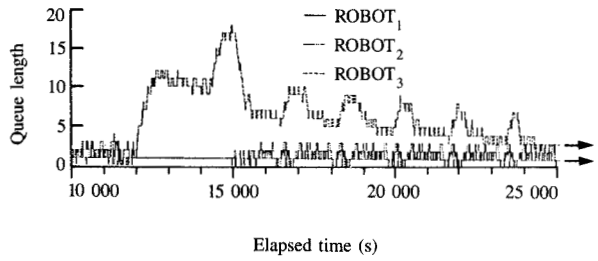

Figure 2

Reaction to machine failures (static routing).

If there is one machine in each bank, but some parts skip some banks, it is a special case of the "job shop" environment. In this case, all parts must visit the machines in the same order, whereas in a general job shop each part is allowed to visit the machines in a different order. The only well-studied case in which the banks contain more than one machine is the "identical parallel machine" environment. This is the case of one machine bank. Various optimizing and heuristic algorithms have been developed for all these cases. For a survey, see [1].

2. Loading

The first problem to be discussed is that of scheduling exactly when to load each part. A new algorithm was developed for this kind of loading, called FFLL (Flexible Flow Line Loading).

There are two goals in the loading problem. The main goal is to maximize throughput, i.e., minimize the makespan, or total completion time, of the whole day's mix. A secondary goal is to minimize the amount of buffer space used. One reason to do this is to minimize WIP. The other reason is to minimize the possibility of overflowing a buffer. FFLL effectively assumes that buffer capacity is infinite, so by minimizing the amount of buffer space actually used, the possibility of violating any real buffer capacity constraint is minimized. (To consider buffer capacity as an explicit constraint would be a much harder scheduling problem.)

It is useful to divide the loading problem into three subproblems. The first subproblem is called "machine allocation." Recall that a part needs to be processed by only one machine in each machine bank. Machine allocation is the problem of choosing which machine in each bank will process each part. The second subproblem, "sequencing," is to determine the order in which the parts enter the system. The third problem, "timing," is to determine the exact times at which the parts should be loaded into the system. FFLL handles these problems in the order given above: machine allocation, sequencing, and then timing. However, for ease of exposition, timing is discussed first, followed by machine allocation and then sequencing.

The various subproblems of the loading problem are complex optimization problems. There do not appear to be any reasonably fast algorithms for finding optimal solutions to such problems. For this reason, FFLL uses fast heuristic algorithms to find good (but not usually optimal) solutions to the subproblems.

Suppose a machine allocation has been determined. FFLL uses "periodic scheduling" to do the sequencing and timing. The basic idea of periodic scheduling is to schedule a small representative sample of the day's mix and repeat this schedule at regular intervals until the whole day's mix has been produced. Hitz [2] has developed a periodic scheduling algorithm based on implicit enumeration. The periodic scheduling approach of FFLL is based on heuristics.

The "representative sample" is called the "minimal part set" (MPS). The MPS is the smallest possible set of parts in the same proportion as the whole day's mix. For example, if the mix is 3000 parts of type A, 2000 of B, and 1000 of C, the MPS is three parts of type A, two of B, and one of C. By using periodic scheduling, the problem of sequencing is reduced to finding a good order in which to load each part in the MPS. The problem of timing is reduced to determining the times at which to load each part in the first MPS and determining the time interval after which the schedule is to be periodically repeated. This time interval is called the "period" of the schedule (denoted by P). The number of times the minimal part set must be repeated in order to complete the whole day's mix is called the "frequency" (denoted by F). In the example, $F = 1000$.

The periodic schedule is determined in such a way that each machine processes all of its assigned parts for one MPS before processing any parts of the next MPS. Once the sequence and loading times for the minimal part set have been determined, the best period for the schedule is easy to compute. Define the "MPS work span" of a machine to be the elapsed time between the time it starts processing its first part in the MPS and the time it completes work on its last part in the MPS. The best period for such a schedule is simply the maximum MPS work span among all the machines. A longer period than this would result in unnecessary idle time on all the machines, while a shorter period would result in unnecessary queuing in the buffers.

With the period defined as above, the makespan for the whole day's mix is a little more than $F \times P$. A small amount of time (the transient interval) is spent getting the parts into the system. Essentially, the makespan can be minimized by finding a schedule that minimizes the period P.

For each machine, define its "MPS work load" to be the total time it must spend processing parts assigned to it in the MPS. Thus a machine's MPS work span is equal to its MPS work load plus any idle time it incurs processing the MPS. Define the machine with the greatest MPS work load to be

the bottleneck. Clearly the period cannot be less than the bottleneck work load. Notice that the bottleneck work load does not depend on the timing or sequencing and can therefore be precomputed.

Fortunately, in a flexible flow line, it is easy to find a schedule whose period is equal to the bottleneck work load, i.e., a minimum period schedule. A simple way to do this is as follows. Suppose a sequence for the MPS has been chosen. Conceptually, let all the parts of the MPS enter the system as rapidly as possible. Consider the machines, one at a time, in an order consistent with the "flow," i.e., the order in which parts visit the machine banks. At each machine, process the parts in the order in which they arrive. Begin processing each part as soon as the machine is available. After considering all parts of the MPS that visit the machine, if the work span of the machine exceeds its work load, delay processing the first part by the difference. All subsequent parts are still processed in order, starting as soon as the machine is available. Then proceed to the next machine.

