SEQUENCING TO MINIMIZE WORK OVERLOAD IN ASSEMBLY LINES WITH PRODUCT OPTIONS

Candace Arai Yano
Department of Industrial and Operations Engineering

Ram Rachamadugu
School of Business Administration

The University of Michigan
Ann Arbor, Michigan 48109

October 1987

Revised November 1988

Revised August 1989

Revised November 1989

Abstract

We address the problem of sequencing jobs, each of which is characterized by one of a large number of possible combinations of customer-specified options, on a paced assembly line. These problems arise frequently in the automotive industry. One job must be launched into the system at equal time intervals, where the time interval (or cycle time) is prespecified. The problem is to sequence the jobs to maximize the total amount of work completed, or equivalently, to minimize the total amount of incomplete work (or work overload).

Since there is a large number of option combinations, each job is almost unique. This fact precludes the use of existing mixed model assembly line sequencing techniques. We first consider the sequencing problem for a single station which can perform two different sets of operations. We characterize the optimal solution for this problem and use the results as the basis for a heuristic procedure for multiple stations. Computational results with data from a major automobile company are reported.

SEQUENCING TO MINIMIZE WORK OVERLOAD IN ASSEMBLY LINES WITH PRODUCT OPTIONS

1. INTRODUCTION

Paced assembly lines are commonly used to manufacture different models of the same general product. Although the number of models is normally limited, the presence of customer specified options can enormously increase the number of distinct products that can be ordered. For example, Weiner (1985) points out that there are potentially 2.5 million unique configurations for the Escort, Ford's subcompact car. Most of the proposed approaches for mixed-model assembly line sequencing assume that the number of distinct products is small and that the mix of products is relatively stable. Various versions of the mixed model assembly line sequencing problem exist (see, for example, Wester and Kilbridge (1964), Thomopoulos (1967), Dar-el and Cother (1975), Dar-El and Cucuy (1977), Monden (1983), and Schonberger (1982)). The main theme throughout these articles is one of determining a sequence of models (e.g., ABCABD...) so as to spread out a particular model or models as smoothly as possible. A model might be selected for consideration because of its high labor content or because it uses a component which is being delivered at a constant rate, such as in a just-in-time context. Thomopoulos (1967) discusses some of the issues of concern in determining a sequence for mixed model assembly lines, such as worker idle time and the cost of work incompletion.

This paper focuses on sequencing paced assembly lines with many distinct products. We assume that each product is characterized by the presence or absence of each of a *set* of available options or upgrades. This situation is evident in the automobile industry, where each product can be defined by the presence or absence of air conditioning, power

windows (versus manual windows), deluxe seats (versus standard seats), presence or absence of rear defoggers, manual or automatic transmission, etc.

The main characteristics that distinguish paced assembly lines from other assembly systems are that the jobs arrive at each workstation at equal intervals of predetermined duration, and that work can be performed at a particular station only when a job is in a particular portion of the assembly line (which we call a window). If a worker attempts to perform work outside of that window, it is likely that the worker will interfere with a worker in an adjacent station, or that the necessary power tools will not reach the job. Also, additional unproductive travel (walking) time is spent when a worker strays too far from his workstation. (In the case of robotic assembly, it is evident that the robot may not be able to reach the job, or in doing so, might collide with another robot.) Of course, a human worker has the alternative of performing operations more quickly than normal, but this often leads to quality problems. These problems would need to be rectified later, sometimes at great expense. Moreover, low quality has an adverse impact on market share and on profitability. Thus, these quality problems need to be avoided.

Okamura and Yamashima (1979) proposed a heuristic procedure for a single station for the objective of minimizing the maximum distance an assembly worker would need to proceed downstream from his workstation (if he were permitted to do so) to ensure completion of all work. Coffman et al. (1985) proposed the use of spacing constraints where K out of N sequential orders are permitted to contain an option (K and N may vary by station). This heuristic procedure was designed to

resequence a few dozen jobs after the original sequence was scrambled by an upstream repair process. As each job becomes available for sequencing, the algorithm places it in the first available position for which none of the constraints is violated. One drawback of the procedure is that it can leave gaps in the sequence, and constraints will be violated when the gaps are closed. Burns and Daganzo (1985) studied the relationship between the spacing of jobs having an option and peak capacity requirements for a single station. Bird (1986) developed a Markov chain approach to model the expected time to violation of a (single) spacing constraint for a random sequence. Parrello (1988) describes the difficulty of using an expert system to solve problems of this type.

We address the problem of determining a sequence to minimize the total work overload for a large number of (e.g., 1,000) jobs on a paced assembly line. By work overload we mean the work (measured in processing time) which would not be completed if the assembler works at a normal pace and stays within the boundaries of the window. The impact of sequencing on work overload is most easily explained by an example. Consider a station that deals with the installation of sun roofs, and assume that its window length and staffing level were specified to accommodate sun roofs on 20% of automobile orders. If every fifth vehicle has a sun roof, there will be no work overload. Depending upon the window length and staffing level, it may be possible to complete all the work in a sequence in which the spacing between jobs with sunroofs alternates between four and six. Eventually, however, larger deviations from the "every fifth vehicle" pattern will normally cause some work

overload unless there is a considerable amount of excess labor capacity at the station.

The objective in the Okamura and Yamashima (1979) paper is similar to minimizing the maximum work overload per job. (Under their assumptions, it is equivalent to this criterion when there is one worker per station, but is different when there is more than one worker at the station.) Their objective, however, does not guarantee good average performance, which is our concern. For instance, in the sun roof example, once a subsequence of two sun roofs in a row has been sequenced, the objective in the Okamura and Yamashima paper provides no incentive to prevent additional subsequences with two sun roofs in a row, whereas our objective does.

Normally, the work overload is handled through the use of utility workers who either are dispatched to assist the regular workers during peak load situations, or are stationed at various points along the assembly line to complete the unfinished operations. The other alternative, of course, is to staff the assembly line to handle estimated peak loads. This is an expensive solution, so a typical "real" solution would involve a combination of slight overstaffing and utility work. No matter what the staffing policy is, it is clear that minimizing the work overload contributes to reducing the total labor cost (and improving the product quality).

The remainder of this paper is organized as follows. Section 2 describes the problem of scheduling work on jobs to minimize total work overload at a station for a given sequence and characterizes an optimal policy. These results are used in sections 3 and 4 to develop recursive formulas for work overload and to develop optimal sequencing procedures

when only one workstation which is affected by the presence or absence of an option is considered. Using these results, we develop a heuristic procedure for minimizing the total work overload when many stations must be considered. This is explained in section 5. Experimental results appear in section 6 and concluding remarks are contained in section 7.

