

SEQUENCING AND BATCHING FOR TWO CLASSES OF JOBS WITH DEADLINES AND SETUP TIMES

DAVID L. WOODRUFF AND MARK L. SPEARMAN

*Graduate School of Management, University of California at Davis,
Davis, California 95616, USA*

*Department of Industrial Engineering and Management Sciences,
Northwestern University, Evanston, Illinois 60208, USA*

We formulate a general sequencing problem that includes two classes of jobs with setup times, setup costs, holding costs, and deadlines. The formulation is unique in its explicit recognition of the opportunities to exploit productive capacity increases due to batching. An algorithm based on tabu search is then used as a solution method. Computational results are presented that suggest that the algorithm is effective.

(SEQUENCING; TABU SEARCH)

1. Introduction

This paper describes a general formulation of the single machine sequencing problem. The model considers the cost and time required for setups, the cost of holding inventory, the value of increased throughput, and the desire to meet exogenously set deadlines. The problem of sequencing becomes particularly difficult when the amount of time required to set up between parts is significant and depends on the order of part production. One way to avoid this difficulty is to eliminate setups, but this may not be economically justifiable (Trigeiro, Thomas, and McClain 1989).

Interest in sequencing has been perked by the realization that the sequence of jobs on a production facility's bottleneck is important. This is true both for *pull*-based control systems where the start of work is triggered by the completion of other work and *push* systems where the start of work is scheduled in advance. One of the reasons that setup reduction is so important in the kanban literature (e.g., Hall 1983; Monden 1983) is that sequencing cannot be performed a priori. However, a pull system that can accommodate significant setups known as CONWIP (Spearman 1990) does so by allowing sequencing of the bottleneck. Explicit recognition of the need to sequence the bottleneck is one of the advantages of the push control system "Drum-Buffer-Rope" advocated by Goldratt and Fox (1986). Our sequencing method could be used as the "good rules" for scheduling that Goldratt and Fox suggest must exist but do not specify. Better-known push systems based on Material Requirements Planning (MRP) regard sequencing as a dispatching problem and delegate it to shop floor control. This can sometimes result in schedules

* Received April 1990; revisions received April and August 1991; accepted September 1991.

with due dates that are impossible to meet regardless of the dispatching rules used (Kanet 1988).

When setup times are significant, it is desirable to batch together jobs from the same family in such a way that deadlines are met. Batching also effectively increases capacity by reducing the time spent on setups. We will define a family of parts by the property that intrafamily changeovers do not require a setup. The presence of part families is very important in the operation of our algorithm. When there is more than one order for parts from the same family then a dominance result can be exploited.

Because work is batched to avoid setups, some jobs will finish early. If early shipment is not allowed (see e.g., Kanet and Christy 1984), we must consider the effect that a sequence will have on holding costs. If early shipment is allowed, then holding costs will not vary significantly with the sequence selected (assuming that transfer batches are smaller than process batches).

When there are exogenously set deadlines and significant setups, minimization of makespan subject to deadline feasibility is not always an adequate representation of the desire to increase throughput. We have encountered the situation in industry where most production is in response to orders that have been promised for certain deadlines. Additional work with deadlines is possible but not "booked." Examples include jobs that are conjectured by the sales staff to be a possibility if feasible, work that can be sent to a subcontractor if it cannot be done on time, or jobs can be done in response to forecasts of demand (no deadline). The minimum makespan sequence for the promised jobs will not necessarily result in the best accommodation of "filler" work because it is constructed without knowledge of the setup characteristics, deadlines, or value of the optional jobs. The formulation given here defines two classes of jobs so that the optional work (*filler jobs*) may be considered along with the required work (*firm jobs*). This is in marked contrast to the common practice of using setup costs as a proxy for capacity losses due to setup times. This practice is not desirable because the "cost" of lost capacity cannot be known until the sequence has been generated (this is discussed in a slightly different context in Karmarkar 1987).

An important contribution of this paper is the formulation that is presented. We cast the problem as one of profit maximization rather than cost minimization. We also consider a higher level problem than many others in the literature in that we *decide* whether to accept certain jobs rather than attempting to minimize a cost function for a *given* set of jobs. Other contributions relate to the use of heuristic search techniques in general and tabu search in particular for the sequencing problem. Examples include a proposition related to Earliest Due Date Within Families (EDDWF) that greatly reduces the search space and the use of a new diversification parameter to guide the search. Finally, our computational results show that our implementation is a viable one.

The search for good sequences is done using techniques based on tabu search (Glover 1989a, 1989b, 1989c, 1990). Tabu search is a meta-algorithm for search of a space of solutions and a neighborhood structure that defines a means of moving from one solution to another. Steepest descent forms the basis of the algorithm, but tabu search escapes local optima by forbidding the reversal of recent moves. Sequencing jobs with setup times is a hard problem so we will prove and make use of a theorem that reduces the search space. We also make use of insertions to permute the sequences that seems to be superior to the commonly used method of "swaps." Our computational experience confirms that tabu search is an effective means of attacking the problem.

Section 2 defines the problem mathematically and is followed by a section that discusses relevant literature. Section 4 gives an algorithm based on tabu search and two propositions concerning optimal sequences. Section 5 describes computational experience with the method. Sequences with inserted idle as well as extension to the presence of ready times are discussed in Section 6. The final section gives conclusions.

2. Problem Statement

The mathematical statement of the problem makes use of these parameters:

- Z is the planning horizon (time),
- Ω is all jobs (desired shipments), indexed here by i ,
- D_i is the deadline (time) for job i , ($D_i \leq Z$),
- H_i is holding cost per for job i per unit time,
- W_i is the priority weight associated with job i ,
- K_i is the family of job i ,
- $S(k, l)$ is setup time to switch from family k to family l , $S(k, k) = 0$,
- $C(k, l)$ is setup cost to switch from family k to family l , $C(k, k) = 0$,
- T_i is time required for job i (without setup),
- \mathcal{F} is the set of firm job indexes, and
- \mathcal{E} is the set of filler job indexes.

We use the terms date and time interchangeably because we assume that the conversion makes use only of a constant. The parameter W_i indicates the value of including job i in the sequence. Because all firm jobs must be included in feasible sequences, the values of W_i for $i \in \mathcal{F}$ will not affect the choice of the optimum sequence. Note that if a filler job is being considered in response to a forecast or to build ahead for seasonal demand, then the deadline of the job should be set to Z .