The key to the above procedure is the delaying. The difference between a machine's work span and its work load is idle time. Delaying the first part may also delay some of the subsequent parts, but the delay ultimately cancels out with the idle time. Consequently, the work span of the machine after delaying is exactly equal to its work load. Thus the period resulting from this method is simply the largest (i.e., bottleneck) work load. The work loads depend on the machine allocation, but not on the sequence in which the MPS is loaded. It follows that this method of timing achieves the minimum period (= bottleneck work load), regardless of the sequence.

The "timing" algorithm of FFLL is a version of the above method, modified to reduce queuing in the buffers. (For example, it does not load the MPS as rapidly as possible.) How FFLL chooses the *sequence* for the MPS is explained in Section 3.

Now consider the problem of machine allocation. In the context of periodic scheduling, machine allocation is the problem of choosing which machine in each bank will process each part in the MPS. This is done before the sequencing and timing, to allow the latter two subproblems to be solved with known MPS work loads for each machine. The sequencing method described in Section 3 depends critically on this, as does the timing method described above.

The makespan of the schedule is determined by the minimum period, which is equal to the maximum MPS work load. Thus the goal of machine allocation is, for each bank of machines, to minimize the maximum work load among all the machines in that bank.

The lowest conceivable maximum work load for a machine bank would be obtained if all of the machines in the bank were given equal work loads, but this is usually not possible. To find *nearly* balanced work loads, FFLL uses the following heuristic. Each machine bank is considered separately. At each bank, the "Longest Processing Time First" (LPT) heuristic is applied. This approach sorts parts in order of decreasing processing time on the machine bank. (This sorting is only to aid the allocation. It does not determine the loading sequence.) The algorithm then allocates the machines in the bank to one part at a time, in this order. When a machine is allocated to a part, its cumulative work load is increased by the processing time of that part. For each part considered, the machine with the smallest cumulative work load is allocated to it.

In another context, Graham [3] has shown that the LPT rule results in a maximum work load that is never more than 33.3% worse than the optimal solution. More typically, as in the case discussed in Section 4, it came within 0.5%.

3. Sequencing

In the context of periodic scheduling, sequencing is the problem of determining a good order in which to load the MPS. As explained in Section 2, the minimum period can be achieved by appropriate timing, regardless of the MPS sequence. Thus, choosing a sequence for the MPS does not have a significant effect on the makespan. The goal of sequencing is therefore to minimize the amount of queuing.

FFLL uses a new heuristic to choose a sequence for the MPS, called "dynamic balancing." It is heuristic in two ways. The goal of minimizing queuing is difficult to deal with when choosing a sequence, since this requires knowing the loading times, which have not yet been determined. For this reason, dynamic balancing replaces the real objective with a heuristic objective. It then applies a "myopic" heuristic to minimize this new objective.

The heuristic objective in dynamic balancing is based on the following intuitive observations. Parts tend to queue up in the buffer of a machine if a lot of work is sent to that machine in a short period of time. This occurs if there is an interval in the loading sequence consisting of many parts with large processing times all on the same machine. In contrast, a loading sequence which keeps the cumulative work loads of all machines roughly equal at all times tends to avoid queuing. The heuristic objective of dynamic balancing attempts to achieve this.

To be precise, let

n = the number of parts in the MPS,
m = the number of machines (including all banks),
$t_{g,k}$ = process time of part g on machine k.

When the loading sequence is being determined, the machines have already been allocated, so for each part g, $t_{g,k} = 0$, for all but one machine in each bank. Let

$$l_k = \sum_{g=1}^{n} t_{g,k}.$$

That is, l_k is the MPS work load of machine k.

Suppose a loading sequence has been determined. For each part g in the MPS, let S_g be the set of parts in the sequence up through part g. That is, the set S_g consists of part g and those parts which precede part g in the sequence. The problem of choosing a loading sequence can be viewed as determining S_g for each part g.

Suppose, for part g, there is some number α_g such that for all machines k,

$$\sum_{g' \in S_g} \frac{t_{g',k}}{l_k} = \alpha_g. \tag{1}$$

Then $0 \leq \alpha_g \leq 1$. α_g is the fraction of the work load for machine k that has entered the system by the time part g has been loaded. Since α_g does not depend on k, Eq. (1) asserts that this fraction is the same for *all* machines. This is the "dynamic balance" condition that should be achieved by a good loading sequence, for all g. Unfortunately, this condition cannot be achieved, since it implies $t_{g',k} > 0$ for all parts g'. This, in turn, would imply that each part visits all machines, whereas actually each part visits only one machine in each bank. Thus (1) represents an ideal, and the dynamic balancing heuristic tries to come as close as possible to this ideal.

To see how to do this, let

$$t_g = \sum_{k=1}^{m} t_{g,k},$$

$$T = \sum_{g=1}^{n} \sum_{k=1}^{m} t_{g,k} = \sum_{g=1}^{n} t_g = \sum_{k=1}^{m} l_k.$$

Equation (1) implies that

$$\alpha_g = \frac{\sum_{k=1}^{m} \sum_{g' \in S_g} t_{g',k}}{\sum_{k=1}^{m} l_k} = \frac{\sum_{g' \in S_g} t_{g'}}{T}.$$

Thus (1) is equivalent to

$$\sum_{g' \in S_g} t_{g',k} = \sum_{g' \in S_g} \frac{t_{g'} l_k}{T}. \tag{2}$$

The left-hand side of (2) is the cumulative work load of machine k when part g enters the system. The right-hand side is the "ideal" value for this.

Let

$$h_{g,k} = t_{g,k} - \frac{t_g l_k}{T}$$

and

$$H_{g,k} = \sum_{g' \in S_g} h_{g',k}.$$

Then (2) is equivalent to

$$H_{g,k} = 0. \tag{3}$$

Thus, $H_{g,k}$ measures how overloaded machine k is when part g enters the system. If $H_{g,k} > 0$, machine k has been given too much work when part g enters. If $H_{g,k} < 0$, machine k has been given too little work. If $H_{g,k} = 0$, machine k has been given the ideal amount of work.