2. SCHEDULING (OR TIME ALLOCATION)

In this section we consider how to schedule work on jobs to minimize work overload for a given sequence. We assume that the labor available at a station is constant over the scheduling horizon, which, in our application, would represent one shift or one day. Even though it is possible to vary the work force levels, such changes are planned very infrequently in practice. Scheduling can be done at each workstation independently because the labor availability constraints are separable by workstation. For simplicity, we use a discrete time version of the problem. The scheduling (or time allocation) problem at a station is to

$$\min \sum_{i=1}^{N} \left(w_i - \sum_{t \in I_i} x_{it}\right)^+$$

$$\text{s.t.} \sum_i x_{it} \leq b \qquad \forall\ t$$

$$x_{it} \geq 0 \qquad \forall\ i,\ t \in I_j$$

$$x_{it} = 0 \qquad \forall\ i,\ t \notin I_j$$

where

N = total number of jobs

i : job index $i = 1, 2, \ldots, N$

$(x)^+$ = max $(0,x)$,

w_i = work content for job i (e.g., person-minutes) at the station,

I_i = set of time periods in which job i is available at station,

b = amount of labor available at a station (per unit time), and

x_{it} = amount of time allocated to job i in period t.

The objective minimizes total work overload. The constraints ensure that labor availability limits are satisfied and that each job is worked on only when it is at the station.

Since the jobs are on a paced assembly line, each job spends the same amount of time in the station (or window), whether or not it is completed. Also, since one job is introduced to the line at equal time intervals, both the arrival and departure times of the jobs are monotonically increasing with the position in the sequence. In what follows, we will assume that workers are never idle when there is work to be done at the station. It is clear that there always exists an optimal policy with this characteristic. The only issue is how to schedule work on the jobs.

It can be shown that a non-preemptive first-come, first-served policy provides an optimal solution to the problem. In our analysis, a non-preemptive schedule denotes the practice wherein once the processing of a job is started, it is stopped only when the work on it is completed or when it leaves the workstation due to pacing of the line (whichever occurs first). The first-come, first-served policy indicates that jobs are processed in the same sequence in which they arrive at the workstation.

In Appendix A we prove the following proposition.

Proposition 1: For any given sequence, a non-preemptive FCFS work schedule provides an optimal solution.

We note, however, that the optimal solution is not necessarily unique. Nonetheless, these properties help to simplify decision problems presented later in the paper. These results have some useful implications. The first implication is that the optimal work schedule for jobs 1,..., i does not change when job i + 1 is added to the sequence. Therefore, work overload can be expressed quite simply as we will show in the next section. The second implication is that adding jobs to the end of a sequence cannot decrease work overload since the work schedule does not change for the earlier jobs. We also note that the non-preemptive FCFS schedule is easy to implement.

3. MATHEMATICAL PROGRAMMING FORMULATION

Formulas for Work Overload

The aim of this section is to develop formulas for work overload in a fairly general situation where there may be more than one worker at a workstation and labor efficiency may be less than 100% because of other considerations such as union rules. The results in the previous section allow us to restrict our attention to first-come, first-served, non-preemptive policies.

Let the time unit be the interarrival time of jobs to the first station (i.e., one cycle time). Also let

L = station (or window) length

= number of jobs at the station (or window) at any time

= number of time units a job spends at the station (or window);

R = rate at which work is applied per unit time

= number of workers multiplied by efficiency factor;

p_i = processing time for job i = (work content of job i)/R;

s_i = start time of work on job i;

f_i = finish time of work on job i (this may be a forced finish if work is incomplete when the job leaves the station);

z_i = work overload due to job i.

If we let $s_1 = 0$, we can use the following expressions to find the total work overload for a given sequence.

$$s_i = \max(i - 1, f_{i-1});$$

$$f_i = \min(s_i + p_i, i - 1 + L);$$

$$z_i = R[p_i + s_i - (i - 1 + L)]^+.$$

The expression for s_i says that processing of a job starts when it arrives at the station if the previous job has already been finished. Otherwise, processing of the job starts at the finish time of the previous job. The equation for f_i says that work on the job is terminated at the earlier of its completion time or when it leaves the station. Finally, the expression for work overload simply is the amount of work remaining to be done (if any) when the job leaves the station. (Note that the expressions above can be modified to account for jobs currently in the system.)

We can re-express the work overload as

$$z_i = R\{p_i + [s_i - (i-1)] - L\}^+$$

The expression $s_i - (i-1)$ is the delay of the <u>start</u> of job i relative to its arrival time at the station. Since $s_i = \max(i - 1, f_{i-1})$, and z_i is a function of s_i, the entire effect of the finish time of job i - 1 on all

the work overload incurred by subsequent jobs can be reflected in s_i. This lays the groundwork for sequencing procedures which consider not only total work overload but also this delay factor in finding a good solution.

3.1 Formulation

The problem of minimizing total work overload at a station is equivalent to maximizing the total work completed. First, we will consider the single-station problem. Let

$$x_{ij} = \begin{cases} 1 \text{ if job } i \text{ is assigned to the } j\text{th position} \\ 0 \text{ otherwise} \end{cases}$$

v_j = amount of time allocated to the job in the jth position

Also assume that the first job in the sequence starts at time zero. (Generalization to non-zero starting times to account for jobs already in the system is straightforward.) For a single station, the problem becomes:

$$\text{maximize} \quad \sum_j v_j$$

$$\text{s.t.} \quad \sum_j x_{ij} = 1 \qquad \forall\, i$$

$$\sum_i x_{ij} = 1 \qquad \forall\, j$$

$$v_j \le \sum_i p_i x_{ij} \qquad \forall\, i$$

$$s_j \ge j - 1 \qquad \forall\, j$$

$$s_j \ge s_{j-1} + v_{j-1} \qquad \forall\, j$$

$$s_j + v_j \le j - 1 + L \qquad \forall\, j$$

This is a mixed integer (linear) program which at first appears to be an assignment problem complicated by some side constraints. However, one must also make decisions about the v_js, which are continuous decision variables. The main difficulty of the problem lies in the fact that any assignment is feasible if one chooses $v_j = 0$ for all j. As such, the side constraints do not help to eliminate infeasible assignments; they only help to eliminate some values of v_j. Therefore, even if one chooses to use a greedy policy with respect to time allocations (which is consistent with the optimality of the non-preemptive FCFS policy for a given sequence) a brute force approach would require the evaluation of n! sequences.

An equivalent problem in job shop terminology is as follows: Consider a one-machine job shop. One job is released to the machine every time unit. The job leaves the machine L time units after its release. Any work that is performed on the job at the machine must be performed within L time units after its release. Given these constraints, find a launch sequence of the jobs which maximizes the amount of work completed.