For the purpose of determining the need for an initial setup, the zero indexed element of Ω is taken to be the last job before the planning horizon with $D_0 = T_0 = W_0 = 0$. Zero is taken to be a member of \mathcal{F} . We assume that \mathcal{F} and \mathcal{E} are a mutually exclusive and exhaustive set of the indexes of Ω .

The objective is to find a job sequence, σ , that is an n -tuple of job indexes from Ω that maximizes total weight less holding and setup costs. In our notation, $\sigma(1)$ gives the index of the first job in the optimal sequence, $\sigma(2)$ the second, and so on. $\sigma(0)$ is defined to be zero. We use the notation x^+ to mean x if $x > 0$ and 0 otherwise. Mathematically, the problem is to

$$\begin{aligned} &\text{maximize } \sum_{i=1}^n [W_{\sigma(i)} - H_{\sigma(i)}[D_{\sigma(i)} - a_{\sigma(i)}]^+ - C(K_{\sigma(i-1)}, K_{\sigma(i)})] \\ &\text{subject to: } \quad \mathcal{F} \subset \{\sigma(i) : i = 0, \dots, n\} \\ &\quad \quad \quad a_{\sigma(i)} \leq D_{\sigma(i)} \quad i = 1, \dots, n. \end{aligned}$$

where $a_{\sigma(i)} = \sum_{j=1}^i [S(K_{\sigma(j-1)}, K_{\sigma(j)}) + T_{\sigma(j)}]$.

The first constraint assures that all firm jobs will be included. Subsequent constraints assure that there will be no late jobs, and coupled with the requirement that $D_i \leq Z$, assure that the makespan is within the planning horizon. Because firm jobs will be included in every feasible sequence, the values assigned to W_i for $i \in \mathcal{F}$ will not affect the choice of an optimal sequence, just the value of the objective function.

Determining the existence of a feasible sequence has been shown to be NP-hard by Bruno and Downey (1978), who did not consider a formulation with filler work. The presence of filler work adds difficulty because the formulation can be restricted to the knapsack problem by letting $\mathcal{F} = \emptyset$, all $S(k, l) = C(k, l) = 0$, all $D_i = Z$, and all $H_i = 0$.

3. Relevant Literature

The single machine scheduling problem has been addressed by many authors. The problem has been developed well enough to be the subject of a number of very good texts (e.g., Baker 1974; Conway 1967; French 1982). Unfortunately, most of the previous work is not useful for our problem. The objectives (e.g., minimize makespan or average

flow time) are typically different than ours. The absence of filler work in formulations found in the literature begs the question: could we be producing more? Although these formulations are appropriate in many instances, they do not address this question. When setup times are significant, a reasonable question would be to ask if setups could be further reduced (at the expense of holding costs) in order to increase throughput (to obtain higher revenues). This is the question we attempt to answer by allowing the possibility of filler work. Our formulation may be considered a generalization of the early/tardy problem with infinite tardiness penalties (see Baker and Scudder 1990 for a review). There has been no previous work directly on the problem we have encountered in industry and formulated here, but there has been some relevant work done on sequencing firm jobs when there are deadlines and setup times.

Driscoll (1975) has developed a clever branch and bound algorithm that finds the due-date feasible sequence with the minimum setup time. Driscoll's algorithm allows for sequence-dependent setups. The formulation given here can be reduced to Driscoll's by letting all holding costs equal zero and the setup costs be equal to the setup times. Note that minimum setup time implies minimum makespan.

There is a strong relationship between sequencing and lot sizing formulations found in the literature. Many finite lot sizing formulations such as the capacitated lot sizing problem (CLSP) can be formulated as sequencing problems by discretizing the demands and allowing inserted idle time (see Section 6). The reverse is not true in general because the CLSP assumes that demands can be arbitrarily discretized whereas sequencing formulations allow specification of the discretization. A good example of a CLSP formulation is that given by Trigeiro, Thomas, and McClain (1989), who use lagrangian relaxation to solve the CLSP with setups. Their assumptions are consistent with the CLSP in that they ignore sequencing. They assume that setup requirement production of all parts from a family in a period is done in one batch that requires a setup and that the setup characteristics do not vary with the sequence. The objective function considers holding costs (due to early completion), temporal variations in production costs, and setup costs.

Tabu search has been applied to a sequencing problem involving linear tardiness penalties and setup costs developed by Laguna et al. (1990). Although their objective function is similar to (P), the problem is quite different because of the absence of filler work and the presence of tardiness penalties rather than deadline constraints. The bulk of their work was done on problems with setup costs, but they report success extending their algorithm to handle setup times. Their work is encouraging because they report excellent computational experience.

4. Solution Method

We could have removed the most troubling constraints (all but the first) if tardiness penalty functions could be specified. In many cases this is not reasonable. The penalties are often difficult to specify, not linear or even continuous and not independent. For example, a practitioner may say "we can make this job very late if we make a few jobs for the same customer early, but otherwise we shouldn't be late with this job at all." On the basis of our experience with actual practice, we suggest that a decision support system (DSS) approach is indicated. The algorithms used for sequencing must be able to help the user construct sequences with feasible deadlines even though the feasible deadlines may not all be the same as originally requested. Viewed from the user's perspective, the system provides information to "decide which customers to apologize to."

Our basic solution strategy is to use a form of tabu search in an effort to find both feasible and optimal sequences. This is consistent with our goal of developing an algorithm that can be used in a DSS. Tabu search begins with a solution and aggressively seeks to improve it. Because the problem is NP-hard, any algorithm used may need to be ter-

minated prematurely when working with large problem instances. Tabu search will leave us with the best sequence it has found so far. Also, the search method can take advantage of good starting solutions. This can be exploited if parameters are varied slightly by the user after a solution has been found.

This section is divided into four subsections. The first presents a small example of a problem that can be used later to provide some illustration. The second subsection gives and proves a proposition concerning the problem. This proposition is used in the third subsection that describes the components of the tabu search. The final subsection summarizes the solution algorithm.