The dynamic balancing heuristic uses the following objective function:

$$\text{Minimize} \sum_{g=1}^{n} \sum_{k=1}^{m} H_{g,k}^{+}, \tag{4}$$

where

$$H_{g,k}^{+} = \max\{H_{g,k}, 0\}.$$

This objective seeks to minimize the sum of all the machine overloads throughout the whole sequence. Notice that

$$\sum_{k=1}^{m} h_{g,k} = 0,$$

so

$$\sum_{k=1}^{m} H_{g,k} = 0. \tag{5}$$

Thus, by minimizing (4), the heuristic is really trying to get as close as possible to (3).

Finding a sequence which minimizes (4) seems to be a very hard problem. Instead of finding its absolute minimum, the dynamic balancing approach applies a myopic heuristic to this objective. This approach simply adds one part at a time to the end of the sequence, always choosing the part which minimizes the objective at that point in the sequence.

Specifically, suppose part g^* is the last part in the partial sequence determined so far. The myopic heuristic adds to the end of the sequence the part g, which minimizes

$$\sum_{k=1}^{m} H_{g,k}^{+} = \sum_{k=1}^{m} (H_{g^*,k} + h_{g,k})^{+}.$$

By (5), $H_{g^*,k}$ is positive for some k's and negative for others. The same holds for $h_{g,k}$. The heuristic tends to choose a part g such that $h_{g,k}$ is negative when $H_{g^*,k}$ is positive. Now $h_{g,k}$ is most negative if machine k has not been allocated to part g. Thus the heuristic tends to add to the sequence a part that avoids the machines that are currently overloaded.

4. Loading: Results

The FFLL algorithm consists of three major steps:

1. Allocate machines to MPS (LPT heuristic).
2. Sequence MPS (dynamic balancing heuristic).
3. Compute loading times for MPS.

To test the algorithm, FFLL was implemented in an experimental PL/I code on an IBM 3081. This was then used to determine schedules for typical daily mixes for the card line described in Section 1. CPU run times were under one second. One typical day's mix is given in **Table 1**.

The single-letter part types in the first column of Table 1 are pseudonyms for the real part numbers. The numbers in Table 1 are numbers of carriers' worth of cards, i.e., four-card units. The second column indicates how many carriers of each card type are to be produced in the day. The third column indicates how many carriers of each card type are contained in one minimal part set. Thus the first row indicates that 300 carriers' worth of cards of type A (i.e., 1200 cards) are to be produced in the day, and this translates to 12 carriers' worth for one MPS. The results of scheduling this mix with FFLL are summarized in **Tables 2** and **3**.

After the initial transient interval of 5 minutes and 49 seconds, one MPS is produced every 28 minutes and 44 seconds (the period). After 25 periods (the frequency), the whole day's mix is produced. The total time to do this is 12 hours, 4 minutes, and 9 seconds.

Table 3 shows, for each machine, the MPS work load, the utilization, and the maximum number of carriers queued in the corresponding buffer. The machine allocation heuristic has balanced the work loads very effectively. For each machine bank, the work loads, which are nearly half an hour, vary by only a few seconds. The bottleneck machine, SIP_2, has a work load of 28 minutes and 44 seconds, and this determines the minimum period. The utilization of SIP, the bottleneck tool bank, is nearly 100%. It is slightly less because the machines cannot be fully utilized during the transient interval. Very little buffer space is used, only two carriers per machine. This is due to the use of periodic scheduling and to the dynamic balancing heuristic for sequencing.

Another way in which these algorithms were tested was by the use of simulation. Pasquier [4] has developed a RESQ-based simulation system for flexible flow lines. This system has the unique capability of accepting as input a detailed loading schedule such as would be produced by FFLL or some other algorithm. Pasquier used this system to compare FFLL with simpler policies to deal with the various aspects of the loading problem. He generally found that alternatives to FFLL resulted in more queuing at the buffers, although throughput could be maintained as long as some appropriate periodic schedule was followed.

One of the advantages of using a periodic scheduling policy like that produced by FFLL is its adaptability to machine failures. Pasquier [4] used his simulation to experiment with several ways to adapt FFLL's schedule in response to machine failures. The results are summarized below.

As a minimal response, any part which FFLL has routed to a failed machine must be rerouted to a functional machine in the same bank. Also, the period must be increased to reflect the reduced capacity. In order to compute the new period, the MPS work load is recomputed for each functional machine in each bank that contains a failure. For this purpose, it can be assumed that the work loads are the same for each machine in a bank. If the new work load of any machine exceeds the old period, the period is increased to this new value. To appropriately slow down the loading of the parts into the system, the interarrival times are increased by a factor of

$$\frac{P_{\text{new}}}{P_{\text{old}}},$$

where P_{new} is the newly determined period and P_{old} is the original period. When the machines are repaired, the FFLL schedule is resumed.

Figure 2 shows the result of using this minimal adaptation on a simulation of the mix from Table 1. It shows a plot of queue length versus time for each of the robots. The horizontal axis is elapsed time in seconds. (About four hours are shown.) The vertical axis shows the number of parts (i.e., carriers) waiting in the buffer or being processed by each

Table 1 A typical day's mix.

Part type	*Day's mix*	*MPS*
A	300	12
B	25	1
C	650	26
G	25	1
H	175	7
K	100	4

Table 2 Loading results (hours:minutes:seconds).

