SCHEDULING ALGORITHMS TO MINIMIZE UTILITY WORK AT A SINGLE STATION ON A PACED ASSEMBLY LINE

Ahmet Bolat
Mechanical Engineering Department
King Saud University
Riyadh, Saudi Arabia

Candace Arai Yano
Department of Industrial & Operations Engineering
University of Michigan
Ann Arbor, MI 48109-2117

Technical Report 88-7

Revised October 1991

SCHEDULING ALGORITHMS TO MINIMIZE UTILITY WORK AT A SINGLE STATION ON A PACED ASSEMBLY LINE

Abstract

In this article we address the problem of sequencing jobs for one station which offers two types of operations on a paced assembly line with no buffers. If a subsequence of jobs requires more work than the station can handle, some amount of work, which we call utility work, will remain undone. We develop optimal solution procedures for three of four mutually exclusive and collectively exhaustive problem subclasses, with the goal of minimizing total utility work. For the fourth subclass, we evaluate heuristics that are structurally similar to the optimal procedures for the other subclasses. We provide worst-case error bounds for one of these procedures. Computational results indicate that very good, and often optimal, results can be obtained with a combination of these procedures.

1 Introduction

Paced assembly lines with constant-speed conveyors are often used to produce different models of the same general product. In many of these applications, each model is characterized by a set of customer-selected options, and there may be millions of possible models. For example, Weiner (1985) indicates that the Ford Escort has over 2.5 million possible configurations. When the demand for each distinct model is small and delivery deadlines are short, the various models usually are intermixed on the assembly line rather than being produced in batches.

For such facilities, the assignment of tasks to stations (i.e., assembly line balancing) and the capacity of each station on the assembly line (physical length of the station and number of assembly workers) are difficult decisions. Normally they are established on the basis of the expected mix of options so that, on average, the work required by each job (an order for the finished product) at a particular station can be completed within the boundaries of the station. Since the mix of options may fluctuate from hour to hour, the stations affected by options are designed and staffed to provide excess capacity. This excess capacity (labor and equipment) is expensive, however, so facilities of this type rely heavily upon sequencing procedures to smooth out the flow of work to each station. This helps to reduce the amount of excess capacity required.

One common approach to smoothing out the flow of work is to apply **spacing rules**, which we will define shortly. In such an approach, it is assumed that each station can perform one of two sets of tasks on each job. For ease of presentation, we will refer to each set of tasks as an operation, and to one operation as **basic** and the other as **optional**. Without loss of generality, it is assumed that the optional operation takes longer to perform than the basic operation. As an example, assembly of the drive mechanism on a non-self-propelled lawn mower might be called basic, while assembly of a similar mechanism on a self-propelled

mower might be called optional. In other instances, the distinction may be the presence or absence of a particular customer-selected option. Spacing rules normally state that: (i) at most k out of each l consecutive jobs can have the option; and/or (ii) no more than a specified number of jobs with the option can be sequenced consecutively. Spacing rules of this type have been used by, or are implicit in the work of, Thomopoulos (1967), Monden (1983), Coffman et al. (1985), and Burns and Daganzo (1985), among others.

In the papers cited above, satisfaction of the spacing rules is regarded as a sufficient condition to ensure that all work on the jobs is completed, that the work is performed at the normal pace (i.e., not hurriedly), and that it is performed within reasonable limits of the appropriate work station. All of these factors contribute to maintaining product quality. However, there is no universally accepted view of how one should penalize violations of the spacing rules when it is not possible to satisfy all of them simultaneously throughout the entire sequence. For example, Burns and Daganzo try to minimize the percentage of jobs "exceeding capacity," that is, the percentage of jobs violating a spacing constraint. On the other hand, in a shorter-term resequencing context, Coffman et al. have an implied objective of minimizing the number of constraint violations. Thomopoulos considers several different "inefficiencies" that may result when spacing rules are violated, and uses an objective which is a linear function of these inefficiencies. Implicit in this objective are penalties that increase with both the number and extent of spacing rule violations. Nonlinear objectives such as the sum of squared deviations from the desired spacing have been used, but with somewhat less success than the simpler objectives mentioned above (e.g., see Parrello 1988).

In this article, we consider a measure of performance known as **utility work** and develop related sequencing procedures for a single station. Utility work can be defined as the amount of work (measured in processing time) that would remain undone if the workers proceed at their normal pace and remain within the boundaries of their respective stations. This term historically has been used in circumstances where a utility (or floating) worker is assigned to

finish incomplete operations, either by assisting the operators at the station or by completing it elsewhere. In practice, if utility workers are not used for this purpose, the operators must perform the work more quickly or perform part of the work outside the station. The former contributes to quality degradation, while the latter results in reduced productivity due to excess travel time and interference between operators in adjacent stations.

Although utility work can be expressed as a linear function, the sequence that minimizes utility work and that which minimizes the sum of squared deviations, tend to be very similar if the system is loaded below, at, or slightly above capacity. In highly overloaded systems, the quadratic objective tends to provide a smoother flow of work by discriminating among sequences with the same total amount of utility work. If a system is heavily overloaded, however, the primary problem is not one of sequencing. Thus, we believe that utility work is an adequate measure of performance in most realistic situations.

The concept of utility work was first introduced by Thomopoulos (1967). Okamura and Yamashina (1979) develop a procedure to minimize the maximum utility work (over all jobs) at a single station. Yano and Rachamadugu (1991) develop an optimal $O(N^2)$ dynamic programming (DP) algorithm, where N is the number of jobs, to minimize total utility work at a single station for the case of two types of jobs, but do not report computation times. Their approach is described in more detail later.

We develop a set of sequencing procedures for the goal of minimizing total utility work for a single station. Two of the procedures are based upon spacing rule concepts and are optimal when the systems is sufficiently overloaded or underloaded. For intermediate load levels, we develop another procedure which is *not* based on spacing rules, and is optimal under certain conditions on the parameter values. When these conditions are not satisfied, the optimal solution can be obtained by solving a DP which might be computationally tractable for the problem at hand. On the other hand, our primary motivation for solving this problem is to use it as the basis for multiple-station problems with a large number of jobs (e.g., 1000).

(See Yano and Bolat 1990 for related work.) For this reason, we also wanted to evaluate the viability of spacing rule-based sequencing procedures and other heuristics as substitutes for DPs. In a computational study, we compare the various heuristic procedures with optimal solutions.