4.1. Example

In this small example there are three families: *B*, *Q*, and *R*. All setup times are four except the setup from family *R* to family *Q* that requires six time units. All setup costs are one except the setup from family *B* to family *R* that costs two. The processing for all part families requires one time unit. All holding costs are one per time unit. Table 1 gives the jobs to be considered; a weight of zero is used to indicate a firm order because the weights for firm orders do not affect the optimal sequence.

4.2. Proposition Concerning Sequences

The proposition notes that the optimal objective value can always be achieved by a sequence whose jobs are arranged according to earliest deadline (EDD) *within families* (EDDWF) if the holding costs are zero and a reasonable restriction is placed on setup times and costs.

PROPOSITION 1. *If all H_i are zero, $S(k, l) + S(l, m) \geq S(k, m)$, and $C(k, l) + C(l, m) \geq C(k, m)$ then any sequence with N jobs, σ , optimal for (P) will remain feasible and optimal if reordered so that $D_{\sigma(i)} < D_{\sigma(j)}$, $K_{\sigma(i)} = K_{\sigma(j)} \Rightarrow i < j$ for all i and j in $1, \dots, N$ such that $i \neq j$ (EDDWF).*

PROOF. Retention of feasibility after reordering is seen immediately from Corollary 1 in Monma and Potts (1989). Assume that there is an optimal feasible sequence, σ , that has jobs X and Y indexed by x and y such that $D_{\sigma(x)} < D_{\sigma(y)}$, $K_{\sigma(x)} = K_{\sigma(y)}$, $y < x$ (violates the implication of the proposition). Consider the insertion of job Y immediately after job X to form a new sequence τ . To see that an optimal solution is obtained after the insertion, note that sequence dependent terms in the objective function are either multiplied by H , which is zero by assumption, or are setup cost terms. The setup cost before

TABLE 1
Example Jobs

Index in Omega	Family	Time Due	Weight
1	<i>B</i>	5	0
2	<i>B</i>	6	0
3	<i>Q</i>	13	0
4	<i>Q</i>	15	0
5	<i>Q</i>	15	0
6	<i>R</i>	21	0
7	<i>B</i>	22	0
8	<i>Q</i>	22	0
9	<i>B</i>	23	0
10	<i>R</i>	26	0
11	<i>B</i>	30	2
12	<i>Q</i>	30	4
13	<i>R</i>	30	9

the insertion is $\dots + C(K_{\sigma(y-1)}, K_{\sigma(y)}) + C(K_{\sigma(y)}, K_{\sigma(y+1)}) + \dots + C(K_{\sigma(x)}, K_{\sigma(x+1)}) + \dots$. The cost after insertion is $\dots + C(K_{\sigma(y-1)}, K_{\sigma(y+1)}) + \dots + C(K_{\sigma(x)}, K_{\sigma(x+1)}) + \dots$ because $C(K_{\sigma(x)}, K_{\sigma(x+1)}) = 0$ because $K_{\sigma(x)} = K_{\sigma(y)}$ by assumption. The new setup cost cannot be greater because $C(K_{\sigma(y-1)}, K_{\sigma(y)}) + C(K_{\sigma(y)}, K_{\sigma(y+1)}) \geq C(K_{\sigma(y-1)}, K_{\sigma(y+1)})$ by assumption.

This insertion may be repeated for all jobs X and Y indexed by x and y such that $D_{\sigma(x)} < D_{\sigma(y)}, K_{\sigma(x)} = K_{\sigma(y)}, y < x$. \square

We will use EDDWF as a heuristic on the basis of the reasoning that the theorem will hold reasonably well with nonzero holding costs if there is “plenty” of filler work or the deadlines are “tight.”

4.3. Solution Components

Tabu search, which was developed by Glover, has proven to be very useful in research involving discrete optimization. The search uses a hill-climbing algorithm embedded in a meta-algorithm that makes use of memory to control the search. The hill climbing is based on *moves* that transform one solution into another and a move differentiation mechanism that we will assume to be based on a cost function. Short-term memory (the tabu list) is used to avoid back-tracking and to force the search away from local optimal. Intermediate-term memory is intended to force intensified search in promising subregions of the solution space, and long-term memory forces diversification into previously unexplored subregions. The reader should be aware that the brief discussion of tabu search given in this paragraph has taken some liberties by describing particular instances of concepts that are given more general treatment by Glover.

This subsection describes the components of the tabu search used in the solution algorithm. The problem statement indicated that the objective is to find a sequence that includes only feasible filler work. This is relaxed during the solution and all filler work is considered in the trial sequences. Computation of anticipated completion times and sequence costs must be adjusted to account for the presence of infeasible filler work. By including all filler work in the sequences to be considered, we eliminate the need to consider moves that add or remove filler jobs. In essence, filler jobs are “removed” by altering the sequence so that they will be completed after their deadline.

4.3.a. *Tuning.* One difficulty with sophisticated search strategies such as tabu search or simulated annealing is the need to specify various parameters such as cooling schedules, list lengths, and iteration limits. Often, the best value for these parameters is data dependent. Of course, the best performance can be obtained if these parameters are “tuned” by an expert for the type of data at hand even though the search is often reasonably robust with respect to the parameters. Our goal is to produce a system that finds its own search parameters. This, in turn, requires the specification of a set of higher level parameters. But if the search parameters themselves are fairly robust, higher level parameters should be extremely robust. The user-specified control parameters are summarized in Table 2. These will be explained in subsequent subsections.

4.3.b. *Moves.* In order to search the space of possible sequences, we will rely on moves that transform one sequence into another. As in (Glover 1989a, 1989b, 1989c),

TABLE 2
User-Specified Parameters

Parameter	Use	Suggestion
κ	Tabu list length	30
β	Number of steps in first pass	4
γ	Number of steps in second pass	1
ν	Second pass multiplier for L_d	1

if we consider a set of N jobs to be sequenced and let s represent the move function, we can consider a directed search graph whose nodes are the $N!$ possible sequences and whose arcs are defined by s . It is reassuring to define s so that there is certain to be a path from the starting sequence to some sequence that is optimal for (P). A common form of this transformation used in sequencing problems is the "swap" where jobs i and j exchange places in the sequence. In the absence of any restrictions on the swaps, it is well known that this results in a connected search graph. For our purposes it is more desirable to use "insertion" moves where job i is inserted before job j in the new sequence.