Frequency	25 periods
Period	28:44
Transient	5:49
Makespan	12:04:09

Table 3 Loading results by machine.

Machine	*Work load*	*Utilization (%)*	*Max. queue*
DIP_1	25:31	88	1
DIP_2	25:32	88	0
SIP_1	28:27	98	2
SIP_2	28:44	99	1
SIP_3	28:36	99	2
$ROBOT_1$	26:56	93	2
$ROBOT_2$	26:56	93	2
$ROBOT_3$	27:12	94	2

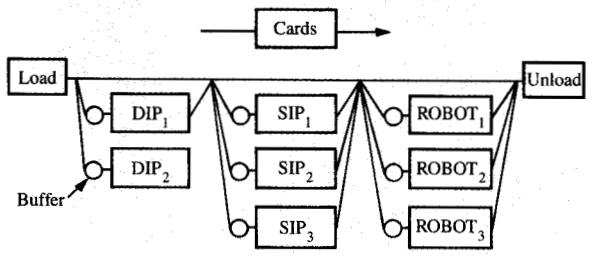

Figure 1

Example machine configuration (schematic).

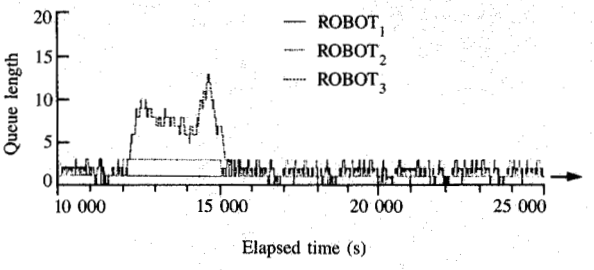

Figure 3

Reaction to machine failures (dynamic routing).

robot. A queue of 0 means that the robot is idle. Robots 1 and 2 fail at time 12 000 and are repaired at time 15 000. This is the worst case for a flexible flow line, in that the most downstream machines have failed. Even though parts are introduced into the system more slowly, the parts already in the system proceed at the normal rate and pile up in front of the one working robot. In spite of this, the buffer capacity of 25 carriers is never exceeded.

In one respect, this adaptation of the schedule is unsatisfactory. After the two robots are repaired at time 15000, a large queue remains in the buffer of robot 3. This is unnecessary, since the other two robots are occasionally idle. This behavior is due to "static routing," i.e., routing parts according to the machine allocation chosen in advance by FFLL. Static routing is not appropriate after machines have failed and been repaired, since the system is no longer in a state predicted by the algorithm. Instead, Pasquier experimented with several "dynamic routing" policies, which routed parts according to the state of the system. The best policy he found was as follows. When a part leaves one machine, it is sent to the machine in the next bank with the shortest completion time, i.e., the machine which will soonest complete processing all the parts in its buffer. **Figure 3** shows what happens when this dynamic routing is used. When the robots are repaired at time 15000, the queue on robot 3 immediately returns to a normal level (as quickly as it can process the parts).

5. Mix allocation

The second problem to be discussed is mix allocation. During a planning horizon of, say, a week to a month, a set of part types must be produced in given volumes. (In the card line example, the planning horizon was a five-day week.) The problem is to choose production volumes for each part type on each day. In other words, the weekly or monthly part mix is to be allocated to each day.

As in the loading problem, the main objective of mix allocation is to minimize makespan, or total completion time of each day's mix. Specifically, the *maximum* of the daily makespans in the planning horizon is to be minimized. The purpose of minimizing the maximum makespan is to maximize the amount of slack time available each day to respond to unpredictable events such as machine failures.

A second and less important objective is to minimize the maximum daily part type count, that is, the maximum number of part types allocated to any day. This simplifies logistics and decreases any off-line setup not explicitly considered.

Finally, there is a requirement that each day's production level of each part type be a multiple of some given unit. This is to allow parts to be produced and transported in batches. For the card line example, the "unit" is 100 cards (although they are transported four at a time).

The mix allocation problem was solved for a somewhat more general environment than the flexible flow line, as described in Section 1. The environment may be called a "reconfigurable flexible flow line." In this environment, the set of machines within each bank can change from day to day. Furthermore, the individual machines within each bank may operate at different speeds. Just as in the ordinary flexible flow line, a part visits only one machine in each bank, since the machines within a bank perform the same "type" of operation. The processing time of a machine on a part is determined by the rate at which the machine performs operations and by the number of operations the part requires.

Since this is a more general environment than that discussed previously, the card line described in Section 1 is still a legitimate example. In this case, each machine type is dedicated to one operation type. The rates at which the machines perform their operations are given in **Table 4**.

Table 5 indicates production requirements and other information for a typical mix allocation problem for the example card line. There are 13 part types (letters A–M) to be produced during the planning horizon of five days. Daily production levels are to be in multiples of 100.

6. Minimizing the makespans

A Linear Programming (LP) formulation was used to solve this mix allocation problem. Let

H = the number of days in the planning horizon.
N = the number of part types.
M = the number of operation types.
d_j = the total number of parts of type j to be produced during the planning horizon.
$h_{i,j}$ = the number of operations of type i necessary for one part of type j.
$S_{i,t}$ = the set of machines that perform operation i on day t.
r_k = the number of operations that machine k can perform in one unit of time (say, an hour).
$u_{j,t}$ = the number of parts of type j allocated to day t.

The $u_{j,t}$'s are the quantities to be found. The LP model allocates the total demand in a way that minimizes the maximum of the daily makespans.

The makespan for a day's mix cannot be computed without scheduling the mix. Thus an approximation is used. Define a machine's *work load* to be the total time it must spend performing its share of the day's mix. Define the work load of an operation (type) to be the maximum work load of all the machines performing it. The makespan of a day's mix is always greater or equal to the maximum operation work load, and this maximum is used as the approximation to the makespan.