A description and formulation of the problem are given in Section 2. We discuss regenerative sequencing procedures and their connection to spacing rules in Section 3. We also show that spacing-rule-based procedures are optimal for two problem subclasses: heavily overloaded systems and sufficiently underloaded systems. In Section 4, we present a more complicated procedure that is optimal for problems with intermediate loads and certain processing time characteristics. For problems with intermediate loads that do not satisfy these processing time properties, we propose heuristic procedures that are structurally similar to the aforementioned optimal procedures. Section 5 contains worst-case error bounds for the spacing-rule-based procedure, and a comparison of the heuristics with the DP in terms of solution quality and computation time. We conclude with a summary and discussion in Section 6.

2 Description and Formulation of the Problem

We assume that there are N jobs to be scheduled at a single assembly station that offers two types of operations. Jobs arrive to the station on a conveyor that moves at a constant rate, and time is scaled so that a new job is introduced to the station at intervals of one time unit. Jobs cannot be removed from their respective positions on the conveyor. We use the term **slot** to refer to a position in the sequence.

We assume that all the operators at the station work on the same job simultaneously. The station is closed to the right and left, so work can be performed only within the limits of the station. The operators move downstream on the line, performing their respective tasks,

then return upstream to meet the next job. We assume that the travel (walking) time to the next job is negligible, which although not exact, is a fairly accurate approximation when the travel time is far less than the processing time at a station, or when allowances for travel are included in the estimated processing time.

The problem is to determine a schedule for the N jobs that minimizes total utility work. Although a sequence specifies the arrival and departure times of the jobs, it does not determine the entire production schedule because it does not specify which job is worked on at each point in time. However, with the objective of minimizing total utility work, it is obvious that non-preemptive, first-come-first-served schedules constitute a dominant set. (This is proved formally in Yano and Rachamadugu 1991.) In this context, first-come-first-served means that the workers attempt to complete the job in slot h before starting work on the job in slot $h+1$. We use this fact in the following formulation. Let

R = number of jobs requiring the optional operation,

o = time required to complete an optional operation,

b = time required to complete a basic operation,

L = length of the station expressed in terms of number of jobs,

= sojourn time of a job in the station,

S_h = starting time for processing of the job in position h,

F_h = work termination time of the job in position h,

U_h = utility work for the job in position h,

$$X_h = \begin{cases} 1 & \text{if a job with the optional operation is assigned to slot } h, \\ 0 & \text{otherwise.} \end{cases}$$

If work terminates on the job in position h (due to its departure from the station) before it can be completed, then some utility work is required for this job. Thus, the utility work for this job is $U_h = (S_h + oX_h + b(1 - X_h) - F_h)^+$, and the objective is to minimize the sum

of the U_hs. We can eliminate the positive part in the objective function by imposing the constraints $F_h \leq S_h + oX_h + b(1 - X_h)$. The problem can be formulated as:

$$
\begin{array}{lll}
\text{Minimize} & \sum_{h=1}^{N}[S_h + oX_h + b(1 - X_h) - F_h] & \\
\text{Subject to:} & S_h \geq h - 1 & h = 1, \ldots, N \\
& S_h \geq F_{h-1} & h = 2, \ldots, N \\
& F_h \leq S_h + oX_h + b(1 - X_h) & h = 1, \ldots, N \\
& F_h \leq h - 1 + L & h = 1, \ldots, N \\
& \sum_{h=1}^{N} X_h = R & \\
& X_h = 0 \text{ or } 1 & h = 1, \ldots, N \\
& S_h, F_h \geq 0 & h = 1, \ldots, N
\end{array}
\tag{1}
$$

The constraints can be explained as follows. The starting time of the job in slot h must be greater than or equal to both its arrival time and the termination time of previous job because the station is left closed and only one job can be processed at a time. The termination of processing on the job in slot h is $F_h = \min\,(S_h + oX_h + b(1 - X_h),\ h - 1 + L)$, that is, the earlier of the completion time and the departure time of the job, because the station is right closed and only one job can be processed at a time. The number of jobs with the optional operation must equal R.

Yano and Rachamadugu (1991) present a backward dynamic programming formulation of this problem in which the stage is the number of jobs scheduled and the state is defined by the number of jobs with the optional operation scheduled thus far and the starting time of the first (i.e., smallest sequence position) job in the partial sequence. The decisions to be made are the type of job to assign and its processing time allocation. In their formulation, time is discretized into units much smaller than the cycle time. The computational complexity of the algorithm although $O(N^2)$, has a proportionality constant which is the product of

three terms: the square of the inverse of this time unit, the station length (L), and the maximum processing time (o). (The complexity is also reported to increase exponentially with the number of job types and the number of stations.) Consequently, it is not clear whether an approach of this type is computationally tractable for large problems. In our computational study, we actually use an equivalent forward dynamic programming procedure which is described in Appendix 1.

In the next section, we develop a simpler sequencing procedure which is based upon regeneration concepts, and show how it is related to spacing rules. We then show that two variations of the procedure are optimal for sufficiently under- and overloaded systems.

3 Regenerative Sequencing Procedure

In this section we develop a sequencing procedure which is based on the concept of regeneration. If this approach is optimal or near-optimal, we would have a good, easy-to-implement procedure. Furthermore, as we discuss later, it can provide guidance on how to choose parameters for spacing rules.

Before discussing the role of regeneration in the sequencing procedure, it is useful to consider the actual movements of the operator within the boundaries of the station. Let us define the origin of the station as the point where jobs enter the station, and without loss of generality, let us assume that it is the leftmost point of the station. Consider a line segment between the left and right boundaries of the station. In the system under consideration, we can define the state of the system as the distance from the origin (left boundary) of the operator on this line segment. We take this one step further and use a two dimensional diagram, which we call an operator movement diagram, to plot the location of the operator over time.

Figure 1 illustrates an operator movement diagram for the sequence O,O,O,O,B,B,B

where O and B stand for jobs requiring the optional and basic operation, respectively, $o = 2.00$ time units, $b = 0.25$ time units, and $L = 4.00$ slots. The abscissa represents the distance from the origin of the origin and the ordinate, the elapsed time. The notation z_i and q_i represent the work-starting point and the work-completion point, respectively, of job i.

Figure 1

If there is a time interval during which no operator movement occurs, the operator is idle. Clearly this can happen only at the origin, since for any other state of the system (i.e., location of the operator), there would be work remaining to be completed. If, on the other hand, the operator reaches the right boundary of the station without completing a job, some utility work is required for that job. Observe that there is neither utility work nor idle time for jobs 1 through 7 in Figure 1.