For the multi-station problem, the equivalent job shop problem has a serial flow shop analogy in which jobs must be processed in the same sequence at all machines. The value of L and the processing times of the jobs may differ across machines.

To generalize the formulation to multiple stations, we must weight the single-station objective by the number of workers at the station and sum over all stations. Let b_k represent the number of workers at station k. The problem then is to

$$\text{Maximize} \sum_k b_k \sum_j v_{jk}$$

$$\text{s.t.} \quad \sum_j x_{ij} = 1 \qquad \forall\ i$$

$$\sum_i x_{ij} = 1 \qquad \forall\ j$$

$$v_{jk} \le \sum_i p_{ik} x_{ij} \qquad \forall\ j,k$$

$$s_{jk} \ge j - 1 \qquad \forall\ j,k$$

$$s_{jk} \ge s_{j-1,k} + v_{j-1,k} \qquad \forall\ j,k$$

$$s_{jk} + v_{jk} \le j - 1 + L_k \qquad \forall\ j,k$$

where

v_{jk} = amount of time allocated to job in position j at station k

s_{jk} = starting time of job in jth position at station k

p_{ik} = processing time of job i at station k

L_k = length of station k

Note that there are essentially different "time zones," one for each station, but one time zone is used as a reference point. It is also important to point out that the complicating constraints are separable by station, so the nature of the problem and its complexity are fundamentally the same as the single-station problem. The major difference is that one needs to find several time allocations for each job.

The single-station problem can be formulated as a dynamic program which can be solved in $O(N^2)$ time (see Appendix B). However, the size of

the state space increases exponentially with the number of job types at a station, and exponentially with the number of stations. Because of the exponential growth of the state space with the number of stations, we will focus on heuristic procedures for the multiple station problem.

In the next section, we consider a special case of the single-station problem where there are only two alternative processing times, a situation which occurs frequently in practice. This is a realistic representation of the problem when each product option requires the installation of special (sometimes additional) parts. At each relevant work station, an order having the option would have a processing time which differs from (and is normally greater than) the processing time for an order without that option. An optimal procedure is developed for this problem and this forms the basis for the proposed multi-station heuristic.

4. OPTIMAL PROCEDURE FOR THE SINGLE-STATION PROBLEM

Consider a station having a window of length L where two different sets of activities (e.g., one set for jobs having an option and one set of jobs without the option) are performed. Suppose that p units of time are required to complete jobs having the option (henceforth called optional jobs) and π units of time are required for those without (hereafter called basic jobs). Without loss of generality, we assume that $\pi < 1 < p$.

Since $p > 1$, we cannot have long sequences of optional jobs without incurring work overload. Indeed, even if the workers were to start at the point where jobs enter the window (which we call the beginning of the window), the maximum number of consecutive optional jobs which can

be completed while avoiding work overload is given by X, where X is the largest integer satisfying

$$Xp \leq X - 1 + L. \tag{1}$$

This relationship arises because Xp time units are required to complete these X jobs, and the Xth job would leave the window X - 1 + L time units after the first of these jobs enters the window.

After the work has been completed on these jobs, the workers will be near the point where jobs leave the window (which we refer to as the end of the window). If the workers continually remain near the end of the window, even a short sequence of optional jobs could cause work overload. Thus, it is desirable to bring the workers back to the beginning of the window. We call such an instance a regeneration. This can be accomplished by sequencing basic jobs after the X or fewer optional jobs. Let m_0 be the number of consecutive optional jobs in a cycle. To fully utilize the available labor, we would like to sequence m_1 basic jobs where

$$m_0 p + m_1 \pi = m_0 + m_1 \quad (m_0 \leq X,\ m_0 \text{ integer}). \tag{2}$$

Of course, there may not be an integer m_2 such that (2) is satisfied as an equality. In such a case, optimal sequence may not have alternating strings of optional and basic jobs. Later, we explain how such situations can be handled. If, however, it is desirable to have a sequence with this property, the maximum utilization (without work overload) is achieved by solving the following nonlinear integer programming problem:

$$\max \quad (pm_0 + \pi m_1) / (m_0 + m_1)$$

$$\text{s.t.} \quad m_0 \leq X$$

$$pm_0 + \pi m_1 \leq m_0 + m_1$$

$$m_0, m_1 \geq 0, \text{ integer}$$

Since the upper bound on m_0 is usually small in practice, it is possible to solve the problem by computing the utilization for each possible value of m_0. For a given value of m_0, maximum utilization is attained when we use the smallest value of m_1 satisfying

$$m_0 p + m_1 \pi \leq m_0 + m_1. \tag{3}$$

Bolat (1988) derives error bounds for the repetitive sequence in which $m_0 = X$ and m_1 is specified by (3), when equation (2) cannot be satisfied by an integer m_1.

When choosing the value of m_1 to use in actual implementation, consideration must be given to the actual option mix. If the fraction of optional jobs is much greater than $m_0/(m_0 + m_1)$, it might be advisable to incur a little (unavoidable) work overload in each cycle by choosing the largest value of m_1 satisfying

$$m_0 p + m_1 \pi \geq m_0 + m_1. \tag{4}$$

This would eliminate all idle time, which in turn, ultimately helps to reduce work overload.

The repetitive schedule of m_0 optional jobs followed by m_1 basic jobs has three attractive features. First, where special tools are involved in the operations, the sequence minimizes changeovers from one set of tools to the other, since it makes each cycle as long as possible and changeovers occur only twice during each cycle. Second, it is quite

simple to construct a schedule that follows this pattern as long as the mix of jobs permits. The balance of the schedule may have some work overload (if there are too many optional jobs), but leaving these jobs until the end of the schedule minimizes total work overload (we prove this result in Proposition 2). Finally, as long as the schedule follows the specified pattern, there will always be exactly m_0 optional jobs in any subsequence of $m_0 + m_1$ consecutive jobs, and the system regenerates after each set of basic jobs.

The second and third points provide the basis for development of an optimal sequencing procedure. We will assume throughout the remainder of this section that an integer m_1 exists satisfying (2). Later in the paper we discuss what happens when this assumption is not satisfied. The sequence is constructed by alternately scheduling m_0 optional jobs and m_1 basic jobs until the necessary positions have been filled. Then, starting at the end of the sequence, jobs are modified until the number of optional jobs is consistent with the actual problem. Thus, if there are fewer optional jobs than in a perfect repetitive sequence, there will be some idle time at the end of the day's schedule, but there will be no work overload. On the other hand, if there are more optional jobs than in a perfect repetitive sequence, there will necessarily be some work overload. We next prove that this procedure minimizes work overload for a single station.