The work load of an operation depends on how the work is distributed among the machines performing it, i.e., the machine allocation. It is assumed that the work load can be distributed equally among the machines performing an operation. This would minimize the operation work load. As noted in Section 2, a nearly perfect machine allocation can be found for a flexible flow line, by using the LPT heuristic.

The number of operations of type i to be performed on day t is

$$\sum_{j=1}^{N} h_{i,j}u_{j,t}.$$

Let $g_{i,t}$ be the number of operations of type i that can be performed during one time unit on day t. Then

$$g_{i,t} = \sum_{k \in S_{i,t}} r_k.$$

In the example of Section 5, $S_{i,t}$ and therefore $g_{i,t}$ are independent of t. Section 10 discusses a case in which these quantities vary with t.

Let

$$w_{i,t} = \frac{\sum_{j=1}^{N} h_{i,j}u_{j,t}}{g_{i,t}}.$$

Then $w_{i,t}$ is the work load of operation type i on day t, assuming that the load is distributed equally among all machines performing the operation. The approximation to the makespan for day t is the maximum work load among all operations on that day:

$$\max_i w_{i,t}.$$

This approximation is always a lower bound to the makespan. This bound can be nearly achieved using FFLL.

Thus the approximation to the maximum daily makespan is

$$w = \max w_{i,t} \qquad i = 1, \cdots, M, \; t = 1, \cdots, H.$$

Let

$$w_i = \max_t w_{i,t};$$

w_i is the maximum work load of operation type i. The goal is to minimize

$$w = \max_i w_i.$$

The operation type i which achieves this maximum is the bottleneck operation for the planning horizon. Unfortunately, if one minimizes the maximum work load of only the bottleneck operation, the nonbottleneck work loads can fluctuate wildly from day to day, which is an undesirable situation. Thus it is worthwhile to minimize the maximum work load of all operations instead of just the bottleneck.

Table 4 Machine information.

Machine type	*Operation type*	*Operations per hour*
DIP	DIP	1180
SIP	SIP	1471
ROBOT	MODULE	810

The *Operations per hour* column gives an average number of operations. This takes into account the expected amount of down time for the machine, which otherwise behaves deterministically.

Table 5 Part type information.

Part type	*Operations required for one card*			*Production requirement*
	DIP	*SIP*	*MODULE*	
A	9	3	2	1200
B	3	8	8	100
C	5	16	9	7000
D	54	11	0	200
E	9	7	12	1700
F	3	20	8	5000
G	33	19	0	700
H	0	8	2	700
I	6	11	11	600
J	4	17	5	400
K	5	20	6	500
L	3	20	8	400
M	14	13	5	1500

A preliminary LP formulation to do this is

$$\text{Minimize} \quad w_i \qquad i = 1, \cdots, M$$

Subject to

$$\sum_{j=1}^{N} (h_{i,j}/g_{i,t})u_{j,t} \le w_i \qquad i = 1, \cdots, M,$$
$$t = 1, \cdots, H,$$
$$\sum_{t=1}^{H} u_{j,t} = d_j \qquad j = 1, \cdots, N,$$
$$u_{j,t} \ge 0. \tag{6}$$

The first set of constraints define w_i as the maximum work load of operation type i. The second constraints force total production to be at the demanded level d_j.

Equation group (6) is a multiobjective LP. It seeks to minimize w_i for all i. For general linear programming problems, one cannot expect to minimize more than one objective at the same time. Trade-offs must be made. However, for this particular multiobjective LP, all objectives can be simultaneously minimized. (This will be shown.)

The constraints of (6) can be used to precompute a lower bound, w_i^*, on w_i. That is,

$$\sum_{t=1}^{H} g_{i,t}w_i \ge \sum_{t=1}^{H}\sum_{j=1}^{N} h_{i,j}u_{j,t} = \sum_{j=1}^{N} h_{i,j}d_j.$$

So

$$w_i \ge \frac{\sum_{j=1}^{N} h_{i,j}d_j}{\sum_{t=1}^{H} g_{i,t}} \triangleq w_i^*. \tag{7}$$

The final LP formulation of mix allocation is

$$\text{Minimize} \quad \sum_{j=1}^{N}\sum_{t=1}^{H} c_{j,t}u_{j,t}$$

Subject to

$$\sum_{j=1}^{N} (h_{i,j}/g_{i,t})u_{j,t} = w_i \qquad i = 1, \cdots, M,$$
$$t = 1, \cdots, H,$$
$$\sum_{t=1}^{H} u_{j,t} = d_j \qquad j = 1, \cdots, N,$$
$$u_{j,t} \ge 0. \tag{8}$$

The cost coefficients $c_{j,t}$ are explained in Section 7. The critical difference between (8) and (6) is that the constraints on w_i are equalities in (8). Thus the work load for each operation is required to be constant throughout the planning horizon.

By a derivation exactly parallel to (7), it can be seen that the constraints in (8) imply that

$$w_i = w_i^*.$$

But since w_i^* was a lower bound to w_i in (6), it follows that any feasible solution to (8) *simultaneously* minimizes w_i for all i subject to the constraints of (6). That is, by finding a feasible solution to (8), one is finding an optimal solution to all objectives of (6). Thus (8) is an appropriate model for mix allocation.

Equation (7) can be used to precompute the daily makespan in (8) as

$$w^* = \max_i w_i^*. \tag{9}$$

Also, since the w_i's are precomputable, they could be treated as constants in (8). However, this would simply introduce redundancy into the constraints, so the w_i's are treated as variables.