Let us now assume that at some time t, no further work is required by jobs at the station. The first job that enters after time t finds the operators waiting idle at the origin, just as if it were entering at time 0. Hence, we say that the station has regenerated at time t. Notice that the station regenerates at the completion of the $7th$ job in Figure 1. Now we will generalize these concepts to develop a regenerative sequencing pattern.

We would like to construct a subsequence that (1) has neither idle time nor utility work; and (2) regenerates at the end of this subsequence if it begins at a regeneration point. Such a subsequence would fully utilize the available labor without incurring utility work, and could be repeated with exactly the same results in each cycle. Moreover, since the system would regenerate at the end of the last complete cycle, the computation of utility work for the remaining jobs is simplified. We discuss this point in more detail later.

Without loss of generality we can assume that the staffing level and the length of the station have been adjusted so that $b < 1$, $o > 1$ and $o \leq L$. (Notice that if the first inequality

is not true, all jobs require utility work because the operators have only one time unit, on average, to work on each job. If the second inequality is not true, there is idle time for all jobs. If the last inequality is not true then O jobs can never be completed within the station regardless of their position in the sequence.) Observe that after completing an O job, the operators are $o-1$ slots farther from the origin than the work-starting point of that job and after completing a B job they are $1-b$ unit slots closer to the origin. Therefore, if we start at a regeneration point, a limited number (k) of O jobs can be sequenced consecutively without incurring utility work. Moreover, starting at the right boundary, a limited number (m) of B jobs can be sequenced consecutively without creating idle time. We can therefore define

$$k = \frac{L-1}{o-1} \tag{2}$$

and

$$m = \frac{L-1}{1-b}. \tag{3}$$

If k and m are integral, a subsequence with k O jobs followed by m B jobs will have zero utility work, zero idle time, and will regenerate after the $(k+m)$th job. Let l be the number of jobs in each repetitive subsequence, G be the number of repetitive subsequences and r be the number of slots left over at the end of the G subsequences. We therefore have $l = k+m$, $G = \lfloor \frac{N}{l} \rfloor$ and $r = N - Gl$. Let H be the total number of O jobs and define $H^* = Gk + min(\, k,\, r)$, which is the maximum number of O jobs for which the repetitive sequencing pattern can be constructed exactly as stated above. (Note that the last cycle may contain fewer than l jobs.) If $H \leq H^*$, it is obvious that this repetitive pattern can be used as a basis for producing an optimal solution. However, for any value of H, the following algorithm produces an optimal solution which has at least G regeneration cycles, each with a duration of l or fewer time units. We first present the algorithm, and then prove its optimality.

ALGSEQ: Given that there exist integral k and m satisfying (2) and (3),

1. Fill the first k slots of the first available l consecutive slots with O jobs (leaving the remaining $l-k$ slots unassigned) and repeat this as long as there are O jobs and the slot index is less than or equal to $Gl-m$.

2. If there are O jobs remaining, go to step 3. Otherwise go to step 5.

3. (a) If $r < k$ then place the remaining O jobs into empty slots in cycle G starting from the end of the cycle. If all m slots are filled and there are still O jobs remaining then go to step 4.

   (b) If $r > k$ then place the remaining O jobs into empty slots of partial cycle $G+1$ starting from the end. If all $r-k$ slots are filled and there are still O jobs remaining then go to step 4.

   (c) If $r = k$ go to step 4.

4. Distribute the remaining O jobs arbitrarily over the available slots and go to step 5. (In some instances, it may be desirable to spread these jobs as evenly as possible. On the other hand, conceptually, it is useful to consider placing these jobs in the latest available slots. In this case, the early cycles regenerate and any utility work is concentrated at the end of the sequence, making it easier to compute total utility work.)

5. Fill the remaining slots with B jobs and terminate.

Proposition 1: If there are integral k and m satisfying equations (2) and (3), then ALGSEQ produces an optimal solution.

Proof: We demonstrate that the sequence is optimal by deriving a lower bound on total utility work and showing that it is possible to construct a time schedule (for the optimal

sequence) that has total utility work equal to the lower bound. The proof is sketched here; see Bolat (1988) for further details.

In any schedule, *total required work plus total idle time must be equal to total available time plus total utility work.* Total work required at the station is given by $Ho + (N - H)b$ and total available time is $N + L - 1$ (the Nth job leaves at time $N + L - 1$). Therefore, for any schedule a lower bound on utility work is equal to $[Ho + (N - H)b - (N + L - 1)]^+$.

Let $H' = H - H^*$ be the number of excess O jobs that are distributed in step 4. Define $H^e = \lfloor (k - r)\frac{m}{l} \rfloor$ and $H^f = \lfloor (r - k)\frac{k}{l} \rfloor$, which are values that help us to determine whether the final full and fractional cycles have utility work. Finally, define U_i as the utility work incurred by the job in the *ith* position in the sequence. By assuming preemption when necessary to force a regeneration every $k + m$ jobs, Bolat (1988) has shown that for the sequence created by ALGSEQ, one can construct a feasible time schedule with the following total utility work:

$$\sum_{i=1}^{N} U_i = \begin{cases} 0 & \text{if } H \leq H^* \\ H'(o-b) & \text{if } H > H^*,\ r = k \\ 0 & \text{if } H > H^*,\ r < k,\ H^e \geq H' \\ H'(o-b) - (k-r)(o-1) & \text{if } H > H^*,\ r < k,\ H^e < H' < m \\ H'(o-b) - m(1-b) + r(o-1) & \text{if } H > H^*,\ r < k,\ H' \geq m \\ 0 & \text{if } H > H^*,\ r > k,\ H^f \geq H' \\ H'(o-b) - (r-k)(1-b) & \text{if } H > H^*,\ r > k,\ H^f < H' \end{cases} \tag{4}$$

Bolat (1988) has also proved that this total utility work is equal to its lower bound and hence the schedule is optimum. (It is important to mention that there may be other optimal solutions that are not constructed on a basis of a repetitive pattern or do not regenerate at all.) ∎

The optimality of ALGSEQ depends on the existence of integral k and m satisfying

equations (2) and (3). If such integral k and m do not exist, then it is still possible to use such repetitive patterns to obtain optimal solutions in certain circumstances. Let us redefine the spacing parameters as the largest value of k for which there is no utility work and the smallest value of m for which there is a regeneration and no utility work, i.e.,

$$k = \lfloor \frac{L-1}{o-1} \rfloor \tag{5}$$

and

$$m = \lceil \frac{k(o-1)}{1-b} \rceil. \tag{6}$$

We have the following result.