Proposition 2: If there are n_0 optional jobs and n_1 basic jobs, where a perfect repetitive sequence has m_0 optional jobs and m_1 basic jobs in each cycle, an optimal sequence has

(a) $C = \min(\lfloor n_0/m_0 \rfloor, \lfloor n_1/m_1 \rfloor)$ cycles of m_0 optional jobs (where $\lfloor x \rfloor$ is the largest integer less than or equal to x) followed by m_1 basic jobs, followed by

(b) $\min[n_0 - Cm_0, m_0]$ optional jobs, followed by

(c) $n_1 - Cm_1$ basic jobs, followed by

(d) $[n_0 - (C + 1)m_0]^+$ optional jobs, where applicable.

Proof: See Appendix C.

Since the proposed sequencing procedure minimizes work overload at a station, it is possible to use it to derive lower bounds on total work overload for the multiple station problem. These lower bounds are used in the heuristic described in the next section.

5. HEURISTIC PROCEDURE FOR THE MULTIPLE STATION PROBLEM

In this section, we consider the situation when each job passes through multiple stations. The amount of work to be performed at each station depends upon whether or not a particular option is required. We assume that each station is affected by (at most) a single option.

The heuristic is a greedy procedure which uses a lower bound on the objective function (for the entire sequence) to sequentially determine which job should be placed in the next position. One can view the procedure as a heuristic branch and bound procedure in which branching occurs for each position in the sequence, but only the node with the lowest lower bound is retained at each level of the branch and bound tree.

Since work overload at each station is measured in time units (not person-minutes), in the lower bound for the system, the single-station objectives are weighted by the number of workers, then summed. This

function is a lower bound for two reasons. First, because each station is considered separately, the sequences giving the lower bounds at each of the stations will not necessarily coincide with one another. Second, in computing the function, the work overload due to the current assignment is computed accurately, but it is assumed that the system regenerates immediately after the job and the work overload for the remaining jobs is computed accordingly. Thus, the actual work overload for the remaining jobs generally would be larger (and cannot be any smaller).

To compute the overload due to placing the candidate job in the current position, one only needs to store the finishing time of the last job in the sequence and to use the formulas in section 3 to compute the actual work overload of the candidate job. Computing the lower bound for the balance of the sequence is also quite simple. Recall that the lower bound is derived from solutions for single station problems. When we are choosing the job in the jth position, single station solution for jobs j, ..., N can only have one of two forms: (1) there is a basic job in the jth position, leaving one mix of basic and optional jobs for positions j + 1 through N, or (2) there is an optional job in the jth position, leaving a slightly different mix of jobs for positions j + 1 through N. Each of the two possibilities has corresponding work overload which can be computed easily. To compute the lower bound for a candidate job in position j, one only needs to sum the appropriate single-station work overload values. It is clear that the computational complexity of the heuristic is $O(KN^2)$ where there are K stations and N jobs to be sequenced. For each sequence position j = 1, ..., N, N - j + 1 lower bounds must be computed for each of K stations and the

sum compared with the lowest lower bound. (A sorting procedure is not required since it is sufficient to store the index of, and lower bound for, the best candidate so far.) Finally, we note that the heuristic can be applied even if there are jobs already in the system (e.g., from the previous day), since the lower bound on the balance of the sequence (jobs j + 1, ..., N) does not depend upon initial conditions.

When (2) is not satisfied by an integer m_1, this basic approach can still be used but the lower bounds on work overload for each station must be computed by using the dynamic programming procedure described in Appendix B. We should note, however, that the values of the parameters (p, π, L) from which m_1 is computed are only approximate in practice. Thus, judicious rounding of m_1 may be adequate.

6. EXPERIMENTAL RESULTS

We obtained data for an assembly line of a major automobile company. Daily production is approximately 1,000 units and approximately 12 workstations (out of several hundred) are considered critical with regard to work overload. For each of these workstations, we obtained relevant processing times and current spacing rules, from which we etimated window lengths. For each of twenty groups of 1,000 arbitrarily selected orders (each group represents customer orders scheduled for assembly on a particular day), we generated two sequences: one using a version of the company's current procedure, and the other using the heuristic described above. The company's current procedure attempts to ensure that for each important option, the cumulative production of orders with that option is close to a specified fraction of total cumulative production. Additional details cannot be disclosed

because of confidentiality considerations. In concept, the procedure is similar to Monden's (1983), but the details differ.

A comparison of the two procedures appears in Table 1. Data are scaled to preserve confidentiality. The proposed heuristic reduces total work overload by 55 percent on average, and the difference is statistically significant at the $\alpha = .005$ level. We also report lower bounds which represent the amount of unavoidable work overload. Each lower bound is computed by finding the difference between the total available processing time at a station and the sum of the processing times of the jobs, then summing these differences over all stations. These lower bounds are very loose since they ignore the interactions among stations and assume that each station can be scheduled with no idle time.

Table 1 here

Both procedures were implemented without additional constraints related to body models (e.g., two door vs. four door) or paint colors. (The cost of paint color changeovers dictates some batching of jobs by color.) In experimentation, we have found that even with additional constraints, the new procedure significantly outperformed the existing procedure. The additional constraints are easily incorporated by confining each assignment to jobs that are feasible with respect to the additional constraints. This can, of course, lead to some infeasible assignments near the end of the sequence if a specific set of N jobs must be sequenced. In practice, this problem is resolved by permitting the procedure to sequence $N' > N$ jobs and imposing the additional constraints only on the first N positions in the sequence.

TABLE 1

WORK OVERLOAD (IN TIME UNITS) FOR THE TWO PROCEDURES

PROBLEM SET#	LOWER BOUND ON OPTIMAL SOLUTION	CURRENT PRACTICE	OUR PROCEDURE	% REDUCTION IN WORK OVERLOAD
1	76	993	423	57.4
2	127	880	355	59.4
3	43	744	222	70.2
4	215	898	392	56.3
5	792	1527	1149	24.8
6	123	871	256	70.6
7	26	659	139	78.9
8	72	931	248	73.4
9	278	1075	481	55.3
10	434	992	675	32.0
11	343	946	507	46.4
12	147	845	254	70.0
13	353	1254	701	43.7
14	129	1109	585	47.2
15	140	922	510	44.7
16	117	987	351	64.4
17	34	808	200	75.2
18	326	1044	550	47.3
19	395	1026	571	45.3
20	228	1066	521	51.1

Many of the ideas from the new heuristic have been incorporated into the company's algorithm. In particular, there has been a move toward identifying the most important options for inclusion in the algorithm on the basis of work overload concepts. Also, there has been an effort to educate production schedulers about tradeoffs among work overloads at the various stations affected by options. At the time that we began our research, there was tendency for production schedulers to insist upon "hard" spacing constraints because the existing algorithm did not adequately consider the relative importance of each option. This resulted in sequences that did not permit the appropriate tradeoffs to be made. Now, most of the unnecessary "hard" constraints have been eliminated and the remaining "hard" constraints are due to technological considerations.