Suppose that the machine configuration does not change from day to day (as in an ordinary flexible flow line), so that $g_{i,t}$ is independent of t. In this case, it is easy to compute a feasible solution to (8),

$$u_{j,t}^* = \frac{d_j}{H}.$$

In general, however, it is possible that (8) is infeasible. This would happen in a pathological case in which the $g_{i,t}$'s fluctuate so wildly that the production levels cannot be balanced. In this case, one must settle for achieving the load balancing constraints as nearly as possible. This can be done by solving some single objective version of (6). [Notice that (6) is always feasible.] One way to do this is to select w_i^* as a cost for each w_i, i.e.,

$$\text{Minimize} \sum_{i=1}^{H} w_i^* w_i.$$

This approach gives the most priority to the bottleneck operation.

In the sequel, (8) is assumed feasible.

7. Minimizing part type count

A second objective for mix allocation is to minimize maximum daily part type count. The part type count for day t is the number of part types allocated to day t, or in other words, the number of $u_{j,t} > 0$ for fixed t. Since the optimal solution to (8) is a basic solution, it has no more than $MH + N$ positive variables. M of them are w_i's, and the rest are allocated part types. Thus the average daily part type count is

$$M + \frac{N - M}{H}.$$

Of course, any one day could have a much higher part type count.

Strictly speaking, minimizing the maximum part type count involves counting the number of positive variables, and this appears to require the use of integer variables. Instead of employing mixed integer programming, a heuristic approach is taken, by defining the LP cost coefficients in (8) so as to encourage an evenly spread part type count. Let

$$c_{j,t} = (t - j) \bmod H.$$

For example, if $H = 5$ and $N = 13$,

$$[c_{j,t}] = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 4 & 0 & 1 & 2 & 3 \\ 3 & 4 & 0 & 1 & 2 \\ 2 & 3 & 4 & 0 & 1 \\ 1 & 2 & 3 & 4 & 0 \\ 0 & 1 & 2 & 3 & 4 \\ 4 & 0 & 1 & 2 & 3 \\ 3 & 4 & 0 & 1 & 2 \\ 2 & 3 & 4 & 0 & 1 \\ 1 & 2 & 3 & 4 & 0 \\ 0 & 1 & 2 & 3 & 4 \\ 4 & 0 & 1 & 2 & 3 \\ 3 & 4 & 0 & 1 & 2 \end{bmatrix}.$$

Let

$$T(j) = [(j - 1) \bmod H] + 1.$$

With the objective defined above, the LP tends to allocate part type j to day $T(j)$, since $c_{j,T(j)} = 0$. Any extra parts of type j tend to be allocated to days $T(j + l)$, where l is as small as possible, since $c_{j,T(j+l)} = l$ for $l < H$. This should produce an evenly spread part type count, since $T(j)$ chooses values from $\{1, \cdots, H\}$ with nearly equal frequency.

8. Producing in units

There is also a requirement to produce in multiples of some unit, say, U. That is, $u_{j,t}/U$ must be an integer. Again, this is integer programming, but it is shown here how to get good answers from the LP solution by intelligent rounding.

The rounded solution must satisfy the demand constraints of (8). The demands d_j are assumed to be multiples of U. On the other hand, requiring the equality work load constraints of (8) to be satisfied would almost certainly result in no feasible solution. Thus (6), which views these constraints as inequalities, is used for rounding purposes.

At this point, a single, scalar objective is required, but it need not be linear. The following quadratic objective is used:

$$\text{Minimize} \sum_{i=1}^{M} w_i^2.$$

This objective tends to minimize all the work loads, with the greatest emphasis on the bottleneck. Here again, the unrounded solution to (8) minimizes the objective.

Let u be the vector of $u_{j,t}$ values. The objective can be expressed as a function of u:

$$f(u) = \sum_{i=1}^{M} \left\{ \max_{t} \left[\sum_{j=1}^{N} (h_{i,j}/g_{i,t}) u_{j,t} \right] \right\}^2.$$

Define integers $x_{j,t}$ and real numbers $v_{j,t}$ such that

$$u_{j,t} = x_{j,t} U + v_{j,t},$$

$$0 \le v_{j,t} < U.$$

Let

$$y_j = \sum_{t=1}^{H} v_{j,t}.$$

Then y_j is a multiple of U. For each part type j, y_j/U of the $u_{j,t}$'s must be rounded up to $(x_{j,t} + 1)U$ and the rest rounded down to $x_{j,t}U$. Of course, most of the $u_{j,t}$'s are 0.

Suppose $u_{j,t'}$ is to be rounded up. (How to choose j and t' is described later.) To maintain the demand constraint, $u_{j,t}$ must be decreased for some or all $t \neq t'$. To do this, each $v_{j,t}$ must be replaced by $v'_{j,t}$ such that

$$\sum_{t \neq t'} v'_{j,t} + U = y_j.$$

The rounding technique decreases all the $v_{j,t}$'s proportionately. That is, for all $t \neq t'$,

$$v'_{j,t} = \alpha v_{j,t}$$

for some α. Thus

$$\alpha = \frac{\sum_{t \neq t'} v'_{j,t}}{\sum_{t \neq t'} v_{j,t}} = \frac{y_j - U}{y_j - v_{j,t'}}.$$

If $y_j = U$, $\alpha = 0$ and no more rounding needs to be done for part type j.

The rounding technique iterates as follows: j and t' are chosen such that $v_{j,t'} > 0$. $u_{j,t'}$ is rounded up and $u_{j,t}$ for $t \neq t'$ is decreased accordingly. All of the $x_{j,t}$, $v_{j,t}$, and y_j are updated accordingly. In particular, y_j is decreased by U. This is repeated until all $y_j = 0$, which implies that all $u_{j,t}$ are multiples of U.