Proposition 2: With $k = \lfloor \frac{L-1}{o-1} \rfloor$ and $m = \lceil \frac{k(o-1)}{1-b} \rceil$, ALGSEQ produces a sequence with no utility work if the fraction of jobs with the option is less than or equal to $[Gk + min(r,k)]/N$.

Proof: The term in brackets is the maximum number of jobs with the option in a sequence using the repeating pattern of k jobs with the option followed by m basic jobs. The result follows immediately. ∎

Similarly, we have the following result.

Proposition 3: With spacing parameters $k = \lfloor \frac{L-1}{o-1} \rfloor$ and $m' = m - 1 = \lceil \frac{k(o-1)}{1-b} \rceil - 1$, ALGSEQ produces a sequence with no idle time, and is therefore optimal, if the fraction of jobs with the option is greater than $k/(k+m')$ and $(H-kD)(o-1)+[(N-H-m'D](b-1) \geq L-1$, where $D = \lfloor \frac{N-H}{m'} \rfloor$.

Proof: Note that D is the number of cycles for which we can form a repeating subsequence of k jobs with the option followed by m' basic jobs, and the subsequences have no idle time. At the end of the D cycles, we have $H - kD$ jobs with the option and $N - H - m'D$ basic jobs remaining to be scheduled. Suppose that we preempt the last job at the end of the

D repeating subsequences and force a regeneration at that point. It is possible to construct a sequence for the remaining jobs with no idle time, as follows:

> Sequence min $(H - kD, k)$ jobs with the option followed by $N - H - m'D$ basic jobs, followed by the remaining jobs with the option (if any).

Case 1: $H - kD < k$

In this case we would sequence $H - kD$ jobs with the option followed by $N - H - m'D$ basic jobs. Since the fraction of jobs with the option in the entire data set exceeds $k/(k+m')$ and the fraction of jobs with the option in the repetitive cycles is exactly equal to $k/(k+m')$, the fraction of jobs with the option remaining at the end of the repetitive cycles must exceed $k/(k+m')$. Consequently, the subsequence for the remaining jobs will not regenerate, and there will be no idle time between jobs. Since we also have $(H-kD)o+[(N-H-m'D]b \geq (H-kD)+(N-H-m'D)+L-1$, or equivalently $(H-kD)(o-1)+[(N-H-m'D](b-1) \geq L-1$, i.e., the total processing time for these jobs is greater than or equal to the remaining time available, there is no idle time at the end of the sequence.

Case 2: $H - kD \geq k$

In this case we would sequence k jobs with the option followed by $N-H-m'D$ basic jobs, followed by any remaining jobs with the option. Since $N - H - m'D \leq m'$ by the definition of D, this subsequence will not regenerate (there are k jobs with the option followed by m' or fewer basic jobs), and there will be no idle time between jobs. Since $(H - kD)(o-1) + [(N - H - m'D](b-1) \geq L-1$, there is no idle time at the end of the sequence. ∎

Note that we forced a regeneration at the end of the D cycles to obtain this result. Consequently, although the condition that the fraction of jobs with the option exceeds $k/(k+m')$ is necessary, the condition $(H-kD)(o-1)+[(N-H-m'D](b-1) \geq L-1$ is sufficient but not necessary for optimality of the spacing heuristic. This does not present a practical

problem, however, since it is easy to check whether the spacing heuristic generates a sequence with no idle time.

Also observe that in the underloaded scenario, the system does regenerate at the end of each cycle, but in the latter highly overloaded scenario, regeneration occurs only if the last job in each cycle is preempted to start the first job with the option in the subsequent cycle just as it enters the station.

The indicated values of k and m (or m') have obvious implications for spacing rules: good (and sometimes optimal) spacing rules are given by "k out of $k+m$," and "no more than k jobs with the option should be sequenced consecutively." Note, however, that the latter rule is generally less constraining than the former.

In this section, we have developed optimal algorithms utilizing the concept of regeneration to sequence jobs for one station offering two types of operations, and have developed related spacing rules. We have shown that when the system is heavily over- or under-loaded, these sequencing procedures are optimal with respect to total utility work.

4 An Optimal Procedure for Intermediate Loads

In section 3, we considered situations where it was possible to avoid utility work because the system was underloaded, or where it was possible to construct a sequence with no idle time because of the heavy load. In this section, we develop an optimal procedure for a class of problems for which we do not have integer k and m satisfying (2) and (3), and for which load levels are such that the spacing procedure is not guaranteed to provide optimal solutions. Since the conditions in which the spacing procedure does not provide optimal solutions are quite lengthy, we will simply use the phrase "fraction of jobs with the option between $k/(k+m)$ and $k/(k+m-1)$ to refer to these conditions. These limits are technically correct, except near the endpoints of the range. For problems with these characteristics, we

know that the system is overloaded, but it is not obvious when and how much utility work should be incurred for each position in the sequence. The following procedure is greedy in that it tries to avoid utility work.

GREEDY PROCEDURE

For each position in turn:

> If there is enough time to complete a job with the option prior to its departure from the station, schedule a job with the option, provided there is a job with the option still unscheduled. If such a job is not available, schedule a job without the option.
>
> If there is insufficient time to complete a job with the option prior to its departure from the station, schedule a job without the option, provided there is a job without the option still unscheduled. If such a job is not available, schedule a job with the option.

This procedure basically tries to use the available time in an opportunistic fashion, scheduling jobs with the option whenever possible without incurring utility work. It has one potential drawback: it may schedule a basic job and thereby cause unnecessary idle time, even when it would be possible to complete nearly all of a job with the option while incurring unavoidable utility work. This procedure is optimal for certain parameter values, as proved below.

Proposition: For the case of two job types, the Greedy procedure is optimal if $o - b < L - 1$ and if the fraction of jobs with the option lies between $k/(k+m)$ and $k/(k+m-1)$.

Proof: See Appendix 2.

Note that the condition $o - b < L - 1$ is a limit on the disparity of the two processing times. Qualitatively, it suggests that one can achieve better results (from the viewpoint

of utility work and related quality considerations) if there is more standardization (smaller difference between o and b) or longer station lengths.