Insertion moves also result in a connected search graph, so in the absence of any restrictions on moves, all optimal sequences for (P) are reachable from all starting sequences. Insertion moves appear to result in a much faster search for solutions than do swaps. Intuitively, this makes sense. It is possible to construct sequences that have desirable insertions and no desirable swaps and sequences that have desirable swaps and no desirable insertions. But any swap can be accomplished with two insertions, so there is a reasonable chance of "discovering" the good swap if the search is occasionally required to make nonimproving insertions. On the other hand, exact duplication of the insertion of the job in position i before the job in position j requires $\max(j - i, i - j - 1)$ swaps making the "discovery" of a good distant insertion using swaps unlikely. Another argument is that an insertion can make a late job early or remove a setup and therefore improve the sequence in one move whereas using swaps can be thought of as adding a constraint by requiring that the movement of the "other job" be "not too harmful."

The superiority of insertions versus swaps is clearly problem specific. Laguna et al. (1990) report that simultaneous consideration of both insertion moves and swaps is superior to either type of move alone for the case of linear completion time penalties. Limited empirical tests confirm that swaps do not work well for (P), but these tests were done with zero holding costs. There may be data instances for which consideration of swaps would provide benefits in excess of the computational effort.

The standard approach used in tabu search is to examine almost all possible moves at each iteration. One severe problem with (P) is the high computation effort associated with evaluating the effect of a move. Generally, there is no way to avoid examining the effect of a move on all jobs after the insertion point in a sequence because the anticipated completion times will be changed. For the example given in Table 1, insertion of a job before the first job in the sequence changes all other completion times. (There is a small class of insertions of $\sigma(i)$ before $\sigma(j)$ that requires examination only of the jobs between i and j .) So the computational cost of evaluating the effect of a move is of order N if there are N jobs in the sequence. This is compounded by the fact that there are N possible jobs to insert and $(N - 1)$ nondegenerate insertion points.

We can make use of Proposition 1 to reduce the number of moves to be considered. If the starting sequence is EDDWF, then insertions may be ignored that result in "jumping over family members." In terms of the search graph, Proposition 1 ensures that consideration only of sequences that are EDDWF is all that is necessary to achieve the optimum objective value. It is clear that a path exists between every sequence that is EDDWF if the moves are restricted to preserve EDDWF. Therefore, we will make this restriction on moves.

To see the effect of the EDDWF restriction on the search space, consider a sequence composed of the three families P , Q , and R that are *evenly distributed*. By evenly distributed, we mean $\dots PQRPQR \dots$ as opposed to uniformly (randomly) distributed. If F families have tasks evenly distributed among a sequence of length N , then the maximum number of nontrivial insertion points to be considered for each will be $\min\{N - 1, 2(F - 1)\}$ (tasks near the ends of the sequences will have fewer nontrivial insertion points). Note that this is an upper bound on the average number of insertion points to consider when the distribution is uneven. With the restriction on the moves, the com-

computational cost of examining all possible moves is of order N^2F . The total number of possible sequences that are EDDWF is clearly less than $\min \{N - 1, 2(F - 1)\}^N$. For example, if there are 30 tasks and five families, then there are less than 8^{30} nodes in the search space. This is a large number, but it is a tiny fraction of $30!$, which is the number of nodes if the search is not restricted to sequences that are EDDWF.

The presence of filler jobs and our desire to avoid having more than one class of moves complicates things somewhat. Filler jobs must be allowed to move to positions that make them infeasible even if it involves "jumping over family members." This is easily accomplished by allowing any feasible filler job to jump to the end of the sequence and any infeasible filler job to jump to any positions that imply a feasible completion time and preserve EDDWF. The addition of these moves increases the number of moves to be considered only by a number on the order of the number of filler jobs. Another complication presented by filler jobs is the possible inclusion of tardy filler jobs in the sequence. Because these jobs would not actually be performed, they must be ignored in some sense during the analysis of moves of feasible filler or firm jobs. Infeasible filler jobs must be eligible to be "jumped over" by members of the same family unless the "jump" makes the "jumped over" filler job timely (which would result in a sequence that is not EDDWF). The computational effort associated with these considerations grows with the fraction of capacity devoted to filler work with deadlines interspersed with those of the firm jobs.

A further complication is the need to consider the preservation of batching accomplished by previous moves. In many tabu search implementations, the tabu list is used to avoid move reversal that, in turn, tends to avoid undoing the effects of previous moves. In this application, we can provide a higher level definition of "undoing" because batch formation is a major aspect of the solution when setups are significant. After a job is selected for insertion, adjacent jobs from the same family are also considered for inclusion in the move. All adjacent family members that can be included without increasing the cost function value are included in the move. This restriction tends to preserve batches unless their decomposition is clearly desirable.

4.3.c. Intensification and Diversification. Tabu search makes use of *long-term memory* to guide the search into regions of the solution space that have not been explored and *intermediate-term memory* to intensify the search in promising regions. A commonly used approach for long-term memory is to penalize solutions that employ frequently used characteristics. One approach that has been used is to assign a penalty to the placement of a job at a certain point in the sequence on the basis of the frequency of its occurrence at that point in the past (as in Laguna et al. 1990). For (P) it is extremely difficult to do this to good effect. The presence of significant setups in a "tight" schedule means that certain jobs will be in certain positions for every solution that is optimal or even near-optimal. As an extreme example, if the EDD sequence begins with five firm jobs from the family corresponding to job zero, then it is easy to show that all optimal sequences will begin with these five jobs (under reasonable conditions on holding costs and setups). Forcing jobs out of these positions will certainly force the search into new regions, but only luck will result in finding "good" regions.

Rather than attempting to develop a complex scheme that tries to find good but untried regions, we have opted to force the algorithm to change its search characteristics as a means of exploring different regions of the solution space. We do this by introducing a diversification parameter, d , that provides diversification in the search and a means for the search to optimize its own characteristics in a fashion somewhat like the search for good values of λ in lagrangian relaxation.