At each iteration, j and t' are chosen in a heuristic attempt to minimize the (ultimate) value of $f(u)$. Let $u(j, t')$ be the value of u corresponding to rounding up $u_{j,t'}$ and adjusting u accordingly. Let

$$V_j = \{t: v_{j,t} > 0\}.$$

Let

$$f^* = \max_{j} \left((\min_{t' \in V_j} \{f[u(j, t')]\} \right).$$

Table 6 LP allocation.

Part type	*Production levels by day*				
	1	*2*	*3*	*4*	*5*
A	1200	—	—	—	—
B	—	100	—	—	—
C	—	—	2600	2200	2200
D	100	—	—	—	100
E	500	1000	—	—	200
F	2600	2200	—	—	200
G	—	400	—	100	200
H	—	—	600	100	—
I	—	—	—	500	100
J	—	—	—	—	400
K	—	—	—	500	—
L	—	—	—	—	400
M	—	—	1100	400	—
Part type count	4	4	3	6	8

Table 8 Unaided allocation.

Part type	*Production levels by day*				
	1	*2*	*3*	*4*	*5*
A	200	400	—	—	600
B	100	—	—	—	—
C	1000	3000	1000	1000	1000
D	100	100	—	—	—
E	600	500	—	—	600
F	1000	—	2000	2000	—
G	100	—	300	—	300
H	200	—	200	—	300
I	200	—	—	—	400
J	—	—	400	—	—
K	—	—	100	—	400
L	—	—	—	400	—
M	500	—	—	600	400
Part type count	10	4	6	4	7

Table 7 Resulting work loads.

Operation type	*Work loads by day* (h)					
	1	*2*	*3*	*4*	*5*	*Max.*
DIP	12.1	12.3	12.0	10.8	12.2	12.3
SIP	13.6	13.5	13.8	13.3	13.9	13.9
MODULE	12.0	12.5	12.4	12.6	12.4	12.6
Makespan	13.6	13.5	13.8	13.3	13.9	13.9

Table 9 Resulting work loads.

Operation type	*Work loads by day* (h)					
	1	*2*	*3*	*4*	*5*	*Max.*
DIP	13.7	12.1	9.7	8.7	15.1	15.1
SIP	12.4	12.2	16.3	16.3	10.8	16.3
MODULE	12.6	13.9	11.5	12.8	11.0	13.9
Makespan	13.7	13.9	16.3	16.3	15.1	16.3

Values of j and t' are chosen which achieve f^*.

A more intuitive heuristic would choose the minimum over j instead of the maximum. However, the above approach seems to work better empirically. (It gave better makespans on several test cases similar to those given in Sections 9 and 10.) For an intuitive explanation, observe that each value of j must be chosen y_j/U times, sooner or later. Values of j such that the minimum over t' is high are those part types for which rounding necessarily causes an imbalance in the work load. Choosing those values first allows the imbalance to be corrected by appropriately rounding the other part types afterwards.

9. Mix allocation: Results

This LP allocation technique was applied to the specific problem defined in Section 5. The method was implemented with a PL/I program which generates the LP matrix, calls IBM's MPSX to solve the LP, and rounds and interprets the solution. Each run took under one second of CPU time on an IBM 3081.

The resulting daily production levels ($u_{j,t}$) and part type counts are given in **Table 6**. The work loads for each operation type for each day ($w_{i,t}$) are given in **Table 7** along with the maximum for the whole week (w_i). Table 7 also lists the maximum operation work load for each day, which is the lower bound on the day's makespan.

The daily makespans vary by only 0.6 hours. Thus, the rounding has not damaged the solution much. Also notice that most part types have been allocated to day $T(j)$. For example, part type H (index 8) has been allocated to day 3.

For comparison, **Tables 8** and **9** show the same information for an allocation of the same week's mix determined by the production manager without the aid of a model.

The LP method improves the makespans by up to three hours while simultaneously achieving better part type counts. Notice that for the LP allocation, the SIP operation is always the bottleneck, i.e., it always has the greatest work load, thus determining the makespan. This is efficient, since the SIP operation is the bottleneck to the whole week's production. With the unaided allocation, the bottleneck operation changes from day to day. Thus, on days 1, 2, and 5, SIP capacity is underutilized. This explains why the LP allocation achieves lower makespans.

The mix shown in Table 1 for FFLL was generated by the mix allocation LP.

10. Reassigning a machine

Tables 10 and **11** give the results of applying the LP approach in the same basic environment, but with a different weekly requirement (indicated in Table 10).

Note that the DIP operation is a serious bottleneck, while the robots, which do the MODULE operations, are greatly underutilized. It turns out that the robots can do DIP or SIP operations as well as the MODULE operations they are already doing. The only restriction is that each robot should only do one type of operation per day (to avoid substantial midday setups). Moreover, even when it does DIP or SIP operations, a robot still operates at the rate of 810 operations per hour. This is the case of the "reconfigurable" flexible flow line.

There seems to be a potential to decrease the makespans by reassigning one or more robots to the DIP operations on one or more days. In terms of the LP, such a reassignment would alter the $S_{i,t}$'s, and through them the $g_{i,t}$'s, so that they depend on t. On some days, the set $S_{i,t}$ of machines doing DIP operations includes a robot, and other days it does not. The corresponding $g_{i,t}$ values are greater on the days for which the robot is reassigned than on the other days. The effect of such a reassignment can easily be determined by computing the daily makespan (of the unrounded solution) using (9). There are really only a few sensible ways to do this reassignment, and by exhaustive search, the minimum value of $w^* = 15.1$ was found for assigning one robot to DIP operations on three days (1, 2, and 3), and to MODULE operations on the other two days. The other two robots are always assigned to MODULE operations. **Tables 12** and **13** show the result of applying the LP method to this reassigned configuration.