5 Heuristics for Intermediate Loads

In this section, we consider the fourth and final set of problem parameters: fraction of jobs with the option between $k/(k+m)$ and $k/(k+m-1)$ and $o-b \geq L-1$. In this case, finding an optimal solution requires the solution of a DP because neither a simple repetitive pattern ("spacing heuristic") nor the Greedy procedure is guaranteed to produce an optimal solution. This raises the question of whether either might provide good solutions.

Worst case error bounds are easy to obtain for the spacing heuristic. (These error bounds also apply to the case of $o-b < L-1$.) Using spacing parameters of k and m (not $m-1$), we would have at most $k+m-(ko+mb)$ idle time in the cycles that have positive idle time. If $o-b \geq L-1$, some of this may be unavoidable because of the combinatorial aspects of sequencing. Since the system is overloaded, any idle time translates into an equivalent increase in utility work. Thus, in the worst case, the error per job is $[k+m-(ko+mb)]/(k+m)$.

Another reasonable heuristic is a variation of the greedy procedure, which is described below.

GREEDY PROCEDURE-II

For each position in turn:

> If it is possible to schedule a job with the option and avoid utility work, schedule a job with the option if there is still one unscheduled.
>
> Otherwise, if it is possible to schedule a basic job and avoid idle time, schedule a basic job, provided there is such a job still unscheduled. If such a job is not available, or if scheduling a basic job will cause idle time, schedule a job with the option.

This version of the greedy procedure tries to avoid all idle time, possibly at the expense of incurring unnecessary utility work. It is intuitively clear that this version of the greedy procedure would perform well in heavily-loaded systems. It is difficult to derive structural properties or worst case error bounds for this heuristic because the resulting sequence may not regenerate or may have very long regeneration cycles.

We now have three heuristic procedures for this final subclass of problems: the spacing procedure, the original greedy procedure, and the Greedy-II heuristic. To evaluate the performance of the heuristics empirically, we coded the DP procedure in FORTRAN77 on an Amdahl 5880 and created a set of 200-job problems in which the fraction of jobs with the option was selected in the interval between $k/(k+m)$ and $k/(k+m-1)$, where k and m are defined by (5) and (6), respectively, reflecting slightly overloaded to heavily loaded stations. Table 1 lists the parameters for the 33 problems. Note that if b is close to 1, the condition $o - b \geq L - 1$ generally leads to $k = 1$ and m large (e.g., $m \geq 10$). In such circumstances, the difference between $k/(k+m)$ and $k/(k+m-1)$ is quite small. Thus, we have included only a few examples for $b \geq 0.8$.

Table 1

We compared the heuristic solutions to the solution from the DP. We also report a simple lower bound on total utility work (total work to be completed less total time available), which is the same as the bound that would be obtained from a linear programming relaxation of the problem. It is interesting to note that in several problems, the optimal objective value differs from the lower bound. In these instances, it is not possible to construct a solution with no idle time, even when the system is overloaded. The objective values from the heuristics and the DP are presented in Table 2, along with the maximum number of states and the computation times for the DP.

Table 2

All three heuristic procedures obtained optimal solutions in five problems (problems 3, 17, 31, 32, and 33) where $ko + mb = k + m$. When the processing times and spacing parameters satisfy this property, it is easy to construct sequences with no idle time, so these results are not surprising.

Overall, the Greedy-II procedure outperformed the others in the remaining 28 problems. It produced the best solution (or tied for the best) for 17 of the problems, of which 11 were optimal. The spacing heuristic had the worst performance of the three heuristics. It obtained the best solution (or tied for best) in only 4 problems, and in none of the 28 problems did it find an optimal solution. The original greedy procedure produced the best solution (or tied for best) in 12 of the problems, of which 7 were optimal. The best of the heuristic solutions was optimal in 15 of the 28 problems. There were, however, a number of problems in which the best solution from the heuristics was quite poor. In no case did a heuristic procedure find an optimal solution when the optimal objective deviated from the lower bound.

The results suggest that the two greedy procedures might be worthwhile to execute before attempting to solve the DP. If the better of the two solutions has the same objective value as the lower bound, we could avoid solving the DP. Otherwise, the DP should be executed.

To illustrate the effect of workloads on the performance of the heuristic procedures, we selected three problems (#1, #9, and #15) and varied the fraction of jobs with the option over the entire range from $k/(k+m)$ to $k/(k+m-1)$. The results are shown in Table 3. As expected, Greedy-II outperforms the spacing heuristic when then system is overloaded (lower bound greater than zero). This occurs because it myopically tries to avoid idle time, while the spacing heuristic has the opposite strategy of avoiding utility work. The converse applies in underloaded situations. However, the original greedy procedure outperformed the spacing heuristic in underloaded scenarios. One reason for this is that the Greedy procedure does not constrain the sequence to a repetitive pattern, and thereby tends to utilize the available time more effectively.

Table 3

We were also interested in determining how well the spacing-rule-based procedure would perform for the domain of problems in which the (original) Greedy procedure is optimal. Although the Greedy procedure is simple, the repetitive pattern is clearly much simpler. We randomly generated over 100 problems, and isolated 15 of them with parameter values for which the repetitive pattern (spacing rule) would be suboptimal and the Greedy procedure would be optimal. Parameters for these 200-job problems are listed in Table 4 and results are presented in Table 5. The results indicate that the spacing heuristic performs only marginally, with objective values falling mid-range between the optimal and worst case objective values. (The worst case objective value is the optimal objective value plus the worst case error for the spacing procedure discussed at the beginning of this section.) Thus, use of the spacing heuristic would not be recommended for this subclass of problems.

Tables 4 and 5

On the basis of these results, it appears that a combination of the procedures would perform well. The simple spacing heuristic would be executed first, using spacing parameters k and m if the fraction of jobs with the option is less than $k/(k+m-1)$, or parameters k and $m-1$ otherwise. If the resulting objective value deviates from the lower bound, which can occur only when the fraction of jobs with the option is between $k/(k+m)$ and $k/(k+m-1)$, the greedy procedures or the DP would be executed, depending upon the processing time parameters and the need for optimal solutions. This reduces the need to use the more complicated procedures, thereby limiting computational effort. These procedures must be executed many times in the context of multi-station problems, so reducing the computational effort may be important in practical applications. In a sequel, we suggest the use of a surrogate objection function and related solution procedures that permit further reductions in computation.