We also were able to utilize the algorithm to evaluate the tradeoff between paint color changeovers and work overload. This was instrumental in moving the company toward a better (lower total cost) color batching policy.

7. CONCLUSION

In this paper we investigated the problem of sequencing jobs, each representing a combination of product options, on a paced assembly line. We have developed an optimal procedure for the situation where a single station is affected by an option. We also provided a heuristic procedure for multiple stations. The procedure was compared with an existing procedure used in industry. Results indicate that the new procedure can provide a substantial reduction of work overload. Computational time requirements for our procedure are minimal and the procedure can be implemented on microcomputers.

In addition to the implications described in the previous section, our research also has implications for the design of assembly lines. In this paper, we addressed the input sequence issue. An important consideration in the design of paced assembly lines is the station length. This is dependent on the mix of options as well as the processing times. Our procedure can help to quantify the effect of station length on work overload, which also affects quality.

In the current study, we addressed the situation when the input sequence can be maintained on the entire assembly line. However, in practice unreliable upstream processes may make this very difficult, if not impossible. A practical example of such a situation occurs in the automotive industry where painting typically precedes the assembly process. In such situations, the integrity of a predetermined input sequence is difficult to maintain because defects in the paint process necessitate rework on some jobs. However, one alternative to overcome this difficulty would be to provide buffers between the two stages for the purpose of resequencing. We are currently researching these issues.

Acknowledgement

We wish to acknowledge constructive comments made by the associate editor and the referees. We appreciate the opportunity and funding provided by one of the domestic automotive companies for investigating this problem. This research was in part supported by the Graduate School of Business at The University of Michigan, Ann Arbor, Michigan.

REFERENCES

Bird, C. G. (1986), "Sequencing Vehicles for Assembly Under Precedence Constraints," Paper presented at the ORSA/TIMS conference in Los Angeles, CA.

Bolat, A. (1988), "Generalized Mixed Model Assembly Line Sequencing Problem," Unpublished Ph.D. dissertation, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, Michigan.

Burns, L. D. and C. F. Daganzo (1985), "Assembly Line Sequencing Principles," Research Publication GMR-5127, General Motors Research Laboratories, Warren, MI.

Coffman, P. E. Jr., S. E. Hoffman, and S. A. Weiner (1985), "An O.R. View of Assembly Plant Modeling," Paper presented at the TIMS/ORSA conference in Boston, MA.

Dar-El, E. M. and R. F. Cother (1975), "Assembly Line Sequencing for Model-Mix," International Journal of Production Research 13(5), 463-477.

Dar-El, E. M. and S. Cucuy (1977), "Optimal Mixed-Model Sequencing for Balanced Assembly Lines," OMEGA 5(3), 333-342.

Monden, Y. (1983), Toyota Production System, Industrial Engineering and Management Press, Institute of Industrial Engineers, Atlanta, GA.

Okamura, K. and H. Yamashima (1979), "A Heuristic Algorithm for the Assembly Line Model-Mix Sequencing Problem to Minimize the Risk of Stopping the Conveyor," International Journal of Production Research 17(3), 233-247.

Parrello, B. (1988), "Car Wars: The (Almost) Birth of an Expert System," AI Expert, 60-64.

Schonberger, Richard J., Japanese Manufacturing Techniques: Nine Hidden Lessons in Simplicity, The Free Press, New York, NY, 1982.

Thomopoulos, N. T. (1967), "Line Balancing-Sequencing for Mixed Model Assembly," Management Science 14(2), 59-75.

Weiner, S. (1985), "Perspectives on Automotive Manufacturing," in The Management of Productivity and Technology in Manufacturing, edited by Paul R. Kleindorfer, Plenum Press, New York, 57-71.

Wester, L. and M. Kilbridge (1964), "The Assembly Line Mixed Model Sequencing Problem," Proceedings, Third International Conference on Operations Research 1963: Dunod, 1964.

Yano, C. A. and R. Rachamadugu (1987), "Sequencing to Minimize Work Overload in Assembly Lines with Product Options," Technical Report #87-22, Department of Industrial and Operations Engineering, School of Engineering, The University of Michigan, Ann Arbor, MI 48109.

APPENDIX A

Proposition: For any given input sequence, a non-preemptive FCFS work schedule provides an optimal solution.

Proof: We show that an optimal work schedule which does not satisfy the above characteristics can be modified into a schedule that does, without increasing work overload. We assume that the worker(s) work on one job at a time. As long as the work accomplished is proportional to the time spent, this is an optimal policy.

The proof is by recursion. We assume that we begin with an optimal schedule. We show that the schedules for the first job in the sequence, J_1, and some other job, J_k, can be modified in such a way that J_1 is processed continuously, and so that it is either completed or leaves the station before work commences on the second job. Then J_1 satisfies the above characterization. Since the same arguments can be applied to jobs 2,..., N in sequence, the final schedule will satisfy the given characterization.

Throughout the proof, J_k is assumed to be a job whose sojourn in the station overlaps that of J_1 (i.e., there is some time interval when both jobs are in the station simultaneously). Also, let us define:

r_i = release time of job i,

l_i = departure time of job i from the station,

f_i = (latest) time at which work on job i terminates.

The proof is divided into two parts. We first consider the case in which the required work on J_1 is not completed. The second case pertains to situations in which the required work is completed.

Case 1: All the required work on J_1 is completed.

Suppose J_1 is preempted from t to t + Δt. (If there is more than one preemption, the arguments below can be applied recursively to the last preemption until all preemptions are eliminated.) If the worker was idle during (t, t + Δt), the schedule can be modified so that the processing occurs continuously from t to $f_1 - \Delta t$. The work overload remains unchanged.

If, on the other hand, the worker was processing J_k during (t, t + Δt), the schedule can be modified so that J_1 is processed continuously from t until $f_1 - \Delta t$, and work is performed on J_k during $(f_1 - \Delta t, f_1)$. Observe that this is feasible since $f_1 \leq l_1 \leq l_k$. (Release times and departure times of jobs are, by definition of the problem, monotonic in time). Again, the work overload remains unchanged.

Case 2: All work required by J_1 was not completed.

For this case, we assume that any preemptions of the type described in Case 1 have already been eliminated. Thus, Case 2 pertains to situations in which processing of J_1 is stopped prior to its completion in order to begin processing of another job.

Suppose that the amount of incomplete work on J_1 is U, where $U \leq l_1 - f_1$, and that work on J_1 was terminated to work on another job J_k (or another set of jobs). Obviously, the worker could not have been idle during (f_1, l_1) since this violates our assumption that the original schedule was optimal. The schedule can be modified so that J_1 rather than J_k (or the set of jobs) is processed during $(f_1, f_1 + U)$. Observe that after the change, J_1 is processed continuously. Moreover, the total work overload remains the same.