Unlike lagrangian relaxation, the dependence on d to achieve these two goals must be designed. We must use d in such a way that the search path will tend to change significantly and so that the cost function value of a solution found for a given computational effort and starting solution is convex in d . For example, the diversification parameter appears in cost functions so that "low" values result in nearly infinite costs for constraint violation

but “high” values allow the search to traverse infeasible sequences. Selection of moves also makes use of d . Low values result in selection of the best available move as is done in tabu search whereas high values result in randomized move selection vaguely resembling simulated annealing (see e.g., Johnson et al. 1989).

After the search has been terminated at one value of d , the value is changed and the search is restarted. The best solutions (based on the appropriate cost function, defined later) found at various values of d are kept in a data structure called `Best_at_dVal` so that the algorithms can be allowed to search for a longer period of time using those values that have shown the most promise (i.e., found the best feasible objective values).

When the diversification parameter is set to an as-yet-untried value, the starting sequence is set to the best sequence found at $d = 0$. When a good diversification parameter, d is reused, the starting sequence is set to the sequence from `Best_So_Far`. The algorithm revisits diversification parameter values in increasing order of quality of solution originally found to as to increase the likelihood that the new starting solution was not visited during the first search using d . Thus, the intensification of the search occurs with promising starting points as well as promising search characteristics (as a function of d).

4.3.d. *Cost functions.* The problem (P) can be expressed as a cost-minimization problem using “big M” penalties for constraints violation. For the j th job in Ω define the filler indicator function as

$$G(j) = \begin{cases} 1 & j \in \mathcal{E} \\ 0 & j \in \mathcal{F}. \end{cases}$$

For the i th job in sequence σ , define the anticipated completion time as

$$a_\sigma(i) = a_\sigma(r(\sigma; i)) + [S(K_{r(\sigma; i)}, K_{\sigma(i)}) + T_{\sigma(i)}],$$

the tardiness function as

$$y_\sigma(i) = [a_\sigma(i) - D_{\sigma(i)}]^+,$$

the earliness function as

$$e_\sigma(i) = [D_{\sigma(i)} - a_\sigma(i)]^+,$$

the tardiness indicator function as

$$v_\sigma(i) = \begin{cases} 1 & y_\sigma(i) > 0 \\ 0 & \text{otherwise,} \end{cases}$$

and the σ index of the most recent job that will be actually done as

$$r(\sigma; i) = \max_j \{j : j < i, [1 - G(\sigma(j))v_\sigma(j)] = 1\}.$$

Let $a_\sigma(0) = 0$. This definition of completion times allows filler work to be included in the sequence but recognizes that a filler job will not be actually worked on unless it can be completed on time.

The cost of sequence σ of length N is then

$$f(\sigma) = \sum_{i=1}^N [[1 - G(\sigma(i))]v_\sigma(i)M + G(\sigma(i))[1 - v_\sigma(i)] \\ \times (C(K_{r(\sigma; i)}, K_{\sigma(i)}) + e_\sigma(i)H_{\sigma(i)} - W_{\sigma(i)})].$$

We will use a finite value of M defined so that every feasible sequence will have a lower cost than every infeasible sequence. A value that clearly has this property is

$$M = \sum_{i \in \Omega} [W_i + H_i D_i + \max_{j \in \Omega} C(K_j, K_i)] + 1.$$

For the purpose of allowing infeasible sequences to be considered at high values of d , an alternative cost function that converges to f as d goes to zero can be used,

$$g_d(\sigma) = \sum_{i=1}^N \left[[1 - G(\sigma(i))] y_{\sigma(i)} (M - d)^+ + G(\sigma(i)) [1 - v_{\sigma(i)}] \left(C(K_{r(\sigma(i))}, K_{\sigma(i)}) + e_{\sigma(i)} H_{\sigma(i)} - W_{\sigma(i)} \exp(d/M) \right) \right] + \epsilon a_{\sigma(N)}$$

where ϵ is very small relative to all other cost parameters. This function has been designed to trade filler work for feasibility so that the relative importance of each varies with d . Table 3 shows the calculations of the terms in the first summation term for g (the ϵ term is omitted) for the example given in Table 1 using a d value of 250. None of the filler work would be completed on time in this sequence so the g terms are zero.

The ϵ term is included for heuristic reasons. Many tabu search implementations make use of move characteristics in determining tabu status; an attempt is made to avoid reversing recent moves. Thus, the tabu list serves the triple role of tending to avoid cycling, tending to avoid undoing the results of previous moves, and tending to force some search trajectory when the move selection mechanism does not (e.g., when a flat region in the search space is encountered). As we shall see, the tabu list mechanism we employ will tend to avoid cycling but will not eliminate move reversals that do not result in revisiting a recent solution. As previously described, we have altered the move selection mechanism to avoid undoing batches. The ϵ term is intended to provide trajectory when other considerations do not. All other things being equal, the algorithm will tend to reduce makespan. This means of providing trajectory is intuitively appealing because makespan reduction often results in reduced tardiness or increased opportunities to insert filler work.

4.3.e. *Tabu List.* Generally, tabu search makes use of a tabu list to guide the search locally and to avoid cycling. Although it might be desirable to specify states that are tabu, the storage and computation required is usually excessive. In many applications, the tabu list is maintained by keeping track of some characteristics of recent moves and making an effort to avoid reversing them. In the interest of economy, the methods employed often rule out moves that will not, in fact, undo previous moves. This problem is mitigated by the fact that the tabu list is short-term memory and the restrictions are removed after a relatively small number of moves. Further mitigation is usually provided by the use of aspiration levels that allow the override of tabu restrictions if the proposed move is good enough.