We observed that the greedy procedure produced sequences that regenerated, but the potential regeneration points were not always at the origin of the station. Furthermore, some regenerative subsequences were quite long (e.g., 50 jobs), even though all of our processing times were multiples of 0.05. With processing times used in practice, the regeneration cycles might be much longer, so it would be difficult to take advantage of the existence of regeneration cycles to speed computation. More effort might be consumed in checking for regeneration than would be required to construct the sequence.

We should note that the CPU time required to solve the DPs was over 4 minutes per problem on the average, making it prohibitive to use as the basis for multiple-station heuristics. The other procedures required only milliseconds of CPU time. We have incorporated the simpler heuristic based on spacing rules in a multi-station sequencing procedure which has proved to be successful in an automotive industry application (see Yano and Bolat 1990).

6 Conclusions

We have developed a set of procedures to determine a sequence that minimizes utility work for one station which offers two types of operations on a paced assembly line with no buffers. We have proved that a simple procedure based on spacing rules is optimal when the load on the system is sufficiently low or high. We have developed error bounds for this algorithm if these conditions are not satisfied. We have also developed a greedy algorithm which is shown to be optimal for moderately overloaded systems, under certain conditions on the processing time parameters. A dynamic programming algorithm is presented for cases in which the spacing rule or greedy procedure cannot be guaranteed to provide optimal solutions. We also propose a variation of the greedy procedure as a heuristic for this last class of problems.

In experimental tests, we have found that the simple spacing-rule-based procedure does not perform well over the range of parameter values where the greedy procedure is optimal.

We have also found that the better of the two solutions from the greedy procedures outperforms the solution from the spacing procedure for the range of parameters where only the DP is guaranteed to provide optimal solutions. We also observed that the better of the two greedy solutions also provides optimal solutions in some instances, thereby reducing the need for the computationally intensive DP. In other instances, however, the solutions were far from optimal. Thus, our results indicate that a combination of the procedures is needed to solve this class of problems effectively.

Recall that the motivation for solving the single-station version is to determine lower bounds for the multi-station problem. In this regard, it is useful to emphasize that our results indicate that unless the fraction of jobs with the option is between $k/(k+m)$ and $k/(k+m-1)$ and $o - b > L - 1$, the optimal objective value is equal to the lower bound, which is very easily computed and involves no optimization. In many practical applications, situations with $o - b > L - 1$ are uncommon. Indeed, we did not find any parameters with these characteristics in examining hundreds of different stations at several different automobile assembly facilities. Such parameters would be regarded as reflecting poor assembly line balancing decisions for two reasons. First, it is difficult to achieve good sequences for these stations, especially when they must be considered in conjunction with other stations. Second, even small changes in the product mix can have a dramatic effect on the total workload.

Considering only two types of operations has allowed us to develop efficient procedures for subclasses of this problem. The insight obtained from these results may permit us to extend the approach to multiple operations at a station. Further research along these lines would also be applicable to flexible flow systems with small buffers.

7 Acknowledgments

This work was supported in part by a research contract from a major U.S. automobile company to the University of Michigan.

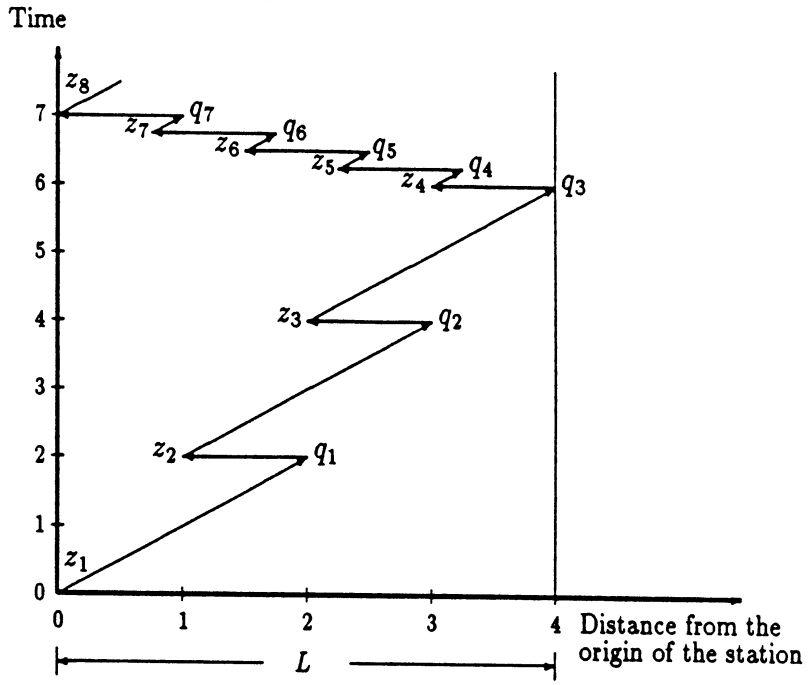

Figure 1: The operator movement diagram for sequence O,O,O,B,B,B,B.

TABLE 1

PROBLEM DATA FOR EVALUATION OF SPACING AND GREEDY HEURISTICS

PROB. NO.	PROCESSING TIMES BASIC	OPTIONAL	STATION LENGTH	NO. OF JOBS W/OPTION	PARAMETERS k	m
1	0.05	2.25	3.0	91	1	2
2	0.05	3.45	4.0	56	1	3
3	0.10	1.45	2.0	145	2	1
4	0.10	3.55	4.0	54	1	3
5	0.15	1.25	2.0	141	4	2
6	0.15	2.65	3.0	69	1	2
7	0.20	2.50	3.0	69	1	2
8	0.20	3.35	4.0	56	1	3
9	0.25	1.85	2.0	85	1	2
10	0.25	3.70	4.0	47	1	4
11	0.30	1.60	2.0	103	1	1
12	0.30	2.50	3.0	56	1	3
13	0.35	2.75	3.0	54	1	3
14	0.35	3.85	4.0	38	1	5
15	0.40	1.80	2.0	99	1	2
16	0.40	3.60	4.0	35	1	5
17	0.45	1.55	2.0	109	1	1
18	0.45	2.95	3.0	49	1	4
19	0.50	2.90	3.0	47	1	4
20	0.50	3.80	4.0	31	1	6
21	0.55	1.75	2.0	82	1	2
22	0.55	3.85	4.0	27	1	7
23	0.60	1.70	2.0	84	1	2
24	0.60	2.75	3.0	39	1	5
25	0.65	2.95	3.0	32	1	6
26	0.65	3.85	4.0	22	1	9
27	0.70	1.95	2.0	46	1	4
28	0.70	3.80	4.0	20	1	10
29	0.75	1.90	2.0	47	1	4
30	0.75	2.80	3.0	24	1	8
31	0.80	2.80	3.0	22	1	9
32	0.85	1.90	2.0	32	1	6
33	0.90	1.90	2.0	22	1	9