Now consider the case in which $U > l_1 - f_1$. Using similar arguments to those above, it can be shown that the schedule can be modified so that J_1 is processed continuously until l_1, and the new schedule remains optimal.

The schedule modifications described in Cases 1 and 2 transform the optimal schedule such that J_1 satisfies our characterization. The same types of modifications can now be applied recursively to jobs 2,..., N. Thus, any optimal schedule can be modified into another which is non-preemptive, FCFS without increasing the work overload.

APPENDIX B

The single-station problem with two types of jobs can be formulated as a dynamic program which can be solved in $O(N^2)$ time. In this formulation, we assume that the schedule is non-preemptive, but for simplicity, we relax the assumption that as much work as possible is performed on a job before starting the subsequent job (FCFS assumption). Thus, the procedure may provide an alternative optimal solution which is not FCFS. However, given the solution, it is possible to construct a FCFS schedule from it. The procedure can be applied even if the value of m_2 satisfying (2) is not integer.

Without loss of generality, we assume that there is some time duration τ such that 1, L, p and π are all integer multiples of τ. Let τ be the duration of the basic time period in the problem. Thus, every relevant time duration can be expressed as an integer multiple of this basic time unit. We define a time interval as a set of consecutive time periods, which is specified by its starting time and duration.

In this dynamic programming formulation, we use backward recursion and start with the last position in the sequence. (For clarity, time is measured forward from time zero, not backward from the end of the horizon.) We use the term *schedule* to denote a feasible allocation of time intervals to positions in the sequence. A schedule does not, however, specify which job is assigned to each position. Our reason for distinguishing these two sets of decisions will become evident as we develop the dynamic programming formulation. We restrict our attention to schedules in which the total available time is allocated. If it is not possible to assign jobs to positions in the sequence in such a way that the allocated times are fully utilized, there will be idle time.

Our objective is to minimize total idle time, and since the total available time is constant, this approach implicitly maximizes the total work accomplished.

The dynamic programming procedure simultaneously constructs a schedule and assigns jobs to positions in the sequence. It is based upon the observation that at any point in time at which a job can commence processing, all decision alternatives can be captured in the following three parameters:

(i) the number of jobs sequenced (which can be reflected in the stage variable),

(ii) the number of optional jobs sequenced, and

(iii) the candidate starting time of the interval for the position under consideration.

For each state (reflected by (ii) and (iii)) within each stage, we must decide the best duration and job type. A formal statement of the dynamic programming formulation follows. Let

n_o = number of optional jobs to be scheduled,

n_1 = number of basic jobs to be scheduled,

N = total number of jobs ($= n_o + n_1$),

k = stage index

= number of positions scheduled

= number of jobs sequenced,

s_k = (candidate) starting time of the interval allocated to position $N - k + 1$ (i.e., the kth position considered),

n_k = number of optional jobs sequenced at stage k,

d_k = duration of interval k (decision variable), $1/\tau \leq d_k \leq p/\tau$,

$D_k(s_k)$ = set of feasible durations at stage k if starting time is s_k,

$= \{d_k \mid s_k + d_k = s_{k-1} \geq (N - k + 1)/\tau$ and $s_k + d_k \leq (N + L - k)/\tau\}$

$$\delta_k = \begin{cases} 1 \text{ if an optional job is selected for position } N - k + 1 \\ 0 \text{ otherwise,} \end{cases}$$

$f_k(s_k, n_k, d_k, \delta_k)$ = idle time in a solution starting at s_k with n_k optional jobs sequenced if duration d_k and job type δ_k are selected at stage k,

$$f_k^*(s_k, n_k) = \min_{\delta_k;\, d_k \in D_k(s_k)} f_k(s_k, n_k, d_k, \delta_k).$$

The dynamic programming recursion equation is:

$$f_k(s_k, n_k, d_k, \delta_k) = [d_k - \pi(1 - \delta_k) - p\delta_k]^+ + f_{k-1}^*(s_k + d_k,\ n_k - \delta_k).$$

The first term is the idle time for the job assigned at stage k and the second term is the optimal value function for the remainder of the horizon. The boundary conditions are:

$$f_0(\cdot) = 0$$

and $s_N = 0$.

Since the total available time must be allocated, there are additional constraints at stage 1:

$$s_1 + d_1 = (N + L - 1)/\tau,$$

where the right hand side of the above expression is the time of departure of the last job from the station. These constraints can be considered within the dynamic programming procedure.

The optimal solution is:

$$\min_{n \leq n_o} f_N^*(0, n).$$

Note that any solution in which fewer than n_o optional jobs are assigned can be modified by replacing each excess basic job with an optional job. The resulting solution is now feasible, and the total idle time remains unchanged. Although $f_N^*(0,n)$ is non-increasing in n, there may be solutions with $n < n_o$ in which there is zero idle time. In this case, it is not necessary to compute the solutions for larger values of n, and the computation time can be reduced accordingly. For a numerical example, see Yano and Rachamadugu [1987].

For each (s_k, n_k) pair we need to evaluate several alternatives. There are at most NL/τ pairs (number of optional jobs sequenced multiplied by number of possible starting times for a stage). The maximum number of alternatives to be considered for each pair is $2p/\tau$ (number of durations multiplied by number of job types). Since there are N stages, the computational complexity of the procedure is $0(2LpN^2/\tau^2)$.

The procedure can be generalized to the case of more than two choices at a station, but the size of the state space increases exponentially with the number of choices. The procedure also can be generalized to the case of multiple stations, but in this case the number of choices is equal to the product of the number of types at the various stations. Thus, even if there are only two choices at each station, the total number of job types (and consequently the size of the state space and the number of decision alternatives) increases quickly with the number of stations.

APPENDIX C

Proof of Proposition 2.

Proposition: If there are n_0 optional jobs and n_1 basic jobs, where a perfect repetitive sequence has m_0 optional jobs and m_1 basic jobs in each cycle, an optimal sequence has

(a) $C = \min(\lfloor n_0/m_0 \rfloor, \lfloor n_1/m_1 \rfloor)$ cycles of m_1 optional jobs (where $\lfloor x \rfloor$ is the largest integer less than or equal to x) followed by m_2 basic jobs, followed by

(b) $\min[n_0 - Cm_0, m_0]$ optional jobs, followed by

(c) $n_1 - Cm_1$ basic jobs, followed by

(d) $[n_0 - (C + 1)m_0]^+$ optional jobs, where applicable.