TABLE 3
Example of g Computations

Index in Omega	Family	Time Due	Weight	Anticipated Completion	g Term
1	<i>B</i>	5	0	5	1.0
2	<i>B</i>	6	0	6	0.0
3	<i>Q</i>	13	0	11	3.0
4	<i>Q</i>	15	0	12	3.0
5	<i>Q</i>	15	0	13	2.0
6	<i>R</i>	21	0	18	4.0
7	<i>B</i>	22	0	23	40.0
8	<i>Q</i>	22	0	28	235.0
9	<i>B</i>	23	0	33	391.0
10	<i>R</i>	26	0	38	470.0
11	<i>B</i>	30	2	0	0.0
12	<i>Q</i>	30	4	0	0.0
13	<i>R</i>	30	9	0	0.0

We have opted to use a direct approach in maintaining the tabu list. That is, we attempt to make previously visited states tabu rather than certain move characteristics. The tabu list is composed of two entries for each sequence visited, the cost (using g_d) and

$$x(\sigma) = \sum_{i=1}^N i\sigma(i)^2.$$

Moves that result in an x value and cost value that match an entry in the tabu list are forbidden. The function x was chosen (somewhat arbitrarily) because it is inexpensive to calculate and does a reasonable job of distinguishing the sequences. There is a nonzero chance that a sequence that has not been visited will be "tabu" because the range of x will be much smaller than the number of possible sequences. This type of error can only be avoided with considerable computational effort as is typically the case when hashing functions are used. The tabu list is kept as a circular array with κ elements, where κ is a parameter. This means that after κ iterations, a sequence may be revisited. In tests involving over 4500 moves using a tabu list of 30 elements for sequences of 30 tasks, just over five percent of moves were tabu. This indicates that extra computational effort to distinguish between sequences is probably not justified.

4.3.f. Move Selection. In order to facilitate some randomness in move selection and to provide an alternative move if the move selected is tabu, a short list of the best moves available is assembled. Tabu search has been conjectured by Skorin-Kapov (1990) to perform poorly in flat regions of the search space (in this case, regions where there are many moves tied for the best and the moves result in little or no change in the cost function). In preliminary tests, we encountered a large number of flat regions in the search space for some problem instances. This was particularly true with zero holding costs. Ryan (1989) has suggested randomness in move selection as a means of creating diversity to help deal with this problem. We have incorporated randomness in move selection in a way that enhances dependence on d . In order to select a move, we must evaluate the change in sequence cost associated with each candidate move i , Δg_i . Suppose that there are I moves in the candidate list, then we will select the move randomly with the probability of move i being selected given by

$$\frac{\exp(-\Delta g_i/d)}{\sum_{j=1}^I \exp(-\Delta g_j/d)}.$$

This selection mechanism results in nearly uniform random selection for very high values of d and in almost deterministic selection for very low values of d . Note that improving moves will be strongly favored over disimproving moves at all but the highest values of d . This selection mechanism bears some resemblance to that used in simulated annealing except that much more information is used here.

The tabu list is checked after move selection. If the move selected is tabu, it is discarded and another move is selected from the candidate list. Note that κ must be less than the length of the candidate list or there is a possibility of an infinite loop so the candidate list length we used was $\kappa + 1$. Table 4 shows a candidate list of length six for the sequence shown in Table 3 ($d = 60$). As an aside, we note that this use of a candidate list is different than that envisioned by Glover (1989a, 1989b, 1989c), where the list is described primarily as a means of reducing the number of moves that must be considered for most iterations and the list is updated using a sampling scheme.

4.3.g. Initialization and Termination. The search for good values of d is referred to as the *first pass*. The best values of d are then retried during a phase called the *second pass*. The first pass goes from $d = 0$ to $d = d_{\max}$ in β steps where β is a user-specified parameter. The starting solution is reset for each of the β iterations in the first pass to a preliminary solution found using $d = 0$. (During debugging tests using sequences of

TABLE 4
Candidate List Example

From	To	Δg	Probability
8	6	-814.0	0.574892
8	10	-470.0	0.145210
8	11	-391.0	0.105867
10	7	-346.0	0.088427
7	3	-263.0	0.063446
7	6	0.0	0.022158

length 10, the optimal was frequently found during the search for a preliminary solution.) The value of d_{\max} is set at M because that is the maximum value that makes sense for cost function g (higher values would result in *rewards* for infeasibility). The second pass is terminated after the γ (a user-specified parameter) best values of d have been retried.

A rule must be developed for stopping the search at a given value of d . A conceptually simple rule is to stop if no improvement has been made after a number of moves have been considered (L_d). We would expect L_d to be an increasing function of d because increasing d increases the possibility of nonimproving moves at every iteration. Appropriate values of L_d are data specific. For some sets of deadlines, long searches are appropriate at high values of d ; for others, there is little point. The method we use is to assume that L_d is a linear function of d and to fit the line using L values found heuristically at $d = 0$ and $d = d_{\max}$. Both points are found by setting $L_d = FN$ (number of part families time number of jobs to be sequenced) and then checking the maximum number of tries between findings of $\text{Best_At_dVal}[d]$. In order to intensify the search during the second pass, L_d is multiplied by a user parameter ν .

4.4. Solution Algorithm

The components are summarized here in a statement of the algorithm. The main algorithm is followed by details for the procedure named SEARCH. Boundary checks are omitted and the solution reset during the first pass is not shown.

Main

1. $\sigma \leftarrow \Omega$; $N \leftarrow \text{cardinality}(\sigma)$; $F \leftarrow$ number of families
2. sort σ increasing in deadline
3. $M \leftarrow \sum_{i \in \Omega} [W_i + H_i D_i + \max_{j \in \Omega} C(K_j, K_i)] + 1$
4. $d_{\max} \leftarrow M$
5. $L_d \leftarrow N$; $d \leftarrow 0$; SEARCH(Tries_0)
6. $d \leftarrow d_{\max}$; SEARCH(Tries_Cmax)
7. $Ld_Slope \leftarrow (\text{Tries_Cmax} - \text{Tries_0}) / d_{\max}$
8. for $d \leftarrow d_{\max}/\beta$ to d_{\max} step d_{\max}/β
 - $L_d \leftarrow \text{Tries_0} + Ld_Slope * d$
 - $\sigma \leftarrow \text{Best_At_dVal}[0].\text{Sequence}$
 - SEARCH(dummy)
9. sort Best_At_dVal increasing in sequence cost
10. for $i \leftarrow 1$ to γ
 - $d \leftarrow \text{Best_at_dVal}[i].d$
 - $\sigma \leftarrow \text{Best_So_Far}$
 - SEARCH(dummy)

11. output Best_So_Far

procedure SEARCH(Tries_Reqd)

1. Tries_Reqd \leftarrow Tries \leftarrow 0

2. $I \leftarrow N * \exp(d/d_{\max})$

3. Candidate_List $\leftarrow I$ moves preserving EDDWF with the lowest values of Δg .

4. sort Candidate_List increasing in Δg

5. Tries \leftarrow Tries + 1

6. URan \leftarrow random(0, 1)

7. Move \leftarrow element k in Candidate_List such that URan is closest to, but less than

$$\frac{\sum_{j=1}^k \exp(-\Delta g_j/d)}{\sum_{j=1}^I \exp(-\Delta g_j/d)}$$

8. if Move in Tabu_List

then

Remove Move from Candidate_List

if Candidate_List = \emptyset then goto 3 else goto 5

9. update Tabu_List

10. DO_MOVE (Move, σ)

11. if σ is the best (using f) sequence found so far at d

then

update Best_At_dVal and Best_So_Far as appropriate

if Tries > Tries_Rqd then Tries_Rqd \leftarrow Tries

Tries \leftarrow 0

12. if Tries > L_d then return

13. goto 3.