TABLE 2
COMPARISON OF SPACING AND GREEDY HEURISTICS WHEN NEITHER IS GUARANTEED TO BE OPTIMAL

PROB. NO.	TOTAL UTILITY WORK					MAX. NO. OF STATES	CPU (SEC.)
	LB	SP. HEUR.	GREEDY	GREEDY-II	DP		
1	8.20	35.40	9.55	8.20	8.20	2906	1529
2	0.00	15.25	1.90	4.05	0.90	772	174
3	14.75	14.75	14.75	14.75	14.75	541	92
4	3.30	10.05	10.50	4.20	3.90	1000	292
5	0.00	3.00	0.00	0.95	0.00	1332	515
6	0.50	3.40	3.75	4.05	2.25	2035	981
7	0.00	2.70	3.20	2.40	1.00	1533	576
8	13.40	15.05	15.80	13.40	13.40	3423	2441
9	0.00	16.70	5.90	8.25	1.05	845	147
10	9.15	19.65	20.55	9.15	9.15	513	85
11	0.00	2.20	2.60	4.20	1.00	1339	302
12	0.00	8.30	0.00	0.80	0.00	1357	489
13	0.00	6.95	12.00	6.70	2.60	744	166
14	0.00	11.85	12.80	9.10	3.50	489	81
15	17.60	31.40	23.80	19.40	18.20	689	64
16	0.00	2.20	0.00	0.00	0.00	654	137
17	8.90	8.90	8.90	8.90	8.90	541	59
18	10.50	18.95	19.75	21.25	11.90	535	90
19	10.80	14.30	14.60	10.80	10.80	513	85
20	0.00	4.90	4.90	1.50	1.10	459	73
21	7.40	14.95	16.25	7.40	7.40	893	173
22	0.00	3.30	0.00	3.75	0.00	452	71
23	11.40	16.10	11.40	11.40	11.40	1240	317
24	1.85	8.85	3.55	1.85	1.85	730	166
25	1.60	5.80	5.80	6.15	2.90	474	77
26	0.00	3.10	0.00	0.90	0.00	863	193
27	0.00	5.70	0.90	2.15	0.30	502	82
28	0.00	2.60	0.00	0.00	0.00	659	129
29	3.05	6.55	6.85	3.05	3.05	513	85
30	0.00	1.60	0.00	0.00	0.00	1042	270
31	2.00	2.00	2.00	2.00	2.00	361	32
32	2.60	2.60	2.60	2.60	2.60	554	90
33	1.00	1.00	1.00	1.00	1.00	267	31

TABLE 3

IMPACT OF WORKLOADS ON PERFORMANCE OF HEURISTICS

PROB. NO.	NO.JOBS W/OPTION	TOTAL UTILITY WORK LB	SP. HEUR.	GREEDY	GREEDY-II	DP
1-1	67	0.00	0.00	0.00	1.05	0.00
1-2	71	0.00	4.40	0.00	1.05	0.00
1-3	75	0.00	10.60	0.00	1.20	0.00
1-4	79	0.00	16.80	0.00	1.20	0.00
1-5	83	0.00	23.00	0.00	1.20	0.00
1-6	87	0.00	29.20	0.70	1.35	0.45
1-7	91	8.20	35.40	9.55	8.20	8.20
1-8	95	17.00	41.60	18.30	17.00	17.00
1-9	99	25.80	47.80	27.05	25.80	25.80
9-1	67	0.00	0.00	0.00	6.45	0.00
9-2	71	0.00	3.40	0.00	6.85	0.00
9-3	75	0.00	7.20	0.00	7.25	0.00
9-4	79	0.00	11.00	0.00	7.65	0.00
9-5	83	0.00	14.80	3.25	8.05	0.55
9-6	87	0.00	18.60	8.55	8.45	1.70
9-7	91	0.00	22.40	13.85	8.85	4.10
9-8	95	1.00	26.20	20.25	9.25	6.50
9-9	99	7.40	30.00	25.55	9.65	8.90
15-1	67	0.00	0.00	0.00	13.00	0.00
15-2	71	0.00	3.40	0.00	13.80	0.00
15-3	75	0.00	7.40	0.00	14.60	0.00
15-4	79	0.00	11.40	0.00	15.40	0.00
15-5	83	0.00	15.40	3.00	16.20	2.20
15-6	87	0.80	19.40	8.20	17.00	6.20
15-7	91	6.40	23.40	13.40	17.80	10.20
15-8	95	12.00	27.40	19.00	18.60	14.20
15-9	99	17.60	31.40	24.20	19.40	18.20

TABLE 4
PROBLEM DATA FOR EVALUATION OF SPACING HEURISTIC
WHEN GREEDY IS OPTIMAL

PROB. NO.	PROCESSING TIMES BASIC	OPTIONAL	STATION LENGTH	NO. OF JOBS W/OPTION	PARAMETERS k	m
1	0.30	3.15	5.0	42	1	4
2	0.00	3.95	5.0	52	1	3
3	0.05	2.75	5.0	70	2	4
4	0.05	2.30	4.0	87	2	3
5	0.10	2.45	5.0	77	2	4
6	0.15	2.50	5.0	79	2	4
7	0.20	3.10	5.0	59	1	3
8	0.25	1.90	3.0	99	2	3
9	0.30	3.35	5.0	46	1	4
10	0.30	2.25	5.0	72	3	6
11	0.35	1.10	2.0	174	10	2
12	0.40	2.40	4.0	62	2	5
13	0.45	1.35	3.0	124	5	4
14	0.45	2.95	5.0	43	2	8
15	0.60	3.50	5.0	28	1	7

TABLE 5
COMPARISON OF GREEDY AND SPACING PROCEDURES WHEN GREEDY IS OPTIMAL

PROB. NO.	GREEDY (OPTIMAL)	SPACING HEURISTIC	WORST CASE*
1	0.00	1.05	26.00
2	1.40	3.85	3.90
3	0.00	4.30	10.00
4	2.75	11.10	12.75
5	0.00	13.75	23.33
6	11.65	20.45	24.98
7	7.10	18.10	22.10
8	11.35	21.50	29.35
9	0.00	11.30	18.00
10	0.00	5.85	10.00
11	0.00	2.80	5.00
12	1.00	6.00	6.71
13	0.00	5.20	10.00
14	0.00	3.00	10.00
15	0.00	4.40	7.50

*Optimal objective value plus worst case error.