Proof: The first C cycles have no work overload and no idle time. Moreover, a regeneration occurs at the end of this subsequence, so it is not possible to make any modifications to this portion of the sequence to reduce the possibility of work overload in the remainder of the sequence.

Observe that after the first C cycles, the set of remaining jobs must fall into one of the three categories below:

(1) there are fewer than m_0 optional jobs and m_1 or more basic jobs;

(2) there are fewer than m_0 optional jobs and fewer than m_1 basic jobs; or

(3) there are fewer than m_1 basic jobs and greater than or equal to m_0 optional jobs.

If one of the three conditions is not satisfied, it would be possible to construct another cycle, which cannot be true by the definition of C. Plots of the position of the worker in the window for these three

mutually exclusive and collectively exhaustive scenarios are shown in Figures 1, 2 and 3.

Figures 1, 2 and 3 here

If either case (1) or (2) above occurs, there are fewer than m_1 optional jobs, and a sequence with no work overload can be constructed by scheduling the remaining optional jobs followed by the remaining basic jobs.

We now consider case 3. We will show that there is no idle time, which clearly minimizes work overload. There can be no idle time if the system never regenerates. Consider the location of the workers during their execution of these jobs (see Figure 3). While they are working on the initial m_1 optional jobs, they are moving away from the beginning of the window and upon completion, are near the end of the window. They then proceed to work on the basic jobs. However, since there are fewer than m_2 basic jobs available, they are unable to return to the beginning of the window. For the remainder of the schedule, they work on optional jobs which progressively move them toward the end of the window. Thus, the system never regenerates within this portion of the schedule, and there is no idle time. Consequently, the proposed sequence minimizes work overload. ■

Formal algebraic proofs of these results are omitted here for the sake of brevity. The interested reader is directed to Yano and Rachamadugu [1987].