5. Computational Experience

In preparation for implementation of a version of our algorithm in a large circuit board plant, we have conducted tests on data that capture the characteristics of the demand and production environment in the plant. The data have been altered to avoid disclosure of sensitive information and to ensure that the optimal objective value is known. Direct setup costs are zero and early shipment is allowed so holding costs are considered to be zero. The exact mechanisms for test generation are given in the Appendix.

The first sets of tests were run using the parameter values given as suggested values in Table 2. Twenty test cases were generated. The average objective function quality was 0.88 and the optimal objective function was achieved in 17 of the 20 cases. Running on a Dell 310 personal computer, the average time to find the best sequence was 101 seconds with a maximum time of 513 seconds. Total run time of the algorithm averaged 1100 seconds. In the three cases where the optimal sequence was not found, the algorithm terminated very quickly, suggesting that higher values of L_c could be used to improve the quality.

The same 20 test cases were given to a program modified so that step 5 of the algorithm set L_d to $F * N$ rather than N and ν was set to two rather than one. The same 17 test cases were solved optimally, but the average objective function quality rose to 0.97. The average time to find the best sequence was 283 seconds and the average total run time rose only by 15 seconds. The improved quality came during searches using d values other than zero, which supports the contention that long-term memory becomes important for hard problem instances.

Because we will use this algorithm in a DSS, we may be willing to sacrifice some quality in the interest of saving time. To study the effect of abandoning the use of the diversification parameter, we used a program that terminated after $\beta + \nu$ iterations of step 5 using the same 20 test cases. (Of course, we did not reset the starting solution.) The number of problems solved correctly fell to 16 and the average objective function quality to 0.85. The average time to find the best sequence was 71 seconds whereas the average total run time was 250 seconds. Some of the time savings was directly due to the degradation in quality. In the cases where the optimal was not found, each of the five calls to SEARCH terminated very quickly because no improvement was possible. Perhaps these 20 cases could be solved more quickly than with long-term memory by using an appropriately long value of L_d , enough calls to search, and a very long tabu list (to guarantee against cycling). We did not conduct a search for such parameters because the point of using the diversification parameter is to eliminate the need for such tuning.

We had two goals for these tests. First, we wanted to verify that tabu search produced adequate speed and quality for use in the industrial setting in which we are currently interested. Our results indicate that a mainframe computer should be used in practice. Because the test cases were generated in such a way as to represent problem instances that in some sense are harder to optimize than the ones we will face, we are satisfied that the algorithm can be used as the engine for a DSS. Our second goal was to establish a benchmark for this problem formulation. The problem of sequencing with deadlines, sequence dependent setup times, and filler work has not appeared in the literature. We think it is an important problem and have shown that a reasonable heuristic exists; perhaps better solution methods can be found. In many cases, variations on the problem may be appropriate. Some of these are discussed in the next section.

6. Extensions

The use of tabu search as a basis for optimization allows easy modification of the objective function. For example, a nonlinear discontinuous penalty for setups per unit time has been added to the objective function used in the circuit board plant. The presence of earliest start times (ready times) and inserted idle time (scheduled starts) can also be accommodated.

6.1. Earliest Start Times

Various situations may give rise to ready times. For example, if the *single machine* to be sequenced is in a production facility, the presence of WIP in the plant or engineering release times can invalidate the assumption that all tasks are available at time zero.

In this case, define E_i as the earliest start time for job i and modify the anticipated completion time function used in the definition of (P) to be:

$$a_{\sigma}(i) = \max \left(\sum_{j=1}^{i-1} a_{\sigma}(j), E_{\sigma(j)} \right) + S(K_{\sigma(j-1)}, K_{\sigma(j)}) + T_{\sigma(j)}$$

and make the corresponding change to the definition of a_{σ} used in the cost functions.

The algorithms described go through without modification except that within each family, ready times must be increasing with increasing deadlines in order to prove Proposition 1. If there are ready times for a significant number of the jobs, then the search space can be reduced using the dominance results of Erschler, Fontan, Merce, and Roubellat (1983).

6.2. Inserted Idle Time

From a mathematical standpoint, it is obvious that one can reduce holding costs by inserting idle time between tasks. This is done by scheduling the start of work. The formulation given assumes that a job will start as soon as the job sequenced ahead of it

finishes. If holding costs are significant and the deadlines are not “tight,” it may be desirable to delay the start of each job until the last possible moment. In this case, the cost computations must be altered. After the anticipated completion times for every job in the sequence have been calculated, they must be modified to be

$$a'_{\sigma(i)} = a_{\sigma(i)} - \min_{i < j \leq N} (D_{\sigma(j)} + a_{\sigma(j)}).$$

The cost functions must be modified to use a' .

7. Conclusion

We have presented a general formulation of the single machine sequencing problem that addresses problems related to significant setup times. The algorithm presented considers tradeoffs between holding costs, setup costs, and increased revenue. The ability to explicitly indicate what types of work can be used to increase revenue is important when sequencing for significant setups and can be accomplished using the methods described here. This moves task sequencing from the cost minimization arena to profit maximization.