References

Bolat, A., 1988. "Generalized Mixed Model Assembly Line Sequencing Problem." Unpublished Ph.D. Dissertation, University of Michigan, Ann Arbor, MI.

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

Coffman, P.E., S.E. Hoffman and S.A. Weiner, 1985. "An O.R. View of Assembly Plant Modeling." Working Paper, Ford Motor Company.

Monden, Y., 1983. *Toyota Production System.* Industrial Engineering and Management Press, IIE, Atlanta.

Thomopoulos, N.T., 1967. "Line Balancing-Sequencing for Mixed-Model Assembly." *Management Science*, Vol.14, No.2, pp. 59-75.

Okamura, K. and H. Yamashina, 1979. "A Heuristic Algorithm for the Assembly Line Model-Mix Sequencing Problem to Minimize the Risk of Stopping the Conveyor." *Int. J. Prod. Res.*, Vol.17, No.3, pp. 233-247.

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

Yano, C.A. and A. Bolat, 1990. "Survey, Development, and Application of Algorithms for Sequencing Paced Assembly Lines," *Journal of Manufacturing and Operations Management*), Vol. 2, No. 3, pp. 172-198.

Yano, C.A. and R. Rachamadugu, 1991. "Sequencing to Minimize Work Overload in Assembly Lines with Product Options." *Management Science*, Vol 37, No. 5, pp. 572-586.

Appendix 1

Forward Dynamic Programming Formulation

The forward dynamic programming formulation of the problem is as follows. Let τ be the greatest common real divisor of $\{1, L, o, b\}$. Thus, all relevant time units can be expressed as an integer multiple of the basic unit of time, τ. Also define:

n_o = number of jobs requiring the optional operation which have been sequenced thus far;

n = total number of jobs sequenced thus far;

d_n = duration for which the nth job is processed;

t_n = time of completion or termination of nth job (in time units);

$$X_n = \begin{cases} 1 & \text{if a job requiring the optional operation is selected for position n} \\ 0 & \text{otherwise.} \end{cases}$$

$G(n_o, n, t)$ = total utility work given n_o, n, and t (in time units).

For simplicity, we present a formulation in which an operator may terminate processing on a job before it leaves the station, even if the work on it is not complete. The objective is to

$$\text{minimize } G(R, N, (N + L - 1)/\tau)$$

and the recursion equations are

$$G(n_o, n, t) = \min_{X_n, d_n} \{[b/\tau + (o - b)X_n/\tau - d_n/\tau]^+ + \min_{z \le t - d_n} G(n_o - X_n, n - 1, z)\}.$$

The first term in braces is the utility work for the job in the nth position in the sequence. The internal minimization allows us to consider schedules in which there is idle time between z and $t - d_n$.

The boundary conditions are:

$G(0, 0, 0) = 0$ and

$G(n_o, n, t) = \infty$ for any (n_o, n) if $t < 0$.

The latter condition ensures that all job starting times are non-negative.

The size of the state space can be reduced by noting that we must have $R - n_o \leq N - n$, that is, the total number of jobs with the option remaining to be sequenced must not exceed the number of remaining positions in the sequence. Also, for any n, the only feasible values of t are n/τ through $(n + L - 1)/\tau$, i.e., the completion or termination of the nth job must be during its sojourn within the station. To reduce computation time, it is also possible to restrict our attention to non-preemptive, FCFS schedules. In this case, there is only one choice of d_n (i.e., as large as feasible) for each value of X_n. As such, this procedure is essentially a forward reaching algorithm. Note that it is easily generalized (in concept) to handle multiple types of jobs, but the state space increases rapidly with the number of jobs types.

Appendix 2

Proof of Optimality of Greedy Procedure for a Class of Problems with Two Job Types

We need to show that the greedy procedure always constructs a sequence with no utility work and no idle time as long as there are jobs of both types remaining to be sequenced if $o-b < L-1$. If the only remaining jobs are basic, there will be idle time but no utility work, and the sequence is clearly optimal. If only jobs with the option remain, there may be some unavoidable utility work, but no idle time, so the sequence again is optimal. Throughout the proof, we assume that the fraction of jobs with the option exceeds $k/(k+m)$, so that it is logical to attempt to construct a sequence with zero idle time. We also assume that the fraction of jobs with the option is less than $k/(k+m-1)$, since if the fraction is greater than or equal to $k/(k+m-1)$, we can reduce m by 1 and obtain an optimal sequence using the repeating pattern.

By construction, the procedure generates a sequence with no utility work as long as there are jobs of both types to be sequenced. We show that the sequence also has no idle time as long as there are jobs of both types to be sequenced and if $o-b < L-1$.

Let τ = greatest common (real) divisor of $\{o, b, L, 1\}$, and

Y_n = position of operator within the station (distance from origin) at the start of job n, measured in time units.

Observe that under the greedy procedure,

$Y_{n+1} = Y_n + (o-1)/\tau$ if $Y_n \varepsilon [0, (L-o)/\tau]$
(i.e., if a job with the option can be completed in position n), and

$Y_{n+1} = \max\{Y_n - (1-b)/\tau, 0\}$ if $Y_n \varepsilon [(L-o)/\tau + 1, L/\tau]$
(i.e., if a job with the option cannot be completed in position n, so a basic job is sequenced).

Idle time occurs in the latter case if $Y_n - (1-b)/\tau < 0$. Note that the minimum value of Y_n in this case is $(L-o)/\tau + 1$. Thus, if $Y_n - (1-b)/\tau \geq 0$ for this minimum value of Y_n, the inequality also holds for all larger values of Y_n. We need to determine whether

$$(L-o)/\tau + 1 - (1-b)/\tau \geq 0, \text{or}$$
$$(L-1)/\tau + 1 \quad \geq (o-b)/\tau.$$

The previous inequality can be written as

$$(L-1) + \tau \geq o - b, \text{or}$$
$$L - 1 \quad > (o-b).$$

Thus, we have shown that when $L - 1 > o - b$, there is no idle time in the schedule as long as there are jobs of both types remaining to be sequenced. This completes the proof. ∎