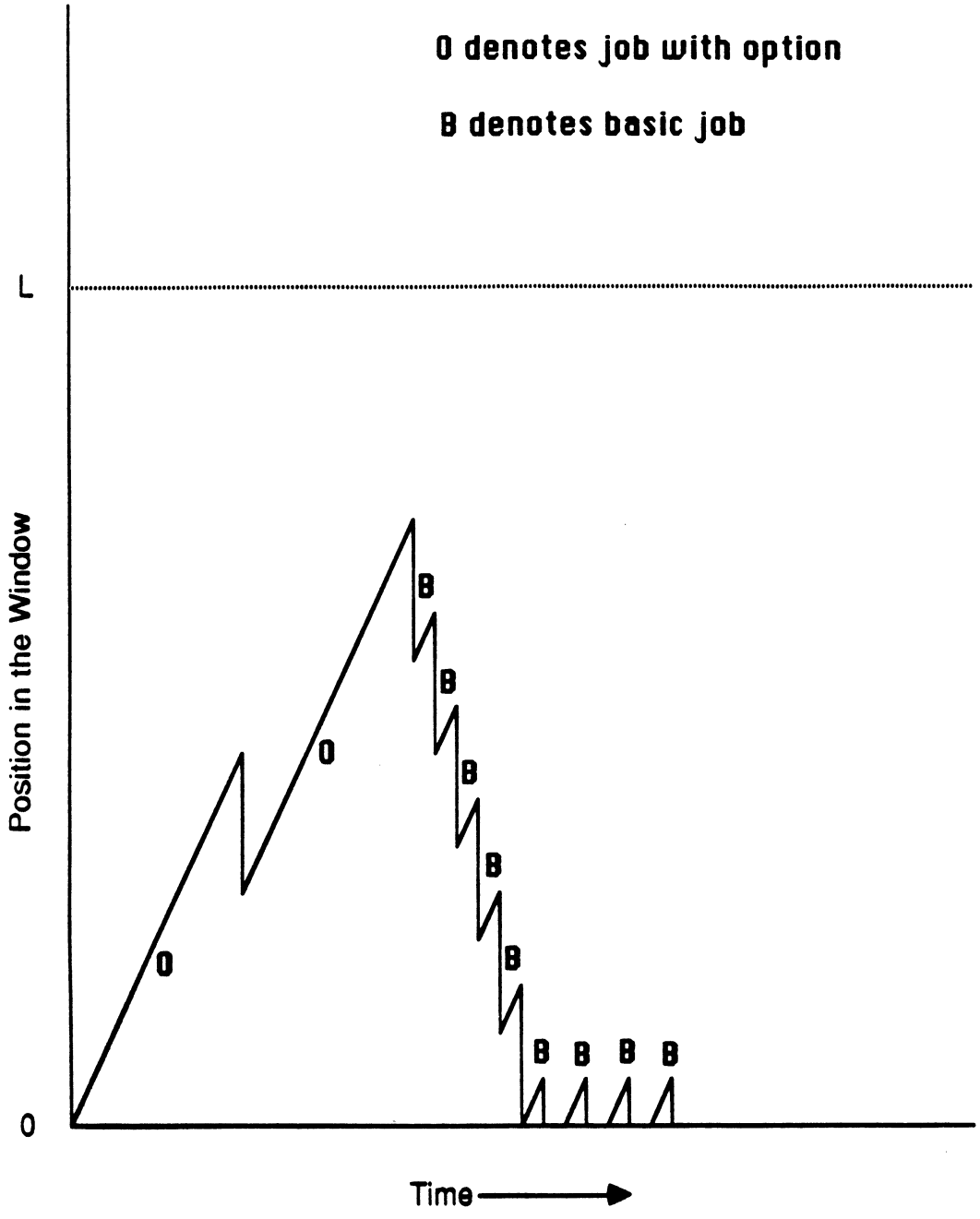

Figure 1

Worker Position Diagram for Case 1

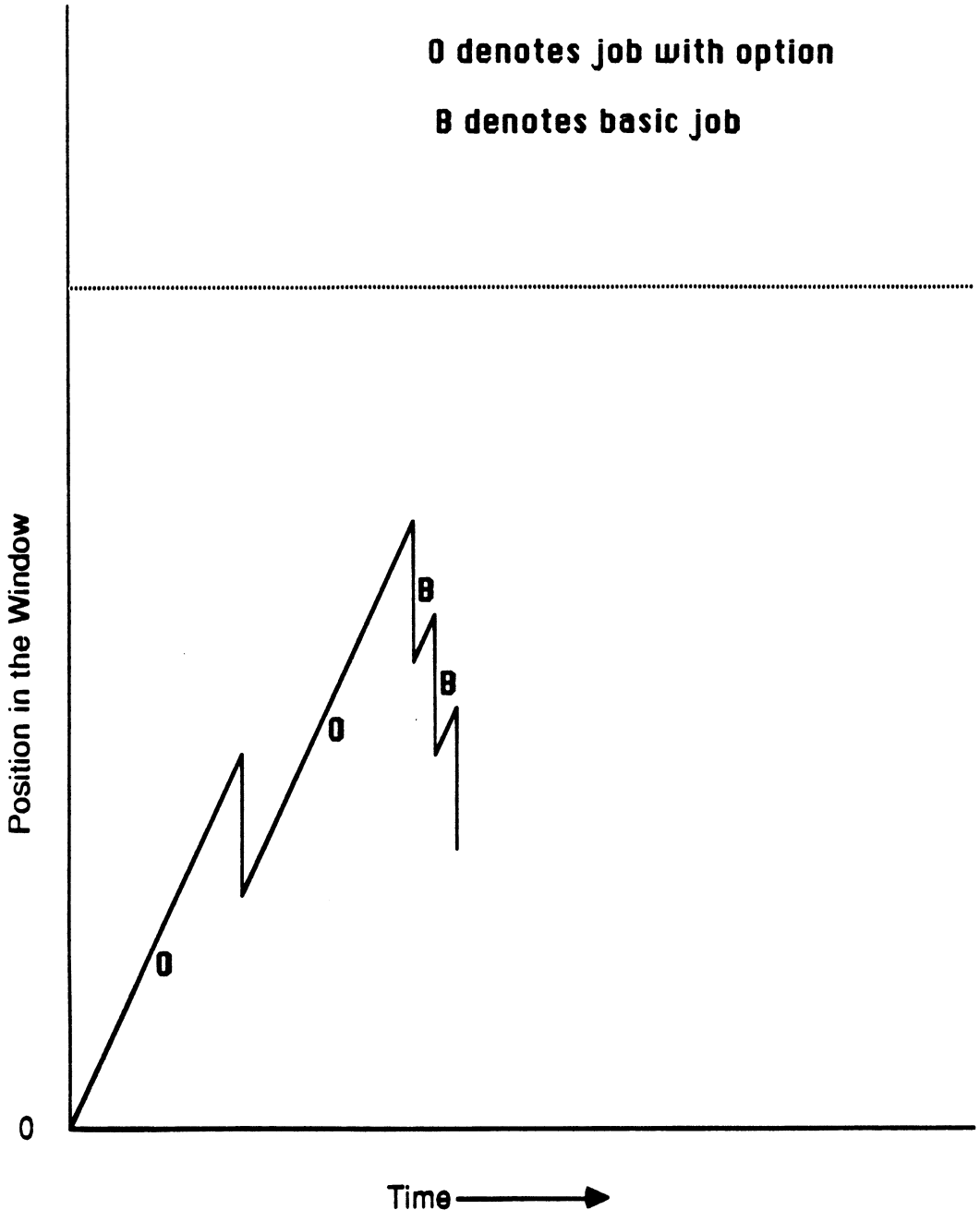

Figure 2

Worker Position Diagram for Case 2

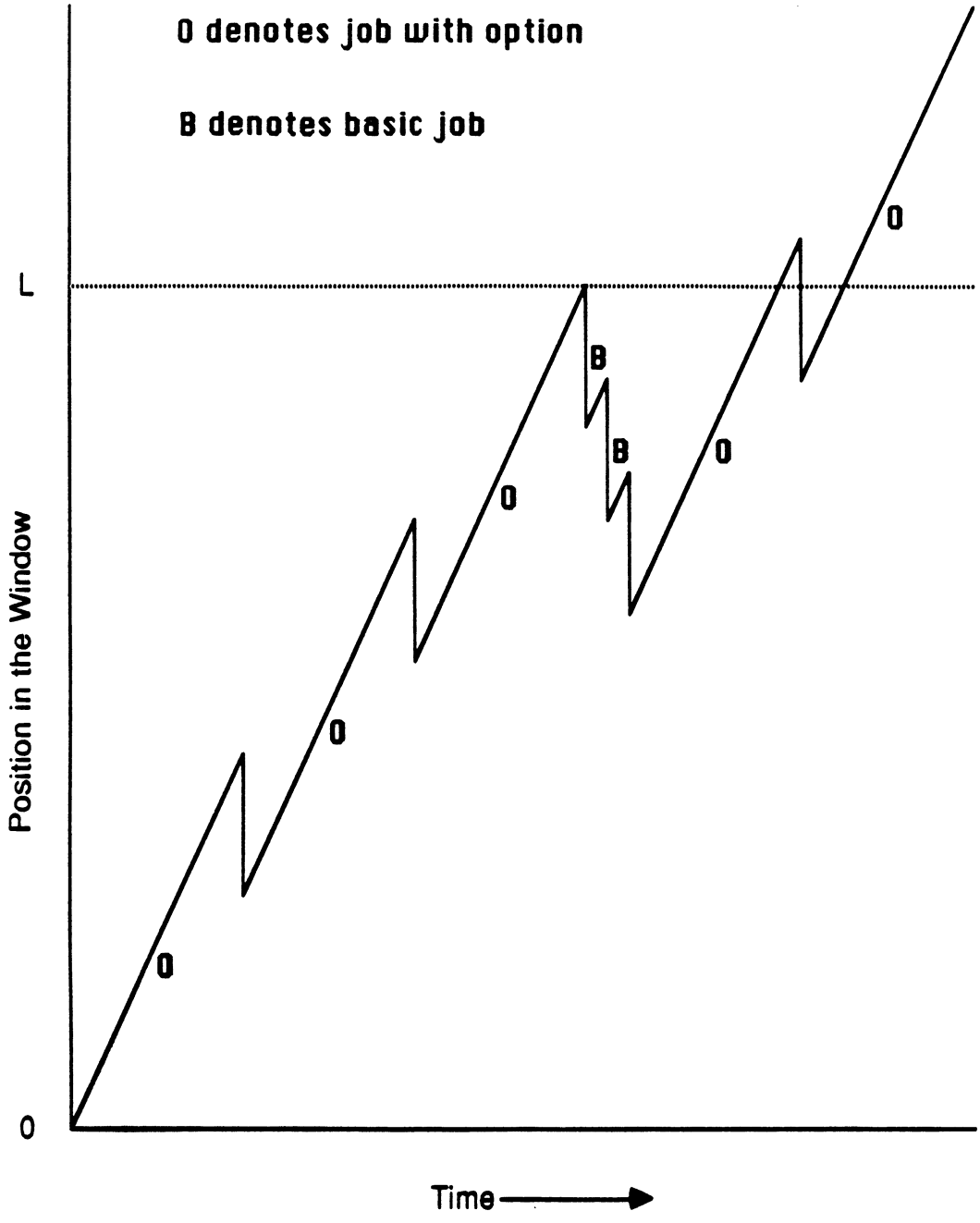

Figure 3

Worker Position Diagram for Case 3