Appendix. Test Problem Generator

The sequences are randomly generated with several meaningful macroscopic parameters. These are as follows:

- a. Expected fraction of jobs that are filler ($F_F = 0.1$).
- b. Target Number of families ($N_F = 5$).
- c. Number of jobs in the sequence ($N_J = 30$).
- d. Average process time as a fraction of the average setup time ($F = 1.0$).
- e. Target number of setups in the sequence ($N_S = 15$).
- f. Setup matrix, i.e., time to setup between family i to family j

$$S = \begin{pmatrix} 0 & 50 & 50 & 100 & 200 \\ 50 & 0 & 70 & 100 & 200 \\ 70 & 70 & 0 & 100 & 200 \\ 100 & 100 & 50 & 0 & 100 \\ 200 & 200 & 200 & 100 & 0 \end{pmatrix}.$$

Due to the way the procedure works, target values are often an overstatement of the actual values for the sequences generated.

To create the sequences we must compute, for each job j , the completion time, the part number, the job quantity, and the job deadline. This is accomplished in the following steps:

- 1. Compute the average setup size from the setup matrix, \bar{S} .
- 2. Compute the “Time Horizon,” T as,

$$T = \bar{S}(N_S + FN_J).$$

- 3. Compute the probability of setup as, $P = N_S/N_J$.
- 4. Generate the completion dates, C_j , as uniform random integers over the range $[0, T]$.
- 5. Make the schedule tight by setting T equal to the last completion date.
- 6. Assign part numbers and quantities to the jobs on a single pass through the set of completion dates. At this step we generate a uniform $(0, 1)$ random number and use (P) to determine whether a setup has occurred. If there is a setup, we choose the next part number with equal probability over the other part numbers. If, however, there is insufficient time for the setup to occur, we continue with the current part number. For this reason, there may be fewer setups than the specified value (N_S).
- 7. Assign deadlines to the jobs on a single pass. Potential deadlines are generated as uniform random integers on $[0, T]$. If the potential deadline is greater than or equal to the completion date, it becomes the deadline, otherwise, the completion date becomes the deadline. In this way, the deadlines for the optimal sequence are tight but EDD is not necessarily optimal.

8. Assign filler or firm status on a single pass. For each job, the status is set to filler if a uniform real on $[0, 1]$ is less than F_F . The weight for each filler job is set to be the quantity for the order.

Because it is known that all filler jobs can be sequenced, the optimal objective value is known. In order to measure the relative “tightness” of the sequences generated, we compute a *EDD capacity utilization* defined as the makespan for a sequence arranged according to earliest due date divided by the last deadline. The EDD capacity utilizations for the tests we used were nearly uniformly distributed between 1.1 and 1.5.

References

- BAKER, K. R. (1974), *Introduction to Sequencing and Scheduling*, John Wiley and Sons, New York.
- BAKER, K. R. AND G. D. SCUDDER (1990), "Sequencing with Earliness and Tardiness Penalties: A Review," *Opns. Res.*, 38, 22-36.
- BRUNO, J. AND P. DOWNEY (1978), "Complexity of Task Sequencing with Deadlines, Set-up Times and Changeover Costs," *SIAM J. Comput.*, 7, 393-404.
- CONWAY, R. W., W. L. MAXWELL AND L. W. MILLER (1967), *Theory of Scheduling*, Addison-Wesley, Reading, Massachusetts.
- DRISCOLL, W. C. (1975), "Scheduling Production on One Machine with Changeover Times," invited paper given at the Fall 1975 ORSA-TIMS Joint National Meeting.
- ERSCHLER, J., G. FONTAN, C. MERCE AND F. ROUBELLAT (1983), "A New Dominance Concept in Scheduling n Jobs on a Single Machine with Ready Times and Due Dates," *Opns. Res.*, 31, 114-127.
- FRENCH, S. (1982), *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*, John Wiley and Sons, New York.
- GOLDRATT, E. M. AND R. E. FOX (1986), *The Race*, North River Press, Croton-on-Hudson, New York.
- GLOVER, F. (1989a), "Artificial Intelligence in Heuristic Solution Methods," In *Proc. 1989 Annual Meeting of Dec. Sci. Inst.*, 16-23.
- (1989b), "Tabu Search—Part I," *ORSA Journal on Computing*, 1, 1989b 190-206.
- (1989c), "Candidate List Strategies and Tabu Search," Technical Report, Center for Applied Artificial Intelligence, Box 419, University of Colorado, Boulder, CO, 80309, July, 1989.
- (1990), "Tabu Search—Part II," *ORSA Journal on Computing*, 2, 4-32.
- HALL, W. R. (1983), *Zero Inventories*, Dow Jones-Irwin, Homewood, IL.
- JOHNSON, D. S., C. R. ARAGON, L. A. MCGEOCH AND C. SCHEVON (1989), "Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning," *Opns. Res.*, 37-6, 865-892.
- KANET, J. J. (1988), "MRP 96: Time to Rethink Manufacturing Logistics," *Prod. and Inv. Mgt. Jour.*, 29, 57-61.
- AND D. P. CHRISTY (1984), "Manufacturing Systems with Forbidden Early Order Departure," *Int. J. Prod. Res.*, 22, 41-50.
- KARMAKAR, U. S. (1987), "Lot Sizes, Lead Times and In-Process Inventories," *Mgmt. Sci.*, 33, 409-423.
- LAGUNA, M., J. W. BARNES AND F. GLOVER (1990), "Scheduling Jobs with Linear Delay Penalties and Sequence Dependant Setup Costs and Times Using Tabu Search," Research Report, Department of Mechanical Engineering, The University of Texas-Austin, 1990.
- MONDEN, Y. (1983), *Toyota Production System: Practical Approach to Management*, Industrial Engineering and Management Press, Norcross, GA.
- MONMA, C. L. AND C. N. POTTS (1989), "On the Complexity of Scheduling With Batch Setup Times," *Opns. Res.*, 37, 798-804.
- RYAN, J. (1989), "Final Report of the Mathematics Clinic: Heuristics for Combinatorial Optimization," Mathematics Department, University of Colorado, Boulder, CO.
- SKORIN-KAPOV, J. (1990), "Tabu Search Applied to the Quadratic Assignment Problem," *ORSA Journal on Computing*, 2, 33-45.
- SPEARMAN, M. L., D. L. WOODRUFF AND W. J. HOPP (1990), "CONWIP: A Pull Alternative to Kanban," *International Journal of Production Research*, 28, 879-894.
- TRIGEIRO, W. W., L. J. THOMAS AND J. O. MCCLAIN (1989), "Capacitated Lot Sizing with Setup Times," *Mgmt. Sci.*, 35, 353-356.