The makespans are reduced by an average of three hours per day, a substantial saving. This has been achieved by reallocating parts with high DIP requirements (e.g., part type G) to the first three days, and those with high MODULE requirements (e.g., part type I) to the last two days. Thus the LP approach determines how to take maximum advantage of the reassignment of machines. The choice of how to reassign was done by a hand calculation, but this could be automated.

Finally, suppose that the robots could do all types of operations without setup. In this case, a line consisting only of robots could be used. Each part would only visit one machine. Since there is only one bank of machines, mix allocation is trivial. Assuming that eight robots are used

Table 10 LP allocation (second case).

Part type	*Production levels by day*					*Requirement*
	1	*2*	*3*	*4*	*5*	
A	1000	—	—	—	—	1000
B	—	100	—	—	—	100
C	—	—	2300	2400	2300	7000
D	200	—	—	200	—	400
E	700	400	—	—	400	1500
F	1900	1100	—	—	—	3000
G	300	900	300	600	800	2900
H	—	—	—	500	—	500
I	—	800	—	300	—	1100
J	—	—	—	—	300	300
K	300	—	—	—	—	300
L	—	500	—	—	—	500
M	—	—	1400	—	—	1400
Part type count	6	6	3	5	4	

Table 11 Resulting work loads.

Operation type	*Work loads by day* (h)					
	1	*2*	*3*	*4*	*5*	*Max.*
DIP	18.3	18.3	17.4	18.8	18.1	18.8
SIP	13.5	13.9	13.8	13.4	13.6	13.9
MODULE	11.3	11.2	11.4	10.7	11.1	11.4
Makespan	18.3	18.3	17.4	18.8	18.1	18.8

Table 12 LP allocation (reassigned configuration).

Part type	*Production levels by day*				
	1	*2*	*3*	*4*	*5*
A	1000	—	—	—	—
B	—	100	—	—	—
C	—	—	1800	2700	2500
D	300	—	—	100	—
E	300	500	—	—	700
F	1900	1100	—	—	—
G	400	1100	700	200	500
H	—	—	500	—	—
I	—	300	—	800	—
J	—	—	—	—	300
K	300	—	—	—	—
L	—	500	—	—	—
M	—	—	1100	300	—
Part type count	6	6	4	5	4

Table 13 Resulting work loads.

Operation type	*Work loads by day* (h)					
	1	*2*	*3*	*4*	*5*	*Max.*
DIP	15.2	15.0	14.6	15.5	15.1	15.5
SIP	13.6	13.7	13.7	13.8	13.5	13.8
MODULE	14.0	14.1	14.0	14.2	13.3	14.2
Makespan	15.2	15.0	14.6	15.5	15.1	15.5

instead of the eight specialized machines, (9) can be used to compute the corresponding makespans. For the production requirement of Table 10, the makespan is $w^* = 20.1$. This is substantially worse than the 15.1 for Table 13, since the DIP and SIP inserters are much faster than the robots. To achieve a comparable makespan would require 11 robots, with $w^* = 14.6$.

11. Concluding remarks

The experience with FFLL and with the LP method for mix allocation indicates that these methods are valuable tools for a flexible flow line. Future research on this subject will proceed in two directions. The methods will be extended to other scheduling environments. Most importantly, setup time for a machine switching from one part type to another must be considered. It seems that in many production environments, part types are grouped into a few families such that a machine incurs setup time only when it switches from one family to another. Also, nonperiodic scheduling must be developed for environments in which the processing time for a single part is so long as to make periodic scheduling impractical. Finally, an implementation of these algorithms will be developed for online use. Since the CPU and storage requirements of the algorithms are minimal, the implementation can be on a PC.

Acknowledgments

I wish to thank C. Abraham of the IBM Yorktown Manufacturing Research Center for his guidance in this research and in the preparation of the paper, and J. Pasquier, also of the IBM Yorktown Manufacturing Research Center, for his contribution to the research. Thanks also to K. Joshi and W. White, R. Kofira, S. Cooper, and R. Groves of the IBM Tucson Printed Circuit Board Manufacturing Organization for the encouragement and information they have provided which made the research possible.

References

1. E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Recent Developments in Deterministic Sequencing and Scheduling: A Survey," *Deterministic and Stochastic Scheduling*, M. A. H. Dempster et al., Eds., D. Reidel Publishing Company, Boston, MA, 1982.
2. K. L. Hitz, "Scheduling of Flexible Flow Shops—II," *Technical Report LIDS-R-1049*, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 1980.
3. R. L. Graham, "Bounds on Multiprocessing Timing Anomalies," *SIAM J. Appl. Math.* **17**, 416–429 (1969).
4. J. Pasquier, "High-Level Simulation of Flexible Card Assembly Lines," *Research Report RC-10881*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1984.

Received December 19, 1984; revised February 12, 1985

Robert J. Wittrock *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Wittrock is a research staff member in the manufacturing logistics group of the Manufacturing Research Department at Yorktown, where he joined IBM in 1983. Currently he conducts research in mathematical models for logistics for IBM manufacturing environments. He received his B.S. in mathematics in 1978 from the University of the Pacific, Stockton, California. He received his M.S. and his Ph.D. in operations research in 1979 and 1983, both from Stanford University, California. He is a member of the Operations Research Society of America.